SOA Infrastructure Tools

Concepts and Methods

Acknowledgment

The research presented in this paper was partially supported by the European Union in the scope of the European Regional Development Fund program no. POIG.01.03.01-00-008/08.

POZNAŃ UNIVERSITY
OF ECONOMICS

SOA Infrastructure Tools
Concepts and Methods

Editors
Stanisław Ambroszkiewicz
Jerzy Brzeziński
Wojciech Cellary
Adam Grzech
Krzysztof Zieliński

Poznań University of Economics Press
Poznań 2010

Coordinator of IT-SOA
Scientific Network: Krzysztof Zieliński

Editors: Stanisław Ambroszkiewicz
Jerzy Brzeziński
Wojciech Cellary
Adam Grzech
Krzysztof Zieliński

Reviewers: Alex Galis
Claude Godart

Technical Editor: Anna Zielińska-Krybus
Layout: Donata Latusek
Cover Design: Bartosz Kwolek

ISBN: 978-83-7417-544-9

Publisher: POZNAŃ UNIVERSITY OF ECONOMICS PRESS
ul. Powstańców Wielkopolskich 16, 61-895 Poznań, Poland
tel. +48 61 854 31 54, +48 61 854 31 55, fax. +48 61 854 31 59
www.wydawnictwo-ue.pl, a-mail: info@wydawnictwo-ue.pl
Correspondence address:
al. Niepodległości 10, 61-875 Poznań, Poland

Printed and bound in Poland by ZPW Pozkal

Contents

Introduction

This book contains a collection of papers which present concepts central to the research performed during the first year of the Project funded by the Polish Ministry of Science and Higher Education and supported by th European Union in the scope of the European Regional Development Fund program no. POIG.01.03.01-00-008/08.

The Project Consortium consists of five academic institutions: Institute of Computer Science of the Polish Academy of Sciences, Poznań University of Economics, Poznań University of Technology, Wrocław University of Technology and AGH University of Science and Technology (coordinator).

The strategic goal of the Project is to facilitate scientific research into innovative methods and tools for practical application of the Service Oriented Architecture (SOA) paradigm in the development of modern IT solutions. The project aims to promote the competitiveness of Polish businesses, enhance the national e-Economy and assist in further development of the information society.

The Project consists of two phases, each divided into four tasks. Work reported upon in this book concerns Task 1 (Conceptual studies of technologies, methods and tools), and Task 2 (Experimental implementation of methods and tool prototypes). Thus, the book focuses on the initial status of the proposed methods and tools, all of which will be further elaborated during the course of the Project.

The 17 chapters of this book can be characterized as follows:

Chapter 1 describes the Enterprise Service Bus (ESB), an extension of the Service Oriented Architecture (SOA) providing faster and cheaper integration of existing IT systems. ESB vendors do not adopt a single specification, which creates problems with enterprise-wide harmonization of governance processes. This is the primary reason behind extending the ESB model to cover modern SOA infrastructures. The study introduces the extended ESB model, encompassing such aspects as the concept of adaptive ESB, monitoring patterns for distributed ESB environments, adaptive services, BPEL engine monitoring and governance. This model positions ESB as an intermediary technology, simplifying the provision of core and extended functions.

Chapter 2 reports on virtualized infrastructures, an innovative execution environment for SOA applications that consists of a set of infrastructural services supporting the application lifecycle. These services are essential for provisioning, discovery, monitoring, management and operation of virtualized execution resources deployed over physical infrastructures. This chapter describes the concept of service-oriented infrastructures (SOI) and explains its role in the context of SOA application deployment and execution. The architectural elements and

their fundamental functionality which needs to be implemented to support the managed, predictive and adaptive behavior of SOI, are specified.

Chapter 3 — SOA management introduces new challenges and problems, heretofore unknown in legacy distributed systems. A vast amount of effort is being invsted in all aspects of SOA management, mainly in areas related to Web Service standards and technologies. Nevertheless, the REST architecture still lacks methods, tools and standards when it comes to management. In this chapter, we discuss selected aspects of management: performance, reliability and security.

Chapter 4 — This chapter reports on ongoing work on transaction support for SOA within three separate subprojects. The chapter starts with a discussion of common goals. Subsequently, the discussion is streamlined into three main parts: extended business transactions, rollback-recovery service and distributed software transactional memory for SOA.

Chapter 5 — This chapter describes the concept of replication tools for SOA-based systems. Replication is expected to ensure availability, atomic consistency, and partition tolerance; however, at most two of these features can be guaranteed. The proposed mechanisms cover various combinations of these features: the replicated state machine approach guarantees strict consistency and availability under the assumption that messages are successfully delivered, while the SAS system for mobile ad-hoc networks aims at ensuring availability at the expense of consistency.

Chapter 6 — The chapter proposes a conversion of the problem of automated Web Service composition to the problem of building a graph of worlds consisting of formally defined objects. Basic rules for defining ontologies for service execution environments are presented, where automatic reasoning about sequences of service calls can result in fulfilling user requirements. Such requirements can be specified in a fully declarative language, without any prior knowledge about services. In turn, the services are treated as independent "black boxes", performing their activities regardless of the flows in which they participate. The requirements and descriptions of services are the only source of knowledge used to generate abstract plans. The above planning process is the first phase of automated composition. The chapter also presents a tool implementing the proposed composition algorithm, along with some experimental results.

Chapter 7 — As business applications are progressing towards an "Internet of Things", more flexible and adaptive concepts for the integration of enterprise services with smart embedded devices are necessary. This chapter addresses two related aspects: (i) exposing embedded devices as Device Services, and (ii) implementation of embedded devices as hardware components without any software elements. We begin with an outline of the SODA (Service Oriented Device Architecture) approach, supporting integration of Service Devices with ESB. Subsequently, an FPGA-based implementation of Device Services is presented. An extension of the SCA programming model to sensor networks is also shown. Finally, technical

problems related to exposing mobile phone functionality in terms of services and accessing services on mobile devices are described.

Chapter 8 — This chapter introduces the concept of dynamic, searchable conglomerates of Internet of Things (IoT) devices to be accessed by SOA services. The main goal is to provide a single, consistent solution for the following problems: (1) efficient searching for IoT devices to be used "at the place", (2) dynamic (re)configuration of device groups to fulfill extended needs by providing complex functionality, (3) dynamic programming of device behavior, and (4) efficient support for IoT devices with handicapped access and/or functionality. The solution is based on a dedicated ontology covering a classification of IoT devices and their capabilities, and also on system functionality requirements, seen from the viewpoint of SOA services. The ontology is to be interpreted at runtime, enabling contextual selection of optimal devices according to dynamically changing circumstances, interaction loci, user/SOA service grants and restrictions, etc.

Chapter 9 — Recent evolution of software architectures has resulted in rising prominence of the Service Oriented Architecture (SOA) concept. This architecture paradigm facilitates construction of flexible service systems. Services can be deployed in distributed environments, executed on various hardware and software platforms, reused and composed into complex services. This chapter proposes an efficient computational and communication resource allocation model for complex service requests. The requests are examined in accordance with the SOA request description model. Their functional and non-functional requirements are analyzed in conjunction with monitoring of communication links and performance data, in order to properly distribute requests and allocate both services and resources.

Chapter 10 — In this chapter a method for the QoS-aware complex service composition in SOA-based systems is presented. The proposed service composition process consists of three stages: complex service structure, scenario and plan composition. Complex service structure involves a set of required functionality aspects and precedence relations, and is derived directly from a service level agreement. Each service scenario is derived from the service structure by choosing an optimal (with regard to a certain quality criterion) order of execution. Finally, the service plan emerges from the scenario by selecting the best versions of atomic services for delivery of the required functionality. Methods for complex service scenario and execution plan optimization are proposed.

Chapter 11 — In this chapter a method for assessment and optimization of the security level of composed Web Services (assuming a layered security architecture and multiagent approach) is presented. As security evaluation requires a precise definition of the set of evaluation criteria, the key criteria for each functional layer of SOA have been proposed. An information fusion scheme based on the Subjective Logic formalism is used to integrate information derived from different layers and agents. The framework also assumes that opinions about the credibility of agents may be used in conjunction with security assessment results.

Chapter 12 — Modern IT infrastructures provide technological means for supporting collaborations between various organizations through several virtualization solutions. These can include simple communication software, collaboration by means of shared data access or advanced teleconferencing. Given the high level of heterogeneity of IT infrastructures used by various organizations, the problem of setting up distributed collaborations, ensuring proper privacy and security as well as enforcing adherence to a set of rules, remains a key issue. This is where the concept of Virtual Organizations comes in. This chapter presents the Framework for Intelligent Virtual Organizations (FiVO) which provides several features supporting the process of creating and managing Virtual Organizations.

Chapter 13 — The Virtual Organization Breeding Environment (VOBE) is an accepted concept in the research area of collaborative networks. Existing VOBEs are based on an infrastructure providing only limited support for efficient integration of VOBE members and virtual organization partners on both technical and business levels. Thus, the Service Oriented Architecture (SOA) is proposed as an approach to implementing VOBE. A SOA-compliant VOBE is called a Service-Oriented Virtual Organization Breeding Environment (SOVOBE). A SOVOBE is systematically organized around the concept of services, which are not limited to Web Services, but include services performed by humans (organizations). In this chapter a set of core services is specified, provided by SOVOBE to support SOVOBE members and virtual organization throughout their lifecycle. The core services include the Competence Management Service, the Social Network Service, the VO Collaboration Service, the VO Creation Service, and the VO Monitoring Service.

Chapter 14 — e-Economy calls for public authorities to apply IT solutions in servicing their clients - businesses and citizens. To achieve this goal, authorities must base their work on an architecture that enables collaboration and integration of heterogeneous IT systems which are the norm when considering public agencies as a whole. An approach to organizing public authorities around the Service Oriented Architecture concept is presented in this chapter.

Chapter 15 — The classic version of the SOA paradigm has its origins in software engineering. From the point of view of e-business processes, the concept of a service is not necessarily related to RPC (as in SOA) where a service is passive and waits for a client to invoke it. In other words, a service may be active and in search for client tasks that can be accomplished. This corresponds to reverse auctions in business practice. A business service has a well-founded structure where its operations are related to one another. These relations cannot be expressed in WSDL. Hence, the concept of an SOA service should be discussed. Generally, from the e-business perspective, the problems of service architecture and communication protocols seem to be important in the scope of business processes. These problems must be solved by providing standards necessary for creating open, heterogeneous

and scalable systems for complex e-business processes. A revision of the SOA paradigm related to the above issues is presented in this chapter.

Chapter 16 — The problem of integrating heterogeneous back-end platforms using the SOA paradigm has been widely addressed in a number of research and development projects. However, in some application domains (such as the construction industry) there is an additional requirement: integration of heterogeneous front-end platforms — from desktop PCs to mobile PDA. In this chapter, a new method of building adaptable user interfaces for SOA applications is described. In the adaptation process the displayed content is adjusted, the best way of presenting content is selected and interaction methods are adapted to the capabilities of particular front-end devices, as well as preferences and privileges of the end users. In this way, the adaptable user interfaces proposed in this chapter make SOA services easily accessible on any device.

Chapter 17 — In this chapter we present the idea of agent-based SOA systems which apply various solutions and mechanisms from the multi-agent domain to ensure flexible construction and execution of business processes. We focus on the problem of composition and execution of complex processes consisting of Web Services. These may be based on the notion of negotiations, inspired by multi-agent domains, or rule-based agents selecting the most appropriate business process from those described using BPEL. We use the management and coordination activities performed in the scope of an integrated rescue action system for our first demonstration of the presented algorithms and prototype pilot tools.

Kraków, April, 2010 Krzysztof Zieliński

ESB — Modern SOA Infrastructure

Tomasz Masternak, Marek Psiuk, Dominik Radziszowski, Tomasz Szydło, Robert Szymacha, Krzysztof Zieliński, and Daniel Żmuda

Department of Computer Science, DSRG,
Al. Mickiewicza 30, 30-059 Kraków, Poland
{tomasz.masternak,marek.psiuk,dr,tomasz.szydlo,robert.szymacha, krzysztof.zielinski,daniel.zmuda}@agh.edu.pl
http://www.soa.edu.pl

Abstract. The traditional Enterprise Service Bus (ESB) concept supports Service Oriented Architecture (SOA) in providing faster and cheaper integration of existing IT systems. The ESB technology has arisen from principles like Enterprise Application Integration (EAI), but it is still not sufficient to satisfy SOA requirements with respect to governance. Vendors of ESB have not adopted a single specification and this creates problems with enterprise-wide harmonization of governance processes. This is a motivation for an extended ESB model being a Modern SOA Infrastructure. The study introduces an extended ESB model, which encompasses among others aspects like the concept of adaptive ESB, monitoring patterns for distributed ESB environment, adaptive services, and BPEL engine monitoring and governance. This model positions on ESB as an intermediary technology for providing core and extended functions, making it easier to use in world-wide IT environments.

Keywords: Enterprise Service Bus, services monitoring, QoS management, services adaptability

1 Introduction

The Enterprise Service Bus (ESB) is a natural evolution and convergence of Enterprise Application Integration (EAI) and Message Oriented Middleware (MON) implementing the Java Message Service (JMS). It is also inspired by the Web Service technology, leading to the following traditional definition [1]: ESB is an intermediary that makes a set of reusable business services widely available.

The traditional ESB leverages technology advancements in the scope of these platforms, establishing a distinct form of middleware that can: (i) support Service Oriented Architecture (SOA) integration requirements, (ii) avoid vendor lock-in by using industry standards, and (iii) standardize integration by utilizing broker functions and cross-application communication based on open standards. Some of the early implementations of ESB were constructed around these principles.

As a whole, traditional ESB implementations are divergent from the typical deployment descriptor/packaging standard. This is due to the fact that ESB vendors have not adopted a single specification, such as e.g. Java Business Integration (JBI) proposed for Java-based ESB implementation, leading to situations where multiple ESBs are deployed within enterprise boundaries. Some implementations are specific to a business unit or application while others are tasked with external communications. This creates problems with enterprise-wide harmonization when attempting to distribute operations and configuration responsibilities for enabling collective governance. A good example is the increasing role of OSGi [5], which applies the service orientation principles within the boundary of a single Java Virtual Machine (JVM), or the Service Component Architecture (SCA) accepted as a common framework for wiring components together.

In this context the traditional ESB does satisfy SOA requirements with respect to governance, regardless of the middleware or technologies used to implement and host services. Expectations in this area are growing as large enterprises have recently begun to exploit the ESB Federation concept. This leads to an extended ESB model, defined as follows [2]: ESB is an intermediary that provides core functions, making a set of reusable services widely available along with extended functions that simplify the use of the ESB in real-world IT environments.

The goal of this chapter is to present the most important aspects of extended ESB as a modern SOA infrastructure and to summarize the work performed in this area by the DCS team at AGH UST. Extended ESB is investigated in reference to the SOA Solution Stack proposed by IBM. This clarifies the role of integration technologies in SOA-based system construction and deployment. We introduce extended ESB technologies as elements of the ESB compound pattern [3]. The functionality of each element of this pattern is briefly specified, defining a basis for the more detailed analysis presented in subsequent sections. The concept of adaptive ESB is introduced first as a key aspect of the presented study. Then, a detailed analysis of the adaptive ESB requirements leads to the specification of its architectural and functional models. The models facilitate the definition of mechanisms necessary for implementing an adaptive ESB. As these mechanisms cover the most important aspects of ESB governance, they are very much in line with the functionality of extended ESB. The monitoring ESB activity is also described as it plays a fundamental role in making governance decisions. The proposed monitoring system is very generic and refers not only to JBI-compliant ESB monitoring but also to OSGi service tracing. The mechanisms required by the different adaptability strategies are described in Section 6. A related concept is the construction

of adaptive services. Such services, based on the Service Component Architecture, are proposed in Section 7. Suitable extensions of the SCA execution infrastructure are implemented for this purpose. As orchestration processes play a crucial role in the development and deployment of SOA applications, Business Process Execution Language (BPEL) engine monitoring and governance is also investigated (Section 8). Finally, since ESB is an integration technology originating from MOM, the problem of message brokering is analysed.

2 SOA Solution Stack Reference Model

SOA application development and deployment should be considered in the context of the SOA Solution Stack (S3) proposed by IBM, which provides a detailed architectural definition of SOA split into nine layers. This model is depicted in Figure 1. Each layer has a logical and physical aspect. The logical aspect includes all the architectural building blocks, design decisions, options, key performance indicators, and so on. The physical aspect covers the applicability of each logical aspect in reference to specific technologies and products and is out of scope of our analysis.

The S3 model is based on two general assumptions:

— The existence of a set of service requirements which are both functional and non-functional, which collectively establish the SOA objective. Non-functional service aspects include security, availability, reliability, manageability, scalability, latency, and the like.
— A single layer or some combination of layers can fulfill specific service requirements and for each layer the service requirements are satisfied by a specific mechanism.

The nine layers of the S3 stack are as follows: Operating Systems, Service Components, Services, Business Process, Consumer, Integration, QoS, Information Architecture, and Governance and Policies. There is no separate layer for business rules. Rather, business rules cut across all layers. The business process and governance layers intersect in defining the rules and policies for a given business process.

A brief characteristic of each layer is presented below as a background for further discussions contained in this chapter.

Operating systems — This layer includes all application and hardware assets running in an IT operating environment that supports business activities (whether customized, semi-customized, or off-the-shelf). Because the layer consists of existing application software systems, implementing the SOA solution leverages existing

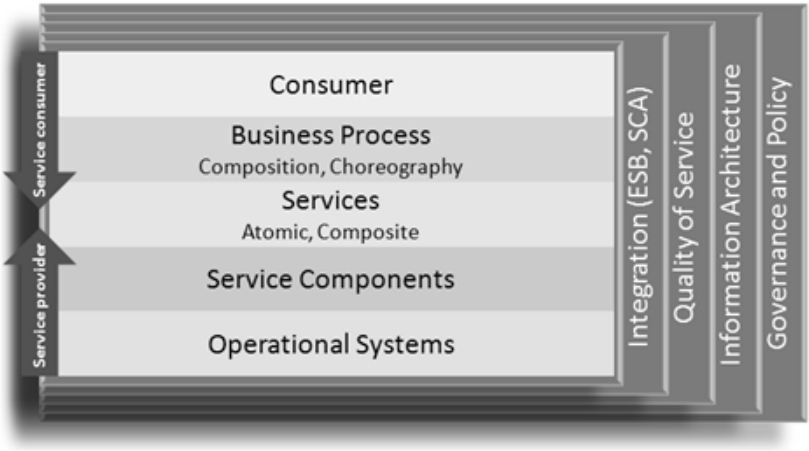

Fig. 1. SOA Solution Stack Model

IT assets. Nowadays this layer includes a virtualized IT infrastructure that results in improved resource manageability and utilization.

Service components — This layer contains software components, each of which is the incarnation of a service or operation on a service. Service components reflect both the functionality and QoS for each service they represent. Each service component:

— provides an enforcement point for ensuring QoS and service-level agreements;
— flexibly supports the composition and layering of IT services;
— hides low-level implementation details from consumers.

In effect, the service component layer ensures proper alignment of IT implementations with service descriptions.

Services — The service layer consists of all the services defined within SOA. In the broadest sense, services are what providers are offering and what consumers or service requesters use. In IBM model [18] S3, however, a service is defined as an abstract specification of one or more business-aligned IT functions. The specification provides consumers with sufficient detail to invoke the business functions exposed by a service provider — ideally in a way that is platform independent. Each service specification must include:

— an interface specification — a collection of operation signatures;
— service endpoint information — the network location to which invocation messages are sent;
— invocation protocol details;

— service semantics, such as measurement units and business context.

It is necessary to point out that services are implemented by assembling components exposed by the Service Component layer. The Service Component Architecture plays an important role within this layer as a promising concept.

Business process — In this layer, the organization assembles the services exposed in the Services layer into composite services that are analogous to key business processes. In the non-SOA world, these business processes are similar to custom applications. On the other hand, SOA supports application construction by introducing a composite service that orchestrates the information flow among a set of services and human actors.

Consumer — The consumer layer handles interaction with the user or with other programs in the SOA ecosystem.

Integration — This layer integrates layers 2 through 4. Its integration capabilities enable mediation, routing and transporting service requests from the service requester to the correct service provider. These capabilities include, but are not limited to, those found in ESB and will be further explained in the following sections.

Quality of service — Certain characteristics inherent to SOA exacerbate well-known IT QoS concerns: increased virtualization, loose coupling, composition of federated services, heterogeneous computing infrastructures, decentralized service-level agreements, the need to aggregate IT QoS metrics to produce business metrics, and so on. As a result, SOA clearly requires suitable QoS governance mechanisms.

Information architecture — This layer encompasses key considerations affecting data and information architectures required to develop business intelligence through data marts and warehouses. The layer includes stored metadata content required for data interpretation.

Governance and policies — The governance and policies layer covers all aspects of managing the business operations' lifecycle. This layer includes all policies, from manual governance to autonomous policy enforcement. It provides guidance and policies for managing service-level agreements, including capacity, performance, security, and monitoring. As such, the governance and policies layer can be superimposed onto all other S3 layers. From a QoS and performance metric perspective, it is tightly connected to the QoS layer. The layer's governance framework includes service-level agreements based on QoS and key process indicators, a set of capacity planning and performance management policies to design and tune SOA solutions as well as specific security-enabling guidelines for composite applications.

3 Extended ESB Functionality

The presented S3 model provides a general landscape for the extended ESB functionality specification.

This functionality emerges even more clearly in the context of ESB Federations used by large enterprises. An ESB Federation consists of many ESB instances, which are organized into several groups:

— working with service units — focus on loose coupling;
— working with networks — focus on resource virtualization;
— linking services together — focus on agile composition;
— doing 1–3 economically — focus on scalability across several dimensions;
— enabling new offerings based on the new technology — focus on the future.

As enterprises deploy multiple ESB instances, they need to provide an infrastructure to manage, secure, mediate, and govern these instances. The capabilities inherent in an ESB Federation solution are delivered through a standard-based SOA infrastructure. These capabilities include:

— Security — ensuring the privacy, authenticity, authorization, non-repudiation and auditing of all messages moving within and between ESB instances and other service-oriented applications. This also includes decoupling the security management model from the application programming model;
— Mediation — many applications and ESB instances will use and support different standard protocols and technologies along with different invocation, synchronicity, reliability, and security models. An ESB Federation solution provides policy-based mediation between the various synchronicity, reliability, programming and security models, technologies, messaging styles and standards, ensuring seamless interoperability.
— Management — as ESB instances proliferate, enterprises will require a holistic view of the transactions that traverse their platforms and applications. An ESB Federation solution provides monitoring, management, SLA and alert reporting capabilities for all managed platforms including ESB instances.
— Governance — the ESB Federation solution provides a consistent policy definition, implementation, administration, management and enforcement mechanism for all SOA-enabled platforms and ESB instances within the enterprise. It verifies that all services implement and enforce the same set of policies, and can comply with the policies that will be enforced downstream.

The ESB Federation solution is an infrastructure solution. It is not an ESB for multiple ESB instances — rather, it simply provides core policy-based infrastructure services fir heterogeneous ESB instances. A successful deployment of the ESB Federation requires the presence of suitable capabilities in each ESB instance.

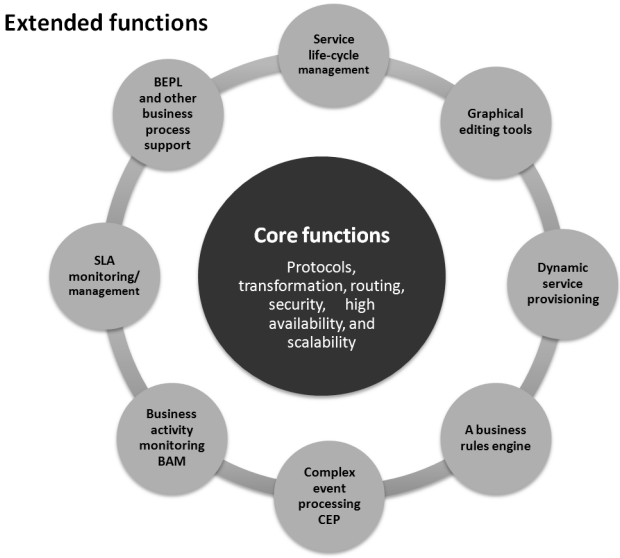

Fig. 2. Core and extended ESB functions

The list of core functions, which provide the basic operational capabilities of the ESB is rather long:

1. Support for multiple protocols.
2. Protocol conversion.
3. Data transformation and data-based routing.
4. Support for multiple connectivity options.
5. Support for composite services through lightweight orchestration.
6. Support for multiple standard business file formats.
7. Integrated security features.
8. Comprehensive error handling mechanisms.
9. Support for both synchronous and asynchronous operations.
10. Highly available and scalable infrastructure.
11. Support for many options in each of the above categories.
12. Extensibility.

Extended functions are those ESB capabilities that lie beyond the operational machinery listed above. They make services widely available to consumers and offer broad support for SOA application development, provisioning and running. The development of these extended functions is currently the focus of attention by ESB vendors. Figure 2 summarizes the most important extended functions.

Table 1. Extended ESB functions

Graphical editing tools	Graphical editors for ESB flows such as itineraries or lightweight orchestration.
Service Level Agreement (SLA) monitoring and management	Reporting and registration of Quality of Service (QoS) and Quality of Experience (QoE). Load balancing to meet SLA. Endpoint security and management.
BPEL and other business process support	Design, simulation, and execution of business processes using BPEL.
BPEL engine activity monitoring	Business Activity Monitoring (BAM) definition of business-centric metrics (Key Performance Indicators) [6] and their presentation in near-real time. Generating alerts when KPIs cross specified thresholds.
Service lifecycle management	Service definition, implementation and installation.
Dynamic service provisioning	Dynamical provisioning of new ESB operations. Modification of flows without restarting ESB components. Dynamic control over the number of running service instances in accordance with SLA targets.
Complex event processing (CEP)	Ability to define and recognize complex events during business process execution.
Business rule engine (BRE)	Ability to specify and impose policies related to business or system activity. Easy integration with CEP and EDA.

A more detailed specification of these functions is presented in Table 1. It is necessary to point out that many related requirements are still not fully satisfied and require further research. The following sections elaborate on this issue in more detail.

From the software engineering point of view ESB can be treated as a Compound Design Pattern, which in turn represents collection-related patterns. The relationship between these patterns is immaterial as compound patterns focus on results of their combined applications. A compound pattern represents a set of patterns that are applied together to a particular application in order to establish a set of design characteristics.

Figure 3 shows the ESB compound design pattern. It is fairly complex as it comprises other patterns (such as the Service Broker) which are also compound patterns. Moreover, the ESB pattern can be considered as an element of the Canonical Schema Bus. Patterns connected by solid lines represent the core functionality of ESB, while remaining patterns address extended ESB capabilities. The functions of extended ESB, as shown in Table 1, cover the orchestration process. This

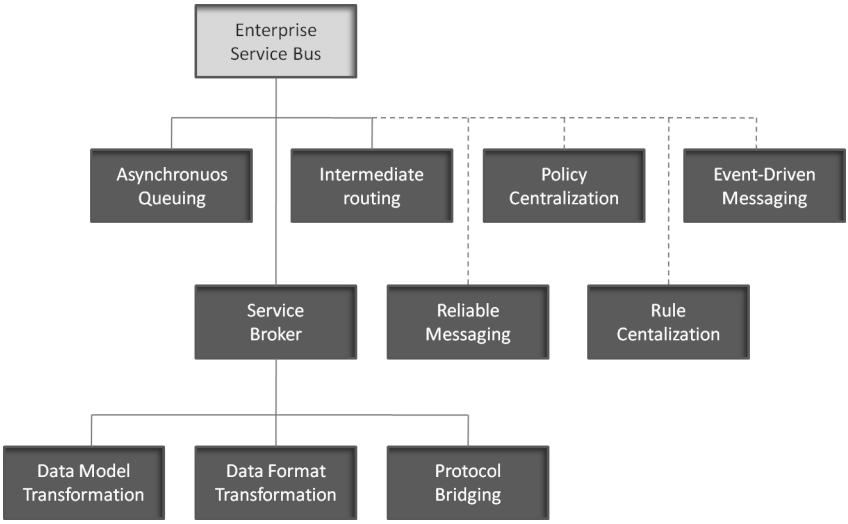

Fig. 3. ESB compound design pattern

leads to the conclusion that the extended ESB compound pattern should be further enhanced with Orchestration compound patterns, which play a crucial role in SOA.

4 Model of Adaptive ESB

The detailed analysis of SOA infrastructures presented in the previous sections shows that its elements should offer some level of adaptability. In fact, adaptability looms as a dominant aspect, cutting across all extended ESB functions. An *S3 Layer* is defined as a set of components, such as architectural building blocks, architectural decisions and interactions among components and layers. This definition emphasizes the existence of many options that can be subjected to architectural decisions taken during the SOA application design phase or postponed until runtime. A generic structure of an adaptive ESB functional element is shown in Figure 4. The policy engine perceives the system as an abstraction exposed by the exposition layer and is able to perform a set of actions on this abstraction. The abstraction can be defined as a complex service execution model, or a simple set of services involved in execution. In both cases an adaptation policy has to be defined. This policy is used to represent a set of considerations guiding decisions. Each adaptation policy may express different goals of system adaptation, such as minimizing system maintenance costs or ensuring particular QoS parameters.

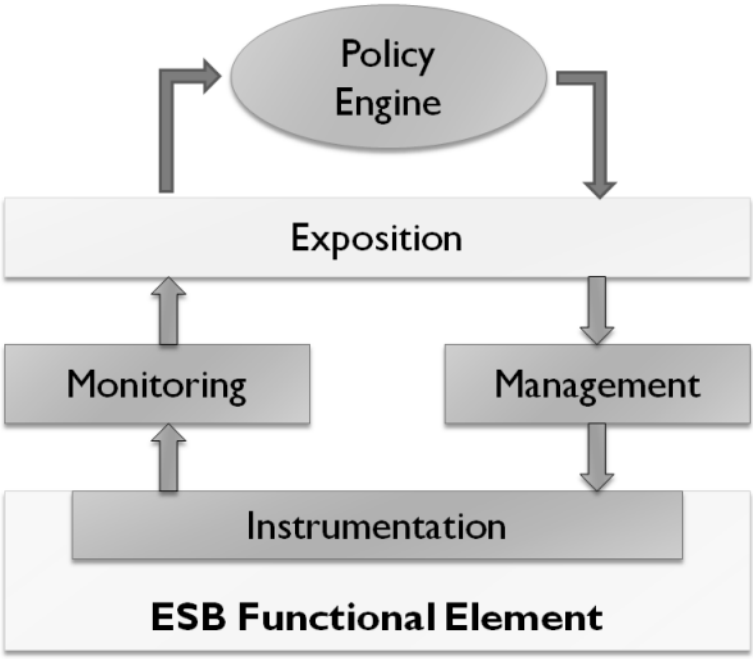

Fig. 4. Adaptive ESB functional elements

Design-time decisions cannot, however, take into account every context of service execution and lower-layer architectural configuration. Hence, runtime architectural decision processing is particularly important for the S3 stack. Each layer must therefore be equipped with suitable adaptability mechanisms enabling the enforcement of these decisions.

4.1 System Elements

The Adaptive ESB functional element is constructed around the well-known closed loop control schema. A fundamental part of this element is the adaptability loop which consists of the following blocks:

— **Instrumentation** — the Enterprise Service Bus is enriched with additional elements that gather monitoring data from the ESB and perform adaptability operations.
— **Monitoring** — determines system state and traces monitoring service invocations. The stream of monitoring data is processed by the Complex Event

Processor and listeners are notified of complex monitoring events. The monitoring layer exposes sensors for the *Exposition layer*.

— **Management** — components used to manage the execution of services deployed in the ESB and for lifecycle management of interceptors. The management layer exposes effectors for the *Exposition layer*.

— **Exposition** — components that expose the state and events occurring in the ESB as facts for the *Policy Engine Layer*. This layer also contains components used to model system behavior in the model domain.

— **Policy Engine** — rule engines responsible for service adaptation in order to meet high-level goals defined as policy rules.

A number of distributed adaptive elements can be managed by a global Policy Engine in accordance with a high-level strategy. The instrumentation layer enriches the ESB with additional elements. These elements provide adaptability transformations necessary to achieve adaptive ESB and are described in the following sections. The monitoring layer is responsible for supplying notifications of events occurring in the execution environment. As the volume of monitoring information gathered from the ESB could overwhelm the *Exposition layer*, events are correlated with one another and notifications are sent only about complex events. Complex Event Processing can be compared to an inverted database containing stored statements: as data arrives in real time, these statements are executed and notifications are sent to registered listeners. The policy engine layer analyses facts and infers decisions which are then implemented in the execution environment. Facts representing the state of the system or events occurring in the system are supplied by the exposition layer.

The approach presented in this section is a model-driven adaptation policy for the SOA. The system analyzes composite services deployed in the execution environment and adapts to QoS and QoE changes. The user composes the application in a chosen technology and provides an adaptation policy along with a service execution model. The architecture-specific service composition layer continuously modifies the deployed service in order to enforce the adaptation policy. The abstract composition, providing input for architecture-specific service composition, can be hidden and used only by IT specialists during application development. The system behavior is represented by the service execution model.

The composite service execution model is an abstraction of the execution environment. It covers low-level events and relations between services and exposes them as facts in a model domain for the *Policy Engine*. Decisions taken in the model domain are translated to the execution environment domain and then executed. The *Model Analyzer System* gathers monitoring data from the execution environment via monitoring and management agents.

This process can be described as architecture-specific service composition adaptation that is performed in order to achieve the required value of QoS or QoE guided by the adaptation policy. The adaptation loop addresses service selection,

binding protocol and interaction policy choices. Thus, the adaptability process mainly concerns integration mechanisms.

5 ESB Monitoring

The service orientation paradigm enables IT architects to identify independent parts of business logic and expose them as services. Additional service design methodologies [4] classify services into types and hierarchies, stressing the most important service attribute: composability. The composability of services significantly increases the potential size to which a fully operational service-based enterprise system can be extended. In order to provide sufficient adaptability to constant changes in business requirements occurring in such large and complex infrastructures, proper monitoring mechanisms have to be introduced. The Enterprise Service Bus is the most frequently used design pattern for enterprise service backbones. Therefore, ESB monitoring is a vital element of service-oriented IT systems and is essential for the Adaptive ESB Model (cf. Figure 4). This section defines the goal of ESB monitoring and presents the problems, which have to be solved in its context.

5.1 ESB Monitoring Background

Our study focuses on the SOA paradigm realized using Java-centric technologies; therefore we will focus on OSGi [5] and Java Business Integration (JBI) [6]. JBI is a Java-centric standardization of the ESB pattern while OSGi introduces a dynamic service-oriented modularization model in the context of a single JVM process. OSGi adds significant flexibility by transforming the flat dependency space into a well-structured hierarchical one and by introducing dynamic single-process service repositories. OSGi is often referred to as single-process SOA [7]. The JBI specification has not been widely adopted [8] and currently provides a point of reference rather than an actual ESB implementation standard. Modern ESB infrastructures tend to take OSGi as the basis for single-process SOA and extend it, by means of federations [9], to a distributed service backbone. Therefore our study begins with a definition of OSGi monitoring and assumes JBI as a point of reference in the construction of monitoring mechanisms for ESB Federations. Figure 5 depicts the monitoring structure in a service-oriented environment. Monitoring is divided into domains and perspectives. The infrastructure domain refers to monitoring of the system environment with focus on attributes of the operating system and any virtualization layer between the OS and actual hardware. The platform domain relates to the platform on which the container of a higher domain is executed.

JVM monitoring is the subject of the JVM platform domain on which this study focuses. The container domain is the actual element where the system part of OSGi and ESB monitoring is located. The monitoring of application-related service attributes takes place in the application domain and further evolves into monitoring of important Key Performance Indicators [10] in the business domain.

Fig. 5. Monitoring layers in a service-oriented environment

Thus ESB-based SOA monitoring spans the following domains: container (system monitoring), application (application monitoring) and business (business activity monitoring (BAM) [11]). In all domains monitoring can be performed either from the consumer's or provider's perspective. For ESB monitoring, both perspectives are important. The presented monitoring structure can be mapped onto layers of the S3 model (cf. Fig. 1). The container domain corresponds to the S3 Service Components and Integration, while application and business domains are counterparts of the S3 Services and Business Processes.

The goal of the presented ESB monitoring is to provide a complete solution for monitoring of both perspectives in the three ESB domains. The functional requirements of ESB monitoring can be summarized as follows:

— Gathering complete information on the OSGi and ESB level: monitoring service deployment – current service topology as well as service invocation parameters.
— Applying a design which enables transformation of system-level monitoring information with no semantics into enriched information providing business value.

— Ensuring flexible, declarative monitoring target specification, which can be changed at any time.
— Monitoring should adapt to changes in the distributed ESB environment in order to ensure fulfillment of the defined goals.

In addition to functional requirements there is also a set of nonfunctional design assumptions which influence the architecture to a significant degree. These assumptions are tightly related to the *monitoring scenario* concept, which specifies the desired monitoring outcome from the user's point of view (the following section provides more details about *monitoring scenarios*). The nonfunctional assumptions are as follows:

— Monitoring has to be selective: only information required to implement *monitoring scenarios* at a given should be gathered.
— Monitoring mechanisms should not influence the performance of elements, which are not currently monitored and any performance deterioration of monitored elements should be minimal.
— Monitoring data should be propagated only to locations which are needed to realize a given *monitoring scenario*.
— System reconfiguration in response to changes in *monitoring scenario* definitions should not take longer than several seconds.

These requirements outline a large problem space, which has to be explored. The most challenging aspects are selectivity and business activity monitoring. In order to satisfy selectivity requirements, the described approach introduces a methodology for mapping the *monitoring scenario* to a proper configuration of *interceptors* (cf. next section) which collect monitoring data. The first attempt at implementing BAM focuses on enriching the *monitoring scenario* with a business context and introduces *interceptors* capable of providing information with business value.

5.2 Monitoring Model

The general outline of ESB and OSGi monitoring, described in the previous section, introduces the need for *monitoring scenarios*. To describe such scenarios in compliance with all stated requirements the following aspects of ESB and OSGi monitoring have to be considered:

— topology monitoring — refers to monitoring of services, modules and containers interconnection in the SOA environment;
— parameter monitoring — gathering the values of predefined metrics such as performance, reliability, composability etc. Each metric installed in any of the

containers introduces a set of attributes which have to be defined in order to gather monitoring data;
— content monitoring — refers to monitoring particular service invocation parameters or business application errors.

In order to specify a *monitoring scenario* two steps have to be performed. First, the topology of the monitored container must be acquired. It is impossible to specify a *monitoring scenario* without proper information about its current topology state. In accordance with these assumptions, a scenario definition encompasses topology elements for which the scenario will be applied. Such a specification may be represented by a set of patterns, matching particular topology elements to their names. The second step of scenario specification is to define a metric, which will be applied to the predetermined topology elements. Each metric has an individual set of parameters. Information about the values of selected parameters has to be included in the *monitoring scenario* definition. One monitoring scenario can contain many metrics and topology elements — each pair is called a *monitoring subscription*.

Having defined a *monitoring scenario*, we need to specify a mechanism capable of flexibly plugging into the monitoring logic of OSGi and ESB. The proposed model introduces a non-invasive mechanism called the *interceptor socket*. In both cases, such a component must provide functionality for plugging interceptors and providing them with monitoring data. The interceptor socket is an element, which enables the monitoring system to be dynamically connected to the monitored environment. It seems reasonable to state that in the ESB environment interceptor socket mechanisms have to be capable of processing messages passed to services installed in a container. In the latter case we need to ensure that service invocations are intercepted along with their context (i.e. invocation parameters). Such a unified approach for both OSGi and ESB containers enables dynamic management and installation of interceptors. As can be seen in Figure 6, interceptors installed in sockets feed monitoring data to higher-level monitoring logic through the monitoring backbone. This backbone is one of the core components in every distributed monitoring system and can be the subject to extensive analysis.

In order to describe the possible methods of creating interceptor chains it is first necessary to describe what an interceptor is. The presented study introduces the concept of Interceptor as a Service (IaaS). The OSGi-based IaaS realization facilitates reconfiguration and management of interceptors while fulfilling one of the most important non-functional assumptions, i.e. selectivity of the monitoring model. Service-oriented interceptors can be divided into two groups: agnostic and business interceptors. The former category defines interceptors which do not need to analyze invocation parameters (content), but instead focus on information gathered from the invocation context (e.g. the processing time of a particular service). The latter category defines business interceptors, which analyze content (service

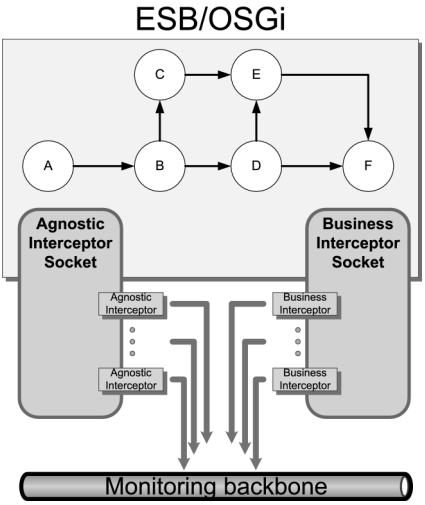

Fig. 6. Connection between the monitored environment and the monitoring backbone

parameters in OSGi or message content in ESB) and process such data in accordance with the implemented business logic. These two categories are organized into separate chains. Each chain is an ordered list of interceptors with a defined sorting strategy. It determines the order in which interceptors process data and pass monitoring results to the backbone.

The presented interceptors need to be created and configured in accordance with *monitoring scenario* details. *Monitoring subscriptions* of the current scenario are installed in the monitoring environment and serve as a basis for the creation and configuration of interceptor chains. Special interceptor configurations can cover services, which do not exist at subscription time. If a new service is activated and matches a given topological pattern, there is no need for reconfiguration. Such a service can be automatically monitored in accordance with specified metrics. *Monitoring scenarios* allow flexible transformation of business domain monitoring goals into low-level system monitoring configurations. The business analyst has to specify a set of *monitoring scenarios*, which have to be implemented in order to extract business domain data from system events. Those scenarios, along with sets of monitoring subscriptions, are propagated throughout the distributed ESB.

The most important (and most problematic) aspect is the distribution of specific *monitoring scenarios* in the distributed ESB environment. They not only have to be periodically propagated (in order to publish changes), but also synchronized to maintain the consistency of *monitoring scenario* definitions. On the

other hand, as described in the previous section, the solution must ensure *selectiveness* of monitoring capabilities. Fulfilling those goals is not an easy task and must be approached with care. Those aspects, along with the monitoring backbone design architecture (described earlier), will be the main direction of future work originating from the presented study and aiming at implementation of BAM in a Federated ESB environment.

6 ESB Adaptation Mechanisms

Monitoring ESB allows management actions to be performed. The presented concepts cover selected issues related to improving communication over ESB by separating traffic into several flows, which can then be independently handled. This leads to increased scalability and is required by many aspects of infrastructural functionality, including monitoring, data collection, management information distribution and security considerations.

The proposed mechanisms are compliant with existing integration technologies. Interceptor mechanisms enable dynamic control of service or component invocation, along with sensors and effectors necessary to close the adaptation loop.

As mentioned earlier, the Enterprise Service Bus is an integration technology that allows architects to compose applications with services using various communication protocols. ESB provides mechanisms for message normalization and routing between selected components. The proposed adaptation mechanism provides elements necessary to close the control loop for ESB. It is used for compositional adaptation in systems which modify service composition while retaining their overall functionality. ESB is suitable for implementation of such an adaptation, since one can modify message flows between components in accordance with high-level goals.

6.1 Sensors

An adaptive system should adapt itself in accordance to various goals, requiring several types of information from ESB. In most cases, this information will be disjunctive, so one would expect to deploy specialized types of sensors rather than generic ones. Interceptor design patterns fulfil these requirements, allowing interceptors to be deployed or undeployed at runtime.

A message is created in a service, passes through the specialized Service Engine and is sent to the Normalized Message Router (NMR), which reroutes it to a particular destination through the same components. Common usage of this concept includes:

— **QoS Measuring** — the functionality of monitoring interceptors is not limited to creating copies of messages sent through them, but may include more complex tasks, providing quantitative information related to service invocation. In the QoS interceptor, a message sent to a service is stored in the internal memory of the interceptor. When a response is generated, the previous message is correlated and response time is evaluated.
— **Tagging/Filtering** — interceptors are commonly used for message tagging and filtering.
— **VESB** — it is possible to implement the Virtual ESB (VESB) pattern, which can be compared to solutions currently used in computer network traffic management, such as VLAN [12].

6.2 Effectors

Adaptive software has to adapt itself to changes in the execution environment. This adaptation requires performing actions (via effectors) that modify the execution characteristics of a system. For any sort of system, a set of operations has to be defined, along with places where these modifications should be introduced. It has been found that modifying message routes can affect the adaptation of complex services deployed in the ESB.

Rerouting messages to other instances is justified only when these instances share the same interface and provide identical functionalities. Current implementations of ESB that are compliant with the JBI specification share some common attributes used in the course of message processing. Each invocation of a complex service is described by its CID (Correlation ID) and is constant for one invocation, even when passed among services. A particular message sent between services is described by an EID (Exchange ID). Generally speaking, the invocation of a complex service is described by a CID and consists of several message exchanges described by an EID. Extending the Normalized Message Router with a routing algorithm able to modify message routing would yield an adaptive ESB.

Routing Algorithm. Once the message reaches the NMR, the routing algorithm relies on matching the message with routing rules in the routing table. If the message matches a routing rule, that rule is fired and the Service Name from the routing rule substitutes the intended destination Service Name. Routing rules are split into groups with different priorities, analyzed in a particular order. In every message, parameters such as VESB tag, Correlation ID, intended Service Name and Exchange ID are matched to routing rules. If the message matches several rules, one of them is selected on a round-robin basis to provide load balancing.

Rule priorities are assigned in accordance with the abstraction levels depicted in Figure 7. Rules are analysed in the order of increasing priority values.

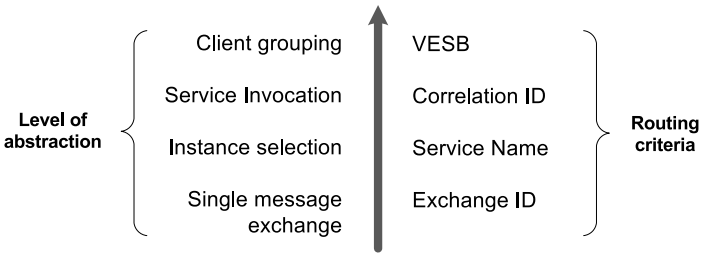

Fig. 7. Abstraction level of message flow

Priority	VESB	CID	Intentional Service Name	EID	Decision
1	n/a	n/a	n/a	+	Service Name
2	n/a	+	+	n/a	Service Name
3	+	n/a	+	n/a	Service Name
4	n/a	n/a	+	n/a	Service Name
5	n/a	+	n/a	n/a	Service Name
6	+	n/a	n/a	n/a	Service Name
7	n/a	n/a	n/a	n/a	Service Name

Fig. 8. NMR routing table

Depending on the priority value, different parameters are matched. The presented matching criteria are summarized in Figure 8. The lower the priority value the more important the routing rule. Some attributes are omitted when processing rules, although each routing rule returns a Service Name as a result. The decision element which closes the adaptation loop presented in chapter 2 gathers information about VESB from sensors and uses effectors to dynamically modify the routing table at runtime.

7 Adaptive Components

Adaptive Components are used in the extended ESB model to provide adaptability for the Service Component layer, as described in the SOA Solution Stack model. The main goal of Adaptive Components is to provide a background for creating atomic services used by upper layers.

In SOA systems, atomic services are mainly comprised of components. The need to ensure implementation and communication protocol independence forces the selection of an independent integration environment. The Service Component Architecture (SCA) specification [13] provides such a solution. It supports several

component implementations (such as Java, C++, BPEL and Spring) as well as various communication protocols (Web Services, RMI and JMS). Moreover, it is highly extensible, as evidenced by different SCA implementations [14], which introduce additional technologies not covered by the original blueprint.

It should be noted, however, that the SCA specification lacks adaptability mechanisms, which play a crucial role in ensuring conformance between the provided services and changing user requirements (i.e. Quality of Service). The following section describes adaptability mechanisms introduced into the SCA specification.

7.1 SCA Adaptability Requirements

The SCA specification involves the concept of a composite, which consists of a set of components connected by wires. A wire consists of a service representing the functionality exposed by one component, a reference representing the functionality required by another component (delegate object) and a binding protocol, which represents the communication protocol between the reference and the service. Several components in an SCA composite may expose services for external communication. Such services are further exposed e.g. via ESB.

Services created using SCA composites should be able to adapt to changing customer requirements and service provider capabilities (such as different CPU speeds resulting from infrastructure changes etc.) Non-functional customer requirements are recognized as QoS metrics and may be provided by the QoS layer of the S3 model. On the other hand, the measured service parameters are called Quality of Experience (QoE), and represent actual capabilities of a service. In an ideal case, QoE metrics should be as close to QoS requirements as possible.

The presented discussion leads to augmenting component descriptions with additional information about the provided QoS and monitoring of actual QoE values during runtime.

7.2 Service Model

An SCA composite may be perceived as a directed graph with nodes represented by components and edges represented by wires (directed from references to services), further called a *Composition Instance* (CI). A CI is connected to a specific QoE description derived from a composite. A set of *Composition Instances*, which meet the same functional requirements and differ only with respect to non-functional ones (QoS) is called a *Composition*. *Compositions* may also be represented as directed graphs, created by joining all CIs which provide a given element of functionality. The rules for joining CIs are as follows: if CI1 and CI2 use the same component as

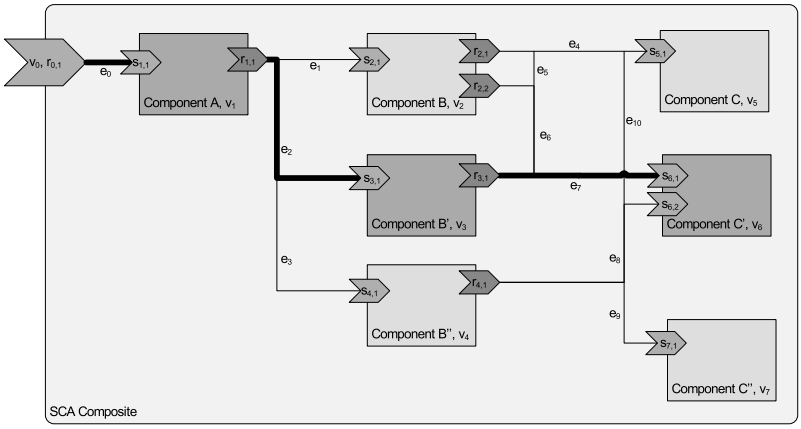

Fig. 9. Sample Composition with a selected Composition Instance

a specific node, they can be joined; otherwise both need to be added to the graph. Such a solution reduces the amount of resources required by all CIs (by exploiting shared nodes) and enables more efficient CI processing. A sample *Composition* with a selected *Composition Instance* is depicted in Figure 9.

7.3 Adaptation Mechanisms for SCA

Adaptation of services consisting of components may be perceived as selection of a proper Composition Instance from those available, in accordance with the stated QoS requirements and observed QoE values. This leads to a classic adaptation loop which includes monitoring, planning and selecting proper Composition Instances at runtime.

The SCA specification and its existing implementations do not provide adaptability mechanisms. Likewise, dynamic wire changes are not allowed. To provide support for changing wires at runtime an interceptor pattern may be used. Once intercepted, invocations between components may be sent via wires, as determined by given adaptability policies. Such interceptors may also be used for monitoring purposes, exposing data flows to other layers in the S3 model. The SCA monitoring and management functionality is exposed via a specialized Monitoring and Management Interface, further used by the inferring components, as described below. A sample Adaptive Component built using enhanced SCA is depicted in Figure 10.

Selecting a *Composition Instance* which best fits the stated non-functional requirements may be a very complex task, due to the large amount of existing components (the number of possible *Composition Instances* rises exponentially along

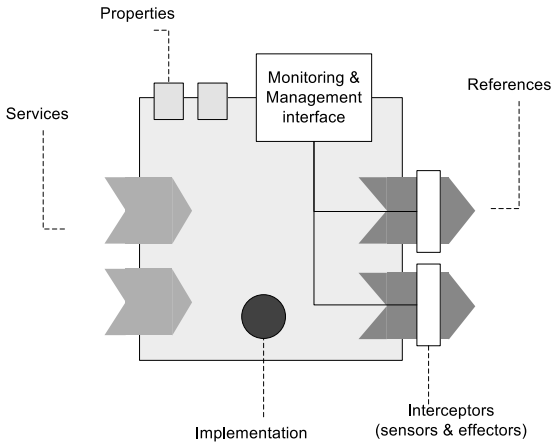

Fig. 10. Adaptive SCA Component

with the number of existing components in a *Composition*) and the large set of rules. Thus, advanced rule engines (RE), such as Drools [15] or JESS [16], can be used. Applying REs for selecting CIs provides a policy enforcement point and simplifies the S3 Policies layer.

7.4 Further Work

Adaptive Components are a basis for further work with the aim of enhancing existing component descriptions to automatically generate *Compositions* based on semantic descriptions. Such descriptions should contain information about functional component capabilities, similar to the OWL-S [17] standard used in the Web Services environment. Starting with a custom-defined goal, a service should be built from components and further adapted using the mechanisms described above.

8 BPEL Monitoring

The IBM S3 model [18] places business processes near the top of the stack, in direct support of customer needs. Such processes hide the complexity of lower-level services and components. According to the Workflow Management Coalition [19] a business process is a set of one or more procedures realizing common business aims or policies, usually in the context of an organizational role and dependency

structure. BPM (Business Process Management) [20] systems usually exploit the declarative system functionality construction model. They determine aims but do not state how to achieve them. Some notations, such as BPMN v1.0 (Business Process Management Notation), are used to describe the entire process in abstract terms, while others, like BPEL (Business Process Execution Language), BPELj (Business Process Execution Language for Java), XPDL (XML Process Definition Language) or jPDL (Java Process Definition Language), can be automatically run in execution environments called business process engines [19][20][21]. Currently, the most widely supported business process definition language seems to be BPEL, with more than ten commercial and open-source process engine implementation. As SOA orchestration processes are gaining importance, the existing monitoring and governance tools for vendor-specific BPEL engines lag behind. Therefore, an experimental prototype of a common multi-vendor distributed monitoring tool has been designed, implemented and validated.

8.1 Existing BPEL Design, Monitoring and Management Tools

Existing BPEL execution environments are augmented with sets of tools simplifying the creation, deployment and management of BPEL processes. Most such environments are based on Eclipse or NetBeans IDEs, with some commercial products claiming to enable process definition over web interfaces. (Results generally correspond to product prices.) Even though each process definition tool is dedicated to a particular process execution engine, they generate BPEL-conformant process definitions, which may be run on almost any engine. In the area of BPEL process monitoring and management the situation is somewhat more complex. Almost each engine provides basic business process monitoring and managements capabilities, sometimes exposed via standardized application GUIs (e.g. BPELMaestro by Parasoft, WebSphere by IBM) and sometimes exploiting vendor-specific APIs (Oracle, ApacheODE, Glassfish). The provided monitoring consoles are not only vendor-specific but also very inconvenient and unwieldy. They only expose basic information about the running process instances and steps (activities), which the processes actually perform. They rarely support filtering, data aggregation and bottleneck analysis, even on a basic level. The following implementations have been investigated in the context of process monitoring and management: BPEL Process Manager by Oracle, WebSphere by IBM, ActiveVOS by Active Endpoints, LiquidBPM by Cardiff, BPELMaestro by Parasoft, bizZyme by Creative Science Systems, Biztalk by Microsoft, ApacheODE by Apache nad jBPM by jBoss. The usability of custom monitoring GUIs is, at best, disputable. All are limited to parsing data provided by their own engines; moreover, they do not enable correlations of subprocesses with base processes and expose only simple query mechanisms

based on pull interfaces (if an API is available at all). Often there is no direct support for selection of monitoring data (only an on/off monitoring switch is exposed, with all the efficiency drawbacks resulting from collection and storage of unneeded monitoring data).

8.2 Motivation

Current BPEL monitoring tools do not fully meet the stated expectations: the information they provide means nothing for business users and is insufficient for IT specialists. They have to be rethought and redesigned to address the needs of particular user groups. A BPEL process definition is more than a simple blueprint for orchestrating process components — it contains process variables and structured activities that have business names and meanings. Such a metainformation, combined with process monitoring data, can be presented to non-technical business users able to validate process correctness from the business/customer perspective. This goal cannot be achieved with ESB and OSGi monitoring, operating at a lower level and presenting complex data in a manner understandable for IT specialists. In order to efficiently monitor a BPEL process, specific monitoring data has to be collected during the entire (usually very long) process execution cycle. This data should include information about individual actions (steps) executed by the process (e.g. execution of a Web Service method, sequence of operation invocations, conditional instructions, exception handling, etc.) together with execution time, additional parameters (e.g. Web Service address), status (e.g. correctly completed, faulty or having generated an uncaught exception) or even the values of BPEL process variables in the context of a particular activity. All this information needs to be collected, filtered, stored, aggregated and presented in a clear and user-friendly way, preferably not specific to any BPEL engine. As none of the existing tools support such functionality, a new, extended, multi-vendor, distributed BPEL monitoring tool is needed.

8.3 System Architecture

The architecture of the proposed BPEL monitoring system is presented in Figure 11. Data acquisition is realized by two cooperating modules instantiated for each monitored engine: Monitoring Event Manager and Monitoring Event Emitter. Together, they are responsible for subscription management and data propagation. Monitoring data is obtained from BPEL engines by the instrumentation module or via the engine API, depending on its availability and functionality [22]. The BPEL Management Agent is the main system component, exposing an interface for management of multiple engines, enabling monitoring data filtering, exposing process

metadata and ensuring historical data access and registration for monitoring event notifications generated by the BPEL Monitoring Agent. The responsibility of the BPEL Monitoring Persistence Agent is to store and query monitoring data and metadata along with the current tool configuration.

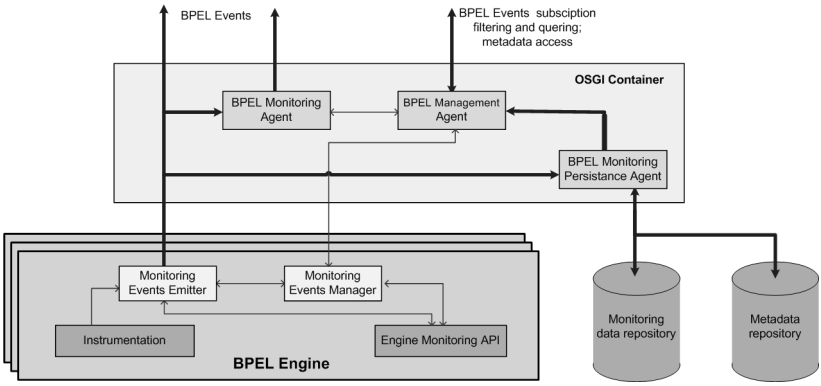

Fig. 11. AS3 — BPEL monitoring architecture

The proposed BPEL monitoring concept fills, in a natural way, the gap between current BPEL monitoring tools and cutting-edge, extremely expensive commercial BAM (Business Activity Monitoring) solutions. BPEL process monitoring must not ignore business users as it does today. It has to shift from measuring low-level, process-specific metrics of BPEL engines (execution time, number of executed process or faults) to more business-oriented goals (number of processes with specified business constraints such as the number of orders with a total amount greater then 100 $). The monitoring tools need to enable business users to trace BPEL process execution, basing not only on the status of a process but also on its variables and parameters. This is the challenge we intend to overcome.

9 Messaging

The Enterprise Service Bus, currently recognized as the foremost SOA infrastructure implementation, builds upon the concept of message-oriented communication, i.e. JMS. Asynchronous communication inherent in MOM solutions is a key enabler for loose coupling and autonomous operation — basic tenets of service orientation. In the early implementations of ESB only basic mechanisms provided by

the messaging infrastructure were used. Newer approaches extend the set of basic capabilities, taking advantage of QoS assurances provided by messaging solutions.

9.1 Communication Models and Routing

The most frequently used communication model assumes a single recipient. In such a case, asynchronous message queuing is used, where each participating side reads incoming messages from a local queue and processes the received data. Message Exchange Patterns extend this basic approach by defining interaction schemes in which correlations between groups of messages are introduced. As a result, even though there is no direct support for synchronous communication, it can be applied when necessary. Single-receiver delivery is one of two commonly available possibilities, the other one being the publish-subscribe model. In this latter case, each message sent by a single publisher is received by a group of clients which share common interest in the data being produced. The notion of a topic and optional peristent data delivery discussed in the following paragraphs provides a high degree of decoupling both in space and time domains.

In addition to basic addressing schemes, more complex mechanisms are available. JMS extends basic message routing with the notion of message selectors. In such a case messages are routed on the basis of message header properties. Furthermore, some currently available platforms provide mechanisms for content-based routing, heavily used in EIA. All of the above mentioned mechanisms can be used in conjunction with one another. Multipoint data delivery is very crucial from the EDA perspective, where events produced at a source have to be delivered to all interested parties. Different types of information used for routing purposes are presented in Figure 12.

Apart from the basic functionality, message-oriented middleware provides more advanced capabilities in the form of QoS assurances, summarized in Table 2.

9.2 Message Oriented Middleware

Message oriented middleware as a core part of the SOA infrastructure, plays a key role in the scope of nonfunctional system properties. In order to enable advanced functionalities, the communication infrastructure should be appropriately constructed. From the system's point of view, MOM can be seen as a network of interconnected brokering nodes. The properties of such a network and the routing algorithms used for message distribution directly affect the overall system efficiency, scalability and robustness.

In typical scenarios, message-oriented middleware is used for sending messages which represent asynchronous service requests or generated events. However,

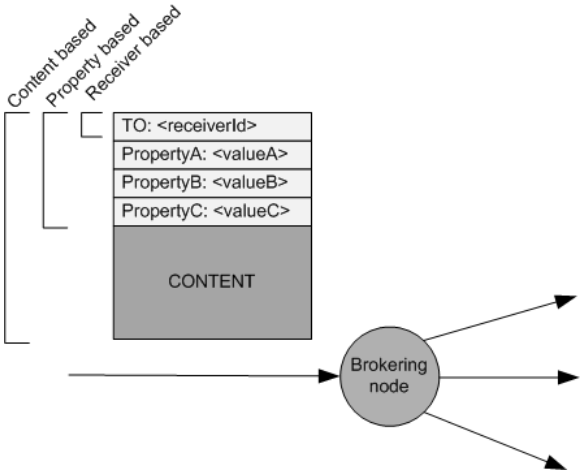

Fig. 12. Information used for message routing

Table 2. Message-oriented middleware QoS summary

Capability	Description
Transactions	Defined per communication session (in single-receiver or many-receivers model). Ensures that messages are sent and received in an all-or-nothing manner.
Message ordering	Specifies message ordering requirements. Some commonly available options include preserving the order of messages sent by a single user and delivered to every receiving site. Other possibilities include total ordering of messages or less restrictive casual order.
Persistency	Transport-level message persistency which enable communication even when some receivers are not currently available. Depending on infrastructure configuration, in-memory or data storage persistency can be used.
Flow control	Ensures that the rate of the message stream produced by the sender doesn't exceed the processing capabilities of the receivers.
Delivery and dispatch policies	Policies which specify what measures should be taken in case of delivery or dispatch failures.

rapid evolution of the SOA paradigm has resulted in additional demands placed on MOM, including efficient batch data transfer in the many-receivers model. Publish-subscribe delivery can be seen as application-level implementation of multicast communication. Apart from local network settings administered by a single authority, such services are unavailable in the network layer. Currently supported QoS assurances, such as message ordering and flow control mechanisms, already enable MOM for batch data delivery. However, in order to achieve a high level of effectiveness and scalability, the infrastructure has to be extended with additional mechanisms. One of them is multipathing, where data is transferred not only along a distribution tree constructed by the messaging middleware but also via additional paths. Extending the basic set of routes results in accelerated data transfer.

Another MOM enhancement from which data delivery solutions would benefit is caching. Assuming that a certain amount of disk storage is available at each brokering node, specific data access patterns can be identified. In such a case data can be cached in nodes close to the clients which most often request it. In currently available middleware solutions delivery buffers at each brokering node can be considered as basic caching mechanism. However, in order to provide suitable QoS, this approach is not satisfactory.

Finally, certain applications demand data transformation services. Prior to data consumption, clients have to preprocess data that is being sent. Assuming that not all clients are able to perform appropriate transformations due to insufficient hardware resources or available software services, it appears appropriate to embed such capabilities in the data delivery infrastructure.

10 Summary

SOA application deployment and execution should take into account end-user requirements expressed as nonfunctional properties. Most of these requirements refer to QoS, reliability or security aspects which must be enforced during runtime. As many SOA applications are deployed as collections of communicating services, their governance should be QoS-aware and mindful of integration aspects. This explains the increasing significance of various extensions of the ESB technology, seen as a cornerstone of modern SOA execution infrastructures. The place and role of this technology is very well defined by the S3 model. Adaptive ESB addresses many governance aspects related to nonfunctional requirements. The proposed mechanisms and their building blocks create a consistent framework which can be further refined according to specific system needs.

The elements of the adaptive ESB framework, such as the monitoring system, exposition layer and policy engine are generic and can be applied to different layers

of the S3 model. This is evidenced by the SCA extension supporting adaptive activity and the proposed BEPL monitoring system. The presented study underline the fundamental role of the SOA execution environment and application monitoring as a starting point for SOA governance in general. Adaptive systems enable mature management of dynamic software execution environments.

Another key trend in the emergence of dynamic software systems. A dynamic system can be modified at runtime, without having to be restarted. This crucial feature is supported by the OSGi technology as far as Java-based systems are concerned. The OSGi model, often described as SOA for JVM, matches the concept of adaptive ESB or SCA. Dynamic software behavior can be further extended by AOP programming. This technology is very useful for instrumenting complex software systems such as ESB.

The presented overview can be summarized by stating that the proposed adaptive system concepts are well supported by modern software tools, opening a wide area for deployment and uptake by the software industry.

References

1. The Forrester Wave: Enterprise Service Buses. Q1 2009, `http://www.forrester.com`.
2. Chappell D., Erl T., Little M., Loesgen B., Roy S., Rischbeck T., Simon A. Modern SOA infrastructure: technology, design, and governance. Prentice Hall PTR, Upper Saddle River, New Jersey, 2010.
3. Erl T. SOA design patterns. Prentice Hall PTR, Upper Saddle River, New Jersey, 2009.
4. Erl T. Service-oriented architecture: concepts, technology, and design. Prentice Hall PTR, Upper Saddle River, New Jersey, 2005.
5. Alliance T.O. About the OSGi Service Platform. July 2004, `http://www.osgi.org/documents/osgi_technology/osgi-sp-overview.pdf`.
6. JSR 208: Java business integration (JBI). `http://jcp.org/en/jsr/detail?id=208`.
7. Hayman C. The benefits of an open service oriented architecture in the enterprise. 2005, IBM.
8. Taft D.K. Can JBI 2 succeed where JBI 1 did not? eWeek, May 2007, `http://www.eweek.com/c/a/Application-Development/Can-JBI-2-Succeed-Where-JBI-1-Did-Not/`.
9. Hutchison B., Schmidt M.-T., Wolfson D., Stockton M. Soa programming model for implementing web services, part 4: An introduction to the IBM enterprise service bus. `http://www.ibm.com/developerworks/library/ws-soa-progmodel4/` [Accessed 01/04/2009].

10. Lohman C., Fortuin L., Wouters M. Designing a performance measurement system: A case study. Eur. J. Oper. Res. 156, 2, 267–286, 2004, http://www.sciencedirect.com/science/article/B6VCT-48GP7GD-1/2/7bb13c9633ae1ea4b746043c758c2297.
11. Amnajmongkol S., Ang'ani J., Che Y., Fox T., Lim A., Keen M. Business activity monitoring in websphere business monitor v6.1. IBM Redbooks, Tech. Rep., 2008.
12. IEEE 802.1q: VLAN. IEEE, http://www.ieee802.org/1/pages/802.1Q.html, 2005.
13. SCA Service Component Architecture. Assembly Model Specification, version 1.00. OASIS, May 2007.
14. Apache Tuscany Home Page. http://tuscany.apache.org/, accessed September 2009.
15. Drools: Business Logic integration Platform. http://www.jboss.org/drools/, accessed November 2009.
16. JESS: the Rule Engine for the Java Platform. http://www.jessrules.com/, accessed December 2008.
17. Burstein M., Hobbs J., Lassila O., Mcdermott D., Mcilraith S., Narayanan S., Paolucci M., Parsia B., Payne T., Sirin E., Srinivasan N., Sycara K. OWL-S: Semantic Markup for Web Services. Website, World Wide Web Consortium, 2004, http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/.
18. Ali A., Liang-Jie Z., Michael E., Abdul A., Kishore C. Design an soa solution using a reference architecture, improve your development process using the soa solution stack. IBM developerWorks, 2007, http://www.ibm.com/developerworks/library/ar-archtemp/.
19. Coalition T.W.M. The workflow management coalition. 2009, http://www.wfmc.org/.
20. Coalition W.M. Workflow management coalition, terminology & glossary. Workflow Management Coalition, Tech. Rep., 1999, http://www.wfmc.org/standards/docs/TC-1011termglossaryv3.pdf.
21. Initiative B.P.M. Business process management initiative. 1997, http://www.bpmi.org/.
22. Duszka T., Janczak J. Master thesis: Badanie wydajności systemów w architekturze SOA. AGH, 2007.

Embedded SOA

Paweł Bachara, Jacek Długopolski, Piotr Nawrocki, Andrzej Ruta, Wojciech Zaborowski, and Krzysztof Zieliński

Department of Computer Science, DSRG,
Al. Mickiewicza 30, 30-059 Kraków, Poland
{pawel.bachara,jacek.dlugopolski,piotr.nawrocki,andrzej.ruta, wojciech.zaborowski,krzysztof.zielinski}@agh.edu.pl
http://www.soa.edu.pl

Abstract. Business applications are progressing towards an "Internet of Things". It is why more flexible and adaptive concepts for the integration of enterprise services with smart embedded devices are necessary. This chapter addresses two related aspects: (i) exposing embedded devices as Device Services, and (ii) implementation of embedded devices as hardware components without any software elements. In first context the SODA (Service Oriented Device Architecture) is outlined. This approach supports integration of Service Devices with ESB. Next FPGA-based implementation of Device Services is presented. Extension of SCA programming model to sensor networks is also shown. Finally technical problems of exposing mobile phone functionality as services and accessing services on mobile devices are described.

Keywords: FPGA, Mobile SOA, mobile device, SCA, sensor network, SODA

1 Introduction

Enterprises are moving towards service-oriented infrastructures where applications and business processes are modeled on top of cross-organization service landscapes. Currently, shop-floor intelligent manufacturing systems and businesses incorporate system intelligence in a limited amount of monolithic computing resources, accompanied by large numbers of embedded devices with constrained resources.

Business applications are progressing towards an "Internet of Things", where millions of devices are interconnected, providing and consuming data about the real world and making it available for business processes that run in the digital

world. Thus, more flexible and adaptive concepts for the integration of enterprise services with smart embedded devices are necessary. Recent years have witnessed the emergence of Service Oriented Architectures (SOA) that use loosely-coupled services to effectively support business processes and user requirements. In this context, implementing services as embedded devices or exposing them on small mobile platforms becomes very important.

Exposing an embedded device as a Device Service means that its design should follow service orientation principles [1]. These include: (i) Standard Service Contract, (ii) Service Loose Coupling, (iii) Service Abstraction, (iv) Service Reusability, (v) Service Statelessness, (vi) Service Autonomy, (vii) Service Discoverability, and (viii) Service Composability. Their aim is to guarantee cooperation of Device Services as distributed, autonomous, and reusable entities, which leverages a new dynamic infrastructure, able to provide better insight into shop-floor activities for higher-level business processes and adapt to changing business needs.

An even more challenging problem is implementation of embedded devices as hardware components without any software elements. In many cases the FPGA (Field Programmable Gate Array) technology could be used for this purpose. FPGA-based Device Services offer many advantages in comparison to the standard microprocessor-based implementation: increased efficiency, low energy consumption, reliability, specialization, etc. Services implemented in this way create a foundation for a new step of SOA called H-SOA (Hardware SOA).

This chapter addresses both aspects of the Device Services implementation. First, in Section 2, the SODA (Service Oriented Device Architecture) is outlined, basing on the integration of Service Devices with ESB (Enterprise Service Bus). Next, in Section 3, an FPGA-based implementation of Device Services is presented. This technology is used for constructing intelligent components such as a camera for license plate verification, sensors with embedded processing capabilities, and a mobile robot. The advantages of FPGA vs. standard processors are analyzed in detail. Section 5 presents MSOA (Mobile SOA), in light of the importance of mobile devices and sensor networks acting as Service Devices. In Section 6 the problem of exposing sensors as services with SCA (Service Component Architecture) is discussed. Since SCA composites can be integrated with ESB, the proposed solution can also be used for ESB-sensor integration. Finally, in Section 7, the technical problems of exposing mobile phone functionality as services and accessing services on mobile devices are described. The chapter ends with a summary.

2 Service Oriented Device Architecture

The Service Oriented Device Architecture (SODA) is an adaptation of the Service Oriented Architecture (SOA) which integrates business systems through a set of

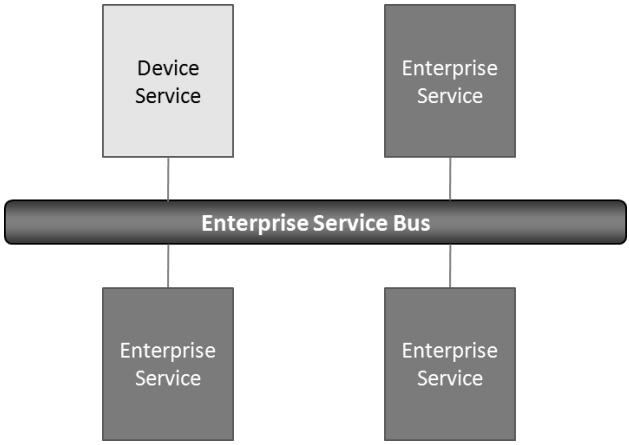

Fig. 1. Device is accessible as any other enterprise service

services that can be reused and combined to address changing business requirements. The SODA approach to designing and building distributed software is to integrate a wide range of physical devices into distributed IT enterprise systems. These systems are typically integrated with the Enterprise Service Bus (ESB). Hence, services offered by hardware devices should also be accessible in this way, as shown in Figure 1.

SODA focuses on the boundary layer between the physical and digital realms. In other words, a sensor such as a digital thermometer or an RFID (Radio Frequency Identification) tag reader can translate a physical phenomenon into corresponding digital data. Alternatively, an actuator such as a heater relay or alarm beacon can translate a digital signal into a physical phenomenon. Sensors and actuators are combined either physically or conceptually to create complex devices and services such as a climate control module or a geofencing system that monitors a vehicle's position within a restricted area.

SODA aims to:

— provide higher-level abstractions of the physical realm,
— insulate enterprise system developers from the ever-expanding number of standard device interfaces, and
— bridge the physical and digital realms with a known service or set of services.

SODA implementations can use existing and emerging device and SOA standards. SODA should be straightforward to implement, unlocking the potential to enable a new level of Internet-based enterprise systems.

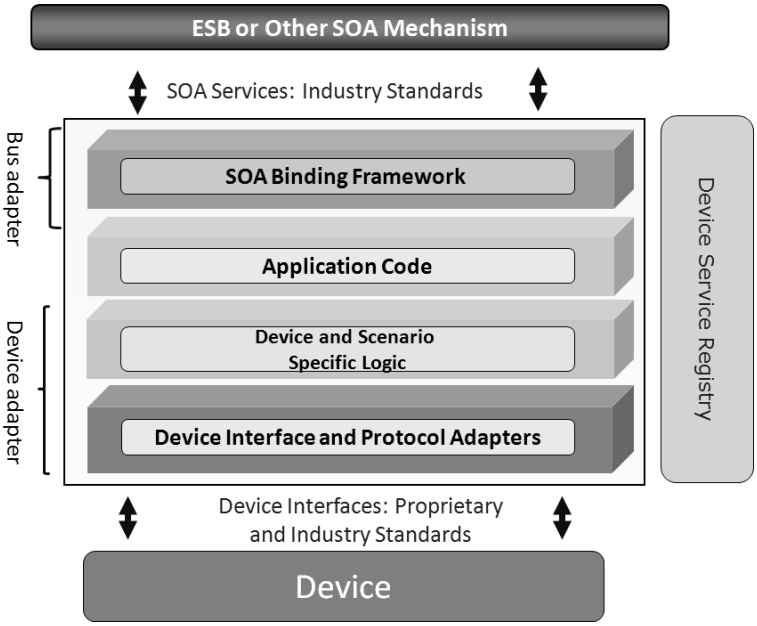

Fig. 2. A Service-Oriented Device Architecture solution stack

Conventional approaches to device integration often center on custom software communicating with enterprise applications through a variety of IT middleware and API technologies. This approach has served enterprises well, but SOA standards and open software initiatives are now moving beyond such middleware architectures. Although IT applications are being adapted to SOA, standards for defining low-level device interfaces are still emerging. Even so, technology exists today to leverage SOA across the entire spectrum of critical events and data originating from devices. Mechanisms for building and sharing service interfaces, capabilities for remote software maintenance and loosely-coupled messaging models present highly effective technologies for the implementation of SODA.

SODA requirements include [2]:

— using a device adapter model to encapsulate device-specific programming interfaces;
— employing loosely-coupled messaging on the service side, capable of supporting multiple streaming services commonly used in SOA enterprise systems, such as the Enterprise Service Bus;
— using open standards, where available, at the device and service interface level;

— providing a means to present standard or open service interfaces to devices that have proprietary protocols or where it might not be practical to embed standards into the low-level device interface;
— supporting the implementation of a spectrum of device adapters — from simple, low-cost sensor data to complex device protocols;
— supporting the upload of remotely configurable logic components into device adapters for maintenance, upgrades, and functionality extensions;
— adapting security mechanisms as required for the domain.

A SODA implementation comprises three main components, shown in Figure 2. Device adapters talk to device interfaces, protocols, and connections on the one side and present an abstract device-specific service model on the other. The bus adapter moves device data over network protocols by mapping a device service's abstract model to a specific SOA binding mechanism used by the enterprise. The device service registry enables the discovery and access to SODA services.

Wherever device interface standards do not exist, interface and protocol adapters within SODA implementations provide a common model for the software used to create service interfaces. Widespread adoption of standards at the device-interface level will reduce the development and maintenance costs of device adapters and their corresponding SODA services. The proposed SODA device interface model is shown in Figure 3. The device layer accepts commands and exposes measurements and signals. This layer provides an interface to the hardware device and understands the meaning of messages and any parameters within a message. The transport layer understands the format of a message and parses input bytes into valid messages. The connection layer supports reading and writing byte streams to the hardware device. Device I/O represents physical interconnections, e.g. an IP network or a serial interface.

Referring to Figure 2, there are two possible ways of integrating Device Services with ESB:

— as an external service, available via a binding component;
— as an internal service, implemented with a service engine component.

It is difficult to say in general which approach is better. Solutions should be adapted to specific situations in the context of the characteristics of the Device Service. Implementation effort and access overhead are the key factors which should be taken into account. It is necessary to point out that services exposed as Web Services can be easily combined with ESB using the standard WS binding component provided by any ESB implementation. In such a case the former approach would be preferable. If a highly specialized access or communication protocol has to be used, the latter solution could more appropriate. From the point of view of

other services deployed in the ESB, both solutions are equivalent as they manage access in a similar way. This topic will be further discussed in the following sections.

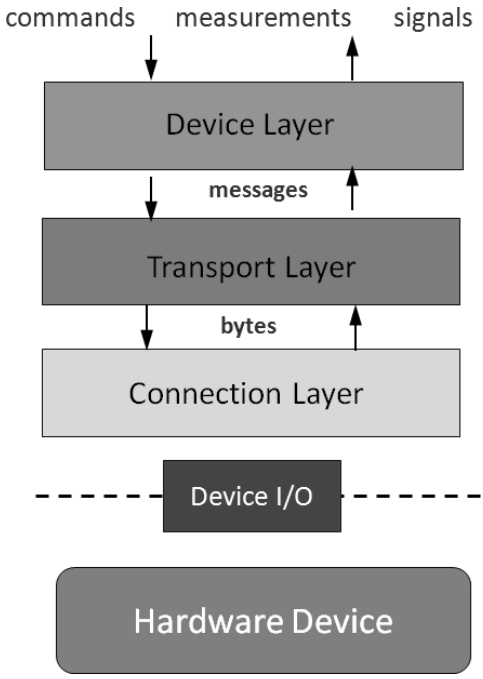

Fig. 3. SODA device interface model

3 Field-Programmable Gate Array (FPGA)

In a modern fast-paced environment the traditional computing paradigms often do not match our expectations in terms of the quality of service and/or response time. This inadequacy is apparent whenever algorithmically complex and resource-intensive processing is required on non-standard input data such as sensor readouts or live video streams. The traditional approach to solving this type of computation problem is by using an existing stationary infrastructure composed of PCs or dedicated servers where input data is fed in and responses are delivered to the remote consumer via available network channels, e.g. Ethernet. In such a scenario

the role of hardware system components is limited to data acquisition while the entire computational burden falls onto the applied software. Major drawbacks of such systems include:

— Unnecessary network traffic generated as possibly large amounts of input data, e.g. video frames, need to be transferred from sensors to processing units;
— Significant latency arising from the necessity to delegate computing to a remote location;
— Lack of hardware acceleration of performance-critical computing processes;
— High cost, including e.g. high power consumption and costly infrastructure maintenance;
— Poor scalability and adaptability as the traditional server-based systems cannot be easily reconfigured to deal with new tasks.

Following the natural tendency to miniaturize the computational environment, growing demand has emerged for intelligent devices which could take over the most computationally intensive tasks and thus help overcome the aforementioned delegation problem. Field-Programmable Gate Array (FPGA) is a leading technology offering reconfigurable devices where any desired functionality can be implemented on chip with massive hardware acceleration enabled. FPGAs are integrated circuits designed to be configured by the customer after manufacturing, where the configuration is generally specified using one of the available Hardware Description Languages (HDLs). This feature makes them very flexible compared to the Application-Specific Integrated Circuits (ASICs) which are custom-made for a particular use, e.g. in a cell phone.

FPGAs contain programmable logic components called *logic blocks*, and a hierarchy of reconfigurable interconnects that allow these blocks to be wired together. Logic blocks can be configured to perform combinational functions, from simple gates, such as AND or XOR, to very complex ones. Typically, logic blocks also contain other structures, such as lookup tables (LUTs) or memory elements. Architecturally, FPGAs are not too far from the popular Complex Programmable Logic Devices (CPLDs), but come with a much higher ratio of interconnections to logic elements. This property extends the range of designs that are suitable for implementation using FPGAs. In practice, FPGA devices are particularly useful for the implementation of algorithms that are in some way decomposable and can make use of the massive parallelism offered by the FPGA architecture.

We believe that FPGA devices can be effectively used as part of the HSOA infrastructure to handle various performance-critical operations, such as complex real-time image processing. We would like to deploy such hardware-accelerated processing units as Device Services conforming to the SOA standards, i.e. exposing a well-defined interface (operation signatures), service endpoint information

(network location), invocation protocol details and service semantics. A proof-of-concept application on which we are currently focusing is the license plate verification system outlined in Section 4 — *FPGA-based Pattern Recognition.*

3.1 FPGA Programming

FPGA programming involves several major steps. First, the user must provide a specification of the desired behavior, either as a program written in one of the HDLs (usually VHDL or Verilog [3, 4]) or a schematic design. Subsequently, using an Electronic Design Automation (EDA) tool, a technology-mapped *netlist* is synthesized which describes the physical resources used in the design and their connectivity. This netlist is then fitted to the actual FPGA architecture in the so-called *place-and-route* process. It is typically performed using a dedicated software suite provided by the vendor of the target FPGA and involves transferring the netlist to the chip via a serial interface (JTAG) or to an external memory, e.g. EEPROM.

Describing complex functionality in a Hardware Description Language is burdensome and requires, apart from HDL programming skills, substantial hardware design experience and intuition from the system developers. To simplify FPGA programming, many libraries of predefined, highly optimized functions and circuits exist, among others various memory and peripheral device controllers. These libraries are called *Intellectual Property Cores* (IP Cores) and are usually released under proprietary vendor licenses. Before any design is tested on real hardware, it is typically simulated at multiple stages throughout the design process. For the initial, functional verification of the design, the HDL description is simulated within a software test bench. Once a netlist has been generated, it is translated to a gate-level description where simulation is repeated to confirm that the synthesis proceeded without errors. Finally, the design is laid out in the FPGA which enables observed propagation delays to be added and the simulation to be rerun with these values back-annotated onto the netlist.

3.2 Programming Using High-Level Languages

Even extensive use of IP Cores brings little improvement as far as an algorithm design is concerned, especially when a custom-made functionality has to be implemented: substantial knowledge of hardware is still required. Moreover, transition to HDL requires a fundamental mental shift from the software-oriented developers. They must adopt a completely different programming philosophy, where the issues transparent to them when using a conventional instruction-based processor,

such as parallel execution, process synchronization or elementary operation timing, begin to matter. The most popular approach adopted to bring FPGAs closer to C/C++ or Java programmers is introducing high-level synthesis tools. They enable designing and expressing the desired behavior of the system in a familiar programming environment, and then automatically translating the code to a selected HDL. C is traditionally used for such high-level synthesis.

In general, constructing a HDL description involving a collection of parallel hardware processes from an essentially sequential block of C code is a very complex task. Very few vendors currently offer true C-to-HDL translation tools. The C language is often supported only for functional verification purposes, i.e. to build software test suites for simulations. Examples of commercial products where HDL code generation is possible include *Catapult C Synthesis* by *Mentor Graphics* and *Impulse C CoDeveloper* by *Impulse Accelerated Technologies*. We focus on the latter solution as this particular programming environment has been chosen for the implementation of our hardware-accelerated image processing algorithms.

3.3 Benefits of Algorithm Implementation on FPGA

Implementation of many computational algorithms on a reconfigurable FPGA chip offers apparent benefits when compared to the traditional approach involving code execution using a sequential processor, be it CISC or RISC. These benefits are mostly related to the extensive optimization possibilities that arise from the parallel nature of the FPGA architecture. In Section 4, *FPGA-based Pattern Recognition*, we will show these advantages using an example of simple image filters. Table 1 summarizes the available mechanisms of algorithm parallelization and their realizations, including programming conventions and constructs in the *Impulse C* environment. A more detailed discussion of programming FPGAs with *Impulse C* can be found in [5].

It should be noted that combining both approaches, i.e. using a processor and FPGA chip simultaneously, is not only possible but also very popular.
In such a scenario an embedded soft-core CPU, such as *NIOS II* for *Altera* FPGAs or *Microblaze* for *Xilinx* family of FPGAs, is implemented entirely in the programmable logic and memory blocks of the target device. It is used to control the execution of the computationally-intensive, performance-critical operations offloaded to a separate FPGA logic circuit which itself acts as a hardware accelerator. This approach is particularly useful when a complex system involving a number of peripheral devices, such as a VGA or an Ethernet controller, is built. In this case all the control tasks and I/O management are typically handled using a high-level programming language, such as C/C++, which is easier and much more intuitive for system developers.

Table 1. Major types of algorithm optimization that admit parallel execution of the program's code on FPGA

Optimization	Description	Programmer's perspective
Algorithm decomposition	This is the first and primary step to be taken when implementing a parallelized algorithm on FPGA. Certain algorithms can be decomposed into logically disjoint units which can be processed independently. For instance, if certain image processing functions involve only pixel-level operations without any operation affecting other pixels, the image can be decomposed geometrically and each part can be processed simultaneously.	In Impulse C, algorithm decomposition is realized by partitioning the code into functions, each defining the behavior of an independent computational process. Functions `co_process_create` and `co_process_config` are available for associating a new process with an appropriate function and for mapping the process definition to hardware resources. Logically, Impulse C processes correspond to Verilog processes or VHDL entities.
Parallel memory access	For fundamental reasons related to the way addressing mechanisms work, accessing a single bank of memory in parallel is not possible. Pseudo-parallelism is, however, achievable as multiple memory read/write operations can be performed in a single clock cycle when using a multi-port memory controller. True parallelism is achieved when multiple RAM chips are available in a circuit and memory access operations are appropriately split between them. For operations on small arrays a scalarization mechanism is also available (see below).	An Impulse C programmer can define two types of memory: local and shared. The `co_array_config` function can be used to specify the type of physical memory (synchronous/asynchronous, single-/dual-port) used to implement the array. Mapping shared memory to hardware is configured in vendor-specific Platform Support Package files. In general, it is the programmer's responsibility to schedule memory access operations appropriately, taking into account the number of physical chips available.
Array scalarization	Small-size arrays may be converted to registers which, if no data dependencies are involved, can be accessed simultaneously within a single clock cycle.	Array scalarization is attempted for each array satisfying several conditions, including constant and integer indexing and whether a "Scalarize array variables" option is enabled in the Impulse C project configuration.

Table 1. (continued)

Optimization	Description	Programmer's perspective
Loop unrolling	A loop with a pre-determined number of iterations, n, can be unrolled, i.e. its body can be duplicated into n copies, each executed in parallel. The iterations of the loop must not include interdependent calculations and/or assignments that would prevent the loop from being implemented as a parallel structure in hardware.	The `CO UNROLL` pragma specified at the top of the loop instructs the Impulse C compiler to attempt unrolling it. Only loops with a small number of iterations should be unrolled to avoid involving excessive amounts of hardware.
Loop pipelining	Statements within the innermost loop can be pipelined with the goal of reducing the number of instruction cycles required to process the entire iteration. The idea of pipelining is to allow the operations of one iteration to execute in parallel with operations of one or more subsequent iterations.	The `CO PIPELINE` pragma placed at the top of the loop instructs the Impulse C compiler to attempt pipelining it. The programmer is responsible for scheduling access to the variables used within a pipeline in a way that prevents conflicts when the pipelining is applied.
Control logic flattening	Control logic flattening causes conditional expressions to be implemented in combinational logic instead of using a state machine.	The `CO FLATTEN` pragma instructs the compiler to explicitly flatten the logic for a block of code.

4 FPGA Application in SOA System

FPGA is rapidly gaining popularity as in many situations it allows complex data processing to be performed in an efficient, flexible and inexpensive way. In light of Section 3 — *Benefits of Algorithm Implementation on FPGA*, hardware implementations of computing algorithms offer numerous advantages over the traditional approach involving stationary computing infrastructure and existing software development paradigms. Therefore, we recommend employing reconfigurable FPGA chips to implement a number of specialized H-SOA-compliant services. In subsequent sections three types of such services are outlined and the progress of work in each area is reported.

In Section *FPGA-based Pattern Recognition* we discuss an FPGA-based machine vision component integrated with a digital camera. The implementation of this component is based on an example of the popular image filter – an edge detector. Future work on the project, involving implementation of a license plate

verification functionality and other complex image/video processing hardware services, is also discussed. In Section *Mobile Robots* we present a six-legged spider robot exposed as an H-SOA Service. The robot features a dedicated FPGA circuit used to simultaneously control eighteen servomotors that are responsible for putting the robot in motion. Possible applications of the spider robot include inspection, exploration, or rescue tasks in low-accessibility areas. Finally, in Section *Reconfigurable Hardware Data Processing Board for Sun™ SPOT*, an FPGA-based extension board for the SPOT (Small Programmable Object Technology) device is introduced. The proposed use of this board involves extending the number of sensors connected to SPOTs, specialized processing of sensor data and other operations intended to reduce the volume of data sent through a sensor network to the final processing node.

4.1 FPGA-based Pattern Recognition

Figure 4 depicts a schematic diagram of a simple image processing operation consisting of two concatenated filters: a smoothing filter and a Sobel edge detector [6]. The stream of input gray-scale pixels (coming, for instance, from a frame grabber) is sent on a row-by-row basis to the first process which simply splits it into five streams. Local buffers are used to enable sending the output as a marching column of five pixels from five consecutive rows. Each column is received by the smoothing process and stored internally in five separate registers (out of a total of 25). Pixel values are shifted column-wise between the registers before each new column is read, in order to simulate the smoothing kernel sliding through the image. The smoothing process calculates a weighted sum of the pixel values stored in the registers and sends the result for each pixel to the output stream. It is then again split, but only into three streams. Finally, yet another process applies two 3×3 Sobel filter masks to extract directional image gradients for each input pixel and calculates the sum of their squares. Applying a predefined threshold to the resulting squared gradient magnitude produces an output stream of *true*/*false* values, where a *true* value is indicative of the pixel lying on an edge. This output stream can be plugged into a VGA controller to send the filtered image to an external device, or into another processing module.

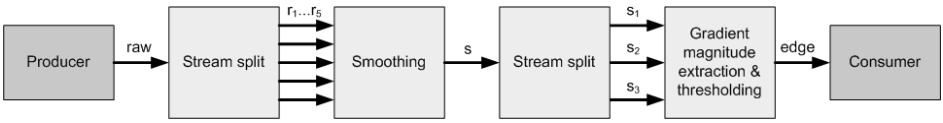

Fig. 4. A simple concatenation of two hardware-accelerated image filters jointly acting as an edge detector

The sample image filter introduced above is a convenient illustration of the benefits brought by programmable logic. Referring to Tab. 1, the following parallelization mechanisms are used to make this filter work much faster compared to its implementation on a standard PC:

— The algorithm is decomposed into four parallel and pipelined processes. They can be designed in such a way that each process produces an output value in a single clock cycle. This implies a non-blocking operation of the entire pipeline, i.e. that no process needs to wait for the input stream of the subsequent process to accept data.
— The stream splitting processes buffer the consecutive pixel rows in individual arrays. If the target FPGA supports dual-port RAM and loop pipelining is enabled, read/write operations on all arrays can execute in parallel in a single clock cycle.
— The filter processes store the kernel mask coefficients and the input pixel values in registers instead of arrays, which guarantees single-cycle calculation of the filtered pixel values.
— The critical loops in all four processes are pipelined, which enables single-cycle generation of the output value by each process.

We have extended the above filter by implementing a more advanced Canny edge detector [7] shown in Figure 5. The hardware module can be used at the pre-processing stage of many real-world image processing applications including optical object detection or image segmentation. The design depicted in Figure 5 includes multiple parallel processes. Since in most cases only fragments of the image need to be analyzed to achieve the desired result, our module first loads the image into separate banks of the SDRAM memory and then crops a predefined region of interest (ROI). The edge extraction pipeline is composed of the parallel processes used in the previous example, augmented with several extra processes, used to refine the output of the Sobel edge detector by making the resulting ridges pixel-wide and hence eliminating redundant edge information. The added processes require direct access to non-contiguous locations in the shared memory where the input image region is stored. Therefore, they can no longer operate in a pixel-by-pixel manner using streams. These particular processes are connected with dashed-line arrows in Figure 5. It is important to note that externally the entire module is still seen as a black box with stream-based input and output.

The hardware implementations of the image manipulation routines presented above are themselves of little practical use. However, they can be used as building blocks when assembling more complex applications. Our short-term goal is to build, for demonstration purposes, a self-contained camera-equipped hardware component performing license plate verification (LPV) on the basis of input images or video feeds depicting road traffic. In the long run we would like to develop

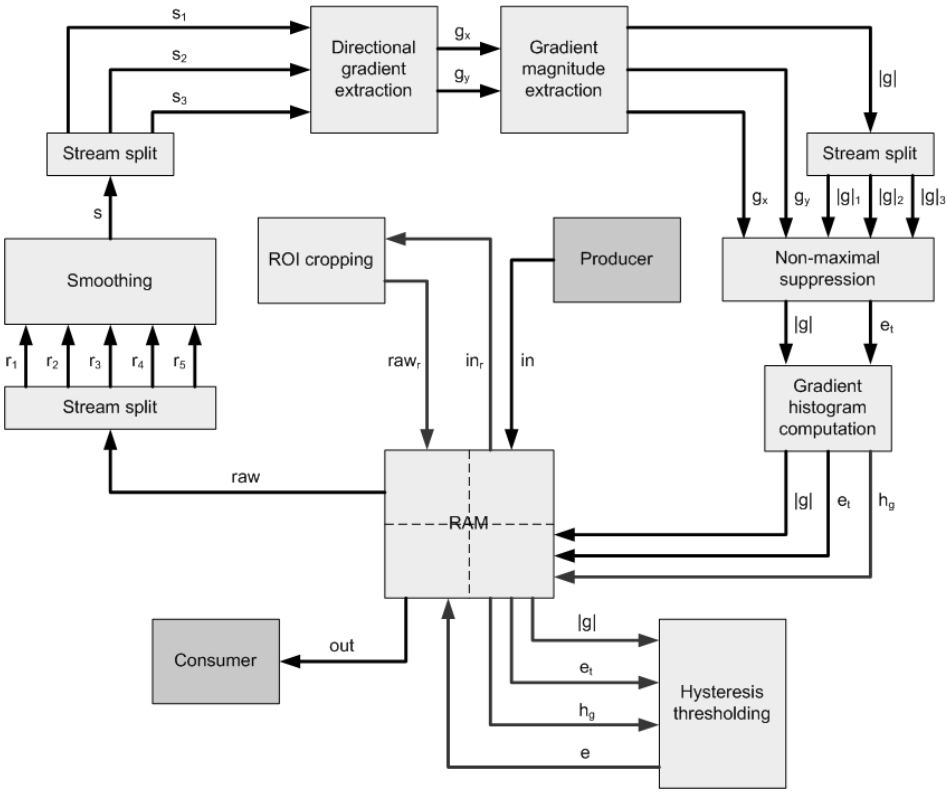

Fig. 5. A more complex design involving a region of interest (ROI) extracted from an input image and a Canny edge detection algorithm [7] applied to the specified ROI

other machine vision algorithms, such as object tracking or automatic scene categorization.

We have already developed a set of suitable yet sequential C++ algorithms for the LPV task and simulated them on a conventional PC, with promising results. At the moment we are developing parallel versions of these algorithms using the *Impulse C CoDeveloper* software suite. HDL code is generated via high-level synthesis and subsequently integrated with existing hardware interfaces available for the *Altera DE2* FPGA development board that we use. The Canny edge detector presented above is part of the design and is already running in real time on our hardware where input images are acquired from the *Terasic's* TRDB-D5M digital camera module, and the output is sent to a VGA display. The main part of the LPV system, a parallelized edge-based shape detector implementing a Generalized Hough Transform [8, 9], is currently under development. This detector will be used

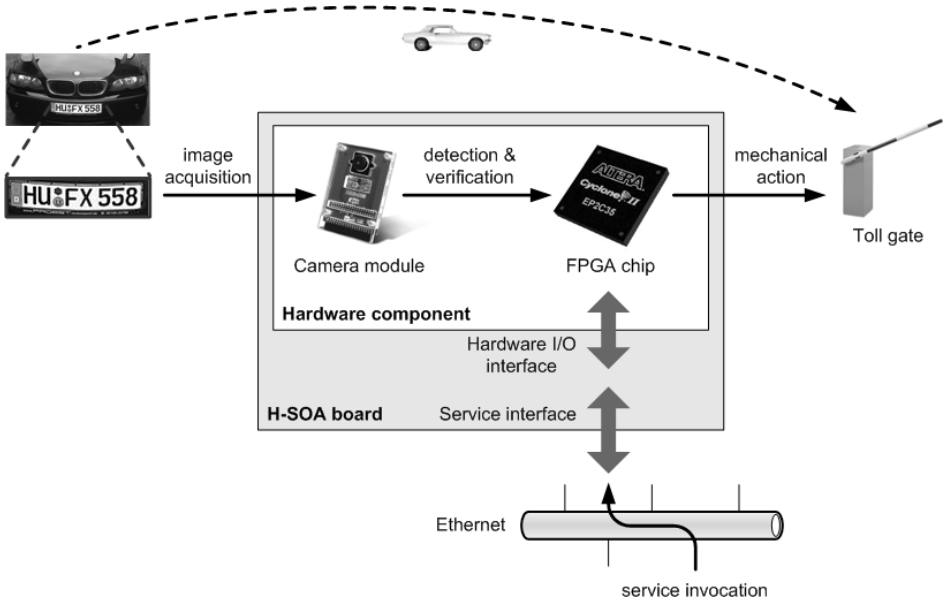

Fig. 6. Operation of an FPGA-based license plate verification system deployed as an H-SOA service

both for license plate localization and for checking the license number against an input list of "wanted" numbers.

As the next step, we plan to integrate our image processing algorithms with the H-SOA board which will expose the hardware component as a Web Service using the Ethernet controller. A conceptual diagram illustrating the usage of the target license plate verification service is shown in Figure 6. Thin solid arrows symbolize the runtime data flow between the hardware components of the system. Internal and external interfaces are marked with wider arrows. This service will expose two operations: 1) retrieving a report of license plate matches over a given time period, and 2) updating the list of "wanted" license numbers.

4.2 Mobile Robots

In many aspects of our life, stationary and purely software-based services may not be sufficient for achieving business, scientific and technological goals. Concurrent tasks spread out over certain areas may benefit from the use of mobile robots based on the FPGA technology. SOA hardware opens new possibilities while also presenting new challenges for designers and programmers.

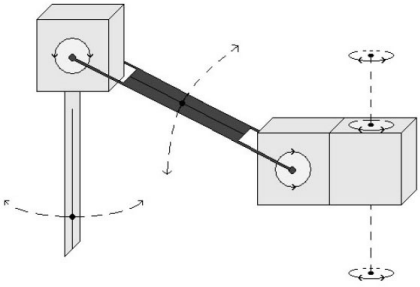

Fig. 7. Hexapod leg construction scheme

Possible Applications. Mobile robots armed with sets of tools and sensors are a powerful instrument for scanning open areas and performing certain tasks, such as:

— ploting up temperature and humidity charts;
— creating three-dimensional terrain maps;
— examinining hard-to-access places;
— inspecting dangerous areas.

Mobile robots might prove very useful in the course of scientific research, as well as during times of war, natural disasters or terrorist attacks [10]. They could search for people trapped inside destroyed buildings and help transport important equipment, e.g. medicine, water containers, food or even mobile phones.

Construction Aspects. There are many different ways to build mobile robots [11], including, in particular, wheeled and hexapod designs. The first one introduces serious mobility problems whenever the path of the robot is blocked by obstacles. Hexapod robots, equipped with six legs, each of which contains three servomotors, seem to be a better solution. See Figure 7.

Thanks to the use of its legs, a hexapod enjoys an extra degree of freedom and can operate in rough terrain, but in comparison to wheeled robots, such devices require more complicated movement algorithms. As mentioned above, a hexapod robot requires three servomotors for each of its six legs, which means that a total of eighteen servomotors are required. To achieve smooth, synchronized movement, many of the servomotors should be activated simultaneously. A typical servomotor controller is based on a microprocessor, which results in pseudo-parallel operation and movement. Our proposed servomotor controller, based on an FPGA circuit, solves this problem. This chapter presents a H-SOA solution, a hexapod spider robot built as a service-oriented hardware tool. See Figure 8.

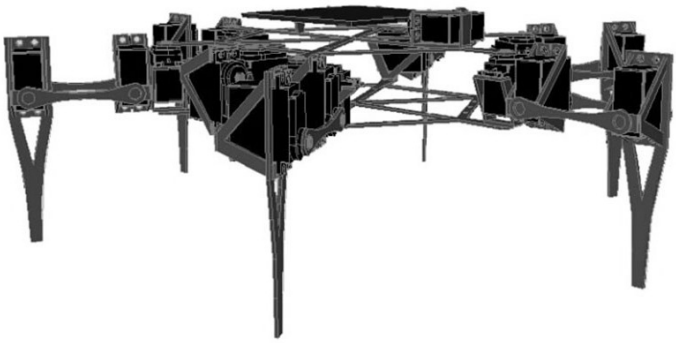

Fig. 8. H-SOA robot visualization

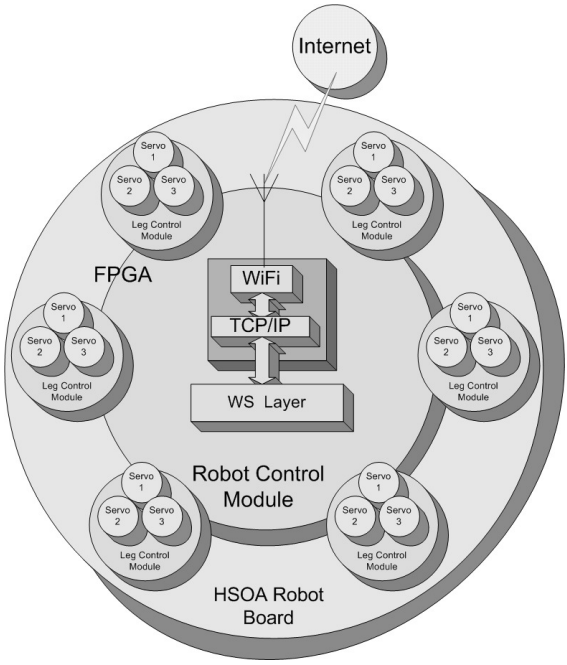

Fig. 9. Simple H-SOA robot model

The robot is supposed to be controlled through an H-SOA module based on an FPGA circuit. A simple model of the robot is shown in Figure 9. Each servomotor receives its own independent controller. The robot has six legs. Each of the legs occupies three servomotors and has its own seperate Leg Control Module.

Apart from legs, the robot can also be equipped with a set of sensors, a camera, a GPS module, a WiFi adapter and a photovoltaic battery. The camera gives the robot a local orientation and the ability to avoid obstacles while the GPS module provides it with global orientation and the ability to reach the desired coordinates. The WiFi adapter allows the robot to expose its services to clients. The small DIGI Connect Wi-Me module shown in Figure 10 is suggested for use with our robot [12].

Fig. 10. Digi Connect Wi-Me

The module supports a 802.11b WLAN, TCP/IP and data transfer of up to 230 kbps. While relatively slow, this transfer rate seems to be fast enough for sending commands to the robot and receiving useful data. To ensure wireless freedom for the robot, it should be backed by a Lithium-Ion (Li-Ion) battery and a photovoltaic solar battery.

H-SOA mobile robots, implemented as hardware-based SOA devices are potentially powerful tools for modern challenges. In accordance with the SOA paradigm, they can be easily used by any SOA client. In this way, H-SOA mobile robots become remote sensors and remote actuators, offering their services to any local or stationary SOA application.

4.3 Reconfigurable Hardware Data Processing Board for Sun™ SPOT

Sun™ SPOT [13] is a small electronic device developed by Sun Microsystems and used as a sensor node in sensor networks [14]. The device is based on the Squawk Java Virtual Machine. The detailed description of SPOT device can be found in section 6 or in [15]. To extend hardware capabilities of the SPOT device we have developed and built an FPGA-equipped extension board. A schematic block diagram of the extension board is shown in Figure 11.

The prototype board uses the ALTERA Cyclone III EP3C25 device. It is also equipped with an FPGA configuration memory (EPCS16) and a serial flash memory (Atmel AT25F512) used to store board properties (as required in the SPOT extension board specification).

SPOT can exchange data with a FPGA using one of its standard interfaces: I2C, SPI, USART, or a user-defined 8-bit unidirectional or bidirectional data bus. The usage of these interfaces depends on the user and application-specific needs.

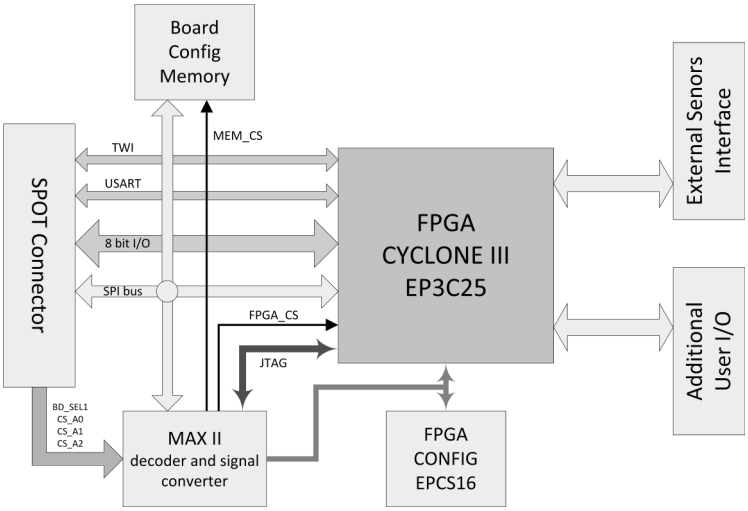

Fig. 11. Block diagram of the SPOT hardware data processing board

The design of the board gives the SPOT device the ability to reconfigure thr FPGA at any point in time. Moreover, the default configuration of the FPGA, stored in EPCS16, can also be freely changed.

The most important feature of the board is flexibility. In the simplest scenario, the FPGA can be configured to provide the user with a large number of additional digital I/O lines (to connect a large number of simple sensors or actuators). By using the same hardware it is also possible to connect one or more different sensors to SPOT (through the hardware data processing board). Each of these sensors (for example, a small digital camera) can produce a large amount of rapidly-changing data. In this case, FPGA can be configured to process such data, reduce its size, and extract useful information. In both cases a properly-configured FPGA device reduces the amount of data that should be sent through a sensor network to a final processing (gateway) node.

Unfortunately, the FPGA configuration cannot be changed freely. Whenever a new configuration is needed, the user has to describe the new functionality of the device in one of its Hardware Definition Languages (Verilog, VHDL, etc) and prepare an appropriate configuration using dedicated software (for Cyclone III devices, ALTERA provides a free version of Quartus II software). Once a new configuration is prepared it can be uploaded to the FPGA device at any time. One of the features of this board is the ability to reconfigure FPGA without direct access to the SPOT device. Wireless access to the host device through a sensor network is sufficient.

The hardware part of our board is at an early prototype stage. The current prototype is too large to fit inside a SPOT case. The final version of the board should be of an appropriate size.

5 Mobile SOA

An important issue with systems based on the SOA paradigm is that the characteristics of mobile environments should be considered during the design process. This type of environment is characterized by wireless communication (such as WLAN, HSPA, Bluetooth etc.) and the use of mobile devices (smartphones, palmtops etc.) or sensors (medical, industrial). Extending SOA applications to lightweight mobile devices (the MSOA concept, i.e. Mobile SOA [16]) allows business processes to operate "everywhere" and services to be provided using wireless technologies. Unfortunately, adapting existing services to mobile applications is not easy and requires changes in application code.

Many kinds of services can be used in MSOA:

— Information services (stock exchange/business services, news);
— Messaging (Skype, Google Talk);
— Multimedia (Youtube, MobiTV);
— Social networking (Facebook, Twitter);
— Gaming (mobile gambling achieved through wireless communication);
— Medical services (diagnostic services);
— Billing and payment services;
— Localization services;
— Company-specific services.

Below are some of the requirements for Mobile SOA [17]:

— End-to-end software infrastructure that enables quick and dynamic deployment of user-focused, network-based services;
— Client-side applications/services (on the mobile device) that provide access to key network-based services for compelling user experience;
— Set of core network-based services and the ability for operators to add and manage these services across all enabled (mobile and stationary) devices on their network;
— Tools for developers to create new services that the operator can easily deploy to enabled devices.

The Mobile SOA (Fig. 12) consists of three layers: the Application Layer, the Service Framework Layer, and the Services Layer. The highest (Application) layer allows consumers to run applications (and services) on mobile or desktop devices.

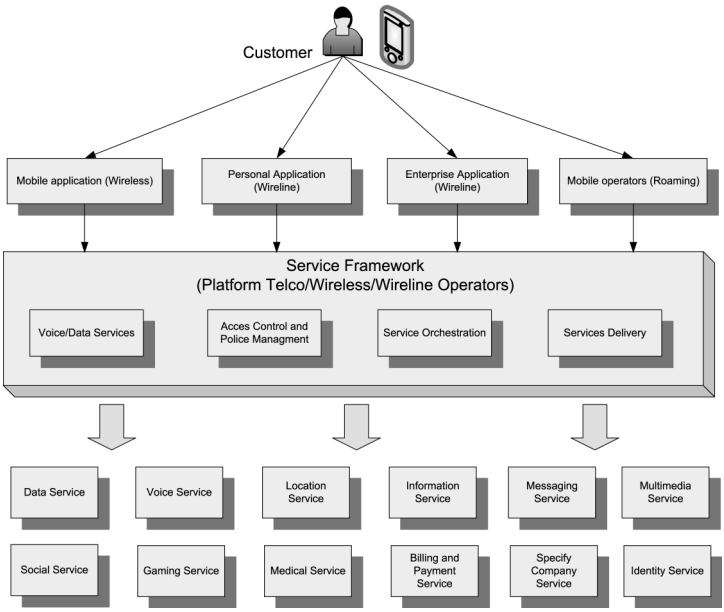

Fig. 12. Mobile SOA

Running applications use the Telco/Wireless/Wireline operator's platform (the Service Framework Layer) which delivers and manages the services. The lowest (Services) layer contains a set of available services.

In the Mobile SOA concept, mobile devices can enter three roles: that of a client, a server or a client/server (Fig. 13). In the first case, the user of the application can find and execute services that are located on other computers (servers). In the second case, the mobile device is itself a server of services, although this mode is rarely encountered because mobile devices possess limited capabilities and typically aren't able to act as servers. Having said that, the area of mobile environments (such as wireless or cellular networks) is rapidly expanding and there are some devices (e.g. complex smartphones, PDAs etc.) that are suitable for fulfilling the role of a server. In the third case, the performance of the device should be sufficient (in terms of processor speed, memory etc.) to play the client/server role.

The benefits of Mobile SOA include:

— The ability to install and uninstall services on mobile devices;
— Their availability on many device platforms (limited dependency on device features);
— Availability "everywhere" and "anytime";
— The ability to support service delivery to multiple screens (mobile, TV, PC).

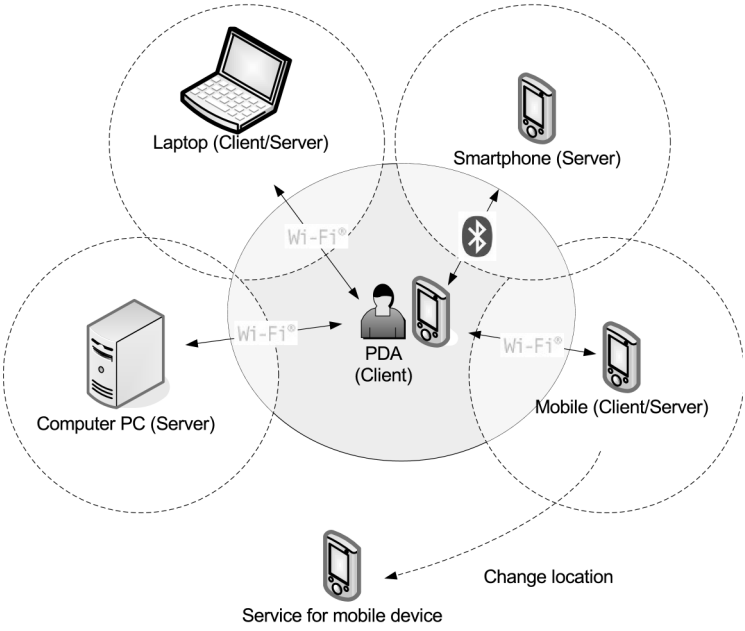

Fig. 13. Device roles in the Mobile SOA concept

The above-mentioned benefits are attainable because MSOA is a distributed architecture with a number of potential applications (wireless, enterprise, etc.) and mobile characteristics provided by the service framework (telco and wireless operators). This architecture can be implemented in a variety of ways (Web Service, OSGi, etc.), providing many kinds of services (such as voice, location, medical or messaging platforms).

6 Sensor Networks

A sensor network [14, 18] is an infrastructure comprised of sensing, computing, and communication elements providing the ability to instrument, observe, and react to events and phenomena in a specified environment. Wireless sensor networks emerged from military applications such as battlefield surveillance; however, they are now used in many industrial and civilian areas of application, including industrial process monitoring and control, machine health monitoring, environment, and habitat monitoring, healthcare applications, home automation, and traffic control. There are four basic components in sensor networks (Fig. 14):

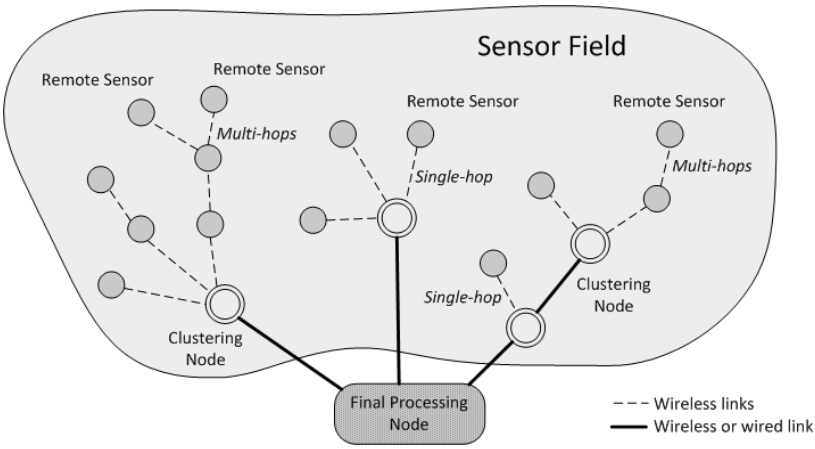

Fig. 14. Sensor network

— a number of distributed or localized sensor nodes;
— the interconnecting network (usually wireless-based);
— the clustering node (may be integrated with one of the sensor nodes);
— the final processing and sharing data node.

Sensor nodes are small electronic components capable of sensing many types of information from the environment, including temperature, light, humidity, radiation, the presence or nature of biological organisms, geological features, seismic vibrations, specific types of computer data, and more. A typical sensor node can be visualized as a small computer. It consists of a processing unit with limited computational power and limited memory, sensors equipped with specific conditioning circuitry, a wireless or wired communication device, and a power source. It can optionally be equipped with dependent modules such as an energy harvesting module, additional communication interfaces, and a hardware data processing module. Essential features of sensor networks are: limited power, the ability to withstand harsh environment conditions, mobility of nodes, dynamic network topology, robustness against communication errors and node failures, heterogeneity, large-scale deployment, and unattended operation.

In specific conditions some types of sensors may acquire large amounts of data. Due to the limitations of the interconnecting network, the sensor node should be able to reduce the size of the collected data. The limited computational power of the sensor node can make this task very difficult. In such a situation we have to consider equipping the sensor node with additional hardware which can not only extend its computational abilities but also help to process the excess data. Collecting and processing data through a single sensor node or network can be

considered as a service. We can therefore try to integrate all sensor nodes or the sensor network itself into a SOA system.

To describe the idea of integration of sensor networks with SOA systems we have to present the technologies chosen for our research, discussing their benefits and restrictions.

Sun SPOT. The small mobile device domain is represented by Sun SPOTs. A Sun SPOT device (Fig. 15) is a small, wireless, battery-powered experimental platform. The device is programmed almost entirely in Java to allow regular programmers to create projects which used to require specialized embedded system development skills. The Sun SPOT includes a range of built-in sensors as well as the ability to easily interface with external devices [15]. Sun SPOT devices are programmed in Java Mobile. A SPOT device is built by stacking a Sun SPOT processor board with a sensor board and battery. There is also a smaller version called base station which does not include sensors nor battery, and whose main responsibility is maintaining a connection between free-range devices and a PC. A Sun SPOT has a 180MHz 32-bit ARM920T core processor with 512KB RAM and 4MB Flash. Each device has a 2.4GHz IEEE 802.15.4 compliant radio with an integrated antenna and a USB interface (used to connect to a PC). The basic Sensor Board includes the following:

— a 3-axis accelerometer (with two range settings: 2G or 6G);

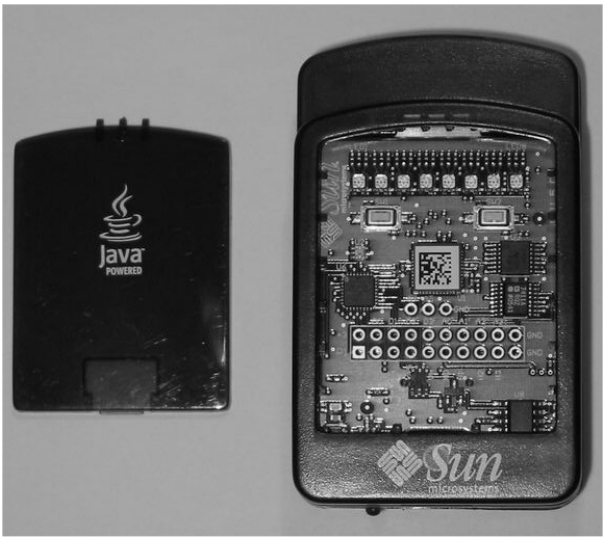

Fig. 15. Sun SPOT device

— a temperature sensor;
— a light sensor;
— 8 tri-color LEDs;
— 6 analog inputs readable by an ADC (Analog to Digital Converter) ;
— 2 momentary switches;
— 5 general-purpose I/O pins and 4 high-current output pins.

The architecture of SPOT devices is designed to use a dedicated sensor or custom board which can replace the standard sensor board. Sun SPOTs run the Squawk Java Virtual Machine (JVM). Squawk is an open-source virtual machine for the Java language whose goal is to examine new ways of building virtual machines. The aim of the Squawk project is to develop as much as possible in Java for portability, ease of debugging, and maintainability. Squawk is Java-compliant and CLDC 1.1-compatible, with a limited memory footprint. It is designed for use in small, resource-constrained devices. To reduce memory use on mobile devices many tasks are performed on the host machine: verification, optimization, and transformation of client classes into Squawk's internal object representation. An important feature of the Squawk VM is its isolation mechanism whereby an application is represented as an object. One or more applications can run in a single JVM and each application is completely isolated from all other applications. In addition to the standard semantics of isolates, Squawk implements isolate migration. An isolate running on one Squawk VM instance can be paused, serialized to a file or over a network connection, and restarted in another Squawk VM instance [19]. CLDC also imposes some restrictions, such as:

— Java Reflection API is not available;
— Package java.io.Serializable cannot be used;
— Remote Method Invocation (RMI) is not supported;
— User-defined class loaders have to be used.

These restrictions complicate operating Sun SPOT devices in enterprise frameworks which sometimes rely extensively on the aforementioned capabilities. For instance, Tuscany uses the Reflection API to wrap managed components.

Service Component Architecture. To integrate mobile devices with enterprise systems a middleware layer is required. In order to avoid creating everything from scratch we have chosen an existing platform: Service Component Architecture (SCA). SCA is a promising emerging standard for enterprise software development. Its specification contains a model for composing applications and systems from services. In this sense it is very much in line with the Service Oriented Architecture (SOA) principles [20]. SCA augments prior approaches for implementing services and is based on open standards such as Web Services [21]. The SCA programming model for building applications and systems is based on the idea that a business

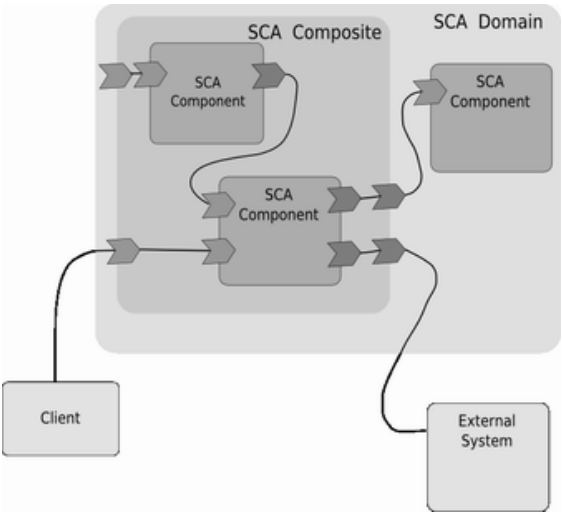

Fig. 16. SCA application

functionality can be provided as a series of services, assembled together to create a solution that serve a particular business need [22]. Applications are constructed as composites which may contain new, custom services and business functions derived from existing systems and applications. This programming model aims at encompassing a wide range of technologies for the implementation of service components, and providing support for communication and cross-invocation of operations [22]. Components can be created in different programming languages with the use of frameworks and development environments commonly applied to these languages. SCA also provides a wide range of communication and service access technologies that are in common use – including Web Services, messaging systems, and Remote Procedure Call (RPC) interfaces. A sample SCA application illustrating the presented concepts is shown in Figure 16. Some details of the SCA framework depend on its implementation. Tuscany is the most widely-used Java-based SCA platform, enjoying the support of the Apache Software Foundation [23]. Being an open-source project, it provides open access to the code developed in Java v. 5 by a large and active group of programmers.

FPGA. To build a flexible and multi-purpose hardware extension for a sensor node we need a solution which gives us the ability to adapt or change its functionality with a minimum of hardware alterations. It should also be able to process a large amount of data effectively and with relatively low power requirements. In our opinion, FPGA-based solutions can meet these conditions. A description of this technology is presented in Section 3.

6.1 Middleware for the Integration of Small Mobile Devices in the SOA

To ensure the integration of the mobile device domain with enterprise SOA systems two architectural approaches can be considered. It is possible to set up a service container and expose services on mobile devices allowing direct access from enterprise applications. This requires the implementation of advanced communication protocols on the mobile devices. On the other hand, we want to keep devices as small as possible and reduce power requirements. Reconciling all these requirements can be very difficult. We can, however, benefit from communication protocols designed for sensor networks and develop middleware which integrates mobile devices and standard computers. Thus, it is possible to expose services while reducing communication complexity.

Requirements. As is clear from the presented discussion, one of the principal requirements of our solution is allowing the interaction of SCA components with mobile devices. While mobile devices remain responsible for integration with the environment, SCA components provide a way to implement business logic and to communicate with other enterprise systems. Together, they form a new type of distributed enterprise applications. Such interactions should satisfy the following functional requirements:

— Synchronous and asynchronous communication and remote operation invocations should be provided;
— Exposing services for other SOA applications should be possible.

These functional requirements may be extended with the following non-functional needs:

— Ease of use — the provided interaction mechanisms should be easy to use and efficient in spite of the complexity of interaction performed between components located in different domains;
— Genericity — the solution should not impose limits on the set of supported applications;
— Conformance with SCA design principles — integration should follow, the SOA and SCA paradigms and should not violate any principle inherent in these concepts.
— Scalability — the integration pattern needs to be usable not only with single, separate mobile devices but also with large sets of devices.
— Performance — the solution should not introduce a substantial overhead in terms of communication with small devices.

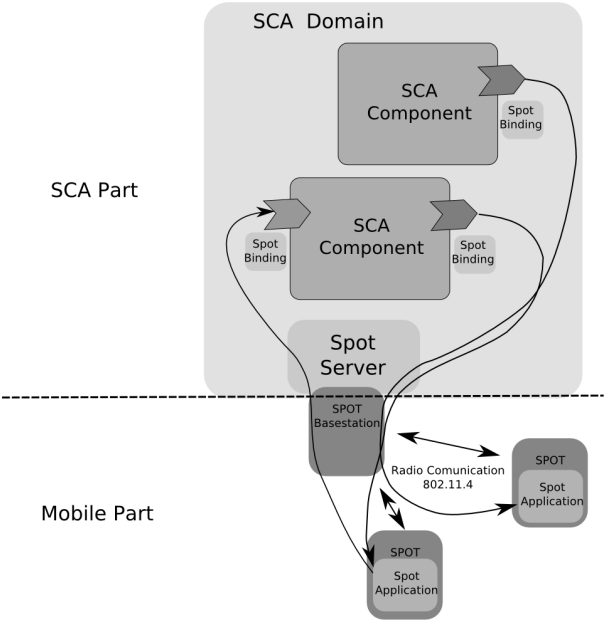

Fig. 17. SPOT binding pattern

Architectural Patterns. In this section the proposed architectural patterns for the integration of components running in the SCA Domain with Sun SPOTs are described. They are discussed in the context of the functional and non-functional requirements defined in the previous section.

The SCA specification introduces an element called *binding*. As already explained in the previous section, bindings are used by services and references to describe the protocol used to call or provide a service. This protocol is required to manage the communication between components in a given domain, but should also support the access to external services. We present our solution in Figure 17. Three aspect of communication are considered:

— a low-level communication API, handling send and receive operations over suitable data links;
— data marshaling and unmarshaling;
— an addressing schema for communicating entities or localization protocols.

All these aspects are supported by the SCA domain and are hidden from programmers whenever a suitable binding is used. For communication with external software components residing on mobile devices these low-level communication is-

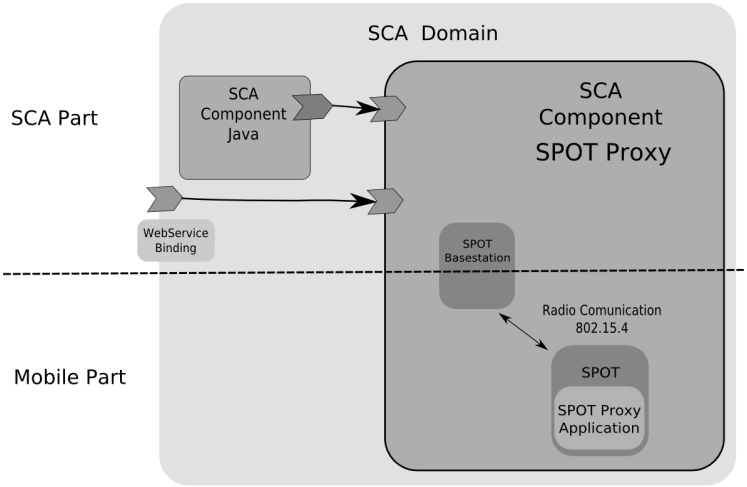

Fig. 18. Spot proxy pattern

sues have to be addressed directly while developing a new binding. The benefits of using the binding pattern are as follows:

— the SPOT binding implements low-level communication aspects with the spot device;
— SCA components are used and constructed in a standard way;
— a full bidirectional control over an application working on the SPOT device is provided.

On the other hand, there are also certain disadvantages:

— all problems related to remote communications have to be resolved by the programmer;
— it is necessary to implement separately a program for a SPOT device to handle all issues related to its deployment;
— Application code has to be altered in many places whenever changes are required.

Based on our experience with the previous version, we decided to create a solution which would be much easier to use and better integrated with the SCA environment. A component was developed which provides access to device functions while hiding the complexity of the communication with mobile devices. We call this component the SPOT Proxy. It is a special SCA Component which exposes the functionality of a physical device and forwards requests to the real devices seen in Figure 18. From the developer's point of view, communication with the mobile

device is greatly simplified. Our code marshals parameters and issues requests to the SPOT device. The device unpacks these parameters and method descriptions, calling the specified actions. Results are retrieved and sent back to the server. Finally, the component method returns the output values. There is also support for communication issues (e.g. request timeouts). The benefits are as follows:

— we do not have to worry about communication with the SPOT device;
— we do not need to provide code for the mobile device;
— all functions can be used in an easy way;
— the component matches SCA requirements very well.

There are only two problems which might potentially prove important:

— we cannot deploy our own code to the device;
— communication performance problems may emerge in programs which frequently call mobile device functions.

Proxy components facilitate the rapid development of distributed applications. We can also benefit from the use of different SCA components and binding implementations, and their automatic cooperation. On the other hand, we have to bear in mind that in some cases communication overhead and bandwidth requirements may become prohibitive.

6.2 Extension Modules for the Sun SPOT Device

To extend the hardware abilities of the Sun SPOT device we propose equipping it with two types of extensions. The first, a reconfigurable hardware data processing board, is described in section 4. The second gives the SPOT device the ability to safely control high-voltage electric devices. The PM1 power module is a simple standalone semiconductor relay. It uses the SHARP S202S12 optocoupled solid state relay, equipped with a zero-crossing circuit. It can be connected to the SPOT device through one of its H0-H3 signals located at the P1 header on the SPOT sensor board. The module can control high-voltage circuitry (powered from the electrical grid). For example, by using this module the SPOT device can control light switches in one or more rooms. Another example is operating the SPOT device as a thermostat. Users can issue a temperature adjustment request remotely (over the network), and the SPOT device can compare this request against its own temperature sensor, subsequently deciding to power a heater on or off without any additional activity in the sensor network.

New, extended abilities of the SPOT device can be exposed as services for other SOA applications with the use of the SCA-based solutions described above.

7 Mobile Devices

The development of mobile technologies such as new wireless standards (HSPA, 802.11n), operating systems (Android, Symbian, Windows Mobile), and devices (HTC, Samsung, Android) allows the use of mobile devices in support of the Mobile SOA concept.

Two main technologies are important for the implementation of MSOA on small mobile devices:

— Web Service (WS);
— OSGi.

7.1 Using Web Services

The WS technology is available for the Java Micro Edition (JavaME) environment (J2ME Web Services APIs — JSR172 optional package [24]), the Android system and the Windows Mobile (Phone) system (.Net Compact Framework). Of these, the most popular and most widely used is the JME environment, as all modern mobile phones running SymbianOS, Mobile Windows (Phones), and specialized corporate operating systems for cellular phones implement the WS solution.

The JSR172 specification describes how a JavaME client can use WS on a mobile device. This specification defines:

— an API for using SOAP and WSDL;
— a JavaME XML parser;
— mappings between WSDL (Web Services Description Language) and JavaME data types.

The J2ME Web Services APIs is a subset of the JAX-RPC 1.1 specification, which introduces some limitations, namely:

— support only for the WS-I Basic Profile;
— support only for version 1.1 of SOAP;
— no ability to define methods of communication with a service registry (such as UDDI).

It is possible to use other Web Service implementations (for example Apache Mirae) on older devices which support JavaME but do not implement JSR 172. The next generation of the WebService JavaME solution on mobile devices will follow the specification of the Service Connection API for JavaME (JSR 279). The main features of this solution are:

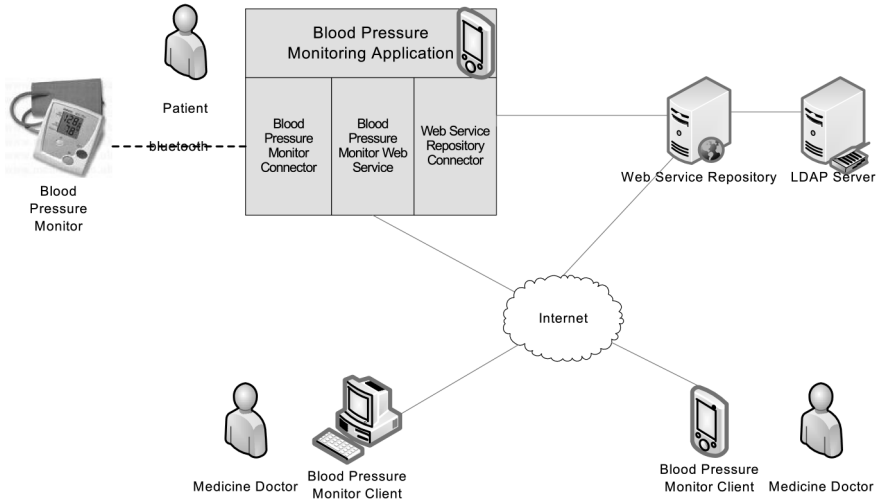

Fig. 19. Blood pressure monitoring application

— mechanisms for locating services;
— authentication;
— support for the Liberty Identity Web Services Framework (ID-WSF);
— support for services implemented in various technologies (for example UPnP).

The Web Service technology (JSR172 package) can be used to develop many types of services on mobile devices. A promising area of application is the medical domain. Figure 19 describes a possible scenario where a medical application (using JSR172 and Bluetooth communication) helps measure blood pressure. The patient uses a cellular phone with Java ME and Bluetooth module to connect to the network and upload personal blood pressure data. The application acquires this data and registers the medical service in the repository (Web Service Registry and LDAP Server). The patient's physician can then select a blood pressure monitoring service from the repository (containing the network address of the patient's device) and, by invoking a specific method, acquire further information on the patient's health. This scenario was implemented at the Department of Computer Science of the AGH University of Science and Technology in Krakow. The presented medical application is merely an example of a solution designed according to service principles. Similar methods and concepts can be used in other scenarios and programs that provide services on mobile devices.

7.2 OSGi — The Dynamic Module System for Java

The OSGi framework is a Java-based service platform that allows remote management and configuration of components (bundles). Some examples where this framework has been implemented include Eclipse Equinox, Apache Felix, Knoplerfish and Concierge on mobile devices. The OSGi framework implements a component model for installing (and deinstalling), activating, and stopping bundles without restarting the application. Every bundle can also export services for use by other components. OSGi allows dynamic management (as well as uploading) of resources such as program code. This feature is very important for lightweight mobile devices which have limited memory and CPU, as it enables optimal usage of such devices. The presented solution can thus be applied in scenarios (and applications) that provide services on mobile devices (the Mobile SOA concept is described in section 5). In previous versions of the OSGi specification (prior to 4.2), communication between bundles (services) was only possible on-device. Implementations of the OSGi framework on mobile devices include Concierge, Sprint Titan (Windows Mobile), and ProSyst (Windows Mobile and Android). The newest version of OSGi (4.2) includes Distributed OSGi (DOSGi), providing remote access to services (bundles). Similar solutions are introduced by R-OSGi [25] and the Distributed OSGi With Web Service Project.

8 Summary

Integration of SOA enterprise software systems with services deployed on mobile devices or intelligent sensor networks is recognized as a very important requirement leading to better exploitation of IT potential in numerous domains, inluding security, medical systems, transportation, logistics, and others.

The development of such systems introduces certain technical challenges. The first refers to the limited hardware resources which can be used for implementation of services. This problem can be solved by the FPGA technology, ensuring efficient implementation of well-defined software services. FPGA chips can be added to existing devices such as cameras, sensors, servomechanisms, etc. creating specialized services implemented as hardware modules. Programming FPGA devices involves technical obstacles which are related to the extensive data processing parallelism supported by such devices. Development of high-performance FPGA code calls for new algorithms and low-level (VHDL) programming. The upside is that once implemented, the functionality performed by low-level code can be exposed as a service and reused in many devices.

Another issue is related to the process of developing software for mobile or embedded systems, based on conventional processors but limited in terms of resources

(i.e. computing power and memory). This precludes the use of software platforms developed for standard computers. Programming such systems is therefore quite complicated and leads to integration problems related to changes in programming models and limitations of JVMs. The diversity of the programming model makes combining standard computers with mobile and embedded devices difficult. Support for a uniform software development platform spanning traditional computer systems and small mobile devices is therefore very important.

In spite of the diversity of small device programming paradigms, the services offered by such devices should be integrated with enterprise systems in a uniform way via binding components available for commonly-used ESB platforms. This is an essential feature of Embedded SOA.

References

1. Erl T. SOA Design Patterns. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2009.
2. Deugd de S., Carroll R., Kelly K., Millett B., Ricker J. Soda: Service oriented device architecture. IEEE Pervas. Comp. 5, 94–96, c3, 2006.
3. Chang K.C. Digital systems design with VHDL and synthesis: an integrated approach. Wiley-IEEE Computer Society Press, New York, 1999.
4. Thomas D.E., Moorby P.R. The verilog hardware description language. Kluwer, New York, 1998.
5. Pellerin D., Thibault S. Practical FPGA programming in C. Prentice Hall, New York, 2005.
6. Duda R.O., Hart P.E. Pattern classification and scene analysis. Wiley, New York, 1973.
7. Canny J. A computational approach to edge detection. IEEE Trans. Patt. Anal. Machin. Intell. 8, 6, 679–698, 1986.
8. Duda R.O., Hart P.E. Use of the hough transformation to detect lines and curves in pictures. Comm. Assoc. Comp. Machin. 15, 1, 11–15, 1972.
9. Ballard D.H. Generalizing the hough transform to detect arbitrary shapes. Patt. Recogn. 13, 2, 111–122, 1981.
10. Billah M.M., Ahmed M., Farhana S. Walking hexapod robot in disaster recovery: Developing algorithm for terrain negotiation and navigation. World Acad. Sci. Eng. Technol. 42, 60, 328–333, 2008.
11. Walking Machine Catalogue. `http://www.walkingmachines.org`, 2008.
12. Digi International. `http://www.digi.com`.
13. Sun™ SPOT Theory of operaton. Red Relase 5.0. `http://www.sunspotworld.com/docs/Red/SunSPOT-TheoryOfOperation.pdf`, 2009.

14. Sohraby K., Minoli D. Wireless sensor networks technology, protocols, and applications. Wiley, New York, 2007.
15. Project SUN SPOT. http://www.sunspotworld.com/.
16. Tergujeff R., Haajanen J., Leppnen J., Toivonen S. Mobile soa: Service orientation on lightweight mobile devices. IEEE Int. Conf. on Web Services, 1224–1225, 2007.
17. Shapiro A., Frank A. Mobile SOA: End-to-end Java technology-based framework for network services. [online], 2008, [presentation - JavaOne Conference].
18. Ilyas M., Mahgoub I. Handbook of sensor networks: compact wireless and wired sensing systems. CRC Press, New York, 2005.
19. The Squawk Project. http://research.sun.com/projects/squawk/.
20. Erl T. Service-oriented architecture: concepts, technology, and design. Prentice Hall, New York, 2005.
21. IBM Service Component Architecture. http://www.ibm.com/developerworks/library/specification/ws-sca/.
22. OSOA — Service Component Architecture Home. http://www.osoa.org/display/Main/Service+Component+Architecture+Home.
23. Apache Tuscany Project. http://tuscany.apache.org/.
24. J2ME Web Services Specification. http://jcp.org/en/jsr/detail?id=172, 2004.
25. Rellermeyer J., Alonso G., Roscoe T. R-osgi: distributed applications through software modularization. Middleware, 2007.

Selected Aspects of Management in SOA

Bartosz Brodecki, Jerzy Brzeziński, Dariusz Dwornikowski, Jacek Kobusiński, Michał Sajkowski, Piotr Sasak, and Michał Szychowiak

Institute of Computing Science
Poznań University of Technology,
Piotrowo 2, 60-965 Poznań, Poland
{Bartosz.Brodecki,Jerzy.Brzezinski,Dariusz.Dwornikowski,Jacek.Kobusinski,
Michal.Sajkowski,Piotr.Sasak,Michal.Szychowiak}@cs.put.poznan.pl

Abstract. SOA is a strongly business oriented paradigm for building heterogeneous distributed systems composed of loosely coupled services. Due to these characteristics, management in SOA introduces new challenges and problems, that were not so crucial in legacy distributed systems. As a consequence, vast amount of work is being done in the field of SOA management, mainly in those constructed according to the Web Services standards and technology. On the other hand, the REST architecture style still lacks methods, tools and standards when it comes to management. In the following chapter, we discuss selected aspects of management: performance, reliability and security. Regarding the performance aspect, we consider the means of adapting SOA systems for optimizing or meeting SLA and QoS requirements with respect to selected efficiency metrics. The proposed solution comprises monitoring and management execution at the level of running environments, operating systems and services. The reliability aspect deals with fault tolerance and with a special insight in distributed failure detection in SOA systems. Finally we consider security policy languages for SOA taking into consideration the system, as a whole.

Keywords: SOA, management, monitoring, failure detection, security policies, security language, SLA, QoS

1 Introduction

Service Oriented Architecture (SOA) is a strongly business focused paradigm for developing heterogeneous distributed systems. Such an environment is composed of loosely-coupled services that provide some business logic through clearly defined

interfaces. Due to this fact, the services can easily cooperate with each other, forming business processes.

SOA systems can also be characterized as being autonomous, under the authority of different management domains and service providers. There are two major technologies for building SOA systems: SOAP-based Web services (also referred to as Big Web services, Web services or WS-* [1]) and REST-based Web services [2, 3]. The first technology is based on the SOAP protocol (*Simple Object Access Protocol*) and the ESB (*Enterprise Service Bus*) paradigm, where functionalities offered by services are described by WSDL (*Web Services Description Language*) documents. A lot of WS-* standards have been developed within the W3C, the OASIS consortium or by vendors themselves. The composition of web services into business processes is driven by definitions provided by XML based languages like BPEL (*Business Process Execution Language*), which is the most widely used nowadays [4]. While SOAP-based Web services originated mainly from vendors and enterprises, REST-based Web services (REST is an acronym for *Representational State Transfer* [2]) originated directly from the community. The RESTful Web services technology is built around resources, that are accessed with HTTP methods (*GET*, *POST*, *PUT*, *DELETE*), with the use of the well known URI (*Unified Resource Identifier*) addressing scheme.

The motivation for management in SOA systems is rather straightforward and similar to that in legacy distributed systems. The constant monitoring and control of the system is used to provide availability and reliability, which together result in retaining SLA (*Service Level Agreement*) and QoS (*Quality of Service*) at the requested levels. However in SOA, due to the strong business orientation, these levels are crucial not only for a client, but also for a vendor. In order for a service to be chosen by a client, vendors have to assure the highest quality of their products, so as they can effectively compete within the electronic market.

Due to the above features, the task of management in SOA systems is challenging. As far as SOAP-based Web services are concerned, there are several standards that focus on management [5, 6, 7, 8], major amount of work having been done by vendors within software platforms and running environments. However these specifications are rather complicated and not always fully implemented in software solutions. Therefore, in some cases the RESTful Web services can be an interesting alternative to WS-*. It is because they are lightweight and can be easily integrated into the existing Web infrastructure, due to the HTTP protocol they are based on. Therefore, management tasks in SOA can be redefined to match the infrastructure already used in the Web, but with the emphasis on providing QoS and SLA to satisfy business client's needs. In order to meet these needs, mechanisms for provisioning high availability and reliability are deployed. However, SOA systems are very dynamic and these mechanisms need to be continuously adapted to the constantly evolving environment and the management itself has to be dynamic in order to be successful.

SOA paradigm is also heavily used in cloud computing [10, 9, 11], as a mean for supporting common interfaces for cloud infrastructures, on which software services and application stacks are deployed by the vendors. However, SOA acts only as an higher level access layer, it is not a core feature of the cloud computing. Therefore, this chapter does not deal with the management in cloud computing, such as virtual machines management, resource reservation and load distribution [12]. Nevertheless, we clearly state that aspects of management presented in this chapter can be easily applied to cloud computing paradigm.

In this chapter, we focus on three main aspects of management: performance, reliability and security.

The performance aspect concerns the means for adapting the system and its components in order to optimize or meet SLA and QoS requirements with respect to selected efficiency metrics of the processing of services. It comprises monitoring and management of execution on the level of running environments, operating systems and services. The reliability aspect deals with the fault tolerance with a special insight in distributed failure detection in the RESTful Web services systems. Finally, the security aspect concerns security policy languages for SOA that take into consideration the system as a whole.

The chapter is organized accordingly to the aspects described above. First, we focus on performance aspects of SOA systems management in Section 2. In Section 3 we describe a failure detector service, as a mean for providing fault tolerance. Section 4 concerns the security policy languages for SOA. Finally, Section 5 contains concluding remarks for the whole chapter.

2 Proactive Management in SOA

2.1 Basic Definitions and Problem Formulation

SOA systems can be generally characterized as multi-layered, as shown in Figure 1. Taking a top down approach, we can distinguish the following layers: virtual organizations, business processes, atomic services, running environment, operating system, and hardware. Monitoring and management concepts exist at every layer, but differently at each layer. In SOAP-based Web services, the emphasis is put on management at business processes and ESB levels, which provide routing support between web service providers and consumers. A vast amount of work in management and adaptation is done in orchestration engines. There BPM (*Business Process Management*) patterns are utilized along with BPEL (*Business Process Execution Language*) which provides extensions for ensuring QoS in distributed executions of business processes.

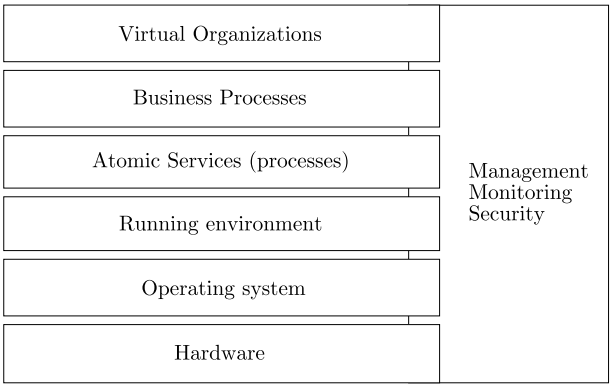

Fig. 1. A multi-layered view of SOA systems

RESTful Web services are lightweight and hide the complexity of WS. They ideally integrate into existing Web infrastructures, instead of adding unneeded burden of standards on software components. Unfortunately, existing Web infrastructures are not ready for adapting RESTful Web services without obstacles. This also concerns the management and monitoring tasks.

2.2 Related Concepts and Contributions

This section deals with concepts and contributions related to the management task in SOA: WS-Management and Web Services Distributed Management (WSDM), Autonomic Computing, intelligent events processing and proactive management (prediction, forecasting and diagnostics).

The WS-Management [13] is a SOAP based protocol for managing different components (e.g. servers, devices, applications). To be complete, the WS-Management utilizes different standards from WS-* for addressing (WS-Addressing), event publishing (WS-Eventing) and security (WS-Security). What is interesting, WS-Management, despite being a pure WS-* standard, shows some REST properties by using the WS-Transfer standard, which adds (*GET*, *PUT*, *DELETE* and *CREATE* commands to its namespace. Unfortunately the similarity ends on this level, as the communication is still SOAP based. Another standard family to mention is the WSDM (*Web Services Distributed Management*), maintained by the OASIS organization [14, 5]. It is strongly based on WS-RF (*Web Services Resource Framework*), hence every resource in WSDM has its WS-RF representation. Each resource is described by an XML document defining its properties and capabilities that can be monitored and managed. In fact, WSDM consists of two separate substandards: MUWS (*Monitoring Using Web Services*) and MOWS (*Monitoring Of*

Web Services). MUWS is responsible for monitoring with the usage of WS-* and the SOAP protocol [15, 16]. Every managed resource is represented by a web service endpoint that can be reached by any capable client — so called, manageability consumer. The properties that can be managed by the client, are published as so called manageability capabilities that list the operations which can be executed on the given resource. MOWS on the other hand is used for monitoring and managing resources constructed with SOAP-based Web services standards [6]. It defines a list of common metrics that can be used to monitor a web service (e.g. *NumberOfRequests*, *LastRequestSize*, *OperationalState*). The WSDM standard family is very mature and supported by many vendors, yet it is only usable in the SOA systems created according to the WS-* technology, protocols and standards. On the other hand, [17] proposes an approach to distributed management of RESTful Web services based on the Atom publishing protocol. However, the authors focus only on the configuration aspect of management, not taking adaptation and control into consideration.

Another concept that can be applied to SOA systems is Autonomic Computing (AC), proposed by IBM in 2001 [18]. The main idea of AC is to develop self-managing systems as an answer to the rapidly growing complexity of computing systems. The goal of these systems is to minimize the need for human interventions in the management process. The AC systems are characterized by the following properties:

— self-configuration — an automatic configuration of the components, integration into the system, discovery of the topology and registration in a distributed environment;
— self-healing — an ability to find, diagnose and correct faults and failures;
— self-optimization — automatic monitoring, capacity planning and parameters tuning, in order to achieve the highest QoS and SLA;
— self-protection — an ability to protect itself from external threats, attacks and adversarial noise, as well as cascading failures in the system.

The AC concept introduces the idea of an autonomic element (Fig. 2), an entity composed of a managed resource (e.g. device, application, system) and an autonomic manager that continuously monitors the resource and undertakes appropriate actions, according to the situation. The idea strongly resembles the control loop concept from control theory, and is often referred to as the *MAPE* loop. This acronym represents the main functions of an autonomic manager: *monitor*, *analyze*, *plan*, *execute*. The idea proposed by IBM nicely integrates with SOA systems, but unfortunately not many full implementations exist. In fact, some of authors claim that autonomic computing will introduce further complexity, due to the fact that the programming languages and software components responsible for acting as autonomic managers will become more complex than the system they manage [19].

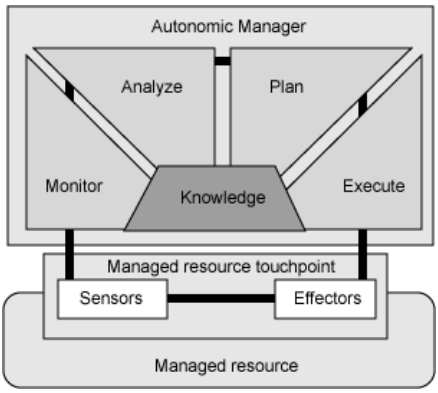

Fig. 2. Autonomic element

The Intelligent Event Processing (IEP) concept is used for continuous monitoring of the managed system's components for any appearances of anomalies, resource exhaustion and failures. Legacy monitoring solutions, in most cases employ polling the probes distributed within the a distributed system. This technique is widely used but treats every source separately and does not take any relationships under consideration. An additional computation has to be performed in a monitoring station, e.g. to correlate or average the acquired data. A solution to this problem can be treating data from the monitoring system as a big stream of events that can be further processed by specialized intelligent event processing (IEP) components. IEPs are event oriented software components that utilize stream processing algorithms. They provide:

— an easy correlation of events of different types and originating from different sources;
— on-line stream processing in time shifting windows;
— pre-processing streams with mathematical functions e.g. average, sum, etc.;
— great efficiency, small memory footprint.

Thanks to the above features IEP can be used as a pre-processing component, which facilitates initial inquiry of a data stream for causality and anomalies, before passing it to the monitoring system, where further analysis can be carried out.

The proactive management concept is aimed at meeting the needs of high availability and reliability, where reacting after a failure took place (reactive management), is not sufficient; proactive mechanisms able to anticipate failures and resource exhaustion need to be deployed in an online and efficient way. One of such mechanisms is the prediction of failures in time series. To achieve it, many

authors use stochastic modeling and statistical methods [20, 21, 22, 23, 24]. However, they mainly focus on offline methods or static analysis of previously gathered data. Also, the models used in these works often assume stationarity of time series, whereas in real life applications such models hardly match. Another problem that is not fully considered is the correlation of different time series and parameters, in order to achieve more accurate failure prediction.

The failure diagnostic mechanism analyzes the failure after its occurrence, finds its root cause and tries to fix it permanently, to avoid it in the future. In systems of the SOA complexity, where plenty of loosely coupled components take part in business process executions, this task becomes rather difficult, yet very important. Several authors employ Bayesian inference and Bayesian networks for the task [25, 26, 27]. After identification of tests, components and dependencies among them, a network is constructed. Upon failure, the evaluation takes place. The result is a set of components that with some probability could be the origin of the failure. The apparent problem of such an approach is that a model has to be constructed by an expert, who has to provide knowledge about the dependencies between components and assign *a priori* probability distributions to Bayesian network. In the case of the topology change, this would have to be done again. The obvious challenge in this field would be to find a way to dynamically detect causality of failures among components and construct such networks automatically. Another problem with this approach emerges directly from the latter challenge. A bayesian network needs to have initially assigned prior probability distributions. Analogically, the challenge is to find a way to automatically assign them without an expert input.

We focus on the monitoring and management of operating systems, running environment (virtualization clusters) and atomic service levels to achieve high QoS and SLA levels. To fulfill that task, we employ proactive mechanisms, such as prediction, diagnostics and anomaly detection, to adapt the system to the ever-changing environment. In our work, we employ simple prediction methods in order to predict future events and act accordingly before they happen. Additionally, we assume that long term, exact prediction is not necessary. Instead, we focus on fast and short term probabilistic methods to compute hints for the decision system, so that it can adapt the system to the needs of distributed processing. Our research concerns mainly the possibilities of using moving average, exponential smoothing and regressive methods, for forecasting in time series.

2.3 Proactive Management System for SOA

As a part of our work, we develop methods and tools for the monitoring and management of SOA systems, as well as research on metrics for the evaluation of such environments. The results comprise prototypes of software modules, as well as

methods and ideas. These modules together form a system that is able to monitor the SOA infrastructure and manage it in order to achieve SLA and QoS goals.

Functional Requirements. We denote functional requirements that should be met by the system. The first requirement is interoperability with external monitoring systems, such as SNMP and Nagios. In order to achieve that, adapters provided for different sources of monitoring data should be able to transform it into a unified format. The unification of data is the second functional requirement we state. Next, we assume the requirement for flexibility in defining the metrics and sources of data by a unified language. This feature would be helpful in the unification of configurations throughout the system. From the performance point of view, we denote the requirement for the system to be able to calculate metrics on the fly, in parallel to data gathering. This will ensure that less processing will have to be done after data acquisition. The ability to change parameters of virtual machines and migrate them automatically is the next requirement. It states, that the virtualization cluster needs to be equipped with mechanisms that enable management system to control the virtual machines hosted within it. Additionally, we consider that modularity, reliability, and scalability are non functional requirements for the considered system.

System Architecture. Figure 3 presents a proposed architecture for the management system for SOA. The *monitoring module* is fused with the *metrics module*. Together they are responsible for the acquisition and transformation of data coming from proprietary agents, external monitoring systems (SNMP, Nagios) and the virtualization environment. The transformation of data is understood by its normalization to the unified format and its processing by the *metrics module*, responsible for metrics calculations. The *prediction module* is responsible for the analysis of the metrics in order to make short term predictions and anomaly detection.

The architecture does not depict it explicitly, but almost every component within the system possesses RESTful Web services API, in order to communicate with other parts of the system. The API also makes it possible to distribute the components of the system in order to achieve the reliability of the management system. The REST-based API was chosen for the prototype implementation, but the system is designed to manage all types of SOA systems and the SOAP API can be easily added in further implementations.

Metrics for SOA systems. In order to manage the system, the knowledge of its state has to be acquired. This process is called monitoring. The key question which arises is: what needs to be monitored to obtain the most valuable knowledge. Thus, definition of metrics is the first step in the process of management of the system. In our work we focus on reliability, replication and performance metrics. However,

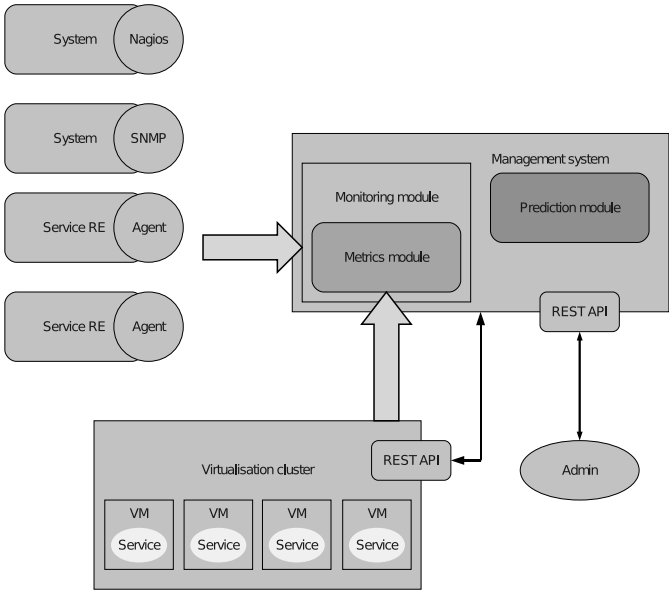

Fig. 3. Management system architecture

we maintain an open list of metrics, in order to be able to add new ones, when there is such a need. The list is as follows:

— Performance metrics (from a system and a process point of view, where applicable):
  - system related metrics (average load, CPU usage, memory usage, number of I/O operations, storage device usage, network interface traffic);
  - utilization of service/node;
  - requests per second to a service;
  - average request processing time for a service.

— Reliability metrics:
  - mean time to failure (MTTF);
  - mean time between failures (MTBF);
  - mean time to recovery (MTTR);
  - errors in a time period;
  - availability.

— Replication metrics:
  - number of replicas of a service;
  - number of requests to a replica in a time period;
  - service replicas consistency level.

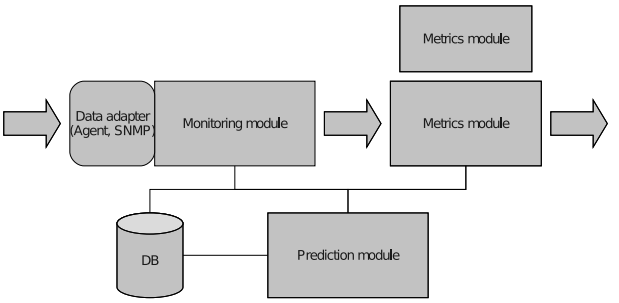

Fig. 4. Metrics module

We additionally enable using metrics which can be composed as an aggregation and normalization of several performance metrics. The function used to determine the value of this metric can be an average, a weighted average or e.g. a logistic function. Such a calculated metric can be used as an input for a decision system or traffic load balancers, such as the QoS Proxy module, described later.

In order to describe metrics and relations between them, we created a XML based language for defining metrics with features of metric abstraction (templating), inheritance, mathematical transformations and aggregations. The first attempt to define that language, as well as examples of usage, can be found in [28].

Figure 4 shows the concept of a data flow within the *metrics module*. Having been acquired by *adapters* from the *monitoring module*, the data is normalized to one unified internal format. The stream is then passed to the *metrics module* which calculates metrics values according to definitions passed by the *metrics manager*. The *prediction module* is connected to the database source (relational database or plain files) and both the *metrics module* and the *monitoring module*. The values of the calculated parameters can be passed further to metrics consumers, e.g. modules responsible for rules execution or graphics modules.

The *metrics module* (Fig. 4) is implemented with the Intelligent Event Processing component — Esper [29].

Monitoring and metrics modules. Being strictly connected with the *metrics module*, the *monitoring module* is responsible for data collection from different sources, that are accessed by the *adapters*, e.g. Nagios, SNMP or a failure detector. Thanks to the modular construction, additional *adapters* can be easily developed if a need for new data sources appears. Additionally, we develop proprietary *agents*, which are also based on IEP components. Moreover, the *agents* have an ability to calculate metrics on their own and in the case of the lack of connectivity with the central module, they can retake its role.

Along with the *monitoring module*, a module responsible for prediction and anomaly detection is developed. It provides a convenient API, which can be used to easily implement different methods. Currently it supports a statistical anomaly detection method, based on comparing the empirical distribution of historical data with the current values. Figure 5 shows an architectural concept of this module.

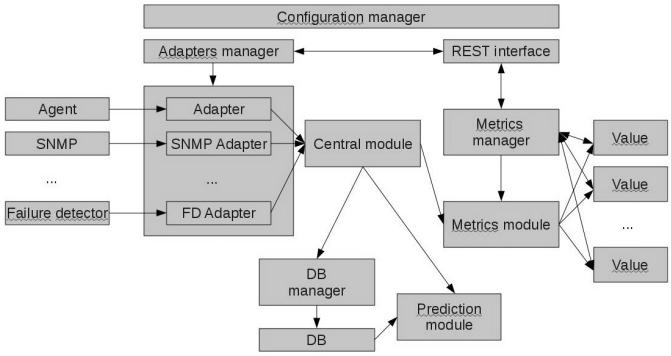

Fig. 5. Monitoring and metrics modules architecture

The data is acquired from *adapters*, either SNMP, Nagios or failure detector. An *adapter manager* is responsible for the creation and control of *adapters*. The stream of data is passed to the *monitoring module*, based on the IEP component. The metrics are defined in the *metrics module*, which is managed by a *metrics manager*. The *prediction module* works in close cooperation with the *storage module*, in order to be able to use the historical data kept in the database or plain files.

Management module. In order to achieve a high level of QoS in a SOA system, one has to manage and adapt the environment to meet this need. The analysis of the data acquired from the *monitoring module* acts as the trigger for the *management module*. We focus on the running environment, operating system and service layers. We use a virtualization cluster composed of six physical nodes that are able to run virtualized operating systems. The cluster is based on the KVM (*Kernel Virtual Machine*) solution for Linux. The effectors we can act on, are the resource configurations of virtual machines (VM), such as: number of processors assigned to the VM, the allocated memory, number of storage devices (virtual hard drives). We are also able to control the effective creation of virtual systems, which is a convenient in terms of capacity planning and security separation. Our environment also supports so called *live migrations* of VMs between cluster nodes. That means that the *management module* can dynamically transfer the running virtual system from one physical node to another, without any visible downtime. That can be

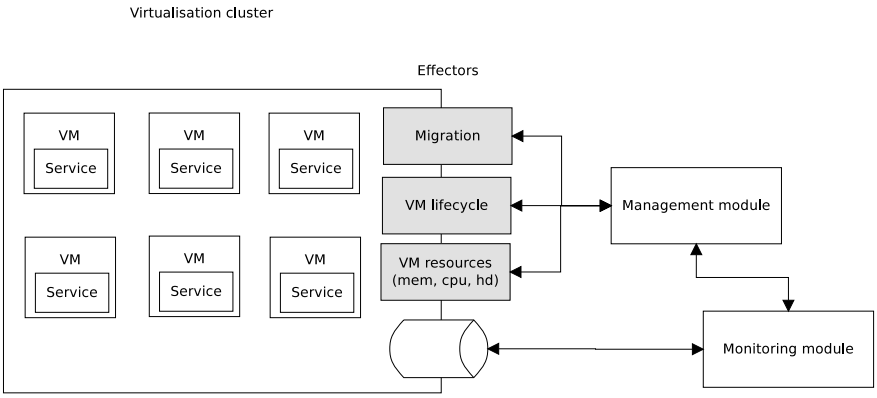

Fig. 6. Architecture of management system

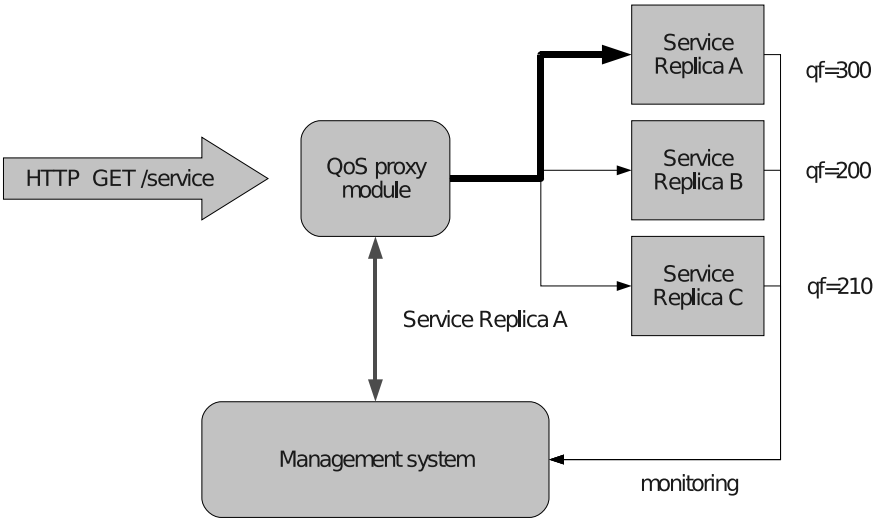

Fig. 7. QoS proxy idea

achieved by using a common storage device for virtual disks, shared by all cluster nodes. In our case we use iSCSI network attached storage. Through continuous analysis of the monitoring data from the whole cluster, every physical node separately and the virtual machines, we are able to guarantee QoS, by a fair balancing of the VMs among physical hosts.

On the service layer, we developed a proxy server component which is responsible for adaptive load balancing of HTTP requests to active replicas of a service.

The balancing is done on the basis of *quality factor*, a parameter that can be loosely described as a metric or a group of metrics aggregated by a mathematical function. An administrator of the system can define which metrics should be part of the quality factor for the whole set of replicas of the service.

Figure 6 presents an architecture of the management system for the virtualization cluster, whereas Figure 7 shows the idea of the QoS proxy. In the latter, a simple scenario of adaptive load balancing of three services can be seen. Upon receiving a HTTP request, QoS Proxy asks the *management system* where it should be directed. The management system constantly monitors the quality factors of the services. The highest value of a quality factor is currently available for the service replica A, hence the decision to direct the request.

2.4 Future Works

From a technological point of view, our future work will be focused on refining the software components we have presented. We plan to fully implement the metric description language module to support sophisticated metrics relationships, especially inheritance. The monitoring agents, which cooperate with monitoring module, will be supported with the ability of dynamic extension by remote code transfer. To achieve that, we incorporate the OSGi framework.

The virtualization cluster is planned to be fully equipped with a RESTful Web services API. Finally, a prediction library will be created, supporting more complex methods like Holt-Winters exponential smoothing and various regressive methods.

On the other hand, from a research point of view, we will focus on exploring methods of prediction in time series. We plan to put an impact on simple statistical and probabilistic mechanisms and try to adjust them, so they could work in an online way. We plan to take a deeper insight in methods that provide capabilities of inference about causality and could work as diagnostics mechanisms upon failure detection.

3 Failure Detection for SOA Systems

3.1 Basic Definitions and Problem Formulation

A *service failure* occurs when a service does not perform its work in a specified way. SOA-based systems composed of dozens or even hundreds of services, hosts and links, a failure occurrence becomes highly probable and its detection is a fundamental starting point to improve the fault-tolerance of the system. A widely

used method to perform failure detection is the heartbeat technique. This method is based on periodical sending messages to other processes. These messages carry information about a process's status, determining whether the process is still alive or crashed. This popular technique assumes that the message maximum delivery time is known, and it uses timeouts to decide which service should be suspected to be the crashed one. A *Failure detector* can be characterized by different properties like *speed of detection*, i.e. the latency between a failure occurrence and its detection, or *accuracy* which describes the capability of not suspecting correct processes.

A traditional binary model of a failure detector's response has serious limitations when considering failure detection as a generic service. These restrictions come from the fact that there are different expectations of detection time and accuracy. Some user applications require aggressive detection even at the cost of accuracy, while others are interested in a more conservative detection with a low level of false suspicions instead of fast but inaccurate decisions. By using a binary response generated by a failure detection service, it is in general impossible to fulfill these requirements. It is because the timeout threshold that marks the line between correct and crashed services should be different, for different failure detection service clients. Therefore, in [30], the authors propose a new type of a failure detector, called *accrual failure detector*. Their proposal assumes transferring the decision about the service's crash to the end user application. The failure detection module only provides some non-negative real value that is further interpreted by the application. This value represents the current *suspicion level* of the monitored service, in opposite to the traditional one, which interprets the state of the service by itself and returns a binary response. The failure detector φ [31], is an implementation of this idea. It can estimate the state of the monitored service on the basis of the query response time.

Analyzing the existing failure detection mechanism, we found that the existing solutions have different restrictions when considering network topology, scalability or client requirements. Moreover, these solutions are often created for other architectures of distributed systems and their adaptation to SOA is not trivial.

3.2 Related Concepts and Contributions

The first frameworks that are worth mentioning are FT-SOAP [32] and FAWS [33], based on the FT-CORBA standard. With regards to the failure mode, they are examples of crash fault tolerant frameworks. They consist of a group of replica services that are managed by external components. FT-SOAP follows the service approach in fault tolerant CORBA and provides transparent fault tolerance by upgrading SOAP with additional components to support fault detection and replication management. It also extends WSDL to inform the clients of the replica

information. On the other hand, FTWeb [34] follows the interception approach from FT-CORBA and proposes an infrastructure, where a dispatcher acts as a proxy for the client requests. In both solutions failure detection is performed by external components (detector and notifier) that interact with the component responsible for replication management. Unfortunately, all these frameworks consist of dedicated components that represent single points of failure within the system.

FTWeb is another attempt to apply fault tolerance techniques from FT-CORBA concept to WS-* domain. This infrastructure has components that are responsible for calling concurrently all the replicas of the service and analyzing the responses before returning them to the client. The main objective of this proposal is to tolerate stop, omission and value faults. To make this possible, it uses an active replication technique. Thema [35] is an example of a framework that can tolerate certain Byzantine faults. It is a demonstration of an application of Castro-Liskov's Byzantine fault tolerant (CLBFT) [36] algorithms in the SOA domain. It uses dedicated libraries on clients, services and some supplementary services to convert normal service requests into Byzantine fault tolerant (BFT) requests. Because of many complex interactions involved in the algorithms mentioned, the performance of this proposal is rather poor, thus it can be problematic. Lightweight Fault Tolerance Framework for Web Services [37] is based on a similar approach. It does not use any specialized components responsible for fault management. Fault tolerance is achieved through a consensus-based replication algorithm. Finally, WS-FTA [38] is an infrastructure that adapts the existing WS-* standards and by providing specialized components allows to tolerate failures. It is responsible for increasing service availability and maintenance of group of service replicas in a consistent state. It proposes a new model of failure detection on group and single replica levels.

All presented frameworks are applied to WS-* domain. Most of them try to adopt certain concepts from the CORBA middleware approach and till now there has been no single proposal that has been applied to RESTful Web services. Therefore, the design and implementation of a flexible failure detection service is well motivated, especially when taking into account the increasing popularity of the RESTful Web services concept.

3.3 Failure Detection Service

The general concept of the proposed failure detection service is presented in Figure 8. As it is shown, the distributed service consists of loosely coupled failure detectors (FDS components) which can be combined into local groups, in order to increase performance and reliability of the service. A client of the service can send a request to any of the known detectors, to detect the failure of some service. Moreover, there is a possibility to setup a callback action. In such a case, the client sends all

the required data and registers its request. When the components of the service continuously monitor the requested services and when the failure occurs, the client is informed about it by the service.

Functional Requirements. The analysis of the existing solutions lead us to formulate requirements that should be fulfilled by the proposed failure detection service. First, the service should be able to monitor any RESTful Web services, HTTP servers, SOAP-based Web services. The list of the monitored services can be changed any time by clients of the service. Moreover, to reduce the client's involvement in the monitoring process, a callback mechanism should be implemented. It allows a client to request a callback in the case when specified conditions are fulfilled. Due to the distributed character of the failure detection service, the configuration of the service should be performed dynamically through Web interfaces or by using XML static files, which makes it more convenient. Modular and open design will ensure that new functionality as well as modification of existing will be easily adopted. Such an open architecture makes the proposed service usable in many practical scenarios.

Failure Detector Architecture. The failure detection service (FDS) is a distributed service composed of detectors that can be located on different nodes. The architecture of such a detector (an FDS component) is presented in Figure 9. The design and implementation of a single FDS component is highly modular. It is composed of several specialized modules that perform dedicated tasks. Such an

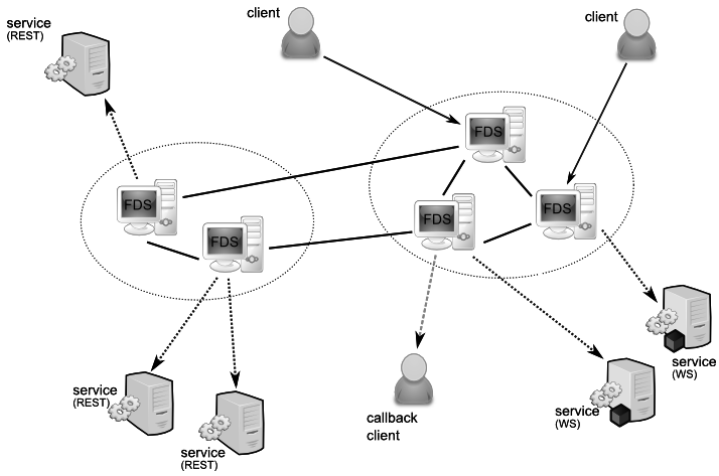

Fig. 8. General concept of failure detection service

architecture allows simple modification of existing elements and future extension and upgrades.

A general overview of FDS components is presented below.

Resources Manager. The Resource Manager is the main module responsible for the management of resources. Its another important role is integrating and exchanging the information between other modules, and interacting with clients. Addressing all the resources is possible through a unique URI. Certain actions are realized using simple HTTP operations, which makes the solution RESTful Web services compliant.

Monitor. The Monitor is a module directly responsible for monitoring requested services. Internally, it implements an accrual failure detector φ, that was mentioned earlier. It calculates the value of the suspicion level of each monitored service by taking into account the query response time. Further, the client can interpret this value according to its preferences and decide about the service state. This module consists of a sub-module called the *Callback Manager*, which is responsible for the management of client's callback requests and for the activation of callback procedures.

Broker. The Broker module is responsible for the communication between the failure detection service components. The communication that occurs between the detectors can be divided into local and inter-group respectively. The module implements mechanisms for both levels. At the local level, the logical topology has

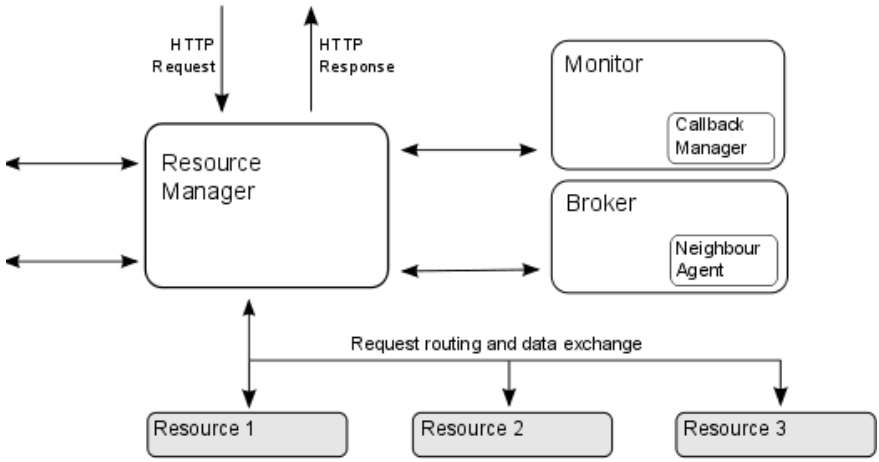

Fig. 9. The architecture of failure detector (FDS component)

a mesh structure. A probabilistic approach used during communication makes the local group more fault tolerant and flexible to network failures. This is the result of using a gossiping (epidemic) protocol which offers more robust spread of information and some form of randomness in peer selection. When the information about the state of a service is locally not available, an inter-group message exchange occurs. The communication scenarios take into account the performance and network bandwidth consumption in order to try to reduce the inter-group communication to the necessary minimum. The sub-module called the *Neighbor Agent* takes care for continuous connection between groups.

Configuration manager. The Configuration Manager module takes care of service configuration. It can read configuration parameters from static XML files, as well as dynamically with a Web interface. Both ways give an access to all configuration parameters. It is possible to configure the way the service operates in many aspects. This makes the service very flexible and customizable.

Registry manager. The Registry Manager is responsible for the management of an internal registry of information about the monitored services and the detectors responsible for their monitoring. This registry is used in the case when the information about a monitored service is unknown and should be located on other service nodes. Instead of starting a costly process of other detector querying, one can try finding required information in the local repository. This repository is updated automatically at the local level, by all group participants.

3.4 Future Works

The construction of an universal failure detection service is not an easy task in a distributed environment, due to the limitation resulting from the system characteristics as well as final users' requirements. Properties like: scalability, adaptability and flexibility seem to be some of the most important ones in the context of building a common service. In order to fulfill these requirements the proposed service has a modular architecture. Such a design allows an easy application of further improvements. Our prototype service currently achieves basic functionality. In the next step, we plan to evaluate its performance and reliability. We believe that there is still place for optimizations in the module responsible for communication. The work to support monitoring of SOAP-based Web services is ongoing. We also expect that the evaluation results will allow better tuning of the monitoring module.

4 Security Policy Languages for SOA

4.1 Basic Definitions and Problem Formulation

A *security policy* represents security-related constraints or conditions imposed on interactions or the interaction participants. Each of these constraints or conditions is expressed in the form of a *policy assertion,* which may be evaluated and is ensured by policy enforcement mechanisms. We consider the following fundamental entities related to security policy: a *target* that is to be protected (e.g. resource or service); a policy *subject* performing actions on targets (e.g. principal or client issuing access requests); and mechanisms, by means of which the target is protected.

First, let us briefly review the key features for an appropriate SOA-compliant security policy language. In fact, a considerable number of separate issues must be taken into account. Although a remarkable effort has been made to find a comprehensive solution, there is still no language able to express all the security requirements.

In order for such a language to be complete, easy to use and manage, and to fully support the SOA-based systems, the following features should be provided.

The first feature is to fully support a distributed loosely coupled service-oriented system with an automatic composition. A SOA policy is required to express not only authorization (access) *restrictions* (authorizing policy *subjects* to access policy *targets*), which is typical today, but also security *obligations* (by which a policy target expresses requirements that any interaction subject is expected to fulfill), and *capabilities* (by which a subject specifies mechanisms it can provide to fulfill target-side requirements) that must be satisfied by all participants in a service interaction (e.g. the communication protection will be set up most likely with negotiation of obligations and capabilities). The ability to express obligations and capabilities is imperative for true SOA environments, where interactions (possibly nested) can be established automatically and dynamically without any human intervention or coordination.

The second feature concerns a distributed decision support. Large scale systems, especially SOA-based, can be composed into a *federation.* In this context we need an ability to acquire policy rules from other federated systems, define distributed trust relationships and rights delegation from one principal to another.

Moreover, modularity and extensibility are important features. It is hard to design a security policy language suitable for any kind of system and any security requirements. There are many security policy languages which are well suited for specific needs. An ideal security policy language should be modular in order to allow easy extensions for custom needs.

Another feature concerns the multilayer design of policies. Security policies are specified by humans but policy definitions must be easily processed by computers. Therefore, a security policy language should have a hierarchical design, in which the bottom layer provides a connection to platform specific security components and security functionalities for the next higher level, and each higher level allows to aggregate policy items in more abstract concepts. At the top, we should have a general human readable policy.

There is also a strong need for simplicity. A policy language should minimize the amount of necessary policy statements in order to keep policies simple and interoperable at a large-scale.

The final feature is a well defined syntax for expressing security relevant issues and allowing formal verification. Policies for large scale distributed systems can be very complex with hardly observable dependencies, incomplete or leading to conflicting decisions. Without an automated evaluation of policies it will be hard to decide about policy correctness.

4.2 Related Concepts and Contributions

Many policy languages for distributed environments have been already proposed in literature. Some of them have reached maturity and received real-world implementations. Here, we review three most representative examples of such security policy languages: XACML, Ponder and SecPAL.

XACML/SAML. XACML (*eXtensible Access Control Markup Language* [39]) is a declarative language based on XML created for managing an access to resources. It describes both, the access control policy and the format of request/response used in access control queries. The current version 2.0 of XACML is an OASIS standard recommended by the W3C consortium.

The XACML's XML Schema definition describes in details its grammar and semantics. The main elements of the language are rules, policies and policy sets. The *rule* element is the basic component of each policy. It consists of: a target, an effect and a condition. The *target* defines a set of resources, subjects, actions and environment to which the rule applies. The *policy* element is described by: a target, a rule-combining algorithm, and a set of rules and obligations. The *rule-combining algorithm* defines a procedure for combining the results of the evaluated rules. The *policy decision* is a value returned by the rule-combining algorithm. *Obligations* describe actions that must be fulfilled in the access control procedure (as logging, for instance). The policy also defines a *policy-combining algorithm* which defines a procedure of combining the results of the evaluated policies. In the following example of policy, a subject (`Administrator`) is permitted to get a full access to a file `/etc/passwd`.

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy RuleCombiningAlgId="urn::first-applicable">
  <Target>
    <Subjects><Subject>
      <SubjectMatch MatchId="urn::function:string-equal">
        <AttributeValue> Administrator </AttributeValue>
      </SubjectMatch>
    </Subject></Subjects>
    <Actions><Action>
      <ActionMatch MatchId="urn::function:string-equal">
        <AttributeValue> full-access </AttributeValue>
      </ActionMatch>
    </Action></Actions>
    <Resources><Resource>
      <ResourceMatch MatchId="urn::function:string-equal">
        <AttributeValue> file://etc/passwd </AttributeValue>
      </ResourceMatch>
    </Resource></Resources>
  </Target>
  <Rule Effect="Permit" RuleId="Permit" />
</Policy>
```

In addition to XACML which specifies access control policies, the Security Assertion Markup Language (SAML [40]) is often used to transport security assertions on service interactions (such as authentication information adjunct to access requests). The SAML assertions can be signed and encrypted. SAML specifies a general syntax and semantics of assertions and describes some facilities to operate on them through bindings and profiles. A binding determines how the SAML requests and responses map onto communication protocols. Profiles [41] illustrate sample use cases with particular combinations of assertions, protocols and bindings. SAML assertions represent statements about a subject of the request, typically a principal (e.g. user or application) requesting the assertion. In general, the assertions contain information that service providers use to make access control decisions. The following types of statements are provided by SAML: authentication assertions, attribute assertions and authorization decision assertions.

Authentication assertions — affirm that the subject was authenticated by a particular authority (using *AuthorityBinding* assertion element) with particular means (*AuthenticationMethod*) at a particular time (*AuthenticationInstant*).

Attribute assertions declare a sequence of attributes associated with the assertion subject (described with *AttributeName* and *AttributeNamespace*).

Authorization decision assertions state whether a requested access has been granted or not. Three Decision element values are allowed: *Permit*, *Deny* and

Indeterminate. These days, the authorization service is likely to be a local service of the service provider, although any external authorization service can be used.

A subject can optionally confirm its identity including a *subject confirmation* evidence manifestation in a SAML request. For example, a subject may identify itself as the owner of a private key associated with a X.509 certificate attached to the request.

Ponder. Ponder [42] is a language for specifying security and management policies developed at Imperial College. In contrast to XACML, Ponder is a declarative, object-oriented and platform-independent language, which maps its records to low level functions in the existing underlying platform.

The basic constructs of the Ponder language are: basic policies, composite policies and meta-policies.

Composite policies are used to group a set of related policy specifications into roles, relationships and management structures. Thanks to that, policies can share definitions and declarations in one scope. *Meta-policies* describe the coexistence of several policies. They can be used to specify the correctness of the policy and conflict detection. Basic policies are security policies related to access control. They include: authorization policies, obligation policies, refrain policies and delegation policies.

Authorization policies specify what actions a subject is permitted to perform to a set of target objects. *Domains* can be used to partition the objects and groups of objects to which the policies apply. Policy subjects and targets are specified using domain scope expressions. Moreover, a *when-constraint* element can be specified optionally to limit the applicability of the policy. Tasks and responsibilities corresponding to the position in the organization are grouped into roles associated with the position. A role is a set of authorization policies describing the rights and obligations for that position.

The body of a basic policy consists of common elements: a subject, a target and a when-constraint. An example of a Ponder's positive authorization policy is shown below:

```
type auth+ serviceManT(subject s, target t) {
 action resetSchedule, enable, disable; } // serviceManT
inst auth+ brService = serviceManT (brManager/, brServices/);
```

The example specifies an authorization policy named `serviceManT` defining actions that a subject of this policy is permitted to perform on a policy target. The last line declares an instance of this policy named `brService` with a concrete subject and a target of the policy.


```
type auth+ serviceManT(subject s, target t) {
 action resetSchedule, enable, disable; } // serviceManT
inst auth+ brService = serviceManT (brManager/, brServices/);
```

Another type of a basic policy is an *obligation policy*. It specifies the actions that a subject must perform on a set of target objects when an event occurs. Obligation policies are always triggered by events and, unlike authorization policies, they are interpreted by subjects. They usually relate to logging the security events concerning the authorization process (which is not what we have defined as *interaction obligations* when specifying policy features).

Refrain policies define actions that a subject must refrain from performing on a target object. The refrain policies can be used when a subject is permitted to perform an action but is asked to refrain from doing so when particular constraints apply.

In Ponder, a subject of a policy has the possibility to delegate actions to another subject. *Delegation policies* are aimed at subjects delegating rights to third-parties to perform action on their behalf. A negative delegation policy identifies which delegations are forbidden.

SecPAL. Security Policy Assertion Language (SecPAL) [43] is a logic-based authorization language from Microsoft. It is a research project intended to balance syntactic simplicity with policy expressiveness. SecPAL allows to define delegation of rights, separation of duties and expiration constrains, among others. Its clear, readable syntax emerges from a very simple structure and statements which are very close to a natural language. Effective decision procedures are formally confirmed by translation into Datalog with Constraints [44]. Finally, extensibility is supported with a possibility to add new constraints to the language. Below, we give a simple example of SecPAL assertions:

```
(1) STS says jbond is a staff
(2) labsrv says jbond can read file://data
(3) jbond says HPC can read file://data/hpc if
      currentTime()<1/04/2009
```

In the example above, the principals are: *jbond*, *labsrv*, *HPC*; and these names stand for public signature verification. Assertion (1) is an identity token issued by STS which is trusted by other servers. This assertion says that user jbond is a member of group (role) `staff`. Assertion (2) expresses a capability for `jbond`, issued by computer `labsrv`, to read its files (`file://data`). Assertion (3) delegates to computer `HPC` right to access a specific file (`file://data/hpc`) on that server, for a limited period of time (`currentTime()<1/04/2009`).

SecPAL has the advantage of automatic translation into XML syntax, which can be widely used by almost any application. Every assertion may be XML-encoded and signed by its issuer.

Comparison. XACML and SAML are widely used standards supported by OASIS and W3C. Their main problem is a very verbose XML notation slowing down

the processing, requiring software assisted policy authoring and making assertion analysis difficult. Their limitations include the lack of interaction obligations and capabilities, no delegation support and no conflict avoidance procedures. Ponder, in contrast, is a mature language but with no wide acceptance. It has a very compact notation but still is hardly readable because of a highly specialized grammar and syntax. Its main advantage in SOA-based environments is the support of the delegation of authority. However, it is deficient in expressing interaction obligations and capabilities. SecPAL is remarkably the most friendly to read. Moreover, it is formally verifiable, has an important feature of flexibly expressing delegation of an authority and offers an automatic transformation to XML. Its ability to express a large set of domain-specific constrains makes it a very complete but complex solution. Limitations include mainly the lack of obligations and capabilities.

As we have seen, none of the presented languages fulfill all the requirements recognized formerly in this section, which limits their applicability for SOA systems (mainly they do not support interaction obligations and capabilities). In the next part of this section, we propose a solution to overcome these limitations.

4.3 ORCA Language for SOA-Based Systems

In the following section, we present our first proposal for a policy definition language for SOA-based systems, which is the first language with an ability to specify not only interaction restrictions, but also obligations and capabilities. Then, a brief overview of the syntax of our ORCA language is also described.

Functional Requirements. Since it is extremely difficult to fulfill multiple requirements that we have recognized, we adopt a layered approach to build a policy language model. Each layer focuses on a particular subset of the requirements, and provides fine tuned building blocks for a global policy of a SOA-based system.

Since the SOA environment spreads across multiple administration domains, it usually requires simple solutions for interoperability reasons. Therefore, we avoid a large number of layers. Our policy language model is decomposed into only four most important layers of abstraction (Fig. 10). At the top of our model, the System Model Layer is intended to specify basic security prerequisites for system model items (i.e. metapolicy). This layer can benefit from UML-based languages, helpful for system model designers. As this issue has been addressed in literature (e.g. [45, 46]) we are not concerned with this layer here. The opposite bottom Physical Layer is device and service implementation-specific, and provides only a low level access to different functionalities managed by the higher layers (examples of such functionalities are file system access control mechanisms or firewall configurations). This layer is also out of the scope of our interest.

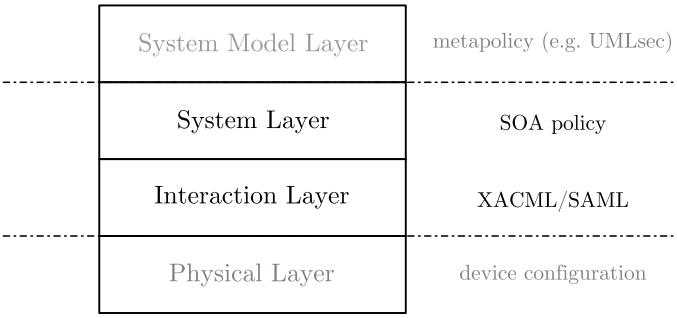

Fig. 10. Layered policy language architecture of ORCA

The System Layer is the best place for a policy language specifying the SOA policy. We believe that the most adequate solution for the System Layer is a high-level human-readable language, well suited for specifying general policy rules easy to manage by people, who do not necessarily have broad computer engineering background knowledge. Unlike suggestions in [47], we use for that purpose a constrained natural language with a very closed syntax and restricted keywords. Thus, we omit the heavily complex problem of parsing a natural language. On this layer, a transformation of policy rules to an abstract logical form is performed for correctness evaluation. The policy correctness is a large and separate issue, and we will not address it here.

The Interaction Layer specifies rules related to particular system components (services) by instantiating general rules inherited from the System Layer into detailed policy assertions for each component. This layer perceives the entire system as a collection of concrete single services, and gathers knowledge about service interactions. On this layer it is possible to make an automatic translation of System Layer rules into XACML/SAML assertions, both being broadly supported in most WS-* implementations.

In the following section, we focus only on the System Layer, for which we have proposed the ORCA (Obligations-Restrictions-Capabilities-Audit) policy language [48]. ORCA provides the following four types of policy rules: restriction rules, obligation rules, capability rules and audit rules.

The *Restriction* rules define access control policy, authorizing subjects with specific rights on targets.

The *Obligation* rules allow a target to express requirements that any subject is expected to fulfill for its requests to be accepted, and vice versa. This may include communication confidentiality protection using one of the supported cryptographic algorithms, or request authentication with an accepted digital signature standard.

The *Capability* rules allow a subject to define a list of alternative mechanisms it can provide to fulfill requirements. For instance, a subject informs that it can

enforce confidentiality protection with only a given subset of cryptographic algorithms or can use only a selected digital signature standard.

The *Audit* rules are used by the security authority to specify logging requirements for interaction events. This corresponds partially to obligations in XACML or Ponder.

Based on a natural language (similarly to SecPAL, for instance) ORCA is simple and easy to understand and to manage in a SOA environment. It uses a *constrained* natural-like language, with only a few syntactical constructions allowed. For example, a generic form of a restriction rule syntax in ORCA is the following:

```
<Subject> X can access <Target> Y for {<action>},
<condition>.
```

where `<Subject>` is a principal (user, role) or a service; `<Target>` is a resource or a service; `X` and `Y` are constants or variables representing names, aliases or identities (local or global, including IP address, URL, UDDI, federated IDs, etc.); allowed `actions` are related to the target (ORCA defines the following keywords: `invoke`, `read`, `modify`, `append`, `create`, `delete`, `full access` or `any access`). Finally, the `condition` restricts the allowed action to a specific time, source, delegation, and it depends on a self-defined predicate (the latter allows simple policy extensions). As we may see, the syntax is pretty clear and easy to follow. A simple restriction rule might be as follows:

```
Role manager can access https://secret/ for {full access},
from local_network.
```

Please note that the above example includes a localization (source) restriction (`local_network` alias). The next example shows a sample prohibition rule:

```
Role manager cannot access https://secret/statistic for {any
access}.
```

A sample predicate condition might be the following:

```
User X can access file://Y for {full_access}, if
owner(file://Y)=X.
```

As stated earlier, a very important condition for SOA-based systems is delegation. A generic syntax of a delegation definition rule is:

```
<Subject> X can delegate <Subject> Y for <Target> Z,
<condition>.
```

The delegation assertion will be included in a request message, e.g. as an SAML assertion or an instance of new type of WSS security token [49]. Then, the allowed delegations may be used to express access restriction:

```
Service http://secret/x-srv can access
```

```
https://secret/x-file for {read, append},
on behalf of user j_bond.
```

Two generic forms of an obligation rule syntax are:

```
<Subject> X must <action> with {<attribute>}, <condition>.
<Target> X requires to <action> with {<attribute>},
<condition>.
```

where `<action>` is `authenticate`, `sign` or `encrypt`; {`<attribute>`} is an ordered set of attribute IDs (e.g. WSS-compliant names of accepted digital signature algorithms).

Generic forms of a capability rule syntax are:

```
<Subject> X can access <Target> Y for {<action>},
<condition>.
<Subject> X can <action> with {<attribute>}, <condition>.
<Target> X can <action> with {<attribute>}, <condition>.
```

Finally, a generic form of a distributed trust rule syntax is following:

```
<Target> X can trust <Subject> Y for {<action>}, <condition>.
```

where `<Subject>` is a principal or an authority.

ORCA Framework Architecture. Components of the ORCA framework are presented in Figure 11 ([48]). It is a fine-grained framework intended to flexibly manage the security policy of a SOA-based system.

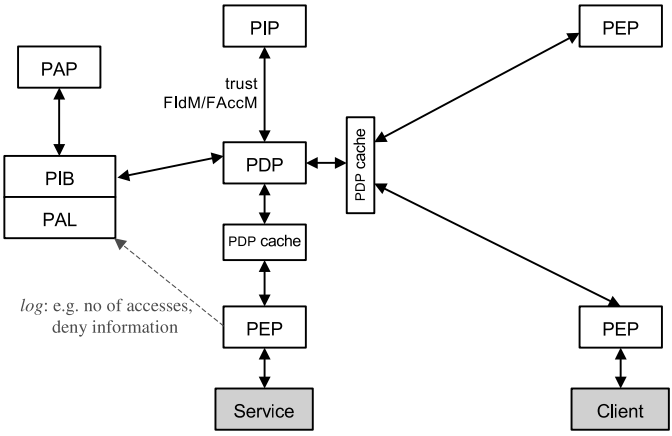

Fig. 11. ORCA framework architecture

The *Policy Administration Point* (PAP) is used to define policy rules and perform any necessary correctness evaluation. It is also responsible for maintaining all inter-domain relationships (e.g. trust definition rules). Rules defined with the use of PAP are stored in policy repository — the Policy Information Base (PIB).

The *Policy Decision Point* (PDP) is a crucial component of the overall policy architecture. It makes the decisions about granting requested access to resources, issued by policy subjects from a local domain.

The *Policy Enforcement Point* (PEP) is responsible for generating decision requests to the PDP, executing decisions received from the PDP and piggybacking security assertions required by PDP to service interaction messages. PEP is additionally in charge of enforcing obligations and capabilities (retrieved from PDP), obtaining the principal's credentials, and managing the credentials accordingly to a delegation, if one is defined.

Typically, each domain has its own local policy, thus it has its own PDP. However, several PDPs can be provided to efficiently evaluate the policy decisions, when necessary. Also optimizations, such as decision caching, can significantly reduce the number of policy evaluations. This motivates the explicit introduction of a PDP cache component, transparent to the PEPs.

The *Policy Information Point* (PIP) is used for acquiring additional information, unavailable locally for the PDP (in a local PIB), necessary to resolve proper decisions. Typically, PIP obtains information from other administration domains. This can be useful in federated systems (or Virtual Organizations) where some part of knowledge needed to take policy decisions can be distributed across multiple autonomous systems. PIP may be e.g. an external Identity Provider for Federated Identity Management (FIdM) or an external Trusted Authority acting as a source of Federated Access control Management data (FAccM).

The *Policy Audit Log* (PAL) repository plays an important role in the policy management, keeping log trails about policy enforcement events.

In fact, the ORCA framework components (PIP, PDP, PDP cache, PIB and PAL) are SOA-compliant services, since they may provide their functionalities through well-defined interfaces. Moreover, we allow a hierarchical relationship of several PDP services, governing fine-grained system sub-domains and composing a larger SOA environment. Interactions between distinct administration domains (top-level) are supported with the use of PIP entities.

4.4 Future Works

Some problems concerning the policy definition language, as for instance, the formal verification of policies, information flow control over the whole SOA system or the definition of reliability requirements, still require further investigation.

The problem of formal verification of policies has already been well recognized and addressed by some languages, usually by using directly (or transforming policy rules into) a logic-based programming language as Prolog or Datalog (e.g. in SecPAL [43], Cassandra [50], Binder [51]).

Information flow control (e.g. covered by DRM languages as XrML) is extremely difficult in loosely coupled service-oriented systems, where the external control over the data is typically limited to communication only, without any possibility to control the internal operations of services.

Finally, reliability requirements should also be expressed by the policy, extending its scope from security only to more compound dependability policy. Solutions for those problems could be interesting extensions to ORCA.

5 Conclusions

The paper has discussed the aspects of performance, reliability and security in management of SOA systems. We have taken special insight in metrics, monitoring, failure detection, security policy languages and management problems.

Software prototypes for proactive management and monitoring system, as well as the set of running environment and service level metrics for SOA systems have been proposed. They have been calculated by an efficient metrics module that takes advantage of intelligent event processing concept. The data for metrics is acquired from monitoring module adapters that provide interoperability with SNMP, Nagios systems. A coherent configuration is supported by the XML language for the metrics and data sources definition. Additionally, we have developed modules and methods of failure prediction and diagnosis, whose output can trigger virtualization cluster to take an action for migrating of virtual machines and tuning their parameters in order to meet SLA and QoS requirements.

A distributed failure detection service has been proposed for web services, based on the accrual failure detector concept, that determines the suspicion level of failure for a monitored service. This output can be further analyzed by our management system, in order to take actions e.g. which aim at failure avoidance.

Finally we have defined requirements for a security policy language for SOA and proposed a concept of such a language — ORCA. ORCA uses a very simple syntax, easy to understand and to manage. This facilitates full interoperability across a large heterogeneous SOA environment, and allows to define policy rules, easily transformable to other representations. ORCA supports access control restrictions, delegations, obligations, capabilities and trust definitions, among others. Its multi--layered architecture allows to express the policy with different levels of details, as desired.

Nevertheless, SOA systems are very complex from many points of view and therefore many issues and problems remain open. A solution for these open problems and the implementation of the appropriate tools are the goals of our further research activities.

Acknowledgments. This work has been partially supported by the Polish Ministry of Science and Higher Education within the European Regional Development Fund, Grant No. POIG.01.03.01-00-008/08.

References

1. Pautasso C., Zimmermann O., Leymann F. RESTful Web Services vs. Big Web Services: making the right architectural decision. 17th Int. World Wide Web Conf. (WWW2008), Beijing, China, 805–814, 2008.
2. Fielding R.T. Architectural styles and the design of network-based software architectures. PhD thesis, University of California, Irvine, 2000.
3. Richardson L., Ruby S. RESTful Web Services. O'Reilly, Sebastopol (USA), 2007.
4. Keen M., Acharya A., Bishop S., Hopkins A., Milinski S., Nott C., Adams J., Verschueren P. Patterns: implementing an SOA using an Enterprise Service Bus. IBM International Technical Support Organisation, Research Triangle Park, NC, USA, 2004.
5. Papazoglou M., van den Heuvel W.-J. Web Services Management: a survey. IEEE Internet Comp. 9, 58–64, Los Alamistos, CA, USA, Nov. 2005.
6. Web Services Distributed Management: Management of web services (WSDM-MOWS) 1.1. OASIS, 2006.
7. Web Service Level Agreement (WSLA) language specification. IBM, 2003.
8. Andrieux A., Czajkowski K., Dan A. Web Services Agreement specification (WS-Agreement). 2007.
9. Hayes B. Cloud computing. Commun. ACM, ACM, New York, NY, USA, 2008.
10. Vouk A.M. Cloud computing — issues, research and implemenetations. J. Comp. Inf. Technol. CIT, Dubrovnik, 2008.
11. Vaquero L.M., Rodero-Merin L., Caceres J., Lindner M. A break in the clouds: towards a cloud definition. SIGCOMM Comput. Commun. Rev., New York, 2009.
12. Rochwerger B., Breitgand D., Levy E., Galis A., Nagin K., Llorente I., Montero R., Wolfsthal Y., Elmroth E., Caceres J., Ben-Yehuda M., Emmerich W., Galan F. The RESERVOIR model and architecture for open federated cloud computing. IBM J. Res. Dev., IBM Corp., Riverton, USA, 2009.

13. American Megatrends About WS-Management. American Megatrends, May 2002.
14. Brown M. Understand Web Services Distributed Management (WSDM). IBM, `www.ibm.com`, July 2005.
15. Web Services Distributed Management: Management Using Web Services (MUWS 1.1) Part 1. OASIS, 2006.
16. Web Services Distributed Management: Management Using Web Services (MUWS 1.1) Part 2. OASIS, 2006.
17. Jim L., Heiko L., Liliana P., Bruno W. REST-based management of loosely coupled services. Proc. WWW 2009, April 2009.
18. An architectural blueprint for autonomic computing, third edition. IBM, Hawthorne, June 2005.
19. Kephart J.O., Chess D.M. The vision of autonomic computing. Computer 36, 1, 41–50, 2003.
20. Kang W., Grimshaw A. Failure prediction in computational grids. Ann. Simulation Symp., Proc. 40th Annual Simulation Symposium, 7, Norfolk, VA, USA, 2007.
21. Kirtane S., Martin J. Application performance prediction in autonomic systems. ACM Southeast Regional Conf., Proc. 44th annual Southeast Regional Conference, New York, 2006.
22. Salfner F., Malek M. Advanced failure prediction in complex software systems. Work, 1–19, 2004.
23. Sahoo R.K., Oliner A.J., Rish I., Gupta M., Moreira J.E., Ma S., Vilalta R., Sivasubramaniam A. Critical event prediction for proactive management in large-scale computer clusters. Int. Conf. on Knowledge Discovery and Data Mining, Washington DC, 2003.
24. Xue Z., Dong X., Ma S., Dong W. A survey on failure prediction of large-scale server clusters. SNPD (2), 733–738, 2007.
25. Agosta J., Gardos T. Bayes Network "smart" diagnostics. Intel. Technol. J. 08, 04, 2004.
26. Fu S., Xu C.-Z. Exploring event correlation for failure prediction in coalitions of clusters. Proc. 2007 ACM/IEEEi Conf. Supercomputing SC '07, 1, Reno, USA, 2007.
27. Rish I., Bodie M., Sheng M., Natalia O., Alina B., Grabarnik G. Adaptive diagnosis in distributed systems. IBM Syst. Technol. 945, 22, 2005.
28. Czajka A., Dwornikowski D., Filipiak D., Pawlak T., Sajkowski M. Opracowanie języka i narzedzi definiowania celu monitorowania na poziomie procesu aplikacyjnego. Inst. Informatyki P. Pozn., Poznań, 2009.
29. Bernhardt T., Vasseur A. Esper: Event stream processing and correlation. `www.OnJava.com`, 2007.
30. Défago X., Urbán P., Hayashibara N., Katayama T. Definition and specification of accrual failure detectors. Proc. Int. Conf. Dependable Systems and Networks (DSN '05), Yokohama, 206–215, 2005.

31. Hayashibara N., Défago X., Yared R., Katayama T. The φ accrual failure detector. Proc. 23rd IEEE Int. Symp. on Reliable Distributed Systems SRDS '04, 66–78, IEEE Computer Society, Washington DC, 2004.
32. Liang D., Fang C.-L., Chen C., Lin F. Fault tolerant web service. Proc. 10th Asia-Pacific Software Eng. Conf. APSEC'03, Los Alamitos, CA, 310, IEEE Computer Society, 2003.
33. Jayasinghe D. FAWS for SOAP-based web services. 2005.
34. Santos G.T., Lung L.C., Montez C. FTWeb: a fault tolerant infrastructure for web services. Proc. 9th IEEE Int. EDOC Enterprise Computing Conf., Washington DC, 95–105, 2005.
35. Merideth M.G., Iyengar A., Mikalsen T.A., Tai S., Rouvellou I., Narasimhan P. Thema: Byzantine-fault-tolerant middleware for web-service applications. SRDS, 131–142, IEEE Comp. Soc., 2005.
36. Rodrigues R., CastroR., Liskov B. BASE: Using abstraction to improve fault tolerance. 18th Symp. on Operating Systems Principles SOSP, Banff, Canada, 2001.
37. Zhao W. A lightweight fault tolerance framework for web services. Proc. IEEE/WIC/ACM Int. Conf. on Web Intelligence, Washington DC, 542–548, 2007.
38. Souza J.L.R., Siqueira F. Providing dependability for web services. SAC '08: Proc. 2008 ACM Symp. on Applied computing, Fortaleza, Ceara, Brazil, 2207–2211, 2008.
39. Moses T. eXtensible Access Control Markup Language (XACML) version 2.0. 2005.
40. Cantor S., Kemp J., Philpott R., Maler E. Assertions and protocols for the oasis security assertion markup language (saml) version 2.0. 2005.
41. Monzillo R., Kaler C., Nadalin A., Hallem-Baker P. Web Services-Security: SAML Token Profile 1.1. 2006.
42. Damianou N., Dulay N., Lupu E., Sloman M. Ponder: A language for specifying security and management policies for distributed systems. tech. rep. Imperial College, London, 2000.
43. Becker M.Y., Fournet C., Gordon A.D. Secpal: Design and semantics of a de-centralized authorization language. tech. rep. Microsoft Research, Cambridge, 2006.
44. Li N., Mitchell J.C. Datalog with constraints: A foundation for trust management languages. 5th Int. Symp. on Practical Aspects of Declarative Languages, London, 2003.
45. Jurjens J. UMLsec: Extending UML for Secure Systems Development. 5th Int. Conf. The Unified Modeling Language, London, 2002.
46. Lodderstedt T., Basin A.D., Doser J. SecureUML: A UML-based modeling language for model-driven security. 5th Int. Conf. The Unified Modeling Language, London, 2002.
47. Schloegel K., Markham T., Heimerdinger W., Schaeffer-Filho A., Sloman M., Lupu E., Calo S.B., Lobo J. Security policy automation — from specification to device configuration. 26th Army Science Conf., London, 2008.

48. Brodecki B., Sasak P., Szychowiak M. Security policy definition framework for soa-based systems. 10th Int. Conf. on Web Information Systems Engineering (WISE 2009), Poznań, Poland, 589–596, Springer-Verlag, October 2009.
49. Nadalin A., Kaler C., Monzillo R., Hallam-Baker P. Web Services-Security: SOAP Message Security 1.1 (WS-Security 2004). 2006.
50. Becker M.Y., Sewell P. Cassandra: flexible trust management, applied to electronic health records. Proc. 17th IEEE Computer Security Foundations Workshop, 139–154, Washington DC, June 28–30, 2004.
51. DeTreville J. Binder, a logic based security language. IEEE Symp. on Security and Privacy, Washington DC, 105–113, 2002.

Transaction and Rollback-Recovery Support for Service-Oriented Architecture

Jerzy Brzeziński[1], Arkadiusz Danilecki[1], Krzysztof Jankiewicz[1], Anna Kobusińska[1], Mariusz Mamoński[2], Tadeusz Morzy[1], and Paweł T. Wojciechowski[1]

[1]Institute of Computing Science
Poznań University of Technology,
Piotrowo 2, 60-965 Poznań, Poland
{Jerzy.Brzeziński,Arkadiusz.Danilecki,Krzysztof.Jankiewicz,Anna.Kobusinska, Tadeusz.Morzy,Pawel.T.Wojciechowski}@put.poznan.pl
[2]Poznań Supercomputing and Networking Center,
Noskowskiego 10, 61-704 Poznań, Poland
mamonski@man.poznan.pl

Abstract. In this chapter, we report on ongoing work on transaction support for Service-Oriented Architecture (SOA) within three disjoint subprojects. We begin with discussing common, general motivations for our work and our goals. Then, we streamline the discussion into three main parts: extended business transactions, rollback-recovery service, and distributed software transactional memory for SOA.

Keywords: Service-Oriented Architecture (SOA), extended business transactions, rollback-recovery, Software Transactional Memory (STM)

1 Introduction

The growing importance of *Service-Oriented Architecture (SOA)* has prompted demands for novel approaches and programming tools to solve problems that are seemingly not new. However, changing system/application requirements (e.g. different models of business processes) and increasing technology advances (e.g. ubiquity of multi-core CPUs) cast new light on some of these problems. In this chapter, we focus on *transaction* support which is vital to the development of the distributed service-oriented systems. We will summarize our ongoing work in this area and the results we have achieved so far; technical details can be found in technical reports and other publications.

According to the SOA reference model [1], in the service-oriented systems, each individual component is an independent *service* providing a predefined, standardized interface (or a contract). A function, which is required of the whole system, is obtained by composing functions of independent services. Using this approach in the development of distributed systems has many advantages. For example, we can point out three most important benefits of dividing a complex system into smaller independent components:

— the possibility of implementing each of the compound services independently;
— the possibility of increasing system throughput by distribution of services on many machines;
— the possibility of reusing services in other systems.

Similarly to almost any distributed environment, in service-oriented systems, it is often required to synchronize many concurrent events (or requests) originating from different components. In the case of *stateless services*, i.e. services that do not store any state, it is sufficient if synchronization of various requests occurs at the level of separate services. This type of synchronization can be relatively easily obtained with the use of programming constructs that are available in the majority of programming languages. However, in the case of *stateful services*, when some services store their state between consecutive requests, the problem of synchronization of different requests is more subtle and complex, since it relates to the whole system, not a single service.

In order to better illustrate the problem, let us consider an example of a system, which consists of:

— two stateful services R and S, delivering in its interface two methods, $R.m$ and $S.n$ (for simplicity, we assume that the methods are executed atomically);
— two concurrent client processes, k and l, each of which calls in turn methods $R.m$ and $S.n$ of services R and S, possibly executed by other processes on remote machines.

Below, there are three examples of possible sequences of service executions[1]:

$$e_1 = R.m^k, S.n^k, R.m^l, S.n^l \tag{1}$$

$$e_2 = R.m^k, R.m^l, S.n^k, S.n^l \tag{2}$$

$$e_3 = R.m^k, R.m^l, S.n^l, S.n^k \tag{3}$$

An answer to the question, which of these executions is correct, is not explicit, since it depends on the assumed *synchronization policy*. In most cases, the

[1]We write, e.g. $R.m^k$ to denote that client process k calls method $R.m$.

correctness of such kind of computation is defined by fulfilling a condition of *serializability*, which can be defined as follows: Concurrent execution of two or more processes is *serializable* if the effect of this execution, i.e. the state of a system after all processes completed execution, is the same as if all processes were executed in a sequence.

Using the above criteria in our example, correct executions are e_1 (consecutive service execution) and e_2 (concurrent service execution). Execution e_3 is not correct since:

— process l can see the state of service R changed by process k;
— process k can see the state of service S changed by process l;

which is not possible in any sequential execution, in which such circular dependencies cannot occur. In the sequential execution, either process k can see the state of the system changed by process l or vice-versa.

To ensure a correct synchronization policy, a sequence of service calls made by each client could be declared as a *transaction*. Then, a transaction system of the SOA framework (e.g. as part of the *Enterprise Service Bus (ESB)*) will be responsible for serializable execution of concurrent transactions.

The most known notion of transactions are database transactions. A *database transaction* comprises a unit of work performed within a database system against a database. Execution of database transactions is described using four properties: atomicity, consistency, isolation and durability, which are referred to using the acronym *ACID*. In this chapter, isolation and atomicity are the most relevant properties. Let us first define the former property, the latter will be defined later in this section.

The *isolation* property defines how and when the changes made by one transaction become visible to other concurrent transactions. Serializability, defined above, is an example of a synchronization policy, which is used in database systems to ensure isolation of atomic transactions. In practice, many database systems do not guarantee serializable transactions but relaxed isolation properties, which are called *transaction isolation levels* [2].

There are two main classes of concurrency control algorithms that are used to ensure isolation of database transactions: lock-based and non-lock concurrency control. When implementing *lock-based concurrency control*, the highest level of isolation (or serializability) requires that the database system acquires locks on shared data, which may result in a loss of concurrency. But the majority of database transactions can be constructed in such a way as not to require a high isolation level, thus reducing the locking overhead for the system. When using *non-lock concurrency control*, no lock is acquired; however, if the database system detects a concurrent transaction in progress, which would violate the serializability property, it must force that transaction to *roll back* and restart its execution. In database

systems, rollback is also available at the programming interface, as a mechanism to withdraw uncommitted operations on a database.

In this chapter, we summarize our ongoing work on transaction processing tools for service-oriented distributed applications. Many of the aforementioned problems and solutions known from the database transactions are similar in this new context. However, there are also important differences. For example, a model of business processes requires long-running transactions which cannot be rolled back easily. These transactions use compensation to handle failures and compensate changes that cannot be revoked. This means that such transactions are not atomic, where *atomicity* means that either all of the operations of a transaction are performed or none. In the case of business long-transactions, it is difficult to follow this "all or nothing" rule, while it is much easier for database modifications. In Section 2, we present our work on support for transaction compensations, based on *SOAP-based Web services* (or *Web Services* in short) [3]. Web Services are considered to be an important enabling technology for production SOA applications, offering several standards for both atomic and long-running transactions.

Regardless of the transactions, the possibility of periodical *checkpointing* and *rolling back* the state of a distributed application can be regarded as a useful mechanism to handle failures. For example, this mechanism is necessary when considering the *replicated state machine* approach to service resiliency, assuming the *crash-recovery* model of failure. In this case, both the group communication system and the application on top, must be able to recover its state after failure. More on the state machine replication and group communication can be found in the chapter that describes service replication. In Section 3.3 of this chapter, we describe our work on turning the rollback-recovery mechanism into a standalone service for creating SOA applications. The specific features of SOA applications required specific solutions to rollback-recovery.

Turning back to synchronization, many concurrency control algorithms for transactions are based on the rollback mechanism. Unfortunately, contrary to database systems, in networked SOA applications it is not always possible to use this approach, since not all transaction operations are revocable. For example, consider Web services in which some operations require a network message to be sent out. It is not possible to rollback such an input/output (I/O) operation; it can only be suspended till the whole transaction commits. However, this solution cannot be used when the I/O communication is bidirectional. An example of a solution to this problem is the use of concurrency control algorithms that are not based on inadvertent rollback. In our previous work, we have proposed novel concurrency control algorithms that belong to this class, called *versioning algorithms* [4, 5]. In Section 4, we report on Atomic RMI and Atomic Locks — two programming tools that we have been developing for transaction processing in networked, service-oriented applications. They use the versioning algorithms to guarantee serializability.

Below, we present the above three themes in three separate sections, each of which includes basic definitions, related work and goals specific to each theme, a description of the intermediate results and future work. Finally, we conclude. In this chapter, we have omitted most of the technical details. They can be found in the referenced technical reports and papers or other publications to appear.

2 Extended Transactions

2.1 Basic Definitions and Problem Formulation

The introduction of the transaction mechanism for the specification and execution of complex business processes in the SOA environment would allow users, similarly as in the case of database systems and distributed information systems, to define computations with the use of semantic commands that control the transactions. In systems based on SOA, we deal with asynchronous, complex and lengthy business processes, where participants (services) are loosely related. Therefore, in these systems the classic model of flat transactions and nested transactions, for which the entire transaction is a unit of atomicity, consistency, isolation, and durability — is inadequate and inappropriate. An important problem is the irreversibility of the execution of certain operations, which is similar to the irreversibility of the real actions in the databases. For this reason, compensation mechanisms are needed for the irreversible effects of the transaction or its fragments.

2.2 Related Concepts and Contributions

The problem of transaction processing in systems based on SOA has already been noticed a few years ago. As a result, a number of protocols and specifications have been developed, supporting the implementation of transaction processing. Among them are those developed by the OASIS WS-TX Technical Committee: *WS-Coordination version 1.2* [6], *WS-AtomicTransaction version 1.2* [7] and *WS-BusinessActivity version 1.2* [8]. In addition, the issues related to the processing of long-term business processes, which take into account the need for compensation, have been addressed in the *Web Services Business Process Execution Language (WS-BPEL)* [9]. Below we briefly characterize these specifications.

Web Services Coordination (*WS-Coordination*) [6]. In accordance with the *WS-Coordination* (*WS-Coor*) specification, all processes executed in the SOA environment can use many components that make up the distributed activities. These

activities can have complex structures with complex relationships between the participants – Web services involved in the implementation of the activities. The duration of the activities may be both short and, taking into account the circumstances of business and user interaction, very long – taking even many days or months.

WS-Coor defines an extensible framework for the coordination of the distributed activities using a coordinator and a set of coordination protocols. The framework allows participants to agree on the final results and the outcome of the distributed activities. The coordination is performed by protocols operating within the established rules of the framework. Protocol definitions go beyond the framework of *WS-Coor*, making it possible to use a wide range of protocols tailored to different types of activities, with various needs, including requirements for transaction processing.

Web Services Atomic Transaction (*WS-AtomicTransaction*) [7]. The *WS-AtomicTransaction* (*WS-AT*) specification enables the coordination of short-lived transactions, which are distributed activities intended to have the property of atomicity. It is based on *WS-Coor* and defines the two coordination protocols *Completion* and *Two-Phase Commit* (*2PC*). The *Completion* protocol is applied between an application and the coordinator. It is used by an application to inform a coordinator about the intention to either commit or rollback a transaction. The *Two-Phase Commit* protocol is applied between the coordinator and participants of the activities. Its primary task is to coordinate all the participants to achieve the same result (commit or rollback) of the distributed transaction [10].

WS-AT is a specification that is intended only for short-lived activities and is not suitable to coordinate a long-running business activity. This specification allows the coordination of services, which must comply with many features. For example, actions taken by these services prior to commit must be tentative, typically they are neither persistent nor made visible outside the transaction. Only the commit of a transaction allows participants to finalize the tentative actions. Then, these actions are made persistent and visible outside the transaction. This specific way of functioning is a serious limitation that it is not possible to assume in many cases, significantly limiting the possibility of using this specification.

Web Services BusinessActivity (*WS-BusinessActivity*) [8]. Unlike *WS-AT*, *WS-BusinessActivity* specification (*WS-BA*) describes a coordinator service and protocols designed for long-running distributed activities.

It is assumed that long-running activities, also known as business activities, have the following characteristics:

— Activities can use many different resources over a long period of time.
— To obtain a result of a business activity, a large number of atomic transactions can be involved.

— The results of individual atomic transactions, comprising a business activity, may be visible before the activity is completed.
— Response time for individual tasks can be very long. This is mainly due to the features of services needed to perform the tasks of an activity. A business activity assumes that a response to a particular request may demand long-running tasks, such as: user interaction, product manufacture, product delivery, etc.
— Due to the features of tasks undertaken within a business activity, a simple cancellation of a task, resulting from the abort of a business activity, is often not sufficient to restore the state of the system prior to the start of the task. Therefore, mechanisms to abort a business activity require more sophisticated solutions. For example, they may require a compensation of a task to obtain the task state before its execution. Compensation of tasks must be implemented by the developers of services that are participants in the business activity.

Coordination protocols, defined in the *WS-BA* specification, allow a very flexible definition of activities. For example, the *WS-BA* specification defines two types of coordination:

— *AtomicOutcome* — all participants must complete their processes or compensate their actions. The result of an activity is the same for all participants.
— *MixedOutcome* — each participant must obtain a specific final result, although each participant's result may be different.

In addition, the *WS-BA* specification defines two types of protocols which differ in the possibility of terminating tasks related to participant activities. Both protocols are used for the communication between a coordinator and a participant. However, the *WS-BA* specification does not define the protocol between the initiator of an activity (an application) and a coordinator.

The *WS-BA* specification defines only the protocols used for communication between the coordinator and participants of an activity other than their initiator. This is due to the fact that rules of aborting or committing business transactions can be very complex and, in accordance with the intention of the *WS-BA* specification, they belong to the sphere of business. This makes the implementation of the *WS-BA* specification generally very complicated. The creator of a business process must develop his own protocol of cooperation between the initiator and the coordinator of an activity, as well as appropriate protocol services for both sides. An alternative solution is the total integration of the coordinator and the initiator of the activity. This solution, however, leads to mixing elements from business (business process definition) and system mechanisms (coordination of the transaction), which significantly affects the simplicity and clarity of the process definition.

Another difference between the specification of *WS-AT* and *WS-BA* is that the latter does not impose restrictions on participants functioning — actions performed

by participants may be immediately persistent and visible outside the transaction, regardless of how business process ends. As a result, a failure of a business process requires appropriate compensation. According to the *WS-BA*, the participants of an activity are responsible for the compensating operations. The initiation of a compensating operation is triggered by a message sent to participants. The compensation rules of *WS-BA* only allow a full compensation, to be undertaken by all participants or selected ones. There is no possibility to compensate specific actions. In addition, it should be noted that in no way *WS-BA* relates to the problems associated with a concurrent execution of many business processes from the same set of participants.

***WS-BPEL* and Long-Running Transaction [9].** *Web Services Business Process Execution Language (WS-BPEL)* is a declarative markup language, used to describe, coordinate and implement complex business processes using Web services.

Business processes defined using the *WS-BPEL* language may have a long duration. They may also be based on Web services, whose actions are immediately persistent and visible outside the business process. An important issue is to respond to errors that can take place during the operation of business processes. In such cases, actions which will rollback the effects of the operation are necessary. For this purpose, *WS-BPEL* uses a compensation technique, which was already mentioned when discussing *WS-BA* specification. The *WS-BPEL* language allows to define a long-term activity, called the *Long-Running Transaction (LRT)*.

Unfortunately, the solutions of *WS-BPEL* assume that elements, such as exception handling or invoking compensation actions, must be prepared by the creator of the business process. Thus, as it was in the case of the *WS-BA* specifications, we face a situation in which business components are mixed with system actions. This impacts the simplicity and readability of the process definition. Moreover, in contrast to the *WS-BA* specification, for which compensation is triggered with one simple message, the creator of a business process expressed in *WS-BPEL* must invoke the appropriate compensation services. As a result, the definition of properly constructed business processes is a complex task. Moreover, it should be noted that in no way *WS-BPEL* relates to the problems of concurrent executions of multiple business processes.

2.3 Transaction Coordinator

Functional Requirements for Transaction Coordinator. The analysis of the listed above specifications leads to the conclusion that it is appropriate to propose solutions that will significantly support the creation of business processes, separating the business layer from the system actions. This solution should allow a

proper coordination of business processes at the stage of acceptance (or rejection). It should provide an opportunity to make a partial compensation, and should take into account the need for coordination of concurrent execution of processes.

To achieve these goals, a Transaction Coordinator system is being designed and implemented. It will allow:

— the definition of a transaction operating within SOA environment, based on SOAP-based and RESTful Web services;
— the application of the control structures within the transaction code associated with the concept of a transaction (start of the transaction, commit of the transaction, rollback of the transaction, creation of the savepoints, rollback to the savepoint);
— the management of a transaction taking into account the types of transactions and their isolation levels;
— the application of management mechanisms of concurrent access to the same set of services (resources);
— the management of a transaction's completion;
— the compliance with the business process properties of a transaction (atomicity, isolation, consistency).

In addition, the Transaction Coordinator will support non-functional requirements, such as:

— transparency — users need not be aware of the mechanisms controlling a business process and therefore, changes in the definition of transactions introduced in order to use the functionality of the Transaction Coordinator, must be limited as far as possible;
— deployment simplicity — deploying the Transaction Coordinator into a runtime environment of business processes will be based on the provided mechanisms, it will be clearly defined and documented;
— response to system failures (the Transaction Coordinator will have the ability to respond to system failures and failures arising from the concurrency management rules) and to failures arising from concurrent access management rules respectively.

Architecture of Transaction Coordinator. The Transaction Coordinator has a modular design presented in Figure 1. It consists of the following modules:

— Transaction Definition Module — a module, which helps the end-user to define a business process using the control structures of transaction processing.
— Transaction Definition File — the definition of the BPEL process, supplemented by transaction processing commands, and other statements related to the business process definition.

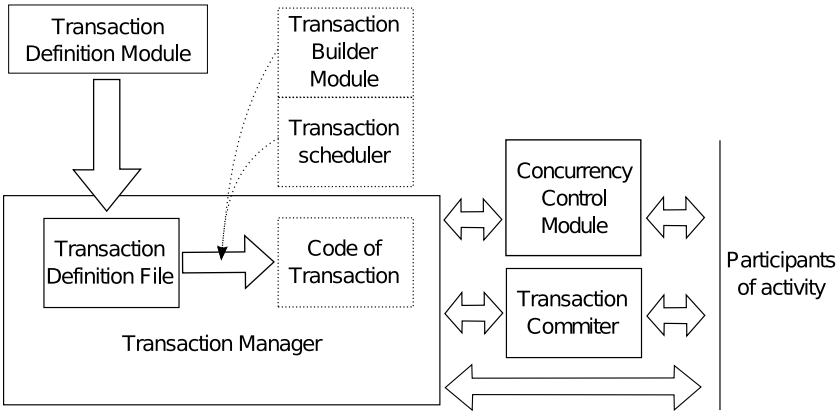

Fig. 1. The Transaction Coordinator architecture

— Transaction Manager — the task of this module is the execution of the Transaction Definition File. In particular, the role of the Transaction Manager can be played by any BPEL server which allows the interpretation of transaction processing commands.
— Transaction Builder Module — the module performs the proper interpretation of the transaction processing commands that are necessary extensions of the *WS-BPEL* language. The Transaction Builder Module operates under the Transaction Manager control.
— Concurrency Control Module — the aim of this module is to manage the concurrent execution of business processes. To do this, it uses the compatibility feature defined at the level of services, processes or even individual resources.
— Transaction Commiter — the task of this module is to coordinate actions associated with the start of transaction, transaction committing, compensation of transaction, savepoint creation, and compensation to savepoints.

The most important modules of the system are thoroughly described below (see Fig. 1).

Transaction Commiter. The Transaction Commiter has been implemented based on *WS-Coor*. From the perspective of *WS-Coor*, the Transaction Commiter serves as a coordinator and, therefore, provides activation and registration services. The activation service creates the so called context of coordination which serves as a transaction ID for all participants used within the particular instance of a business process. The task of the registration services is to register participant of the activity on the basis of his context of coordination. Participants of the activity may play different roles in the process. For the Transaction Commiter the

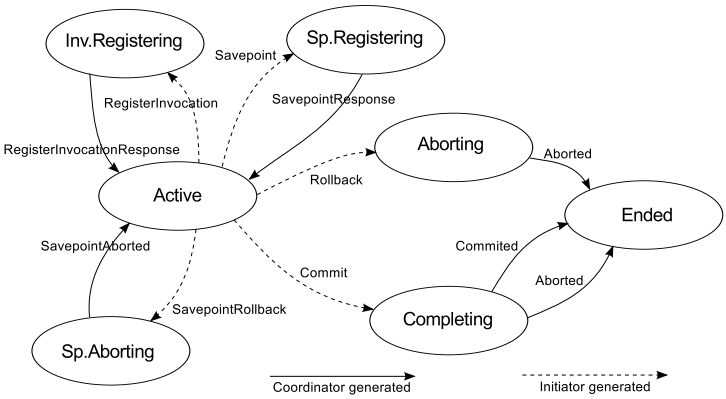

Fig. 2. The *EnhancedCompletion* protocol

active participants are the Transaction Manager, which executes the Transaction Definition File, and services which are called based on the Transaction Definition File. In addition, the Transaction Commiter uses protocol services cooperating with the participants in the transaction based on the protocols described in the following.

Protocols of the Transaction Commiter. Within the Transaction Coordinator individual modules work together using specific protocols. These protocols depend on the type of coordination which is set when the context of the coordination is created by the activation service. In the case of the Transaction Commiter two protocols are used. The first one, *EnhancedCompletion,* is used to cooperate between the Transaction Commiter and the Transaction Manager, which is the initiator of the activity. The *Enhanced2PC*, the second protocol, is used to cooperate between the Transaction Commiter and the other participants of the activity — service components. Figures 2 and 3 show the state-chart diagrams for each of the protocols.

The structure and the meaning of messages used in both protocols have been presented in detail in the technical report [11].

Transaction Definition File. The Transaction Definition is based on the definition of the business process expressed in *WS-BPEL*. Since the *WS-BPEL* language does not contain commands for the processing of transactions, we have proposed some *WS-BPEL* language extensions. Each of these extensions is a language construction processed by the Transaction Builder Module, which takes the appropriate actions according to their semantic. These extensions are the following:

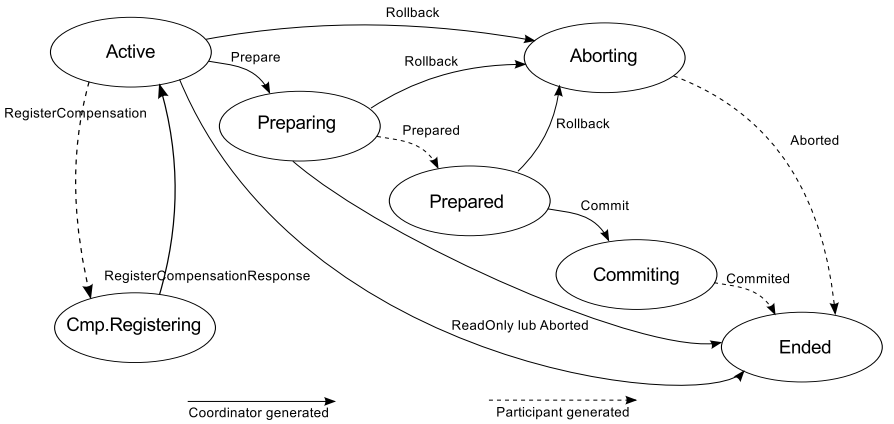

Fig. 3. The *Enhanced2PC* protocol

```
<bpel:extensionActivity>
  <ext:commit/>
</bpel:extensionActivity>
```

Fig. 4. The *commit* extension

— *beginTransaction* — it initiates a transaction;
— *preInvoke* — its task is the execution of additional coordination actions by the Transaction Builder Module, which are necessary when there is an intention to call a participant service. (It is the only extension which has a system and not business meaning);
— *savepoint* — the purpose of this extension is to create a savepoint, which is reference point within a transaction and allows to make a partial (to this point) compensation of a business transaction;
— *rollback* — it implements the compensation of the whole transaction or of only these actions which took place after the savepoint declaration;
— *commit* — its task is to perform the commit and final completion of the transaction.

The form of a *commit* extension is illustrated in the Figure 4.

Transaction Manager. The Transaction Manager is the BPEL server performing the processes defined in the *WS-BPEL* language. In addition to performing actions coming directly from the *WS-BPEL* language instructions, it also cooperates with other modules that are parts of the Transaction Coordinator like the Transaction Commiter, for example. The current implementation of the Transaction Manager

is based on the BPEL server — Apache ODE. The Transaction Manager comprises the Transaction Builder Module which processes the *WS-BPEL* language extensions used to control transaction processing. The implementation and the meaning of each of the extensions have been presented in details in the technical report [11].

2.4 Future Work

The construction of the Transaction Coordinator is not completed. The following elements will be complemented in the near future:

— Concurrency Control Module — the task of this module will be the coordination of the activities performed within the Transaction Coordinator environment by business processes. Its implementation, as well as the implementation of the Transaction Commiter, will be based on *WS-Coor*. The essential difference is that the Transaction Commiter coordinates actions performed within each individual business process, while the Concurrency Control Module coordinates activities at the Transaction Coordinator level. In addition, the participants, in the case of the Concurrency Control Module, will not be Web services but business processes.
— Protocols for the Concurrency Control Module — these protocols are necessary because the implementation of the Concurrency Control Module is based on *WS-Coor*. They enable the cooperation of the Concurrency Control Module with individual participants. The number of developed protocols will depend on the number of different types of participants.
— Protocol services for the Concurrency Control Module — their implementation also derives naturally from the fact that the Concurrency Control Module is based on *WS-Coor*. These protocol services will be responsible for the communication between the Concurrency Control Module and the participants.
— Transaction Definition Module — the core business of the Transaction Definition Module is to assist end-users in the creation of the correct Transaction Definition File. This assistance may include a number of actions, for example: business process visualizations, extension of the business process of the transaction control commands, and the validation of the Transaction Definition File.
— Extension of the transaction model — at this stage, the Transaction Coordinator can be used to process both flat and nested transactions, as in the specification of *WS-Coor*. We are planning to verify the nested transaction processing capabilities, and extend the Transaction Definition File along with all the necessary system elements (including the Transaction Commiter and Transaction Builder Module). The extensions will allow to define a transaction in accordance with one of the advanced models of nested transactions. It is assumed that in this model it will be possible to define more sophisticated

rules of transaction atomicity. These rules will provide the distinction between the critical (mandatory) sections of the transaction, and its optional sections that do not have an impact on the success of the transactions.

3 Service Rollback and Recovery

3.1 Basic Definitions and Problem Formulation

Service oriented systems, due to their loose-coupling and a great number of independent components that are susceptible to failures, are very error-prone. A failure may result in a permanent loss of resources of a failed component and may discontinue a business process. Such a situation is highly undesirable from the viewpoint of SOA clients, who expect that the provided services are reliable and available, and assume an uninterrupted business processing. To improve availability and reliability of SOA-based systems and applications, checkpointing and rollback-recovery techniques may be applied [12, 13]. In general, checkpointing is based on the idea of periodically saving the current system state into stable storage, able to survive failures [14, 15]. Depending on when checkpoints are taken, the existing approaches can be broadly classified into: *coordinated* [16], *uncoordinated* [17], and *communication-induced checkpointing* [18].

When a fault occurs, a system is rolled back to the most recent consistent set of taken checkpoints, called the *recovery line* [19], and then proceeds again. Thus, the amount of lost computation is limited only to the time period from the checkpoint to the failure. Checkpointing may be combined with message-logging protocols, which rely on the piecewise deterministic assumption [20]. Such protocols enable saving non-deterministic events (like receipt of messages) in message logs. Upon a failure, a process state is rolled back to the checkpoint and then logged messages are replayed as they were previously executed. Three primary message-logging mechanisms are considered: *pessimistic*, *optimistic* and *causal* [21]. They differ in where and how the messages are stored.

Although a wide range of rollback-recovery techniques for general distributed systems and distributed databases have been explored in the literature, there are only a few solutions which take into account specific properties of SOA systems. The problem of checkpointing and rollback-recovery in service oriented environments is complicated, among other things, due to the autonomy of nodes, the heterogeneous nature of the environment, and changing workflow of applications.

In general, in SOA systems, services are autonomous components of various characteristics, like concurrency policies, access rights, etc, which run on heterogeneous platforms. Moreover, computation of such services may be distributed

amongst many independent organizational domains that implement their own recovery policies. Therefore, sometimes a service cannot be forced to take a checkpoint, or to rollback, as well as it may refuse to inform other services on the checkpoints it has taken. Furthermore, services can use various transport protocols. Interacting with these protocols requires dealing with such limitations as access latency, timeouts and lost requests. Beside the communication limitations, services may be also unavailable for an unknown reason, and for an unknown amount of time.

That is why the existing approaches to service rollback and recovery usually guarantee fault tolerance only for individual services, instead of providing it for an aggregate set of services that compose a system or a business process. In the context of SOA systems, where services may be dynamically and flexibly integrated, such a situation should not take place. While it is relatively easy to make an individual rollback-recovery, improving fault tolerance of services collaborating in multiple application scenarios is still a challenging task.

This section describes a novel *Business Process Rollback-Recovery Service*, providing fault tolerance for the SOA systems based on the RESTful Web services [22]. The proposed service ensures that after a failure occurrence, the state of a business process is transparently recovered and is consistently perceived by the business process participants: clients and service providers. The proposed solution and transactions are complementary to each other.

3.2 Related Concepts and Contributions

A *business process* is a set of logically related services working together to achieve a common objective. Thus, the definition of a business process specifies the behavior of its participants and describes the sequencing and coordination of the calls to component services. A business process may be specified using, for example, the Business Process Execution Language [23] or BPEL for short. Providing fault tolerance to a business process is related to ensuring the reliability of business process participants: client applications and services performing clients requests.

Different approaches may be used to make business processes reliable. Among them are: replication, transaction-based forward recovery which requires the user to explicitly declare compensation actions, and rollback-recovery checkpoint-based approach.

The existing SOA environments, such as IBM WebSphere, Oracle SOA Web Logic, Microsoft Windows Communication Foundation (WCF), and the technologies based on the Java language (JGroups and JBoss Messaging), apply the checkpointing and rollback recovery concepts. However, a careful analysis of these environments [24] shows that the proposed logging and checkpointing mechanisms are applied only to achieve the reliability of single services, within the limits of

one organizational domain (i.e. one server providing the services). In this way, the dependencies among various services, created when one service invokes other services located on different servers, are ignored. In fact, the distributed dependencies should be taken into consideration while recovering a compound service.

Clients defining business processes may use BPEL engines (for example Oracle BPEL Process Manager [25]) to store process-associated state into a database. The BPEL engines ensure that all database accesses occur within the appropriate transaction and that system failures will never cause inconsistency in a process's internal state. The consistency of distributed business process may be achieved by using WS-Transactions [26], etc. The disadvantage of using this approach is that it may force unnecessary and costly withdrawals in the case of temporary failures. When failures occur, BPEL engines provide automatic recovery removing a significant burden from Web service developers. However, such an automatic recovery is limited to such actions as retrying faulty services, picking another functionally equivalent services (identified by the process designer), etc. [27, 28, 29, 30, 31]. In addition, the utility of an automatic recovery solution is limited by concentrating on stateless services. In many other situations the developer of a business process has to deliver self-written compensation procedures in order to ensure business process reliability, even in the case of failures not related to the business logic of an application.

3.3 Rollback-Recovery Service

Functional Requirements. The proposed rollback-recovery service is introduced as a separate component, responsible for keeping the state of both client applications and invoked services [32]. This service assumes the crash-recovery model of failures, i.e. system components may crash and recover after crashing a finite number of times [33]. Moreover, system components can fail at arbitrary moments and such a failure must be eventually detected, for example by failure detectors [34].

A SOA infrastructure, using such a rollback-recovery service, is shown in Figure 5.

The primary functional requirement of a rollback-recovery service developed for service oriented architectures, is to provide fault tolerance for business processes and their participants. Hence, the aim of the proposed rollback-recovery service is to guarantee the continuity of a business process despite the failures of one or more system components.

With this end in view, the business process rollback-recovery service realizes autonomically and without any client's interference the following functions:

— during the failure-free execution, enabling a client to store its state as well as the state of resources of services being part of a business process;

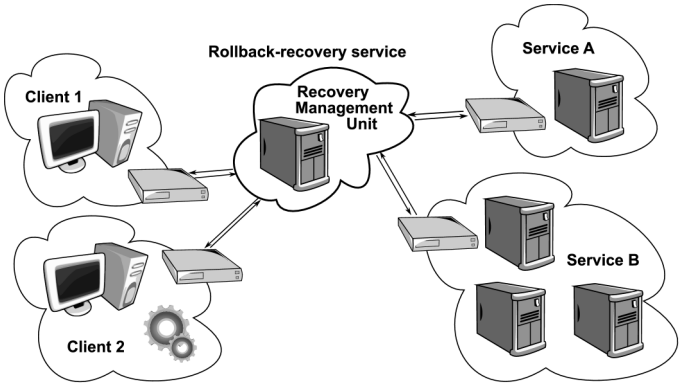

Fig. 5. An infrastructure of service-oriented systems with Rollback-Recovery Service

— in the case of a failure of any system component, ensuring the recovery of a business process state, perceived as consistent by clients and services participating in the business process;
— respecting the dependencies among distributed services, i.e., when the service is composed of other services, the rollback and recovery of the service state should take into account the dependencies among component services, and, if it is necessary also rollback and recover their states;
— providing the required consistency models, i.e., the recovery of state of service resources should consider consistency models guaranteed by the service and required by the clients;
— respecting clients mobility, i.e., it should be possible to recover the state of mobile clients, even if such clients switch over various system components, starting business process on one machine and finishing it on another.

The proposed rollback-recovery service does not require business process architects or service programmers to provide procedures dealing with reliability issues. Therefore, the business process architect can concentrate on business logic only, and can rely on the infrastructure to handle the failures. Also, when a client failure happens, a software defining a specific behavior of the client application is not necessary.

Hence, the proposed service differs from the existing solutions. The service invoked during the business process execution does not need to deliver compensation functions in advance. Also, the business process architect does not need to anticipate or take into consideration the possible failures, which may occur during the business process realization, and is not responsible for preparing appropriate compensation procedure to handle them. In contrast, the proposed business process

rollback-recovery service enables the automatic, transparent rollback-recovery of business process and its participants.

Architecture of Rollback-Recovery Service. The business process rollback-recovery service architecture is presented in Figure 6. The proposed service has a modular structure. The main modules of the rollback-recovery service are the *Recovery Management Unit*, *Client Proxy Unit* and *Service Proxy Unit*. They are introduced to ensure the maximal transparency of the rollback-recovery service and to hide the service architecture details from clients and services.

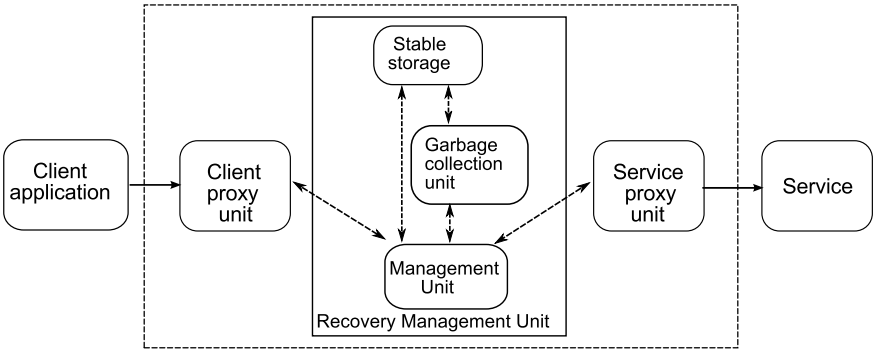

Fig. 6. The Business Process Rollback-Recovery Service architecture

Recovery Management Unit. The *Recovery Management Unit* ensures the durability of the state of all resources important from the business process participants' point of view. It is also used during the rollback and recovery procedure. Within this unit the following modules are distinguished:

— *Stable Storage* — it is able to survive any failure and therefore, it is used to store resources.
— *Management Unit* — responsible for providing access to the *Stable Storage* by executing operations of writing and reading resources; a write operation is realized every time a client issues a request invoking a certain service of a business process, and when the response is sent back from a service to the client; a read operation is used during the recovery procedure, when the resources saved in the stable storage are sent to the client or to the service.
— *Garbage Collection Unit* — it is used to remove from the *Stable Storage* the resources and requests which are not going to be used any longer, neither during the further processing nor the recovery procedure.

Generally, the *Recovery Management Unit* durably records all invocations and responses sent between clients and services. As a result, the complete history of communication among clients and services is kept in the *Stable Storage*, which is used while rolling back and recovery of business processes.

The communication history is also used to provide the requests idempotency. If the *Recovery Management Unit* gets a request which was already performed, it re-sends the response saved in the *Stable Storage* directly to the client, without invoking the service again. Thus, the multiple processing of the same request is avoided.

The amount of data kept by the *Resource Management Unit* may grow indefinitely. To solve this problem, the truncation of the communication history is necessary. It is done by the *Garbage Collection Unit*.

In order to ensure a proper load balancing and a high availability, the *Recovery Management Unit* is replicated.

Client Proxy Unit. In order to provide transparency of the rollback-recovery service, the *Client Proxy Unit* intercepts all service invocations issued by a client, and sends them to the *Recovery Management Unit*, where these invocations are saved in the *Stable Storage* and then passed to the invoked services. If it is necessary, the *Client Proxy Unit* modifies the client invocation accordingly to the *Recovery Management Unit* requirements. To properly identify clients, the *Client Proxy Unit* gives a unique identifier to each client. It also enables clients' migration among various system nodes, while performing a business process.

Service Proxy Unit. The *Service Proxy Unit* is located at the service provider site. Its primary task is monitoring the service state and responding to a service potential failure. In the case of a failure occurrence (the failures are detected by Service Oriented Failure Detection Service [35]), the *Service Proxy Unit* is responsible for initiating and managing the rollback-recovery process.

The *Service Proxy Unit* may work in ordinary, rollback and recovery modes. The *Ordinary mode* describes actions taken by the *Service Proxy Unit*, when failures do not occur. In this mode, the *Service Proxy* directs invocations obtained from clients to the provided service, intercepts, and sends responses generated by the service to the *Recovery Management Unit*. Each response is equipped with the unique service identifier generated by the service. The *Rollback mode* defines the steps which should be taken when the crashed service is rolled back after a failure occurrence. In the rollback mode the service state is rolled back to the latest checkpoint, and the *Recovery Management Unit* is asked to resend to the *Service Proxy Unit* all invocations obtained by the service after the checkpoint was taken. Finally, in the *Recovery mode*, the *Service Proxy Unit* receives from the *Recovery Management Unit* all requests demanded in the rollback mode and sends them to

the service. The requests are appropriately ordered to preserve the same sequence of invoked requests, as before the failure.

All invocations obtained by the *Service Proxy Unit* during the rollback or recovery mode are deferred, and resumed after the rollback-recovery process is finished.

In some situations, service providers may have no stable memory at all. Then, they can use the *Stable Storage* provided by the *Recovery Management Unit* for keeping vulnerable parts of the service state. When such a service fails, the *Recovery Management Unit* may recover the service state by replaying all recorded requests. This possibility, however, has an associated cost of increasing recovery time and increasing strain incurred by business process rollback-recovery service. If a service provider uses its own stable storage, then checkpoints can be made more often and less time is needed for the system recovery. Since one may reasonably expect failures to be rare, this potentially allows service providers to tune the checkpoint ratio in order to balance the cost of checkpointing and recovery time.

3.4 Future Work

Currently, the business process rollback-recovery service presented in this section, is verified and experimentally evaluated.

This service is pessimistic, as *Recovery Management Unit* does not need any preliminary knowledge about the service structure, always assuming that the service invocation may have an impact on other services, and irrevocable consequences. If the *Recovery Management Unit* could refer to any information, such as the possible connections between services, the results of invoked service methods, etc., a more optimistic approach to the construction of the proposed service would be possible.

It was also noticed that when some service is unavailable permanently or for an unacceptably long time, the *Recovery Management Unit* could invoke another, functionally equivalent service. Therefore, our future work includes the adaptation of the existing methods for determining which services are functionally equivalent to appropriately change the architecture of the rollback-recovery service.

The future work on business process rollback-recovery service will also deal with the improvement of the proposed rollback-recovery service efficiency. Moreover, a development of the mechanisms increasing the service transparency and the replication mechanisms of the *Recovery Management Unit* will be carried out.

4 Software Transactional Memory

4.1 Basic Definitions and Problem Formulation

Concurrency control is an important aspect of distributed processing. Concurrent, uncoordinated writes and reads of the shared memory (or shared objects/components) are not deterministic (since they depend on the scheduling of concurrent processes and/or unpredictable network delays), which means that it is not possible to guarantee repeatable results of such processing. In most cases, this behavior is not acceptable and some concurrency control is required. Therefore, the majority of modern programming languages have dedicated language constructs that make it easier for a programmer to ensure synchronization between concurrent processes. The basic mechanisms for synchronization are: monitors, locks, conditional variables and semaphores.

However, despite the availability of the above mechanisms, the programmer is still burdened with the most difficult tasks — an identification of code fragments that require synchronization and an implementation of the synchronization code (e.g. acquisition/release of locks). This stage of developing an application is error prone. Moreover, errors of this type are hard to localize, since they can manifest themselves in a completely different part of an application. For example, forgetting to release a lock may cause deadlock when another process will try to acquire the lock. Another problem is such that, in most cases, software components that use locks cannot be directly reused in other applications. For example, from the fact that each of the two or more software components can be safely used, it cannot be stipulated that a compound of the components will also be correct. The lack of this property is often regarded as not fulfilling the *composability condition* [36].

The ability to effectively reuse and compose services is a critical requirement for achieving some of the most fundamental goals of service-oriented computing [1]. Therefore, for seamless synchronization of different services, we require synchronization constructs to be composable. In Section 1, we have noted that, in database systems, users have at their disposal a declarative mechanism ensuring synchronization, i.e. atomic transactions. By using atomic transactions, they only specify (or declare) what is the consistent state regarding the database operations performed by these transactions. The database system itself is responsible for guaranteeing fulfillment of the consistency requirement. In particular, the database users are not obliged to acquire/release blockades on individual tables or tuples. This feature supports composability. However, database atomic transactions are used for accessing a database, not service components. Our goal was therefore to design and implement alternative methods of synchronization in concurrent and distributed SOA applications, which would be integrated with a programming language and its data structures.

4.2 Related Concepts and Contributions

In recent years, there has been a growing interest in adopting atomic transactions to general-purpose programming languages. These efforts are centered around the notion of *Software Transactional Memory (STM)* [37]. For the last few years, many STMs have been designed and implemented for different programming languages, though almost all these implementations are non-distributed (see e.g. [38, 36, 39, 40] among others). In [41], we have discussed and compared a few examples of STM implementations, each of which proposes different semantics of atomic transactions and a different API.

The STM allows the declaration of *atomic transactions* of code, whose correct processing is enforced by the system, thus ensuring synchronization automatically. These *light-weigh* atomic transactions are similar to transactions known from database systems but they operate on language data structures which are stored in volatile memory. Thus, *durability* of data modifications is not guaranteed in those systems. Another line of research is on type systems for specifying and verifying the atomicity of methods in multithreaded programs (see [42] and follow-up papers), where the notion of atomicity in these systems is similar to the serializability property defined in Section 1.

Atomic Locks [43] is analogous to STM but implemented as an extension of the Java mutex library, with rollback-recovery supported for on-demand requests. The key idea of this extension is based on the calculus described in [44]. The implementation of Atomic Locks uses a versioning algorithm, which means that atomic transactions can have I/O operations. The Atomic Locks library provides two main constructs: `atomic` and `retry` for, respectively, defining a transaction and executing transaction rollback. The `retry` construct enables us to restart a transaction if, for example, some condition is not true. Thus, Atomic Locks offers the functionality of the Software Transactional Memory but does not exclude using locks. However, the Atomic Locks mechanism is not distributed, which restricts its use. More details can be found in [43].

Below, we summarize our ongoing work on extending the concept of STM to service-oriented systems, in which atomic transactions can span several network nodes and can execute input/output operations. In our work, we state that transaction serializability is satisfied if the effect of the execution of two or more concurrent transactions is equivalent to the effect of a sequential execution of those transactions, where an *effect* comprises both memory modifications and the input/output operations.

4.3 Atomic RMI

Functional Requirements. We have been developing *Atomic RMI* [45, 46] — a concurrency control library built on top of Java RMI, which ensures serializable execution of concurrent processes by allowing the developer to define specific sequences of method calls on remote objects as serializable transactions, called *atomic tasks*. We assume that remote objects are implementations of some services. Currently, the implementation does not support transaction rollback recovery. The Atomic RMI implementation is based on Java RMI. However, our transaction support could also be used as an extension of other interfaces to network services, e.g. using SOAP-based or RESTful Web services; see also the Future Work section.

Java Remote Method Invocation (Java RMI) [47] is an extension to the Java language that allows programs to access methods of objects that are defined in other Java Virtual Machines (JVM). The local JVM uses stub objects to interface with objects which may be implemented and executed on a remote JVM. It must be noted that Java RMI does not aim to supply any additional means of synchronization, apart from those available in the Java language by default. Specifically, Java RMI does not ensure that the execution of concurrent processes that access distributed objects is serializable. Therefore, it is not guaranteed that the effect of the execution of concurrent processes (or the state of the system) will be equivalent to the effect of a sequential execution of those processes. In consequence, it is the developer who needs to ensure serializability by implementing a synchronization policy.

In our approach, we have extended Java RMI with the support of distributed transactions whose concurrent execution is serializable. The result of the use of transactions in Atomic RMI is lifting the chore of manual management of blockades (e.g. using distributed locks) from the developer, therefore, reducing his efforts to implementing synchronization policy. However, contrary to non-lock concurrency control approaches, our transaction mechanism does not depend on inadvertent rollback. Thus, I/O operations can be used within a transaction. When compared with the traditional lock concurrency control approaches, our transaction mechanism is more integrated with the programming language. For example, it makes use of the information that can be derived statically from programs, in order to effectively acquire and release blockades on shared objects at runtime; see [4] for details. To facilitate programming, we have developed a precompiler for the automation of the process of analyzing source code of Atomic RMI programs. The static analyzer analyses the code, computes the information required by the Atomic RMI runtime system, and generates the output code for the Java compiler. The tool has been described in [48].

The listing in Figure 7 presents an example of a source code, for the Atomic RMI library, with comments pointing out the main elements. The code defines a distributed transaction that calls two remote services, `a` and `b`, atomically. The

```
// Initialization of a system-wide task manager.
AtomicTaskManager taskManager =
    AtomicTaskManagerService.lookupTaskManager("example.com");

// Declaration of remote objects.
Registry registry = VersionedLocateRegistry.getRegistry(taskManager);

RemoteObject a = (BankAccountInf) registry.lookup("remoteObjectA");
RemoteObject b = (BankAccountInf) registry.lookup("remoteObjectB");

// The header of the atomic task.
RmiAtomicTask task = new RmiAtomicTask(taskManager);
TaskDescription taskDescription = new TaskDescription();

// The body of the atomic task.
task.startTask(taskDescription);

a.call();
if (random()) {
    b.call();
}

task.endTask{taskDescription};
```

Fig. 7. An example of an Atomic RMI code

references for these two objects (implementing two services) are obtained using the `lookup` method, which overwrites the original `lookup` of Java RMI. Atomic RMI extends the RMI registry for service register/lookup operations but the implementation could be modified to support other directory services, too. More details of the code can be found in [45].

Architecture of Atomic RMI. An architecture of a distributed application based on Atomic RMI is presented in Figure 8. It consists of the following system components:

— A-RMI Server: it contains remote objects that can be called by Atomic RMI clients;
— A-RMI Client: it spawns transactions and calls remote objects;
— A-RMI Registry: a name server (or trader) that is used to register and lookup remote objects which are associated with unique names;
— HTTP Server: it enables to fetch missing classes, whose instances have been passed (or returned) in a remote method call.

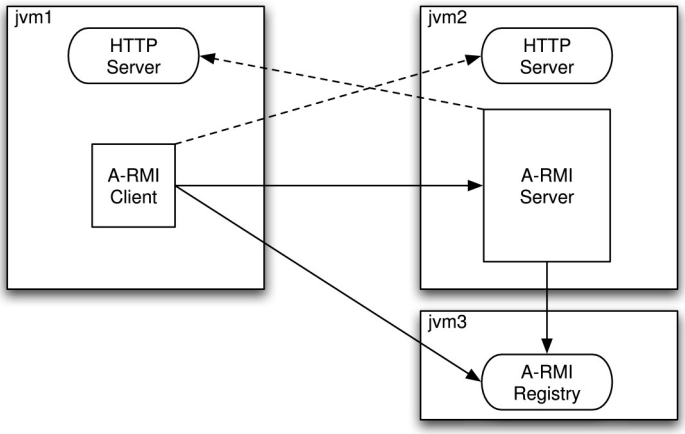

Fig. 8. An architecture of a distributed application based on Atomic RMI

Below, we describe the main classes of the Atomic RMI implementation. A class diagram illustrating transaction management in the Atomic RMI library is shown in Figure 9.

An interface *AtomicTaskManager* delivers methods of atomic tasks management. It is used by other classes of the Atomic RMI library. The goal of the *AtomicTaskManager* is to coordinate the beginning and the end of atomic tasks and to register/unregister remote objects. The only realization of the *AtomicTaskManager* interface is the *AtomicTaskManagerImpl* class which implements the functions defined in the interface. The *AtomicTaskManagerService* class is a service that must be spawned in order to use the Atomic RMI library. The service constructor creates an instance of the task manager class (*AtomicTaskManagerImpl*) and binds it to the RMI registry.

Remote objects that are used by transactions must be registered using an extension of the Java RMI registry. A class diagram that implements this registry is shown in Figure 10.

An interface *VersionedRegistry* extends the *Registry* interface from the *java.rmi.registry* package, with additional methods *bind* and *rebind*, which, in addition to a remote object, also allow to specify the upper-bound on the number of calls to remote objects[2]. An implementation of this interface and an extension of the *java.rmi.registry.Registry* class is the *VersionedRegistryImpl* class. In the

[2]This information is derived automatically by the Atomic RMI precompiler, whenever possible.

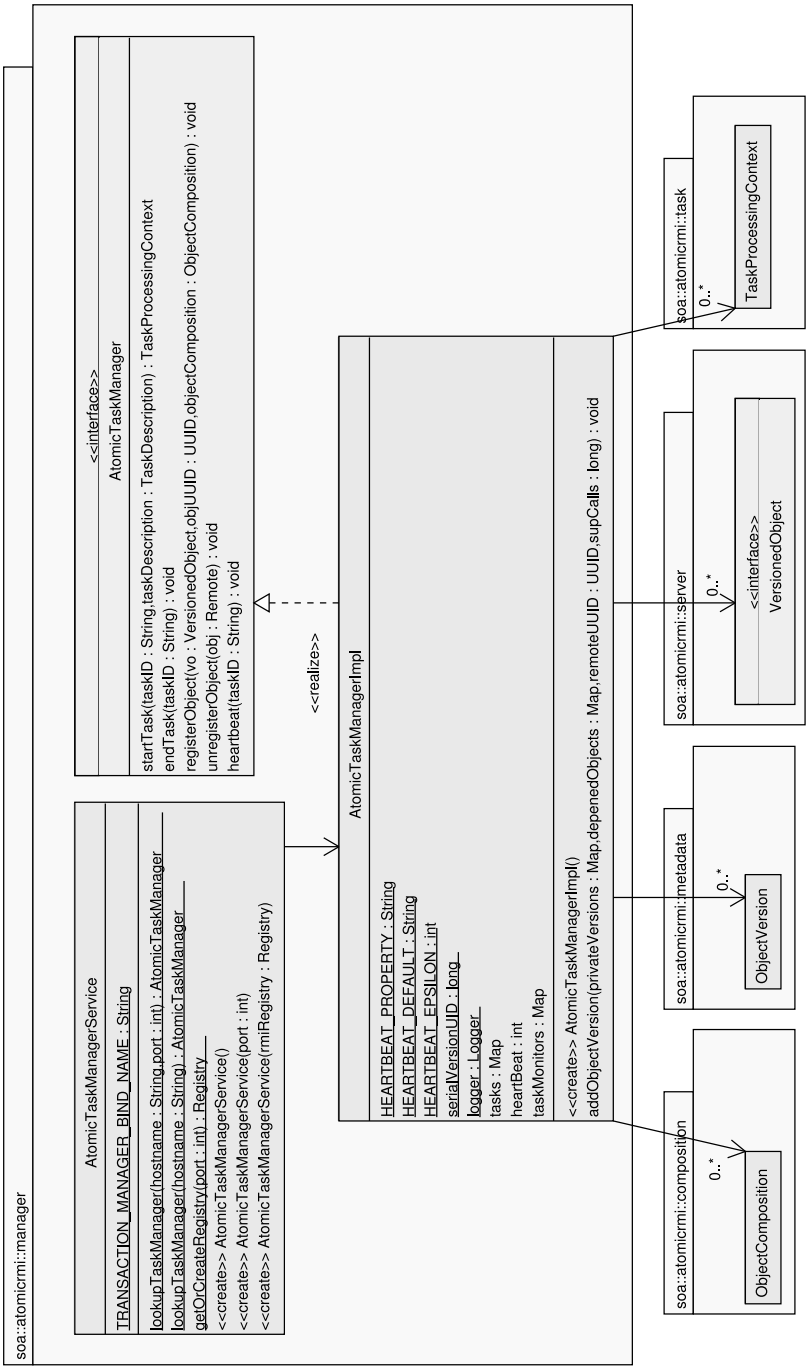

Fig. 9. A class diagram of the atomic task manager in Atomic RMI

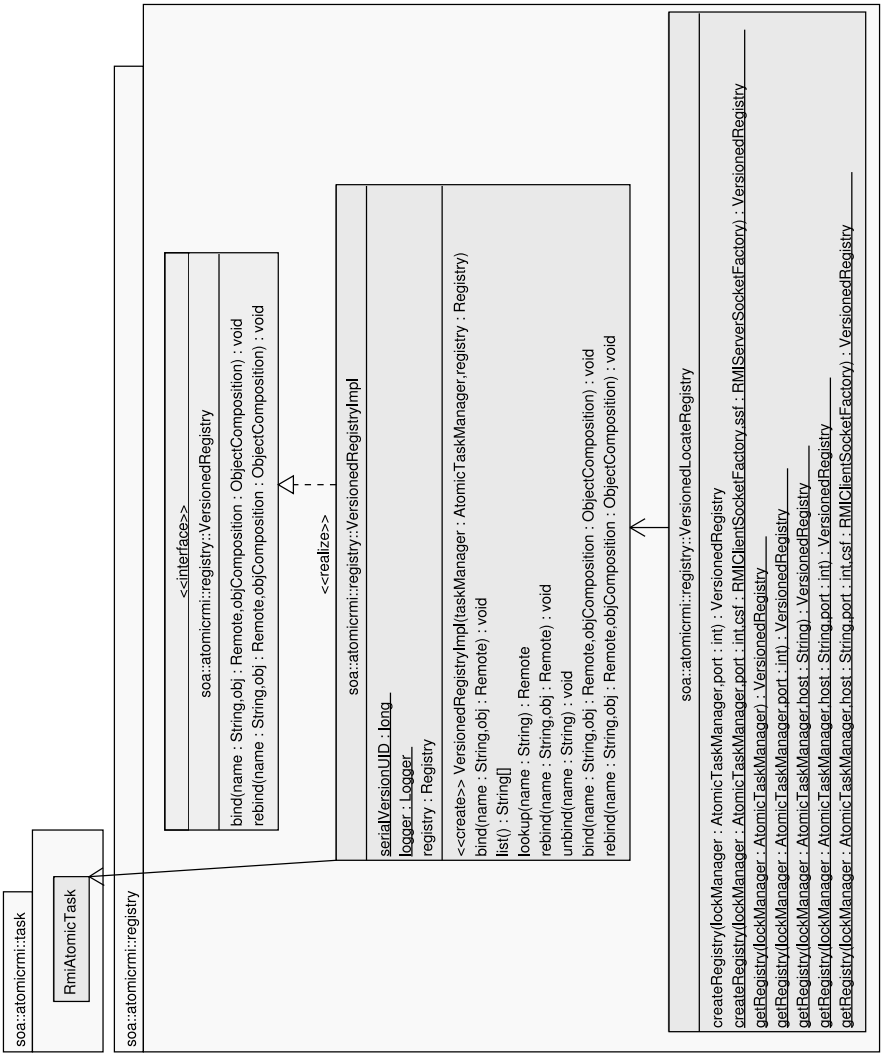

Fig. 10. Classes of the object registry in Atomic RMI

soa.atomicrmi.registry package, there has also been created an auxiliary class *VersionedLocateRegistry*, which has methods that can be used to obtain references to the Atomic RMI registry; the latter can run on any host.

The *RmiAtomicTask* class shown in Figure 11 defines methods that start and terminate the execution of transactions (tasks). It guarantees serializable access to remote objects by a transaction (task).

The execution of a transaction is associated with an object of a *TaskProcessingContext* type, which contains an up-to-date description of the remote object versions which are necessary for correct execution of the versioning algorithms. The upper bounds on the number of accesses to remote objects are given using

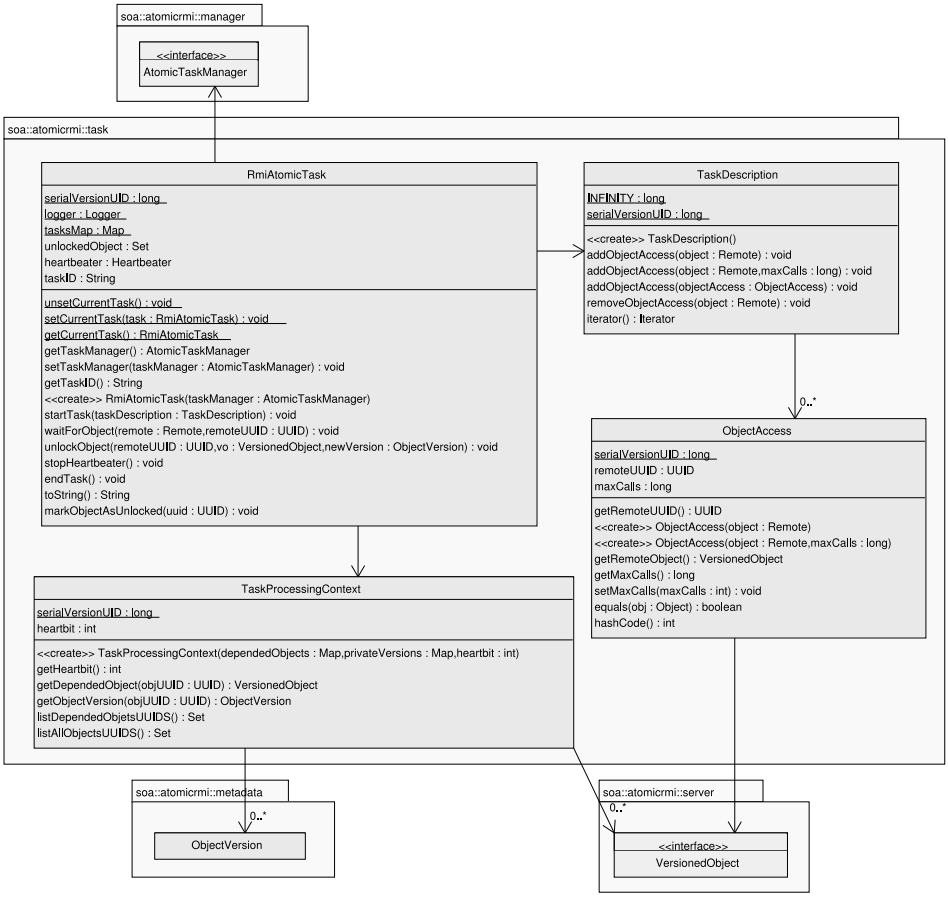

Fig. 11. A class diagram defining atomic tasks and dependencies of objects used by the tasks

the *TaskDescription* class, which specifies an access to a single object with the use of the *ObjectAccess* class.

An alternative way of describing dependencies between remote objects is with the use of the *composition* mechanism. Figure 12 shows the *soa.atomicrmi.composition* package which consists of classes that deliver such a functionality. The *ObjectComposition* class stores for a given object a list of dependencies on other remote objects; the dependency information is stored in containers of a *Dependency* type. References to remote objects are wrapped with the *RemoteObjWrapper* class.

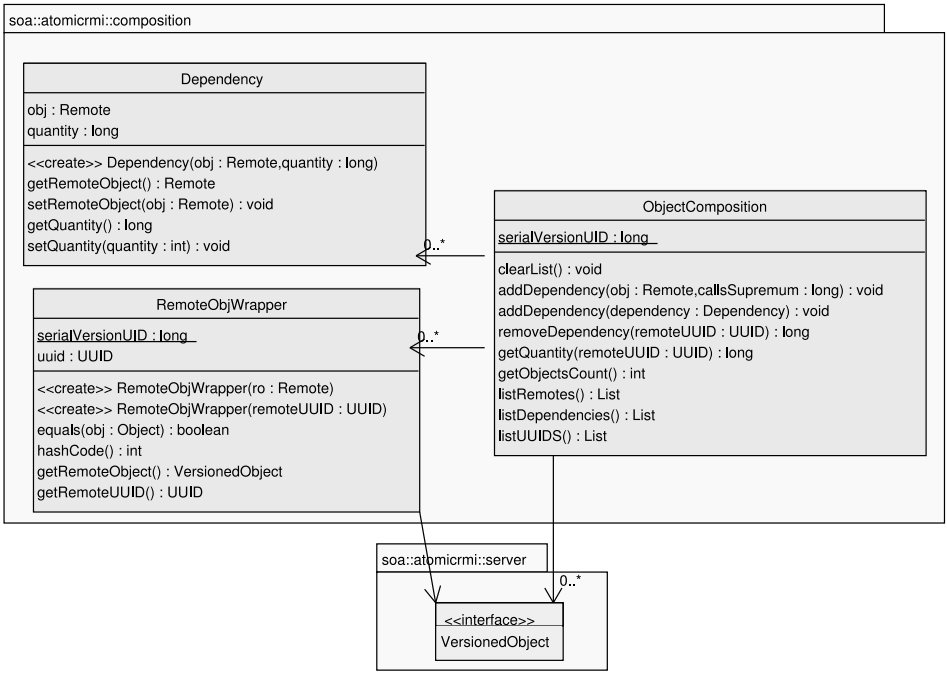

Fig. 12. Classes that enable the composition of dependencies between remote objects

Figure 13 shows classes that are used by the Atomic RMI library internally. The abstract class *VersionedObjectProxy* is the wrapping of remote objects, enriched with methods that control access to the remote objects being wrapped. A current version of an object is described in the *ObjectVersion* class. This version is used by the versioning algorithm. Depending on whether the wrapped remote object is on the same node or on a remote node, a proper implementation of the *VersionedObjectProxy* class is used. In the former case, it is *VersionedObjectServerProxy*, while in the latter, it is *VersionedObjectClientProxy*.

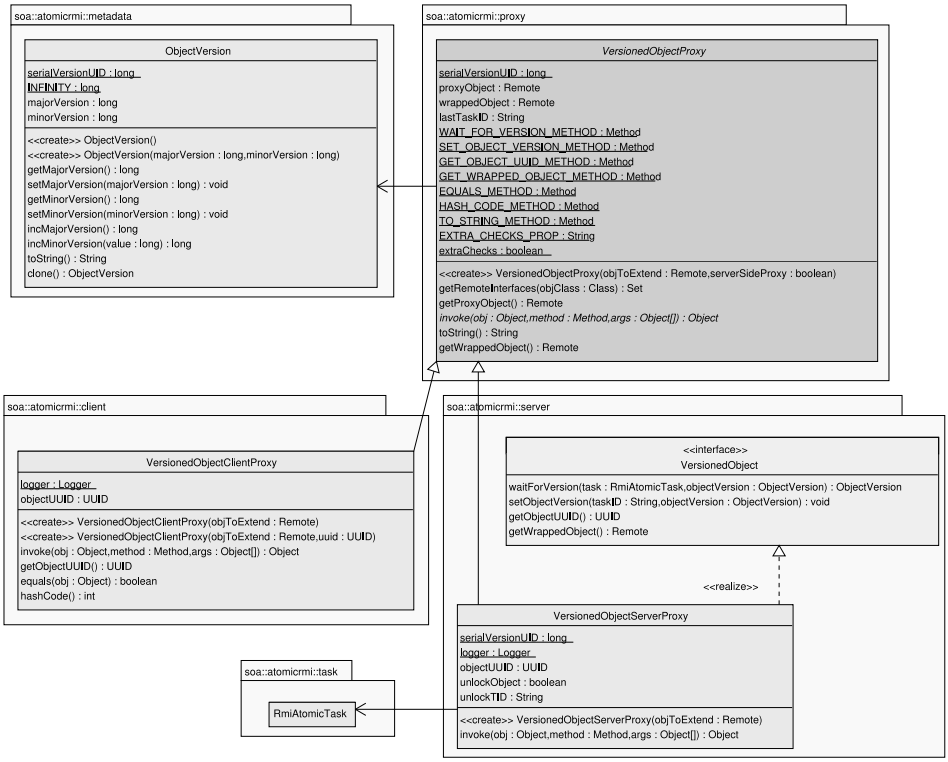

Fig. 13. A diagram of classes that are used by the Atomic RMI library internally

4.4 Future Work

Currently, our implementation of distributed transactions in Atomic RMI does not tolerate fatal errors. If a process executing a transaction lost connection with a remote object (service), it continues after a timeout and the transaction completes partially. Unfortunately, this behavior violates atomicity and may also violate consistency since data integrity can no longer be guaranteed. Therefore, we began working towards extending our transaction mechanism with rollback recovery that will solve this problem. A necessary step in this process, was the design and implementation of Atomic Locks, mentioned in Section 4.2.

In parallel with the development of synchronization tools for SOA, we are also interested in using atomic transactions as the means for increasing service resilience. *Service resilience*, defined as the continued availability of a service despite failures and other negative changes in its environment, is vital in the Service-

Oriented Architecture context. A typical way of increasing service resilience is to replicate services. We plan to develop a tool for *transactional replication* of the SOA services. For more details, see the future work section in the chapter concerned with replication tools.

5 Conclusions

In this chapter, we have presented preliminary results of three disjoint subprojects on transaction processing tools for service-oriented applications. A full presentation of these results has appeared in the referenced technical reports and papers or is waiting for publication. The Transaction Coordinator has proven to be a useful tool for the definition and management of business transactions in the SOAP-based and RESTful Web services. The Business Process Rollback-Recovery Service provides a rich and integrated set of solutions to guarantee seamless rollback-recovery of faulty processes within the Service-Oriented Architecture. The Atomic RMI library for Java delivers a useful programming abstraction for ensuring consistent processing of service requests in a distributed system. In the future, we are planning to continue the described work.

Acknowledgments. This work has been partially supported by the Polish Ministry of Science and Higher Education within the European Regional Development Fund, Grant No. POIG.01.03.01-00-008/08.

The authors would like to thank Mateusz Hołenko, Wojciech Mruczkiewicz, Konrad Siek and Piotr Zierhoffer for discussions and comments. We also thank Wojciech for preparing the UML diagrams.

References

1. Reference Architecture Foundation for Service Oriented Architecture — Version 1.0. OASIS, Oct. 2009, `http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-cd-02.pdf`.
2. Adya A., Liskov B., O'Neil P.E. Generalized isolation level definitions. Proc. ICDE '00: Conf. on Data Engineering, San Diego, California, 2000.
3. Web Services. W3C, 2009, `http://www.w3.org/2002/ws/`.

4. Wojciechowski P.T. Language design for atomicity, declarative synchronization, and dynamic update in communicating systems. Poznań University of Technology Press, 2007.
5. Wojciechowski P.T., Rütti O., Schiper A. SAMOA: A framework for a synchronisation-augmented microprotocol approach. Proc. 18th IEEE Int. Parallel and Distributed Processing Symp. IPDPS'04, Santa Fe, New Mexico, April 2004.
6. Feingold M., Jeyaraman R. OASIS Web Services Coordination Version 1.2. OASIS, Feb. 2009, `http://docs.oasis-open.org/ws-tx/wscoor/2006/06`.
7. Little M., Wilkinson A. OASIS Web Services Atomic Transaction version 1.2. OASIS, Feb. 2009, `http://docs.oasis-open.org/ws-tx/wsat/2006/06`.
8. Freund T., Little M. OASIS Web Services Business Activity version 1.2. OASIS, Feb. 2009, `http://docs.oasis-open.org/ws-tx/wsba/2006/06`.
9. Alves A., Arkin A., Askary S., Barreto C., Bloch B., Curbera F., Ford M., Goland Y., Guízar A., Kartha N., Liu C.K., Khalaf R., König R., Marin M., Mehta V., Thatte S., van der Rijn D., Yendluri P., Yiu A. Web Services Business Process Execution Language. OASIS, Apr. 2009, `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`.
10. Bernstein P.A., Hadzilacos V., Goodman N. Concurrency control and recovery in database systems. Addison-Wesley Longman, Boston. 1986..
11. Jankiewicz K., Kujawiński K., Mor M., Morzy T. Mechanizmy transakcyjności w kompozycji usług w złożone procesy biznesowe. tech. rep., Institute of Computing Science, Poznań University of Technology, TR-ITSOA-OB2-5-PR-09-12, Dec. 2009.
12. Chan P.P.-W., Lyu M.R., Malek M. Making services fault tolerant. Proc. 3rd Int. Service Availability Symp. ISAS'06, Helsinki, Finland, 43–61, 2006.
13. Dialani V., Miles S., Moreau L., Roure D.D., Luck M. Transparent fault tolerance for web services based architectures. Proc. 8th Int. Euro-Par Conf., Paderborn, Germany, 2002.
14. Elmootazbellah N., Elnozahy, Lorenzo A., Wang Y.-M., Johnson D. A survey of rollback-recovery protocols in message-passing systems. ACM Comp. Surv. 34, 3, 375–408, Sep. 2002.
15. Maloney A., Goscinski A. A survey and review of the current state of rollback-recovery for cluster systems. Concurr. Comp. Pract. Exp. 21, 12, 1632–1666, 2009.
16. Cao G., Singhal M. On coordinated checkpointing in distributed systems. IEEE Trans. Parall. Distrib. Syst. 9, 12, 1213–1225, 1998.
17. Mostéfaoui A., Raynal M. Efficient message logging for uncoordinated checkpointing protocols. Proc. 2nd European Dependable Computing Conf. on Dependable Computing EDCC-2, London, UK, 353–364, Springer-Verlag, 1996.
18. Baldoni R. A communication-induced checkpointing protocol that ensures rollback-dependency trackability. Proc. 27th Int. Symp. on Fault-Tolerant Computing FTCS'97, Washington, DC, 68, IEEE Computer Society, 1997.

19. Randell B. System structure for software fault tolerance. Proc. Int. Conf. on Reliable Software, New York, New York, 437–449, ACM, 1975.
20. Damani O., Garg V. How to recover efficiently and asynchronously when optimism fails. Proc. 16th IEEE Int. Conf. on Distributed Computing Systems ICDCS'96, Hong Kong, 1996.
21. Alvisi L., Marzullo K. Message logging: Pessimistic, optimistic, causal, and optimal. IEEE Trans. Software Eng. 24, 149–159, 1998.
22. Richardson L., Ruby S. RESTful Web Services. O'Reilly, Sebastopol, USA, 2007.
23. Tartanoglu F., Issarny V., Romanovsky A., Levy N. Coordinated forward error recovery for composite Web Services. Proc. 22nd Symp. on Reliable Distributed Systems SRDS'03, Florence, Italy, 2003.
24. Danilecki A., Holenko M., Kobusińska A., Rudnicki F., Rzepecki K., Szychowiak M., Zierhoffer P. Rollback-recovery mechanisms in SOA environments. tech. rep., Institute of Computing Science, Poznań University of Technology, TR-ITSOA-OB2-4-PR-09-02, 2009.
25. Documentation for Oracle BPEL Process Manager and Human Wokflow. `http://www.oracle.com/technology/products/soa/bpel/collateral/documentation.html`, Oracle Corporation, 2009.
26. Cabrera L.F., Copeland G., Cox B., Freund T., Klein J., Storey T., Thatte S. Web Services Transactions specifications. 2005, `http://www-106.ibm.com/developerworks/webservices/library/ws-transpec/`.
27. Baresi L., Guinea S., Pasquale L. Self-healing BPEL processes with dynamo and the JBoss rule engine. Proc. Workshop on Engineering of Software Services for Pervasive Environments: in conjunction with the 6th ESEC/FSE joint conference, 11–20, Durbrovnik, Croatia, 2007.
28. Dobson G. Using WS-BPEL to implement software fault tolerance for web services. Proc. 32nd Conf. on Software Engineering and Advanced Applications EUROMICRO'06, Washington, DC, 126–133, IEEE Computer Society, 2006.
29. Juhnke E., Dornemann T., Freisleben B. Fault-tolerant bpel workflow execution via cloud-aware recovery policies. Proc. 35th EUROMICRO Conf. on Software Engineering and Advanced Applications SEAA'09, 31–38, Patras, Greece, 2009.
30. Modafferi S., Mussi E., Pernici B. SH-BPEL: a self-healing plug-in for Ws-BPEL engines. Proc. 1st Workshop on Middleware for Service Oriented Computing MW4SOC'06, New York, 48–53, ACM, 2006.
31. Lau J., Lung L.C., da S. Fraga J., Veronese G.S. Designing fault tolerant web services using BPEL. Proc. Seventh IEEE/ACIS Int. Conf. on Computer and Information Science ICIS'08, Washington, DC, 618–623, IEEE Computer Society, 2008.
32. Brzeziński J., Hołenko M., Kobusińska A., Zierhoffer P. Założenia projektowe, model i architektura usługi zapewniającej odtwarzanie procesów biznesowych. tech. rep., Institute of Computing Science, Poznań University of Technology, TR-ITSOA-OB2-4-PR-09-05, 2009.

33. Guerraoui R., Rodrigues L. Introduction to reliable distributed programming. Springer-Verlag, New York, 2004.

34. Sergent N., Dfago X., Schiper A. Failure detectors: Implementation issues and impact on consensus performance. tech. rep., École Polytechnique Fédérale de Lausanne, Switzerland, SSC/1999/019, 1999.

35. Kamiński J., Kaźmierczak M., Kobusiński J., Nowacki S., Rosiński K. Failure detection mechanism in SOA environments. tech. rep., Institute of Computing Science, Poznań University of Technology, TR-ITSOA-OB1-4-PR-09-02, 2009.

36. Harris T., Marlow S., Peyton Jones S., Herlihy M. Composable memory transactions. Proc. 10th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming PPoPP'05, Chicago, Illinois, 2005.

37. Shavit N., Touitou D. Software transactional memory. Proc. 14th ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing PODCS'95, Ottawa, 1995.

38. Harris T., Fraser K. Language support for lightweight transactions. Proc. 18th ACM SIGPLAN Conf. on Object-Oriented Programming, Systems, Languages, and Applications OOPSLA'03, Anaheim, California, 2003.

39. Ringenburg M.F., Grossman D. AtomCaml: first-class atomicity via rollback. Proc. 10th ACM SIGPLAN Int. Conf. on Functional Programming ICFP'05, Talinn, Estonia, 2005.

40. Ni Y., Welc A., Adl-Tabatabai A.-R., Bach M., Berkowits S., Cownie J., Geva R., Kozhukow S., Narayanaswamy R., Olivier J., Preis S., Saha B., Tal A., Tian X. Design and implementation of transactional constructs for C/C++. Proc. 23rd ACM SIGPLAN Conf. on Object-Oriented Programming, Systems Languages and Applications OOPSLA'08, Nashville, TN, 2008.

41. Kryger P., Wojciechowski P.T. Analiza i porównanie wybranych realizacji Software Transactional Memory. tech. rep., Institute of Computing Science, Poznań University of Technology, TR-ITSOA-OB2-2-PR-09-2, 2009.

42. Flanagan C., Qadeer S. A type and effect system for atomicity. Proc. Conf. on Programming Language Design and Implementation PLDI'03, San Diego, California, 2003.

43. Kryger P., Wojciechowski P.T. Atomic Locks: Projekt i implementacja mechanizmu pamięci transakcyjnej w języku Java. tech. rep., Institute of Computing Science, Poznań University of Technology, TR-ITSOA-OB2-2-PR-09-6, 2009.

44. Wojciechowski P.T. Isolation-only transactions by typing and versioning. Proc. 7th ACM-SIGPLAN Int. Symp. on Principles and Practice of Declarative Programming PPDP'05, Lisboa, Portugal, 2005.

45. Mamoński M., Wojciechowski P.T. Atomic RMI: Projekt i implementacja mechanizmu rozproszonych operacji atomowych w języku Java. tech. rep., Institute of Computing Science, Poznań University of Technology, TR-ITSOA-OB2-2-PR-09-4, 2009.

46. Wojciechowski P.T. Extending atomic tasks to distributed atomic tasks. Proc. of the Workshop on Exploiting Concurrency Efficiently and Correctly $(EC)^2$ co-located with 20th Int. Conf. on Computer Aided Verification CAV'08, Princeton, New Jersey, 2008.
47. Microsystems S. Java Remote Method Invocation (Java RMI). Oct 1998, `http://java.sun.com/products/jdk/rmi/`.
48. Siek S., Wojciechowski P.T. Design and implementation of the precompilation tool in Java for Atomic RMI. tech. rep., Institute of Computing Science, Poznań University of Technology, TR-ITSOA-OB2-2-PR-09-5, 2009.

Replication Tools for Service-Oriented Architecture

Jerzy Brzeziński, Michał Kalewski, Marek Libuda, Cezary Sobaniec, Dariusz Wawrzyniak, Paweł Wojciechowski, and Piotr Wysocki

Institute of Computing Science
Poznań University of Technology,
Piotrowo 2, 60-965 Poznań, Poland
Jerzy.Brzezinski@put.poznan.pl,
{Michal.Kalewski,Marek.Libuda,Cezary.Sobaniec,Dariusz.Wawrzyniak,
Pawel.T.Wojciechowski,Piotr.Wysocki}@cs.put.poznan.pl

Abstract. This chapter describes the concept of replication tools for SOA-based systems. Replication is expected to ensure availability, atomic consistency, and partition tolerance. However, at most two of these features can be guaranteed. The proposed mechanisms cover various combinations of the above features: the replicated state machine approach, implemented on top of a group communication protocol, guarantees strict consistency and availability under the assumption that messages are successfully delivered, and the SAS system for mobile *ad hoc* networks is aimed at availability at the expense of consistency.

Keywords: replication, consistency, replicated state machine, group communication, atomic broadcast, mobile *ad hoc* network

1 Introduction

A service-oriented architecture (SOA) assumes information systems to be decomposed into loosely-coupled interoperable services. Loose coupling in the case of link or host failures may lead to system behavior that seems abnormal or faulty. However, due to the requirement of interoperability, service resilience — defined as the continued availability of a service despite failures or other negative changes in its environment — is vital to SOA-based systems. Therefore, the SOA infrastructure mechanisms must ensure that each service is highly available regardless of unpredictable conditions, such as sudden and significant degradation of network latency or failure of dependent services.

Replication is a means to improve the availability of data or other resources, and thereby the resilience of services. Besides, it can influence efficiency by increasing the throughput of a service available at multiple points on different hosts working in parallel. Moreover, a form of replication — caching — can reduce the response time of a service, because it allows avoiding time-consuming computation to get the same result. Another benefit of caching is the chance to find the result closer to the requesting client, which possibly circumvents the issue of network latency.

The main problem of replication is consistency. Assuming the system model in which a given server handles a number of possibly replicated resources, the problem is raised when the state of a resource replica changes. This is subsequent to a modification of the resource, performed on one of its replicas. A straightforward way of coping with this problem is to prevent accessing each replica of the resource before the modification starts, and release the replicas after all of them have been updated. This approach leads to the strongest consistency model — the atomic one. [1, 2, 3]. One of the simplest implementation of the idea is the invalidation of all replicas except the one being modified. On the other hand, reduction to only one replica decreases availability.

The trade-off between consistency and availability in the context of transaction processing in partitioned networks has been mentioned in [4]. This topic has been raised by Brewer [5], and then formalized by Gilbert and Lynch [6]. According to Brewer's conjecture, only two of the following three properties can be obtained in an asynchronous distributed system:

— atomic consistency;
— availability;
— partition tolerance.

This means that replication can ensure atomic consistency and availability, provided that no network partition occurs (no messages are eventually lost). This "impossibility result" holds even in the case of a partially synchronous network model [6]. Besides, achieving strict consistency provided by the atomic (or sequential) model is costly, because it requires coordination between all replicas. Consequently, the approach to replication aimed at strict consistency and availability is hardly scalable, thus in practice, limited to relatively few hosts working rather in a local area network. In view of the facts and under the assumption of no partitioning, a replicated state machine approach is developed, as a means of improving service resilience in case some hosts fail.

Strict consistency is not always necessary for a replicated service to work correctly. A perfect example of weaker consistency requirements is a directory service [7, 8]. Generally, the approaches to consistency that allow for temporary divergence between replicas are termed *optimistic*, because they are based on the optimistic

assumption that no conflicts appear during updates, thereby the difference remains invisible or unimportant. A potential benefit of optimistic replication is higher performance.

Optimistic replication becomes especially significant in systems which allow for network partitioning. The requirement of strict consistency in such systems might compromise availability, thereby undermining the sense of replication. While network partitioning appears transient in a stable (or wireless) local area network, it cannot be neglected in wider area network. A severe case in this aspect is a mobile *ad hoc* network. Besides the consistency problem, the crucial issue in *ad hoc* networks is proper replica allocation and relocation that should allow for dynamic network reconfiguration. This, in turn, complicates the localization of replicas.

The chapter is organized as follows. Section 2 describes a general idea of the proposed replication mechanism. Section 3 addresses the issue of a replicated state machine, and atomic broadcast as a means of implementing it. Section 4 focuses on dynamic replication (replica allocation and relocation) in distributed *ad hoc* systems. Some conclusions are presented in Section 5.

2 Basic Concepts

2.1 Replication Technique

In order to preserve resilience of services, special care must be taken over the number of replicas. To avoid reducing this number, for the sake of consistency maintenance, update of replicas should be considered rather than invalidation. A replica can be updated in a twofold manner:

— state transfer — the modified state of one replica is passed to other servers;
— operation transfer — the modifying operation is performed on each replica.

A natural consequence of the difference in these approaches to a replica update is the distinction between passive and active replication [9, 10, 11, 12]. *Passive replication* means that there is one distinguished replica server that processes requests from clients and disseminates a new state to other replicas. In *active replication* the requests are processed by each replica server on its own.

Another classification addresses the issue of request distribution. In a *primary-backup* (master-slave) approach one replica server is designated to accept the requests (especially modifications) from clients and communicate with other (backup) servers in order to process them. In a multi-primary (multi-master) approach several servers can accept requests.

Table 1. Replication techniques

	Operation transfer	State transfer
Primary-backup	Semi-active	Passive
Multi-master	Active	Semi-passive

State transfer is more usual in a primary-backup approach, which leads to passive replication in a strict sense. However, operation transfer is also possible in this case, and this technique is called *semi-active* replication [10]. Moreover, *semi-passive* replication can be distinguished, in which a given request is addressed to several servers, but only one of them actually accepts and processes it, then disseminates a new state to other replicas [13]. These techniques of replication are summarized in Table 1.

Except for active replication, all the forms require designation of one replica to accept or process requests from clients. This may limit the potential throughput, thereby compromising efficiency. Hence, in the presented concept of the replication mechanism active replication is considered.

2.2 Replication Infrastructure

Replication can be applied to a given application service in two distinct ways: either as a mechanism tightly integrated with the replicated application service or as an independent protocol layer (or service) transparent to the application. The advantage of the former approach, termed *semantic* [4], is the possibility to adapt the replication mechanism for the application and, if necessary, to adjust the application service to the replication. For instance, this approach is used in the implementation of a naming or directory service [14, 15]. However, it is usually not general enough, to be effortlessly used in the case of another application with different characteristic.

The latter approach (*syntactic*) it is more suitable for the SOA paradigm because it allows separating the replication mechanism as an autonomous service. Nonetheless, effective replication requires some service specific information. The key issue is the distinction between read-only and modifying requests, which results from the semantics of operations. Besides, the relationships between services or resources may also be important.

Since it is difficult (or even impossible) to retrieve this kind of information from the implementation, there must be a supplementary description of the services available to the replication mechanism, or an access protocol must provide a fixed set of operations (methods) with well-defined semantics. This alternative reflects the difference between RPC-based approaches (including SOAP-based Web services, also referred to as Big Web services, Web services or WS-* Web services)

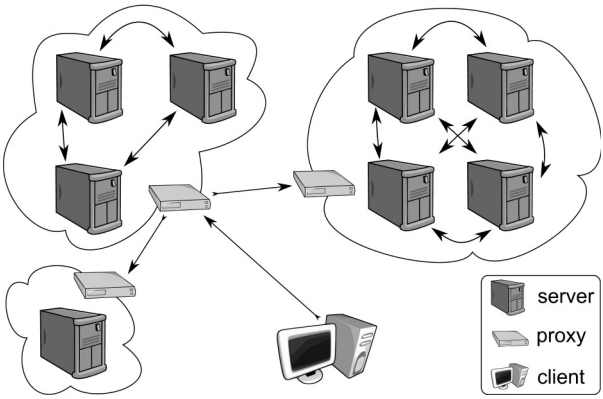

Fig. 1. Replication architecture

[16, 17] and a resource oriented model (REST-based Web services, RESTful Web services) [18, 19].

The replication mechanism described here is aimed at a resource oriented model. Specifically, it is developed for RESTful Web services — Web services designed in a REST style and implemented with the use of the HTTP protocol. One of the principle assumptions of the REST methodology is a **uniform interface**. In the case of RESTful Web services this means that such a service provides some resources that are accessed by one of the HTTP methods: GET, POST, PUT, DELETE. According to the semantics of the methods, GETs do not change the state of the accessed resource, thereby do not raise inconsistency. The problem of consistency maintenance appears in the case of other methods mentioned above. Moreover, some HTTP methods, especially PUT and DELETE, are idempotent, which creates the opportunity for update optimization.

In order to cover replication in a wide area network and also exploit the potential of a local area network, a hybrid replication mechanism can be constructed. One component of this mechanism — a replicated state machine — is implemented on top of a group communication protocol (atomic broadcast). The primary purpose of this component is the resilience to fail-stop crashes of hosts. Although it is intended for replication in LAN, it is not restricted to LAN — it can also work in a wide area network. However, due to efficiency and possible communication failures, optimistic replication (provided by the other component) is applied. This idea of replication shapes the architecture as illustrated in Figure 1. Clouds denote replicated state machines that can be regarded as high-availability clusters. A cluster is available from outside (to the clients and other clusters) by means of a proxy. Proxies implement the protocol for optimistic replication and manage

a buffer cache. To this end, a concept of modular proxy (MProxy) has been developed. MProxy is organized as a chain of dynamically loadable modules that process HTTP messages (both requests and responses). The implementation of MProxy uses HttpComponents [20] for HTTP message handling, and OSGi [21] for dynamic management of modules.

3 Replication Tools for Service Resiliency

3.1 Basic Definitions and Problem Formulation

Replicated State Machine Approach. *Service replication* means deployment of a service on several server machines, each of which may fail independently, and coordination of client interactions with service replicas. Each service replica, called a *process*, starts in the same initial state and executes the same requests in the same order. Thus, the replicated service executes simultaneously on all machines. A client can use any single response to its request to the service. A replicated service is available continuously, tolerating crashes of individual server machines. If required, these machines can be located in geographically distant places, connected via a wide-area network.

A general model of such replication is called *replicated state machine* [9]. In this model, it is guaranteed to maintain a consistent state of all non-faulty processes in a group, if only the group counts a sufficient number of processes. Outputs of a state machine are completely determined by the sequence of requests it processes, independently of time or any other activity of the system. Two properties are guaranteed by the replicated state machine: (1) each non-fault replica receives every request (the *agreement property*), and (2) each non-fault replica processes the requests in the same relative order (the *order property*). These two properties can be implemented using a group communication system.

Group Communication Systems. *Group Communication Systems (GCS)* offer programming abstractions for:

— detection of malfunctioning/crashed processes;
— reliable point-to-point transfer of messages;
— formation of processes into groups, the structure of which can change at runtime;
— reliable message multicasting with a wide range of guarantees concerning delivery of messages to group members (e.g. causally-, fifo- and totally-ordered message delivery).

Notably, the overlay protocols for reliable multicasting do not depend on any central server, so that there is no single point of failure. For this, *distributed agreement* protocols must be used, such as the Distributed Consensus protocol. An example of an application of group communication systems is replication of a service in order to increase its resilience, with the use of the replicated state machine approach, described above. Our goal was to design GCS systems for SOA.

3.2 Related Concepts and Contributions

For the past 20+ years, many group communication systems have been implemented (see e.g. [22, 23, 24, 25, 26, 27, 28, 29, 30, 31] among others). However, there are few commercial systems (e.g. JGroups [31] and Spread Toolkit [30]). Various group communication systems usually have quite different application programming interfaces, which are non-standard, complex and language/platform dependent. Moreover, many of such systems are monolithic, i.e. it is not possible to replace their protocols or add new features. Using these systems to implement SOA services makes the code of services not easily reusable or manageable (adaptable), which is a counterexample to the Service-Oriented Architecture.

The contribution in the area of group communication support for SOA, are CamlGroups and RESTGroups — two group communication programming tools that can be used for implementing resilient services. *CamlGroups* [32, 33] is a modular, easily extendable group communication system that can be used for prototyping new protocols for service resilience. *RESTGroups* [34, 35] is an extension of the Spread Toolkit for developing resilient production SOA applications, based on REST [36] over the standard HTTP [37]. To our best knowledge, RESTGroups is the first RESTful group communication service. Both systems have been described in detail in other papers. In this section, we summarize this work. Firstly, we describe the semantics of broadcast primitives provided by these systems. Then, we sketch an architecture of CamlGroups and RESTGroups. Both systems have been implemented; the distribution files and documentation are available (CamlGroups [33] and RESTGroups [35]).

3.3 CamlGroups and RESTGroups

Functional Requirements. Group communication systems provide various primitives, e.g. for a unicast and broadcast within a group of processes, for maintaining group membership, for maintaining virtual (or view) synchrony, and for detecting process failures. Message broadcasting can be described with the use of two operations: a *broadcast* operation, which is used by a sender process to send a message to all processes in a group, and a *deliver* operation, which is used by a

recipient process to deliver a broadcast message. Below we define joint functional requirements of CamlGroups and RESTGroups — two group communication systems that will be described in the remainder of this section. A more detailed specification of functional requirements can be found in the Technical Report [34] and popular textbooks (see e.g. [38]).

Failure Models. The design and implementation of group communication systems depend on a model of system failures. In the simplest case, we consider a *crash-stop* (or *crash-no-recovery*) model, in which a process that has failed stops its execution. In this model, a process is said to be *correct* if it does not crash; the notion of "correctness" refers to the whole execution time of a process. A process that crashes is said to be *incorrect* (or *faulty*). In this model the processes that crashed do not recover.

In a *crash-recovery* model, processes can be recovered after a failure. In this model, we consider a process to be incorrect if it crashes at least once and eventually cannot recover its state (*eventually down*), or if it crashes and recovers infinitely often (*unstable*). Otherwise, a process is correct, i.e. it never crashes (*always up*) or crashes at least once but eventually recovers and does not crash anymore (*eventually up*).

Broadcast Properties. The simplest broadcast primitive is *Unreliable Broadcast*, which allows a message to be sent to all processes in a group, guaranteeing that if a message sender is correct, then all processes in the group will eventually deliver the message. However, if the sender crashes during the broadcast, then some processes in the group may not deliver the message. Obviously, this primitive is not much useful in systems, in which failures may occur. *(Regular) Reliable Broadcast* solves this problem; it guarantees the following properties:

— *Validity*: if a correct process broadcasts a message, then it eventually delivers the message.
— *Agreement*: if a correct process delivers a message, then all correct processes eventually deliver the message.
— *Uniform Integrity*: for any message, every process delivers the message at most once, and only if the message was previously broadcast.

Note that Reliable Broadcast allows executions, in which a faulty process delivers a message but no correct process delivers the message. *Uniform Reliable Broadcast* is a stronger version of Reliable Broadcast, which satisfies the Validity and Integrity properties defined above but replaces the Agreement property by the following:

— *Uniform agreement*: if a process (correct or not) delivers a message, then all correct processes will eventually deliver the message.

The Regular (or Uniform) Reliable Broadcast primitives provide a basis for stronger broadcast primitives, which have additional properties, e.g.:

— *FIFO order*: this property guarantees that messages sent by a process are delivered in the order of sending.
— *Causal order*: this property means that if some process has sent a message m_1 that caused sending of another message m_2, then each process in a group will deliver m_1 before m_2.
— *Total order*: this property means that messages sent by any different processes in a group will be delivered by all processes in the group in the same order (note that this property does not guarantee FIFO).
— *Generic order*: given a *conflict relation* $C : M \times M \rightarrow Boolean$, where M is a set of broadcast messages, if two messages are in conflict, they must be delivered to all processes in the same order. Otherwise, the messages are not ordered.

The (Uniform) Reliable Broadcast Protocols that support the above properties are called, correspondingly: *FIFO Broadcast*, *Causal Broadcast*, *Total-Order (or Atomic) Broadcast* and *Generic Broadcast*. Note that if the conflict relation C is empty, Generic Broadcast is reduced to Reliable Broadcast. Whereas, if all pairs of broadcast messages are in conflict, then Generic Broadcast is equivalent to Atomic Broadcast.

Below, we briefly describe CamlGroups and RESTGroups. A full presentation of this work has appeared in referenced technical reports and papers or is waiting for publication.

Architecture of CamlGroups. In this section, we describe a software architecture of the CamlGroups group communication system [32]. The system consists of several modular protocols that are composed into a protocol stack. Each protocol in the stack implements some *service*. The protocols can only depend on protocols that are at the lower layer in the stack.

Protocols in the stack communicate using a uniform communication interface, which jointly with the programming interface defines the CamlGroups *protocol framework*. The communication is based on the use of *higher-order* functions and a *callback* mechanism. On a system startup, the protocol framework initiates all protocols composed into the stack, passing to them the functions that are used for flow control in the stack. A given protocol can only call services that are "below" in the stack, where an argument of a service call is usually a message. Responses to these calls are delivered via callbacks.

No implementation details of the protocols need to be known, in order to compose and use them. Thus, this architecture facilitates adaptation and reuse of the protocols, as required by the SOA. Protocols implementing the same service can be easily replaced in the stack. Modularity also makes it easier to understand

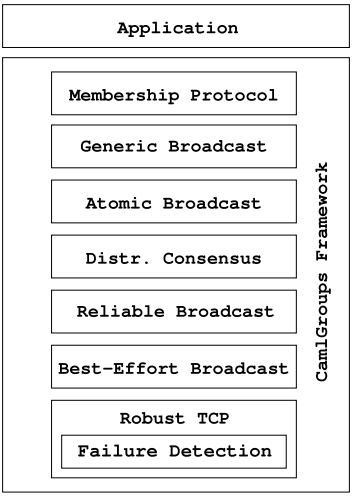

Fig. 2. The CamlGroups software architecture

and extend the system since dependencies between protocols are kept clear and intact both at the design and implementation level.

In Figure 2, we show software modules that are composed into a modular protocol stack for dynamic groups, with the use of a CamlGroups framework. It is a simplified version of the modular group communication stack which has been proposed in [39].

To implement the system, the Objective Caml (OCaml) programming language has been used. OCaml [40] is a functional, strongly typed programming language, whose features, such as: abstract types, pattern matching constructs, interfaces, modules, higher-order functions, and communication of function closures in the network, significantly simplified and fastened the implementation and testing of CamlGroups (see also [41]).

Below, we briefly describe in turn all layers of this architecture, beginning from the lowest communication layer above the transport protocol. Due to lack of space, we omitted the description of the programming and communication interfaces; these and other details can be found in [32]. Preliminary work on the implementation of some protocols, which are used by CamlGroups, has been reported in [42].

Robust TCP and Failure Detector. Above the standard TCP or UDP protocols, there must be a protocol layer implementing a reliable link between two processes, which provides additional guarantees to those provided by TCP and absent in

UDP. For example, UDP can lose network packets, while TCP does not deliver a message, when establishing the network connection fails or when the established connection is broken due to a link failure. The *Robust TCP* protocol, which is based on [43], can deal with temporary link failures by introducing an additional session on the top of TCP, guaranteeing the *exactly-once* delivery semantics for all transmitted data.

The Robust TCP protocol and many other group communication protocols use the *Failure Detector* protocol, implementing the service for detection of faulty processes. In CamlGroups, the Failure Detector implements a *heart-beat* algorithm, and is embedded in the Robust TCP module.

Best-Effort and Reliable Broadcasts. The *Best-Effort Broadcast* protocol just sends messages to all processes using *Robust TCP*, with no guarantees on broadcast message delivery. In particular, it does not guarantee that a message will be delivered to all processes, if the sender process crashes during broadcast. Thus, Best-Effort Broadcast satisfies the Unreliable Broadcast semantics.

The *Reliable Broadcast* protocol implements service for reliable message broadcasting, using a message diffusion algorithm (that appeared in [44]), based on the Robust TCP that is forming the lower layer in the protocol stack. Reliable Broadcast satisfies the Reliable Broadcast semantics, described in Section 3.3. The protocol does not guarantee any order of message delivery. Therefore, additional layers of protocols above it are required.

Distributed Consensus. Another layer in the group communication protocol stack is a service for reaching distributed consensus. The problem of *distributed consensus* lies in reaching an agreement by a set of processes on a single value among a set of values that have been proposed by these processes. In an asynchronous system, where processes have no common clock and run at arbitrarily varying speeds, the problem is unsolvable if one process may crash [45]. The impossibility result does not state that distributed consensus can never be reached, but only that under the system's assumptions, no algorithm can always reach consensus in bounded time. In practice, many distributed consensus algorithms have been proposed (e.g. Paxos [46]).

To implement the *Distributed Consensus* protocol in the CamlGroups stack, Chandra-Toueg's algorithm has been used [44]. The implementation of this algorithm in CamlGroups depends on external protocol modules, implementing Robust TCP and Failure Detector. Robust TCP processes each message broadcast by Distributed Consensus, e.g. a proposed value or decision, and tries to deliver it to its recipient.

Atomic and Generic Broadcasts. The *Atomic Broadcast* protocol implements the total-order broadcast service, based on the algorithm in [44]. To be able to guarantee the same order of message delivery to all correct processes, it uses the Distributed Consensus to agree on a consistent message order.

The CamlGroups software architecture includes the *Generic Broadcast* protocol, implementing the generic broadcast service. The protocol has been developed using the *non-trivial generic broadcast* algorithm in [47]. It enables reliable and efficient delivery of total-order messages in a group of processes, where non-conflicting messages (e.g. control messages) are not ordered.

Group Membership. The group communication systems support *dynamic groups*, i.e. processes can join a group at any time, e.g. to replace processes that have crashed. The *Group Membership* protocol implements a membership service, which is responsible for joining/leaving processes into/from a group and for maintaining a consistent group view on each server. Processes can leave a group at will or they can be excluded from a group when they are suspected as failed. The implementation of this protocol uses the generic broadcast service.

Architecture of RESTGroups. *Spread Toolkit* (or *Spread* in short) [30] provides a messaging service for point-to-point and multicast communication that is resilient to faults across local and wide area networks. Spread services range from reliable messaging to fully ordered messages with delivery guarantees. Spread supports several programming languages, for example C/C++, Java and Python.

RESTGroups [34] has been designed as the platform and language independent *front-end* for Spread, i.e. applications that use it can be implemented using a variety of programming languages and can run on different platforms. RESTGroups represents all Spread services as Web resources, accessible using the REST/HTTP architectural style. Since the Web is ubiquitous and the HTTP ports are usually not blocked, we can run the group communication service truly globally.

The distribution of RESTGroups contains the *RESTGroups Library*, providing a RESTful programming interface to the Spread system, a daemon server program implementing the *RESTGroups Server* for communication with a Spread daemon, and various demonstration programs. Below we sketch the RESTful programming interface and the system's architecture; a complete description appeared in [34].

Programming interface. RESTGroups provides a representation of Spread group communication services as resources, identified by URLs, and a simple but powerful *Application Programming Interface (API).* The API uses the following methods of the HTTP protocol for invoking the services:

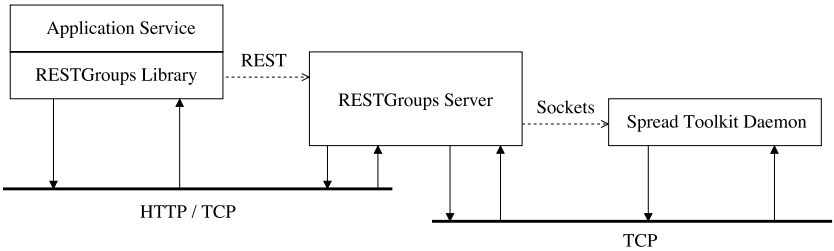

Fig. 3. The RESTGroups system architecture

— GET is used to retrieve data (for example, a message) or perform a query on a resource; the data returned from the RESTGroups service is a representation of the requested resource;
— POST is used to create a new resource (for example, to extend a process group with a new process or to send a new message); the RESTGroups service may respond with data or a status indicating success or failure;
— PUT is used to update existing resources or data;
— DELETE is used to remove a resource or data (for example, to remove a process from a process group); in some cases, the update and delete actions may be performed with POST operations as well.

The RESTGroups system architecture (Fig. 3) shows that RESTGroups is an intermediary between a (replicated) *Application Service* and the Spread group communication system. Instead of communicating with a group communication system (GCS) directly, using its programming interface, a client of the RESTGroups system (i.e. the Application Service in Figure 3), uses exclusively the HTTP methods, following the REST/HTTP architectural style. The methods are provided by the programming interface described above, and implemented by the *RESTGroups Library*.

A suitable request of the HTTP protocol, possibly containing a XML document, is sent to a *RESTGroups Server*, which is a front-end for the GCS. The GCS services are represented as Web resources, which are identified with the use of *Uniform Resource Identifiers (URI)*. The server translates client requests on these resources into concrete calls of the group communication services. In our case, these services are supplied by the *Spread Toolkit Daemon*.

System Deployment. In the case of a concrete RESTGroups application, if the Application Service were replicated on n machines, then, in most cases, we would have: (1) n Spread Toolkit Daemons running on any n machines, and (2) from 1 to n RESTGroups Servers that are communicating with them, usually deployed on the same machines as Spread daemons.

The clients of the replicated Application Service (not shown in Figure 3) can communicate with any replica of the service. Each service replica (Application Service) connects to a dedicated RESTGroups server. Each of the RESTGroups servers can interact with any Spread daemon, using the sockets on TCP. Except when broadcasting of messages can be optimized at the low level of network protocols (which is possible in the local-area networks) the Spread Toolkit Daemons communicate using the *IP unicast* protocol.

3.4 Future Work

In the future, we plan to develop a tool for *transactional replication* of SOA services. The tool's design and implementation will be based on RESTGroups or *Java Paxos* — a new variant of the popular Paxos protocol [46] that will support both the *crash-stop* and *crash-recovery* model of failure. We are currently developing an efficient implementation of Java Paxos.

4 Dynamic Replication of Services in Distributed *Ad Hoc* Systems

4.1 Basic Definitions and Problem Formulation

Mobile ad hoc networks (abbreviated MANET) [48, 49, 50, 51] are composed of autonomous and mobile hosts (or communications devices), which communicate through wireless links. Each pair of such devices, whose distance is less than their transmission range, can communicate directly with each other — a message sent by any host may be received by all hosts in its vicinity (Fig. 4). If hosts in MANET function as both a computing host and a router, they form a *multiple hop ad hoc network*. Hosts can come and go or appear in new places, so with an *ad hoc* network, the topology may be changing all the time and can get partitioned (partitioning fragments the network into isolated sub-nets called partitions or components) and reconnected in a highly unpredictable manner. Mobile hosts in an *ad hoc* network can exchange information in areas that do not have preexisting infrastructure in a decentralized way (the control of a network is distributed among the hosts).

MANET systems are described as having the following characteristic properties: lack of central points, lack of a stable network infrastructure, frequent dynamic changes of a connection topology, limited computing resources and transmission security. They have many applications [52], such as military (e.g. on the battlefield), marine (e.g. communicating ships at dock), or emergency-related (e.g. establishing

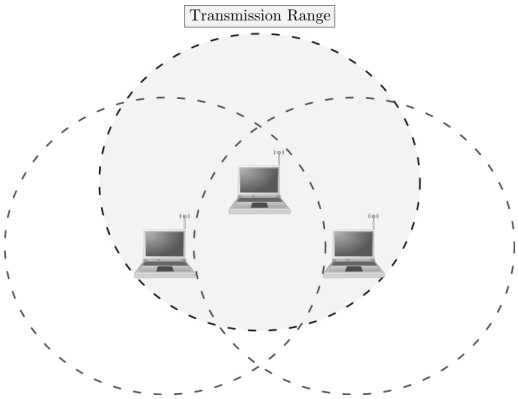

Fig. 4. An example of an *ad hoc* network

communication network for the rescue team). They may also be used to provide Internet connectivity in places that lack infrastructure, to create mobile conferences, to communicate automobiles on streets, or to establish personal wireless networks. Additionally, the potential of MANET-based solutions grows proportionally to the popularity of personal digital assistant (PDA) devices, netbooks and laptops, etc.

The so-called *peer-to-peer* systems (or P2P) have a similar processing model. In a peer-to-peer model the nodes, running a distributed service, may spontaneously join or leave the computing environment. Thus, node inaccessibility is considered a perfectly natural situation, not a failure, in either MANET or P2P model. Moreover, no stable (static) network infrastructure is assumed in both models. These properties make *ad hoc* systems (wired or wireless) and P2P systems very similar, and in consequence, the results of a research in one area are quite often applicable to the other area.

Ad hoc networks created in traditional, wired Internet, have also become popular recently. Such networks are termed "*ad hoc* grids" or "volunteer computing networks" (sometimes also "overlay networks"). The idea behind them is to establish a computing environment between many distributed Internet hosts, which are voluntarily activated by their users to share their processing power. The node becomes active in an *ad hoc* grid only during specific time periods, which were previously accepted by the node user (for example, in a SETI@Home project it is screen-saver time). Since each node is free to leave the environment at any moment (e.g. after the computer is turned off or disconnected from network), node unavailability is quite a normal situation. Because of this property and since failure and repair of nodes and links is unpredictable in both cases, such networks are also categorized as *ad hoc* systems.

Since in *ad hoc* networks the hosts move freely, disconnections may occur frequently and this may cause frequent network partitioning. Thus, services availability in MANETs is lower than in fixed networks and distributed systems. One possible approach to improve this, is to replicate services among nodes in an *ad hoc* network, but a replication creates a problem of inconsistency when partitioning occurs with nodes containing replicated services. Therefore, when designing a replication scheme for dynamic systems with partitioning, the competing and not independent goals of *availability*, *correctness* (in terms of consistency) and *partition tolerance* must be met. As it is known, it is impossible to achieve all three simultaneously [6]. Availability indicates web services and data accessibility with a system's normal function disrupted as little as possible, and correctness indicates accuracy of replicated nodes (with respect to the coherency protocol). So, the replication scheme in MANETs requires addressing the following three main issues: *replica allocation/relocation*, *location management* and *consistency management*.

Thus, the goals of the current research include the following: to analyze the efficiency of the various SOA replica allocation and relocation protocols in distributed *ad hoc* systems, to propose new localization protocols for the distributed *ad hoc* systems, and to formally and experimentally verify the proposed solutions. The research will result in creating a software platform for dynamic allocation and localization of the SOA service replicas in distributed *ad hoc* systems. The current work on the system (called: *SAS*) is summarized in Section 4.3, while the following Section 4.2 briefly describes the dynamic replication in distributed *ad hoc* systems.

4.2 Related Concepts and Contributions

Replica Allocation and Relocation. Replica allocation and relocation in the field of fixed networks and distributed systems has been an extensive research topic. It determines how many replicas of each web service or data item are created and to which nodes and how these replicas are allocated. In fixed networks, the optimal replication scheme depends only on the read-write pattern for each service or item and in many works the *communication cost* and *communication time* of replication scheme are used as the cost functions [53]. The communication cost is the average number of messages required for a read or write of a service or data. The communication time of an operation is its longest message path, so the communication time of a replication scheme is the average communication time of read or write of the service or data item. The problem of finding an optimal allocation scheme has been proved to be NP-complete (for different cost models) for both general static distributed systems [54, 55] and *ad hoc* networks [56]. Unlike traditional distributed systems, where the cost functions do not concern the

mobility of nodes, in MANETs the deployment of replicated SOA services should also reflect changes in a network topology.

Several strategies have been proposed for service or data replication schemes and replica allocation in MANETs — surveys of dynamic replication techniques for mobile *ad hoc* networks can be found in [57, 58]. In [59], three heuristic methods for replica allocation have been proposed. In all three proposed heuristics, replicas are relocated in a specific period (called *relocation period*) and replica allocation is determined based on the access frequency from each node to each item and additionally in two methods (*DAFN* and *DCG*) on the current network topology. However, all these protocols focus only on improving the service availability (in case the network changes) and static access frequencies. Therefore, algorithms, which can adapt the replica allocation scheme to the topological changes as well as to access requests distribution changes has been proposed in [60]. The *Vienna Dynamic Web Service Replicator* [61], in turn, is aimed at the problem of flexible replication and synchronization mechanism for stateful Web services in *ad hoc* networks.

Replica Location Management. The location management or location tracking problem in *ad hoc* networks is to gather information on the state and location of each node, i.e. to track the location of web services and their replicas. As pointed out above, the location changes in MANETs are caused by replica allocation/relocation and unpredictable network topology changes. Therefore, dedicated techniques to manage location of replicas to efficiently forward access requests in such a dynamic environment have been proposed in the literature, e.g. [62, 63].

Consistency Management. Mobility and arbitrary topological changes in *ad hoc* networks create a problem of maintaining consistency if replicas exist in the system, particularly when partitioning occurs. In such a case, there is a trade-off between availability and correctness — availability allows all nodes to process computation, even when partitioning occurs (but this may compromise correctness). On the other hand, correctness requires suspending operation processing in all but one partition and forwarding updates at reconnect (but such an approach compromises availability).

Generally, the strategies that have been proposed for consistency management in partitioned networks can be classified into four classes [4]: pessimistic strategies, optimistic strategies, syntactic approaches and semantic approaches. The first two concern the trade-off between availability and correctness. *Pessimistic strategies* preserve consistency by limiting availability, and *optimistic strategies* do not restrict availability. The two other classes concern correctness. *Syntactic approaches* use one-copy serializability (one-copy serializability is originally defined as: the concurrent execution of operations on replicated web services is equivalent to some

serial execution on a non-replicated web service) as a criterion for consistency correctness. *Semantic approaches* use the semantics of data items, transactions or web services in defining correctness criteria.

The highly dynamic network topologies and limited resources are the reasons why weaker consistency protocols with only probabilistic guarantees have been mainly proposed for use in MANETs [64, 65, 66]. On the other hand, if it can be assumed that a group of collaborating nodes in an *ad hoc* network can be partitioned and that partitions heal eventually, it is possible to develop reliable dissemination (broadcast) protocols, which can support a consensus protocol [67]. The assumption that it does not allow any partition to be permanently isolated, is called *MANET Liveness Property* [68]. Thus, in the case of an *ad hoc* network with a liveness property, it is possible to achieve strong (that are not probabilistic) consistency guarantees. Verification of the liveness property of wireless *ad hoc* networks based on selected mobility models can be found in [69, 70].

4.3 Service-Oriented *Ad Hoc* System

Functional Requirements. The *Service-oriented Ad hoc System* (*SAS*) should allow creating, publishing and accessing replicated web services in *ad hoc* networks. It can use UDP datagrams as a basic communication protocol, can work within 802.11 wireless networks (in *ad hoc* mode) and within wired Internet networks. It is assumed that web services should be implemented and are published, in a RESTful manner, with the use of HTTP as an access protocol and URI to identify them. Every *SAS* service must implement a given programming interface, which defines standard REST methods and two special methods used to initialize a newly created replica with an up-to-date initial state. All web services are dynamically replicated and the system allows locating available services and replicas.

SAS Architecture. The architecture of *SAS* system is presented in Figure 5. There are two *SAS* layers, called *Network* and *Replication*, and they are placed just above the ISO/OSI Transport layer, at the very bottom of the ISO/OSI Application layer. The *Network* layer (consisting of two modules, called `Network Communication` and `Routing`) is responsible for transporting messages in the *SAS* system. It provides the following three communication primitives: *neighbor broadcast* (the message is broadcast only to the neighbor nodes, and not propagated further), *flood broadcast* (the message is propagated to all nodes), and *unicast* (the message is sent to only one receiver). Each of them uses the connectionless UDP protocol for sending messages. Since the unicast operation requires routing, one of the layer's modules, the `Routing` module, implements AODV (Ad Hoc On-Demand Distance-Vector) [71, 72], a popular routing protocol for mobile *ad hoc* networks. The task of the *Replication* layer is to control various aspects of SOA services

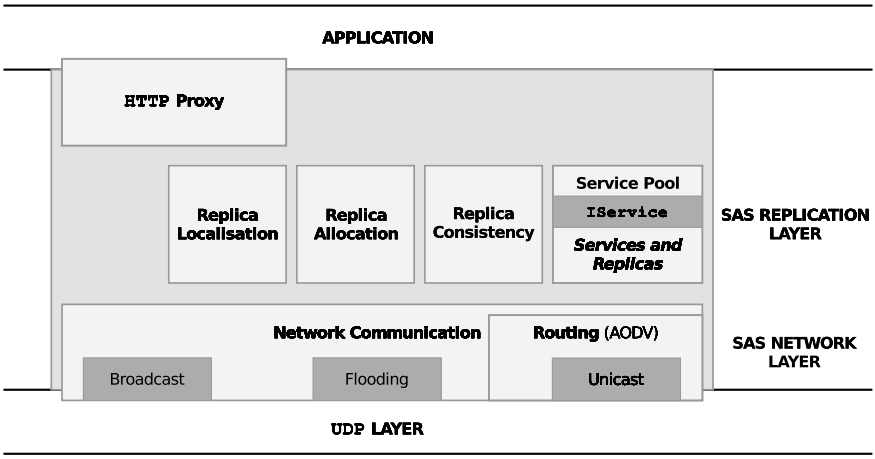

Fig. 5. The *SAS* system architecture

replication. Replication in the *SAS* system focuses on the following four problems: allocating and creating new replicas, activating newly created replicas, localizing replicas in the system, and controlling consistency among replicas according to a given consistency model. Each of these four problems is implemented in one of the four separate modules, called the `Replica Allocation` module, the `Service Pool` module, the `Replica Localiser` module, and the `Replica Consistency` module, respectively.

The Network Layer. The implementation of the *Network* layer, the bottom system layer, is based on the use of FIFO (Fist-In-Fist-Out) queues. For each module the *Network* layer creates a dedicated FIFO queue, which stores messages addressed to this module. Upon arrival, a message is inserted into an appropriate queue by the *Network* layer, and later it is removed and processed by the module with the use of this queue. All insertions and removals preserve the First-In-First-Out order. The queues are identified by integer numbers, assigned to them during module initialization. When initializing, each module calls the *Network* layer to get its queue `ID`, and then uses it to execute any of the communication primitives described earlier. The *Network* layer's internal structure is shown in Figure 6. After receiving the message from the UDP subsystem, the receive thread reads the destination module's queue `ID` (one of the message fields) and stores the message in the right queue. The addressee is then able to consume and process the message.

The structure of messages exchanged in the *SAS* system is presented in Table 2. Three message types have been defined, and these are: *unicast* (one-to-one send, type `0`), *neighbor-broadcast* (only to direct neighbors, type `1`) and *flood-broadcast*

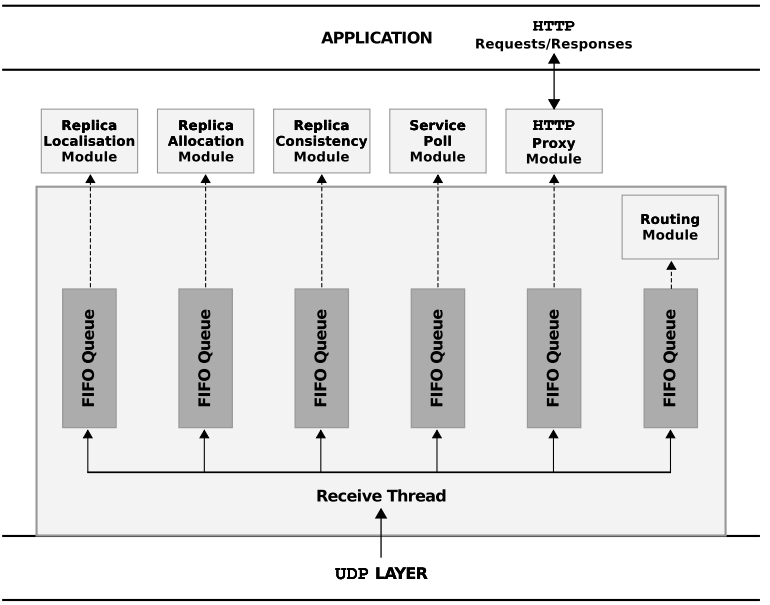

Fig. 6. The *Network* layer's internal structure

Table 2. The network message structure

Start byte	Length [bytes]	Value
0	1	Protocol version
1	1	Message type
2	1	Receiver port
3	2	Sender sequence number
5	4	Receiver IP address
9	4	Sender IP address
13	0-65480	Data

(network-wide, type `2`). The third field, *Receiver port*, contains an `ID` of the receiver module's FIFO queue. The fourth field, *Sender sequence number*, indicates a unique identifier for each sender's message (used in flood-broadcast messages). IP addresses of the sender and receiver node are stored in the next two fields. The last field, *Data*, encapsulates the original message that was sent by the sending module.

The Replication Layer. The task of the *Replication* layer is to control various aspects of replication of the SOA services. Replication in the *SAS* system focuses

on the following four problems: allocating and creating new replicas, activating newly created replicas, localizing replicas in the system, and controlling consistency among replicas according to a given consistency model. Each of these four problems is implemented in one of the four separate modules, called `Replica Allocation`, `Service Pool`, `Replica Localiser`, and `Replica Consistency` module, respectively.

Replica Allocation Module. This module creates and destroys service replicas at remote nodes when needed, and controls the number of replicas in the system, by keeping it between the predefined *minimum* and *maximum* thresholds. If the number of replicas exceeds the *maximum*, then the `Replica Allocation` module of the manager's node requests other nodes to delete the surplus replicas. Conversely, if the number of replicas falls below the *minimum* threshold, then the module requests other nodes to create the additional replicas.

Module operation is as follows. The `Localiser` module periodically provides the `Allocation` module with the list of all services running in the system, together with the list of the IP addresses of all the nodes that host a given service. Then, for each service the `Allocation` module checks if the local node is the manager for this service (the node with the smallest IP address becomes the manager). If it is, it determines the *minimum* and *maximum* parameters and then starts creating (or deleting) service replicas until their number falls into this interval. The whole process starts again when a new list is received from the `Localiser` module.

Replica Localiser Module. The `Localiser` module is responsible for finding service replicas in the *SAS* system. The replica lookup process is based on sending periodic broadcasts that inform about the set of services hosted by the broadcasting node. The `Localiser` module constantly collects and analyzes this information, and therefore it is able to determine to which node requests should be sent.

The module operates as follows. As it has been mentioned, each node periodically broadcasts a message that informs which services (or their replicas) it currently runs. This message is forwarded to all nodes in the system (*flood broadcast*), so that each node has full information about the replica placement. This data is retained in a hash table, which has the service name as a key and the IP address list as a value. The more advanced lookup mode assumes the use of the WRDL (*Web Resource Description Language*) language to describe services, and a rich set of criteria for service lookup (this solution is still being developed). The results generated by the `Localiser` module are also useful for two other modules: `HTTP Proxy` and `Replica Consistency`. The former asks the `Localiser` module for an IP address of any node that can process a given HTTP request and subsequently forwards the request to such node. The latter is given the IP address list of all the nodes that host the replicas of a specific service. It uses that list to execute replica initialization and consistency protocols.

Replica Consistency Module. This module deals with two problems related to the consistency of replicas. The first is to maintain consistency among the group of service replicas, according to a given consistency model. The second problem is to initialize a newly created replica with an up-to-date initial state, and to do it without violating consistency.

The module works in the following manner. In order to keep replicas in a consistent state, the `Consistency` module intercepts each request before it reaches the application. Such interception allows checking if the request delivery should be delayed until the consistency conditions are met. The implementation of the state initialization procedure is based upon the notion of "getters" and "setters". It is assumed that each service replica implements two special methods. One of them is `GetState()` and its purpose is to read the state from one of the already existing replicas. The second method is called `SetState()`, and it is used to write the state that was obtained with `GetState()` to a newly instantiated replica.

HTTP Proxy Module. The `HTTP Proxy` module serves as the system's front end, exposed to external client applications. It communicates clients with the replicated SOA services offered by the *SAS* system.

After receiving an HTTP request from a client, the `HTTP Proxy` module first extracts the service name from the request's URL (the URL should have the following form: `http://localhost:PORT/service-name/resources/`). Then it asks the `Localiser` module for an IP address of any node that hosts a replica of such a service. Depending on the result, the module either forwards the HTTP request to an appropriate node or responds with the HTTP 404 ("Service Not Found") error.

Implementation. The *SAS* system has been implemented in the *.NET Compact Framework*, so it can be run on versatile software platforms, including *Microsoft Windows*, *Windows Mobile* and *Linux*, and on almost every hardware platform (in particular, on PDA devices). Thus, in order to compile and run the system only the *.Net Compact Framework* libraries must be available.

4.4 Future Work

Our prototype system achieved its basic functionality. However, its layered and modular architecture allows further enhancements to be applied easily. In the next step, we plan to improve the reliability of communication protocols and the effectiveness of the localization protocols (particularly in relation to the dynamic replica allocation schema used). We also plan to implement strong consistency guarantees for networks with the *MANET liveness property* (additionally to the

current optimistic strategies). Finally, the proposed solutions will be formally verified (consistency protocols) and experimentally evaluated with the use of the *SAS* system and an elaborated simulator of *ad hoc* networks.

5 Conclusions

Since it is impossible to guarantee at the same time all the expected features of replication — availability, atomic consistency, and partition tolerance — a hybrid mechanism has been proposed. This mechanism can ensure strict consistency in a group of tightly integrated hosts — replicated state machine — assuming that no messages are lost. If this assumption is not valid, optimistic replication can be used. A severe case is a mobile *ad hoc* network where partitioning and subsequent link failures are quite common. In order to alleviate the problem of availability, a dynamic replication mechanism is considered, which, besides the consistency protocol, addresses the problem of replica allocation and relocation.

Acknowledgments. This work has been partially supported by the Polish Ministry of Science and Higher Education within the European Regional Development Fund, Grant No. POIG.01.03.01-00-008/08.

References

1. Misra J. Axioms for memory access in asynchronous hardware systems. ACM Trans. Progr. Lang. Syst. (TOPLAS) 8, 1, 142–153, 1986.
2. Lamport L. On interprocess communication, parts I and II. Distributed Computing 1, 2, 77–101, 1986, `http://cite-seer.ist.psu.edu/lamport85interprocess.html`.
3. Herlihy M.P., Wing J.M. Linearizability: A correctness condition for concurrent objects. ACM Trans. Progr. Lang. Syst. 12, 3, 463–492, 1990.
4. Davidson S.B., Garcia-Molina H., Skeen D. Consistency in partitioned networks. ACM Comp. Surv. 17, 3, 341–370, 1985.
5. Brewer E.A. Towards robust distributed systems (abstract). Proc. 19th Annual ACM Symp. on Principles of Distributed Computing PODC'00, New York, NY, 7, ACM, 2000.
6. Gilbert S., Lynch N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News 33, 2, 51–59, 2002.
7. Birrell A.D., Levin R., Schroeder M.D., Needham R.M. Grapevine: an exercise in distributed computing. Comm. ACM 25, 4, 260–274, 1982.

8. Stokes E., Weiser R., Moats R., Huber R. Lightweight Directory Access Protocol (version 3) Replication Requirements. RFC 3384 (Informational), Internet Engineering Task Force, 2002, http://www.ietf.org/rfc/rfc3384.txt.
9. Schneider F.B. Implementing fault-tolerant services using the state machine approach: A tutorial. ACM Comp. Surv. (CSUR) 22, 4, 299–319, 1990.
10. Powell D., Chéreque M., Drackley D. Fault-tolerance in Delta-4. SIGOPS Oper. Syst. Rev. 25, 2, 122–125, 1991.
11. Budhiraja N., Marzullo K., Schneider F.B., Toueg S. The primary-backup approach. Distributed Systems. S. Mullender (ed.), 199–216, ACM Press/Addison-Wesley, NY, 1993.
12. Guerraoui R., Schiper A. Software-based replication for fault tolerance. IEEE Comp. 30, 4, 68–74, 1997.
13. Defago X., Schiper A., Sergent N. Semi-passive replication. Proc. 17th IEEE Symp. on Reliable Distributed Systems, West Lafayette, Indiana, 43–50, 1998, http://www.cecs.missouri.edu/~joshi/ieeerds.html.
14. Albitz P., Liu C. DNS and BIND. O'Reilly, Sebastopol, CA, 2001.
15. Allen R., Lowe-Norris A.G. Active Directory. O'Reilly, Sebastopol, CA, 2003.
16. Winer D. XML-RPC specification. 1999, http://www.xmlrpc.com/spec.
17. Cabrera L.F., Kurt C. Web services architecture and its specifications: essentials for understanding WS-*. Microsoft Press, Redmond, WA, 2005.
18. Fielding R.T. Architectural styles and the design of network-based software architectures. Ph.D. dissertation, University of California, Irvine, 2000.
19. Richardson L., Ruby S. RESTful Web Services. O'Reilly, Sebastopol, CA, 2007.
20. HttpComponents. The Apache Software Foundation, http://hc.apache.org/.
21. OSGi — the dynamic module system for Java. OSGi Alliance, http://www.osgi.org/.
22. Birman K.P., Renesse V.R. Reliable distributed computing with the Isis toolkit. Wiley-IEEE Computer Society Press, Los Alamitos, CA, 1994.
23. Renesse R.V., Birman K.P., Maffeis S. Horus: A flexible group communication system. Comm. ACM 39, 4, 76–83, 1996.
24. Dolev D., Malki D. The Transis approach to high availability cluster communication. Comm. ACM, 39, 4, 64–70, 1996.
25. Amir Y., Stanton J. The Spread wide area group communication system. tech. rep., Department of Computer Science, Johns Hopkins University, Baltimore, MD, CNDS-98-4, 1998.
26. Hayden M. The Ensemble system. tech. rep., Department of Computer Science, Cornell University, Ithaca, NY, TR98-1662, 1998.
27. Miranda H., Pinto A., Rodrigues L. Appia, a flexible protocol kernel supporting multiple coordinated channels. Proc. 21st IEEE Int. Conf. on Distributed Computing Systems ICDCS'01, Phoenix, AZ, 707–710, 2001.
28. Fortika. EPFL, 2006, http://lsrwww.epfl.ch/crystall/.
29. Samoa. 2008, http://lsrwww.epfl.ch/samoa/.

30. The Spread toolkit. Spread Concepts LLC, 2006, `http://www.spread.org/`.
31. The JGroups toolkit. Red Hat, 2009, `http://www.jgroups.org/`.
32. Wysocki P., Wojciechowski P.T. CamlGroups: Design and implementation of the group communication system in Objective Caml. tech. rep., Institute of Computing Science, Poznań University of Technology, TR-ITSOA-OB2-1-PR-09-5, 2009.
33. CamlGroups. 2010, `http://www.cs.put.poznan.pl/pawelw/camlgroups/`.
34. Kobus T., Wojciechowski P.T. RESTGroups: Design and implementation of the RESTful group communication service. tech. rep., Institute of Computing Science, Poznań University of Technology, TR-ITSOA-OB2-1-PR-09-6, 2009.
35. RESTGroups. 2010, `http://www.cs.put.poznan.pl/pawelw/restgroups/`.
36. Fielding R.T., Taylor R.N. Principled design of the modern Web architecture. ACM Trans. Internet Technol. (TOIT) 2, 2, 115–150, 2002.
37. Fielding R., Gettys J., Mogul J., Frystyk H., Masinter L., Leach P., Berners-Lee T. Hypertext Transfer Protocol — HTTP/1.1. RFC 2616 (Draft Standard), Internet Engineering Task Force, 1999, `http://www.ietf.org/rfc/rfc2616.txt`.
38. Guerraoui R., Rodrigues L. Introduction to reliable distributed programming. Springer, Berlin, 2006.
39. Mena S., Schiper A., Wojciechowski P.T. A step towards a new generation of group communication systems. Proc. 4th ACM/IFIP/USENIX International Middleware Conference Middleware'03, Lecture Notes in Computer Science, M. Endler, D. Schmidt (eds.) 2672, 414–432, Springer, 2003.
40. Objective Caml. INRIA, 2010, `http://caml.inria.fr/`.
41. Biagioni E., Harper R., Lee P. A network protocol stack in Standard ML. Higher-Order Symb. Comp. 14, 4, 309–356, 2001.
42. Gawarkiewicz A., Przybyłowicz J., Witkowski M., Wysocki P., Wojciechowski P.T. Biblioteka komunikacji grupowej: Architektura systemu dla grup dynamicznych w modelu crash-stop. tech. rep. Institute of Computing Science, Poznań University of Technology, TR-ITSOA-OB2-1-PR-09-2, 2009.
43. Ekwall R., Urbán P., Schiper A. Robust TCP connections for fault tolerant computing. J. Inf. Sci. Eng. 19, 3, 503–516, 2003.
44. Chandra T.D., Toueg S. Unreliable failure detectors for reliable distributed systems. J. ACM 43, 2, 225–267, 1996.
45. Fischer M.J., Lynch N.A., Paterson M.S. Impossibility of distributed consensus with one faulty process. J. ACM 32, 374–382, 1985.
46. Lamport L. The part-time parliament. ACM Trans. Comp. Syst. 16, 2, 133–169, 1998.
47. Aguilera M.K., Delporte-Gallet C., Fauconnier H., Toueg S. Thrifty generic broadcast. Proc. 14th Int. Conf. on Distributed Computing DISC 2000, Toledo, Spain, 268–282, 2000.
48. Brzeziński J., Kalewski M., Libuda M. A short survey of basic algorithmic problems in distributed *ad hoc* systems. Pro Dialog A Sci. J. Pol. Inf. Proc. Soc. 1, 21, 29–46, 2006.

49. Misra S., Woungang I., Misra S.C. Guide to Wireless Ad Hoc Networks. Springer, London, 2009.

50. Aggelou G. Mobile *ad hoc* networks: from wireless LANs to 4G networks. McGraw-Hill Professional, New York, 2004.

51. Chakeres I. IANA allocations for Mobile *Ad Hoc* Network (MANET) Protocols. RFC 5498 (Proposed Standard), Internet Engineering Task Force, Mar. 2009, `http://www.ietf.org/rfc/rfc5498.txt`.

52. Frodigh M., Johansson P., Larsson P. Wireless *ad hoc* networking — the art of networking without a network. Ericsson Review 4, 248–263, 2000, `http://www.ericsson.com/ericsson/corpinfo/publications/review/2000_04/124.shtml`.

53. Jajodia S., Wolfson O. Distributed algorithms for dynamic replication of data. Proc. 11th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, New York, 149–163, ACM, 1992.

54. Milo A., Wolfson O. The multicast policy and its relationship to replicated data placement. ACM Trans. Database Syst. 16, 1, 181–205, 1991.

55. Cook S.A., Pachl J.K., Pressman I.S. The optimal location of replicas in a network using a read-one-write-all policy. Distrib. Comp. 15, 1, 57–66, 2002.

56. Jing Z., Yijie W., Xicheng L., Kan Y. A dynamic adaptive replica allocation algorithm in mobile ad hoc networks. Proc. IEEE Int. Conf. on Pervasive Computing and Communications Workshops, 65–69, Orlando, FL, 2004.

57. Padmanabhan P., Gruenwald L., Vallur A., Atiquzzaman M. A survey of data replication techniques for mobile ad hoc network databases. The VLDB J. 17, 5, 1143–1164, 2008.

58. Brzeziński J., Grzmiel J., Kalewski M., Libuda M., Olczak K., Piekarczyk M., Róg P. Analiza architektur i protokołów w istniejących środowiskach SOA dla rozproszonych systemów *ad hoc*. tech. rep., Sieć Naukowa Technologii Informacyjnych SOA, TR-ITSOA-OB5-5-PR-09-02, 2009.

59. Hara T. Effective replica allocation in *ad hoc* networks for improving data accessibility. Proc. IEEE Infocom 2001, 1568–1576, 2001.

60. Zheng J., Wang J., Lu X., Yang K. A dynamic adaptive replica allocation algorithm in mobile *ad hoc* networks. PerCom Workshops, 65–69, 2004.

61. Dustdar S., Juszczyk L. Dynamic replication and synchronization of web services for high availability in mobile *ad hoc* networks. Serv. Orient. Comp. Appl. 1, 1, 19–33, Apr. 2007.

62. Hara T. Strategies for data location management in mobile ad hoc networks. Proc. 11th Int. Conf. on Parallel and Distributed Systems ICPADS'05, Washington, DC, 147–153, IEEE Computer Society, 2005.

63. Lee H., Welch J.L., Vaidya N.H. Location tracking using quorums in mobile *ad hoc* networks. Ad Hoc Networks 1, 4, 371–381, 2003.

64. Rothermel K., Becker C., Hähner J. Consistent update diffusion in Mobile *Ad Hoc* Networks. tech. rep., University of Stuttgart: Collaborative Research Center SFB 627 (Nexus: World Models for Mobile Context-Based Systems), 2002/04, 2002.

65. Hayashi H., Hara T., Nishio S. Replica allocation considering data update intervals in *ad hoc* networks. IFIP TC6/WG6.8 Conf. on Mobile and Wireless Communication Networks (MWCN 2004), 131–142, Paris, 2004.
66. Guerraoui R., Hari C. On the consistency problem in mobile distributed computing. Proc. ACM Int. Workshop on Principles of Mobile Computing, 51–57, Toulouse, France, 2002.
67. Vollset E.W. Design and evaluation of crash tolerant protocols for mobile *ad hoc* networks. Ph.D. dissertation, University of Newcastle Upon Tyne, 2005.
68. Vollset E., Ezhilchelvan P. Design and performance-study of crash-tolerant protocols for broadcasting and reaching consensus in MANETs. 24th IEEE Symp. on Reliable Distributed Systems (SRDS 2005), 166–175, Orlando, FL, 2005.
69. Kosiba M. Właściwość żywotności bezprzewodowych sieci *ad hoc* i jej analiza w kontekście wybranych modeli ruchu. Master's thesis, Poznań University of Technology, 2009.
70. Brzeziński J., Kalewski M., Kosiba M., Libuda M. Verification of the liveness property of wireless ad hoc networks based on selected mobility models. tech. rep., Sieć Naukowa Technologii Informacyjnych SOA, TR-ITSOA-OB5-5-PR-09-07, 2009.
71. Perkins C.E., Royer E.M. *Ad hoc* on-demand distance vector routing. Mobile Computer Systems and Applications, IEEE Workshop on. Los Alamitos, California, 90–100, IEEE Computer Society, 1999.
72. Perkins C., Belding-Royer E., Das S. *Ad Hoc* On-Demand Distance Vector (AODV) routing. RFC 3561 (Experimental), Internet Engineering Task Force, 2003, `http://www.ietf.org/rfc/rfc3561.txt`.

Towards Automatic Composition of Web Services: Abstract Planning Phase

Mariusz Jarocki[1], Artur Niewiadomski[3], Wojciech Penczek[2,3], Agata Półrola[1], and Maciej Szreter[2]

[1]University of Łódź, FMCS,
Banacha 22, 90-238 Łódź, Poland
{jarocki,polrola}@wmi.uni.lodz.pl
[2]Institute of Computer Science, PAS,
Ordona 21, 01-237 Warsaw, Poland
{penczek,mszreter}@ipipan.waw.pl
[3]University of Podlasie, ICS,
Sienkiewicza 51, 08-110 Siedlce, Poland
artur@iis.ap.siedlce.pl

Abstract. The chapter proposes a method for converting the problem of automated composition of web services into the problem of building a graph of worlds consisting of formally defined objects. We present basic rules for defining ontologies for services execution. Automatic reasoning about sequences of service calls leads to satisfying a user's intention. The intention can be specified in a fully declarative language, without knowledge about services. In turn, the services are treated as independent "black boxes", realizing their activities regardless of the flow in which they occur. The intentions and the descriptions of the services are the only source of knowledge used to generate abstract plans. The above planning process is the first phase of an automated composition. The chapter presents also a tool implementing our composition algorithm and some experimental results.

Keywords: service oriented architecture, web services, composition, integration, automated reasoning, ontology

1 Introduction

The service oriented architecture (SOA) paradigm has been successfully implemented in web services technology. One can build applications which invoke functionalities having hidden specifications and developed independently by many

providers. Applications can reach complex goals using web services like procedure/method calls in the classical programming. However, because of a variety of available services, these complex goals can be satisfied by composing a great number of invoke flows.

In recent years there is a growing interest in automating the composition of web services. This can be explained by moving from experimental to real-word environments, with a significantly bigger numbers of more and more complex services. One of the problems hindering the development of automatic composition approaches is scalability. Since most existing composition methods work with concrete instances of web services, even a simple query requires taking all the instances of all the types of services into account. In our work, strongly influenced by the idea of Entish [1], we distinguish between an abstract and a concrete phase of the composition. Moreover, our approach uses no other semantic information than that contained in the ontology. Therefore, the services are seen as black boxes, and we do not take their internal structures into account. This is aimed at making registration of services much easier and feasible from the practical point of view.

1.1 Motivations

There are many Internet services nowadays. Several standards describe how services can be invoked (WSDL [2]), how they exchange information (SOAP [3]), how they synchronise their executions in complex flows (BPEL [4]), and finally how they can be found (UDDI [5]). However, there is still a lack of automatic methods for arranging and executing flows of services, which would use reasoning methods similar to human ones. The main problems are incompatibilities of inputs/outputs of services, and difficulties in comparing their capabilities and qualities. Two services can offer the same functionality, but this cannot be detected automatically without the unification of their interfaces, done by service providers.

Our automatic composition system is based on the following concepts:

— Web services provide their functionalities using stubs, generated from WSDL (or other) descriptions for both automatic composition systems and other systems used so far. We do not require a different interface for each of these forms of usage. However, we need a uniform semantic description of service types in order to build adapters (called *proxies*) translating specific interfaces of concrete services to a common one. The services handled this way can be used in automatic composition systems. Service registration, aimed to be easy to comprehend by a provider, involves building an appropriate proxy for the service which is being registered. In the future we are going to have the registration process based on interface descriptions both in WSDL and in the languages

which contain a semantic information, like OWL-S or Entish. This allows us to adapt a possibly wide class of existing web services.

— A complex client's goal is expressed in a fully declarative intention language. A flow of services which should be invoked to reach the goal (i.e., a *plan*) is built automatically, using no other information than these of the client's query and present in the ontology. The user does not need to know a priori any relations between services. He describes two worlds: the initial and the final one; the goal of the system consists in finding a way of transforming the former into the latter. The user specifies his intention using notions from the ontology, but the relations between these notions and the services are unknown to him.

— Applicability of systems for automatic composition can be seen in environments containing many services which offer the same functionalities, but are of different quality or of other features justifying searching for an optimal solution. To make such a choice, the services should be used in an "offer mode" (not involving irreversible changes of their internal states), and in an "execution mode". This can be reached using the mechanism of proxies.

— Our automating composition consists of three phases. In the first phase, called *abstract planning* or *planning in types*, we create an *abstract plan*, which shows sequences of service types whose executions possibly allow to accomplish the goal (expressed in the query). The second phase makes these scenarios "concrete", which consists in replacing the types of services by their concrete instances. The last phase consists in supervising the execution of the optimal run, with a possibility of correcting it if necessary (i.e., in the case of a service failure). The intermediate results, resulting from the above phases, are to be presented in formats which enable using them in other tools. The tool presented in the chapter can export its results to the BPEL format.

Our further plans of developing the approach involve applying the well-known methods of model checking and theorem proving. We are going to use them mainly (but not only) for building mechanisms of automatic reasoning. To this aim, we change the model of services from functional (meeting the remote procedure call paradigm) to an automata-like, which is more convenient in our research.

The rest of the chapter is organised as follows: Section 2 presents the related work. In Section 3 we sketch the main ideas of our approach. The notions behind them are introduced in Section 4 and illustrated by an example in Section 5. Section 6 shows an algorithm for abstract planning and defines the resulting graph. Section 7 presents an implementation of the abstract planner, illustrated by the results for the example of Section 5. Section 8 contains final remarks and sketches directions for future work.

2 Related Work

There is a significant number of papers dealing with the topic of web services composition [6, 7, 8, 9, 10, 11]. Some of these works consider static approaches where flows are given as a part of the input, while others deal with dynamically created flows. One of the most active research areas is a group of methods referred to as AI Planning [6]. Several approaches use Planning Domain Definition Language (PDDL [12]). Another group of methods uses rule-based planning. Composite services are generated from high-level declarative descriptions. Composition rules describe the conditions under which two services are composable. The information obtained is then processed by some designated tools. The composition is described in the Composite Service Specification Language (CSSL). The project SWORD [13] uses an entity-relation formalism to specify web services. The services have pre- and postconditions; a service is represented as a Horn rule denoting that the postconditions are achived when the preconditions are true. A rule-based expert system generates a plan. Another methodology is the logic-based program synthesis [8]. Definitions of web services and user requirements, specified in DAML-S/OWL-S [14], are translated to formulas of Linear Logic (a propositional logic; LL in short) so that the descriptions of web services are encoded as LL axioms, while a requirement is specified as a LL sequent to be proven by the axioms. Then, a theorem prover determines if such a proof exists. If so, a process model is extracted from it, and a composite service is described either in DAML-S/OWL-S Service Model or in BPEL4WS. While the approaches described above are automatic, there are also semi-automatic methods assuming human assistance at certain stages [15]. Some approaches are based on specifying a general plan of composition manually; the plan is then refined and updated in an automatic way.

Although most of the approaches presented above are oriented to practice, the presented tools have neither become established frameworks nor standards. Most of them are not even publicly available. Moreover, it seems that after a significant activity in the beginning of the current decade, the approach currently slows down. Some works are developed for relatively small and closed domains, and the papers present a preliminary work without delivering a complete solution.

3 Main Ideas

Most problems with automatic composition occur when trying to compose services coming from different providers, as there is no unified semantics for the functionalities which are offered. This problem can be solved by a unified dictionary of notions/types describing inputs and outputs of services (similarly to the Entish project [1]). The problem of service composition is then transformed into a problem

of functions search and composition able to create, from a given set of data (documents in the Entish approach), the set of data (documents), which is requested by the user.

In order to deal with more complicated examples from the real world, one can create a model which allows to describe not only document processing, but also an arbitrary process causing changes in a uniformly described data space. Such a data space will be called a *world*. Worlds can be described by an *ontology*, i.e., a formal representation of the knowledge about them. In order to ensure an easy integration with other current solutions, the ontologies used in our project are defined in the OWL language. The concepts used to model individuals define a hierarchy of types as well as classes and objects.

In our approach, services are understood as transitions between worlds, or functions which transform these worlds. The aim of a composition process is to find a path in the graph of all the possible transitions between worlds which leads from a given initial state to a given final state. The descriptions of the worlds in the specification of a task do not need to be complete.

As in most cases there are many paths which satisfy user's request, there is a need for a technique which allows to evaluate them in an automatic way in order to choose the most optimal one. The evaluation is based on features of services which take part in modifying worlds.

The ontologies used in the approach collect the knowledge not only about the structure of worlds, but also about the ways the worlds can be transformed, i.e., about services. Services are organised in a hierarchy of classes, and described both at the classes level (by a declaration of what all the services of a given class do — in the project such a pattern of behaviour is referred to as *abstract service*), and at the level of objects (a *concrete service* in the project). The description of services involves, besides specifying input and output data types, also the declarations of change effects on a world, i.e., creating, removing and modifying objects.

A user describes its goal in a unified language defined by the ontology. The knowledge contained in the ontology is sufficient for the automatic composition of services which can satisfy the user's request. In Section 7 we show that our approach enables to model automated composition based on matching input and output types of services, characteristic of the Entish project [1].

In contrast to existing solutions [8, 12, 13], we do not assume that the description of the world comes from one source language with a limited semantics. The intention is to enable the planner to cooperate with various service description languages like WSDL [2], OWL-S [14], or Entish [1], by using an intermediate language, which maps the elements of the languages' semantics to the uniform ontology, easy to understand and to reason on. The elements required in the planning, which are not expressible in these languages, are embedded in the ontology during the service registation process.

Our approach requires the hierarchy of services to be common for all the participants of the planning process, i.e., for both the service providers and the consumers interested in composing services into a complex flow. However, our aim is not to create any new world-wide standard, neither general (like UDDI) nor for a particular domain. This aim allows for an easy adaptation of services to be used in the context other than a simple invoking. We hope that this our approach is more likely to be accepted by the business community.

4 Basic Notions

We start with introducing the basic notions used in the chapter.

4.1 Classes and Inheritance

In order to model all the elements of our approach *OWL ontologies* are applied.

Definition 1 (OWL ontology). An *ontology* is a set of definitions of classes (ordered in an inheritance hierarchy), their instances and relations between them[1].

Definition 2 (Class). A *class* is a named OWL template, which defines names and types of attributes.

Inheritance. All the classes are ordered in a multiple inheritance hierarchy. The fact that a class B inherites from a class A means that B contains both all the attributes of A and the attributes specified in its own definition. The class B is called *subclass*, *child class* or *derived class*; the class A is called *superclass*, *parent class* or *base class*. Contrary to object-oriented programming languages, names of attributes are unique in the whole name space (instead in the class definition only). Attribute encapsulation and overloading is not supported[2].

A class is called *abstract* if instantiating it (i.e., creating objects following the class definition) is useless in the sense that the objects obtained in this way do not correspond to any real-world entity. Abstract classes are used, among others, for defining formal parameters of services, which allow an actual value of such a parameter to be any object instantiating an arbitrary non-abstract subclass of the given class. Moreover, on the top of the inheritance hierarchy there is an abstract base class `Thing` with no attributes[3].

[1] OWL ontology definition.

[2] The last two restrictions follow from assuming OWL as the ontology description language.

[3] The rules of class inheritance, no formal definition of abstract class, and the common root of the inheritance tree are also taken from OWL.

Using Subclasses Instead of Instances of Classes. If using an object of a certain class is required in the planning process, then an object of an arbitrary subclass of this class can be used instead. This follows from the fact that the set of the attributes of this object is not smaller that the one of the class required.

4.2 Worlds

Assume that we have a set of objects O containing instances of classes defined in an OWL ontology.

Definition 3 (World and objects). The *universum* is the set of all the objects having the following features:

— each object is either a *concrete object* or an *abstract object*;
— each object contains named attributes whose values can either be other objects or be:
  - values of simple types (numbers, strings, boolean values; called *simple attributes*) or `NULL` (*empty value*) for concrete objects;
  - values from the set {`NULL`, `SET`, `ANY`} for abstract objects.

  If an attribute A of the object O is an object itself, then O is extended by all the attributes of A (of the names obtained by adding A's name as a prefix). Moreover, when an object having an object attribute is created, its subobject is created as well, with all the attributes set to `NULL`[4].
— each simple attribute has a boolean-valued flag `const`.

A *world* is a set of objects chosen from the universum. Each object in a world is identified by a unique name.

By default each `const` flag is set to `false`. If the flag of an attribute is `true`, then performing on the object any operation (service) which sets this attribute (including services initializing it) is not allowed (the value of the attribute is considered to be final).

Interpretation of the values of the attributes for abstract objects is as follows:

— `NULL` means that no value of the attribute has been set (i.e., the attribute has the empty value);
— `SET` means that the attribute has a nonempty value;
— `ANY` means that the state of the attribute cannot be determined (i.e., its value can be either `SET` or `NULL`).

The attributes are referred to by `ObjectName.AttributeName`.

[4]Such an approach in analogous as in C++, and different than in Java.

Definition 4 (Object state, world state).

— A *state of an object* o is a function V_{O} assigning values to all the attributes of o (i.e., is the set of pairs (`AttributeName`, `AttributeValue`), where `AttributeName` varies over all the attributes of o);
— a *state of a world* is a set of states of all its objects.

The worlds are modified by services. A formal definition of a service will be given after describing the reasoning methods about worlds.

There are two ways of adding a new object to the world:

— The object, with some of its attributes set to the values given, is created in the initial world according to the initial clause (Sec. 4.4).
— Adding the object results from invoking a service (Sec. 4.3).

An object can be removed from a world only by means of invoking a service.

In order to reason about worlds and their states we define the following two-argument functions (the second default argument of these functions is the world we are reasoning about):

— `Exists` — a function whose first parameter is an object, and which says whether the object exists in the world;
— `isSet` — a function whose first parameter is an attribute of an object, and which says whether the attribute is set (has a nonempty value);
— `isConst` — a function whose first parameter can be either an attribute or an object. When called for an attribute, the function returns the value of its `const` flag; when called for an object it returns the conjunction of the `const` flags of all the attributes of this object.

Some aspects of the above functions need to be explained. `Exists` can be used in the effect clause (an element of a query, see Definition 11) to express that an object which appears in the user's query as an element of the initial world should not be present in the final world. The function `isSet` aims at testing the initialization of attributes.

In the process of abstract planning, i.e., when values of object attributes are not known but we know whether an attribute or an object was modified by the services used before, one can only use the above three functions. Therefore, the abstract planner allows us only to judge whether the object of interest exists, and what the status of its attibutes (`NULL`, `SET` or `ANY`) is.

4.3 Services

The definition of a service is as follows:

Definition 5 (Service). A *service* is an object of a non-abstract subclass of the abstract class `Service`. A service contains (initialised) attributes, inherited from the base class `Service`. The attributes can be groupped into:

— *processing lists* (the attributes `produces`, `consumes`, `requires`);
— *modification lists* (the attributes `mustSet`, `maySet`, `mustSetConst`, `maySetConst`);
— *validation formulas* (the attributes `preCondition` and `postCondition`).

The grammar and usage of these attributes are presented in Definitions 6-8.

A service modifies (transforms) a world, as well as the world's state. The world to be transformed by a service is called its *pre-world* (*input world*), while the result of the execution is called a *post-world* (*output world*). Modifying a world consists in modifying a subset of its objects. The objects being transformed by one service cannot be modified by another one at the same time (i.e., transforming objects is an atomic activity). A world consisting of a number of objects can be transformed into a new state in two ways (services which create new objects are not taken into account):

— by a service which operates on a subset of its elements;
— by many services which operate concurrently on disjoint subsets of its elements.

Definition 6 (Processing lists). We have the following *processing lists*:

— `produces` — a list of named objects of classes whose instances are created by the service in the world resulting from its execution;
— `consumes` — a list of names of classes whose objects are taken from the world when a service is invoked, and do not exist in the world resulting from the service execution (the service removes them from the world);
— `requires` — a list of names of classes whose instances are required to exist in the current world to invoke the service and are still present in the world resulting from its execution.

The elements of each of these lists are of the following form

```
<formal parameters> ::= <formal parameter>:<parameter class> |
                        <formal parameter>:<parameter class>;
                        <formal parameters>
```

The structure of the lists is similar to the lists of the formal parameters of the procedures. Names of objects are unique in a description of a class of services/an object of a service class. The formal parameters from these lists define an alphabet for modification lists and validation formulas.

Definition 7 (Modification lists). The *modification lists* are as follows:

— `mustSet` — a list of attributes of objects occurring in the lists `produces` and `requires` of a service, which are obligatorily set (assigned a nonempty value) by this service;
— `maySet` — a list of attributes of objects occuring in the lists `produces` and `requires` of a service, which may (but not must) be set by this service;
— `mustSetConst` — a list of attributes of the objects which occur in the lists `produces` and `requires` of a service, which are obligatorily set as being constant in the worlds after executing this service;
— `maySetConst` — a list as above, but of the attributes which may be set as constant.

Each of the above four lists is generated by the following grammar:

```
<attribute spec> ::= <formal parameter>.<attribute name> |
                     <formal parameter>.*
<attribute list> ::= <attribute spec> |
                     <attribute spec>; <attribute list>
```

In the process of abstract planning, each attribute from the list `mustSet` is considered to have the value `SET` (the function `isSet` called for this attribute will return the value `true`), whereas each attribute from the list `maySet` is seen as having either the value `SET` or the value it had before executing the service.

The expression `objectName.*` used in the above lists is understood as "all the attributes of the object `objectName`". It is the only way of referring to the attributes which cannot be listed directly in a given moment, due to the fact that the processed object can be of a subclass of the class specified in the list (so have a bigger set of attributes).

The attributes of the objects appearing in processing lists which do not belong to the union of lists `mustSet` and `maySet`, are not changed when the service is called.

Definition 8 (Validation formulas). The *validation formulas* are as follows:

— a `preCondition` is a propositional formula which describes the condition under which the service can be invoked. It consists of atomic predicates over the names of objects from the lists `consumes` and `requires` of the service and over their attributes, and is written in the language of the propositional calculus (atomic predicates with conjunction, disjunction and negation connectives). The language of atomic predicates contains comparisons of expressions over attributes with constants, and functions calls with object names and attributes

as arguments. In particular, it contains functions calls `isSet`, `isConst` and `Exists`[5].

— a `postCondition` is a propositional formula which specifies the conditions satisfied by the world resulting from invoking the service. The formula consists of atomic predicates over the names of objects from the lists `consumes`, `produces` and `requires` of the service and over their attributes. To the objects and attributes one can apply pseudofunctions `pre` and `post` which refer to the state of an object or an attribute in the input and the output world of this service, respectively. By default, the attributes of objects listed in `consumes` refer to the state of the pre-world, whereas these in `produces` and `requires` — to the state of the post-world.

The validation formulas are built in a way similar to the expressions in high-level programming languages. However, currently (for abstract planning) we use their reduced forms which are DNF formulas (i.e., formulas in a disjunctive normal form), with atomic predicates being (possibly negated) calls of the functions `isSet`, `isConst` or `Exists`. We assume that an arbitrary predicate, being a function over the set of objects and their attributes, can be transformed either trivially, i.e., by replacing arguments of functions by the conjunction of calls of `isSet` over attributes (and `not Exists` over objects in future versions), or in a more sophisticated way, which allows to reason about additional consequences in the resulting plan. We are going to deal with the above problems in our future work.

The following grammar builds a proper validation formula for the abstract planner:

```
<objectName>        ::= <objectName from consumes> |
                        pre(<objectName from requires>) |
                        post(objectName from requires>) |
                        <objectName from produces>
<objectAttribute>   ::= <objectName>.<attributeName>
<atomicPredicate>   ::= Exists(<objectName>) |
                        isSet(<objectAttribute>) |
                        isConst(<objectAttribute>) |
                        not <atomicPredicate>
<conjunction>       ::= <atomicPredicate> |
                        <atomicPredicate> and <conjunction>
<validationFormula> ::= <conjunction> |
                        <conjunction> or <validationFormula>
```

[5]Using `Exists` in `preCondition` is redundant w.r.t. using an appropriate object in the list `consumes` or `requires`. However, the future directions of developing the service description language mentioned in the final part of the chapter include moving modification lists to validation formulas.

It should be noticed that this grammar allows using `post` also in `preCondition`. However, evaluation of such predicates gives always the value `false`.

Service Types, Inheritance, Metaservices. Each class of services (service type) has certain features common for all the instances of this type. They are specified by the instance of the class called *metaservice* (i.e., an object of the service which describes the whole class of services). The (unique) name of a metaservice is derived from the name of the class of services, i.e., is of the form `_class_ServiceClass`.

A description of a concrete service should be understood as a list of differences or as narrowing the template introduced as the metaservice. More precisely, a concrete service can overload the processing lists of its metaservice by using derived classes of objects it works on. This is done by using, in an appropriate list, a formal parameter of the same name and of a restricted domain. Considering the modification lists, a concrete service can extend the lists of its metaservice only by declaring that it modifies attributes added by a derived class of parameters. The definition of a concrete service cannot be inconsistent with the definition of the metaservice.

Comparing with a base class of services, its child class can extend the sets of objects which are consumed, produced and required, using the appropriate processing lists to this aim (so, it is allowed to extend the subset of a world influenced by a service). It can also narrow the types of parameters in the processing lists, which is done by using the same names of formal parameters as in the lists of the base class. By default (when its lists are empty) a child class inherites the specification of the parent class. As the processing lists define the alphabet for the modification lists and the validation formulas, one should ensure the uniqueness of the formal parameters on all the levels of inheritance. The modification lists can be extended in a similar way (i.e., by extending the sets of attributes which are modified). The declarations of setting attributes are not restricted only to the attributes added by the child class in the lists `produces` and `requires` — it is also allowed to modify attributes not changed by the metaservice of the parent class.

Considering the validation formulas, each formula `preCondition` (`postCondition`, respectively) of a concrete service is a conjunction of the precondition (postcondition resp.) of the metaservice and the explicitely given precondition (resp. postcondition) of the concrete service. The case of inheritance is similar — in the derived class, a condition is conjuncted with the appropriate condition from the parent class.

As we mentioned before, the attributes of objects appearing in processing lists which do not belong to the sum of lists `mustSet` and `maySet` are not changed when the service is called. This, however, does not apply to the attributes added to modification lists in narrowed types introduced by concrete services. Potential

inconsistencies, resulting from concatenation of processing and modification lists in child classes of services, are treated as ontology errors[6].

Examples. In order to explain some less intuitive aspects of the definitions, we provide several examples. As the chapter deals with abstract planning only (not concrete planning), the (concrete) validation formulas in the examples are specified using a language which is not introduced in the chapter, but is similar to the language for formulating expressions in popular programming languages. The only parts of these formulas seen by the abstract planner are the variables used (e.g., the fact that an expression contains a variable `x`, but not the fact that the expression is `x>4`).

The first example shows the relation between the processing list of a metaservice and these of a concrete service.

Example 1. If the metaservice of a class `SelectWare` produces an object of a class `Ware`, then a concrete service (e.g. `electroMartSelect`) of the class `SelectWare` can be described as producing an object of the class `Ware` (which is the default) or as producing an object of its subclass (which requires an appropriate content of `produces`). Both versions, together with metaservice specifications, are presented below.

```
_class_SelectWare.produces   = { w:Ware }
electroMartSelect.produces   = { }
electroMartSelect.produces   = { w:HouseholdWare }
```

The next example illustrates the relation between the modification lists of a concrete service and its metaservice.

Example 2. If the metaservice of the class `SelectWare` sets the attribute `x.name`, where `x` is an object of the class `Ware` produced by the metaservice, then the service *`electroMartSelect`* of this class can narrow the type of the objects produced to a subclass of `Ware` called `HouseholdWare`, and can additionaly set the attribute `x.installed` introduced in the class `HouseholdWare` (and therefore not present in `Ware`).

```
_class_SelectWare.produces = { x:Ware }
_class_SelectWare.mustSet  = { x.name }

electroMartSelect.produces = { x:HouseholdWare }
```

[6]We do not assume an "expanded" inheritance hierarchy of services, contrary to hierarchy of types of their "objects". A suggested model of service inheritance is a three-level one: we find at the first level the class `Service` as a "carrier" of basic attributes, at the second level classes carrying additional *quality attributes* (see p. 194), and at the third level classes of services with definitions of their metaservices.

```
electroMartSelect.mustSet  = { x.installed }
```

In the above, the service `electroMartSelect` declares explicitly that it sets the attribute `installed`, and implicitly that it sets the attribute `name` (this follows from the definition of the metaservice).

Next we show some examples of atomic predicates and validation formulas:

Example 3.

— Atomic predicates: `x.number>0, not isSet(x.id)`
— `preCondition:` `x.number>0 and isSet(x.id)`
— `postCondition:` `not x.number = pre(x).number`

The next example illustrates relations between validation formulas.

Example 4. Let `postCondition` of the metaservice `SellingService` be of the form `x.id>0`, and `postCondition` of the metaservice of the derived class `WashingMachineSelling` be of the form `isSet(x.capacity)`. An instance `RacoonShop` of `WashingMachineSelling` (a concrete service) declares explicitely in its `postCondition` that `x.brand="Racoon"`. The above means that in the world resulting from invoking this service the conjunction of these two formulas, i.e. `isSet(x.capacity) and x.id>0 and x.brand="Racoon"`, will be satisfied.

Finally, we show an example of a validation formula and its transformation by the abstract planner:

Example 5. The formula `preCondition`

```
(isSet(x.id) and x.number>0)  or (isSet(x.id) and
x.capacity>0)
```

will be transformed in the abstract planner into

```
(isSet(x.id) and isSet(x.number)) or (isSet(x.id) and
isSet(x.capacity))
```

Additional Attributes of a Service. Service classes can contain additional *quality attributes* which are set while a service is executed. These attributes can be used in user queries (in an execution condition and in a quality function, see Section 4.4). Such attributes can be introduced, among others[7], by base abstract

[7]The decision whether the additional quality attributes are introduced by separate abstract "second-level" classes, or are defined at the level of non-abstract classes of services (with concrete representatives) is left to an ontology designer. The suggestions of the authors are presented in Footnote 6.

services which collect certain common features of services, e.g. "chargeable services" (assigned with the attribute of price) or "time-taking services" (assigned with timing interval). The above attributes are not used in the abstract planning process.

Modifying a World. Separating calls of single services is one of key concepts in our approach. A service processes a world, irrespectively of how many services brought it to the current form. The result of executing a service is a (possibly new) world with a new state.

Definition 9. A service U is *enabled* (*executable*) in the current state of a world S if:

— each object O from the lists `consumes` and `requires` of U can be mapped onto an object in S, of the class of O or of its subclass; the mapping is such that each object in S corresponds to at most one object from the above lists;
— for the objects in S which according to the above mapping are actual values of the parameters in `consumes` and `requires` the formula `preCondition` of U holds;
— the list `mustSet` of U does not contain attributes for which the objects which are actual values of the parameters have the flag `const` set.

Definition 10. A service U executing in a current world S produces a new world S' in which:

— there are all the objects from `s`, besides these which were actual values for the parameters in `consumes` of U;
— there is a one-to-one mapping between all the other objects in S' and the objects in the list `produces` of U, such that each object O from the list `produces` corresponds to an object in S' which is of a (sub)class of O;
— for the objects which according to the above mappings are actual values of the parameters in the processing lists the formula `postCondition` holds;
— in the objects which are actual values of the appropriate parameters the flags `const` of the attributes listed in `mustSetConst` of U are set, and the attributes listed in `mustSet` of U have nonempty values;
— assuming the actual values of the parameters as above, all the attributes of all the objects existing both in S and in S' which do not occur neither in `mustSet` nor in `maySet` have the same values as in the world S; the same holds for the flags `const` of the attributes which do not occur neither in `mustsetConst` nor in `maySetConst`. Moreover, all the attributes listed in `mustSet` or `maySet` which have nonempty values in S, have nonempty values in S' as well.

4.4 Queries

The task of the composition system is to provide sequences of service calls (*runs*) which can transform a world specified by a user into a desired world. Services which can be used for building the above sequences are listed in the repository of classes and objects. From the obtained sequences, the system can choose those satisfying the execution conditions specified by the user, and then select those optimal from the user's point of view (i.e., considering the criteria he gave).

A user defines a query by describing an initial state of the world and a final state required.

Definition 11 (Query). A *query* consists of the following elements:

— an *initial domain* — a list of named objects which are elements of the initial world. The form of the list is analogous to the form of the list `produces` in the description of a service;
— an *initial clause* specifying a condition which is to be satisfied by the initial world. The clause is a propositional formula over the names of objects and their attributes, taken from the initial domain. The grammar of the clause is analogous to the grammar of the `preCondition`;
— an *effect domain* — a list of named objects which have to be present in the final world (i.e., a subset the final world must contain);
— an *effect clause* specifying a condition which has to be satisfied by the final world. The clause is a propositional formula over the names of objects and their attributes from the domains defined above; references to the initial state of an object, if ambiguous, are specified using the notations `pre(objectName)` and `post(objectName)`, analogously to the language used in the formulas `postCondition` of services. The grammar of the effect clause is analogous to the grammar of the `postCondition`;
— an *execution condition* — a formula built over services (unknown to the user when specifying the query) in a potential run performing the required transformation from the initial world into a target world. During the construction of this formula, simple methods of quantification and aggregation are used;
— a *quality function* — a real-valued function over the initial world, the final world and services in a run, which specifies a user's criterion for valuating the quality of runs. The run with the smallest value of this function is considered to be the best one.

The last two parts of a query are used after finishing the abstract planning phase and the first phase of concrete planning, which adjusts types and analyses pre- and postconditions of concrete services. Due to the fact that in this work we deal with abstract planning only, the language of quantification and aggregation

used in execution conditions is in an early stage of development. Its complete formalization will be a subject of our future work.

On the abstract level, the initial clause and the effect clause are specified as DNF formulas over the predicates `isSet` and `Exists`. This means that (not taking into account the variants of worlds following from the disjunctive form) the query can be reduced to enumerating:

— objects in the initial world;
— objects in the final world, carrying an information which was present in the initial world (which is done by using the same names for formal parameters);
— objects that have to be removed from the initial world;
— attributes which must have nonempty values in the final world;
— attributes which must have empty values (must be null) in the final world.

In other words, a list of objects in the initial domain and the DNF form of the initial clause generates a number of alternative initial worlds whose states (values of attributes) are set according to (possibly negated) predicates occuring in the initial clause.

Examples of queries are presented below.

Example 6. User's query:

— Initial domain: empty set
  initial clause: `true`,
— Effect domain: `w:WashingMachine`,
  Effect clause: `w.id>0 and w.owner="Me" and (w.name="Raccoon 1" or w.capacity>5)`,
— Execution condition: `sum(s.price, s:ChargeableService) <= 1000`,
— Quality function: `w.capacity*10 - w.sum(s.price, s:ChargeableService)/100`.

On the abstract level the above query means that we request creating in the final world an object of the class `WashingMachine`, of nonempty values of attributes: either `id`, `owner` and `name`, or `id`, `owner` and `capacity`.

The next example shows an initial world generated from a query, and an "obligatory subset" of a target world.

Example 7. Assume a user's query is:

— Initial domain: `f:Fruits`
— Initial clause: `f.capacity=100 or (not isSet(f.capacity) and f.number=100 and f.weight=1)`
— Effect domain: `j:Juice`
— Effect clause: `j.capacity>0`

From the above query we create the following "working query" for the abstract planner:

— Initial domain: `f:Fruits`
— Initial clause: `isSet(f.capacity) or`
`(not isSet(f.capacity) and isSet(f.number) and isSet(f.weight))`
— Effect domain: `j:Juice`
— Effect clause: `isSet(j.capacity)`

which generates two alternative initial worlds — both consisting of a single object of the class `Fruit`, but of different states: the first one with a nonempty value of the attribute `capacity`, the second one with `capacity` set to null, but of nonempty values of `number` and `weight`. Moreover, the query generates a single "obligatory subset" of the final world, consisting of an object of the class `Juice` with nonempty value of `capacity`.

Definition 12 (Highlighted objects). An object is called *highlighted* w.r.t. a user's query if its name occurs both in the initial domain and in the effect domain of this query.

Definition 13 (Equivalent worlds). Two worlds S and S' are called *equivalent* if the sets of their highlighted objects are equal, and their complements are equal when names of objects are left aside (i.e., for each object `o1` from one set there is exactly one corresponding object `o2` from the second set, such that `o1` and `o2` are objects of the same class, and the values of all the attributes in both the objects are the same).

5 A Complete Example

In order to illustrate our approach we first present a simple ontology, in which abstract planning for our example can be done. For compactness of the description, the notation of triples (similarly as in the language RDF) is used.

5.1 Model for Services Objects

The classes whose objects are processed by services are defined as follows:

```
Ware       id       integer
Ware       name     string
Ware       owner    string
Measurable capacity float
```

```
Juice      extends  Ware, Measurable
Fruits     extends  Ware, Measurable
```

the class `Ware` models a thing which can be sold. The class contains the attributes:

— `id`, carrying the information "being concrete"; setting it to a nonempty value means that a concrete instance or "portion" of a given thing has been selected for further processing;
— `name` which describes what we are working on (but without referring to any concrete instance/portion in the real world);
— `owner` which specifies an owner of the thing;
— `Measurable` which, using Java terminology, is an interface extending the properties of a class with the capability to be measured (expressed for simiplicity by one attribute `capacity`).

`Juice` and `Fruits` are child classes of the classes mentioned above.

5.2 Services

A description of services is:

```
SelectWare  produces    w:Ware
SelectWare  consumes    null
SelectWare  requires    null
SelectWare  mustSet     w.name; w.owner

Selling  produces       null
Selling  consumes       null
Selling  requires       w:Ware
Selling  mustSet        w.id; w.owner
Selling  preCondition   not isSet(w.id) and isSet(w.name)
Selling  postCondition  w.owner!=pre(w).owner

FruitSelling  extends        Selling
FruitSelling  requires       w:Fruits
FruitSelling  mustSet        w.capacity
FruitSelling  postCondition  w.capacity>0

JuiceSelling  extends        Selling
JuiceSelling  requires       w:Juice
JuiceSelling  mustSet        w.capacity
JuiceSelling  postCondition  w.capacity>0
```

```
MakingJuice  produces       j:Juice
MakingJuice  consumes       f:Fruits
MakingJuice  mustSet        j.id; j.name; j.capacity
MakingJuice  preCondition   isSet(f.id) and isSet(f.name) and
                            f.capacity>0
MakingJuice  postCondition  isSet(j.id) and isSet(j.name) and
                            j.capacity>0
```

Below, the word "service" is used to denote a class of services or a metaservice.

— `SelectWare` is a "searching service", typical for e-market, which, given a query, tries to select from the offers of services (not only selling ones) these corresponding to the thing(s) requested. The service creates one or many objects, but does not set their `id`s, leaving this to services which precise the portion or item. It sets, however, the attribute `owner`, initializing it with the place where a ware of interest was found (usually a shop).
— `Selling` is a selling service, operating on an existing object of the class `Ware`, which is not precisely pointed to in the seller's offer (nonempty value of `owner`, but null `id`) but is of a precisely specified kind (nonempty value of `name`). The service declares setting the attribute `id` and changing the owner. The new owner is derived from the query.
— `FruitSelling` is a child service of `Selling`. It narrows the type of the objects processed to an instance of the class `Fruits`. Comparing with `Selling`, it declares additionally setting the attribute `capacity` with the positive amount of the ware sold. The value is derived from the query. `JuiceSelling` is similar to `FruitSelling`, besides the fact that the type of the objects processed is narrowed to an instance of the class `Juice`.
— `MakingJuice` is a service which makes juice from fruits: it removes an object of the class `Fruits` from the world, placing there an object of the class `Juice` instead. The portion of fruits must be "concrete" (with a nonempty value of `id`; we have also to know what kind and how much fruits we have, so a nonempty value of `name` and a positive value of `capacity` are required). The juice produced is also "concrete", i.e., of the list of nonempty attributes analogous to that for the input.

5.3 Query

The query specified as follows:

```
InitWorld    null
InitClause   true
EffectWorld  j:Juice
EffectClause j.id>0 and j.capacity=10 and j.owner="Me"
```

must be understood as "create a world which contains a concrete (`id>0`) instance of juice of a given volume and whose owner is the requester". The results generated by our abstract planner for the above set of data will be shown in Section 7.1.

6 Abstract Planning

The result of the abstract planning phase is an *abstract graph*.

The abstract graph is a directed multigraph. The nodes of the graph are worlds in certain states, while its edges are labelled by services. Notice that such a labelling carries an information about which part of an input world (node) is transformed by a given service (which is specified by actual values of the parameters in `consumes` and `requires` of the service), and which part of the output world (node) it affects (the lists `produces` and `requires` of this service). We distinguish some nodes of the graph: these which have no input edges represent alternative initial worlds, while these with no output edges are alternative final worlds.

A formal definition of the abstract graph is as follows:

Definition 14. An *abstract graph* is a tuple $GA = (V, V_p, V_k, E, L)$, where

— V is a subset of the set S of all the worlds;
— $V_p \subseteq V$ is a set of initial nodes;
— $V_k \subseteq V$ is a set of final nodes;
— $E \subseteq V \times V$ is a transition relation such that $e = (v, v') \in E$ iff $L(e)$ transforms the world v into v', where
— $L : E \longrightarrow U$ is a function labelling the edges with services.

6.1 A Composition Algorithm

Below, we show a forward-search-mode algorithm for automatic composition of abstract services. The input for the algorithm consists of:

— an (OWL) ontology containing the services and objects they operate on;
— a user query specifying an initial world and a final world;
— a natural number k which limits the depth of the search.

The value k bounds the maximal number of services which can be composed to transform the initial world into the final world. The algorithm builds an abstract graph which shows how the worlds are transformed by executing services: its nodes correspond to the worlds, and its edges to the services executed. The graph built

Require: a query $\varphi = (\varphi_S, \varphi_E)$, a maximal search depth k.
Ensure: a graph G
a queue Q
put the world described by φ_S into Q, assign them the depth 0, mark them as initial
while Q is nonempty
  get s from Q
  if s processed **then**
    continue;
  end if
  if s satisfies φ_E **then**
    mark s as final, mark s as processed
    continue;
  end if
  if depth of s greater than k **then**
    mark s as processed
    continue;
  end if
  for each service S defined in the system **do**
    //check whether the world s can be processed by S:
    if s does not contain certain objects from `S.consumes`, `S.requires` **then**
      continue;
    end if
    if s does not satisfy `S.preCondition` **then**
      continue;
    end if
    generate all the subsets of the objects in s satisfying `S.preCondition` and such that the attributes listed in `S.mustSet` have the flag `const` not set
    for all element s_p **do**
      create the world s'_p resulting from transforming s_p by S
      if exists $v \in G$ such that $v \equiv s'_p$ **then**
        add the edge $s_p \xrightarrow{S} v$ to G
        add v to Q
      else
        add to G the node s'_p and the edge $s_p \xrightarrow{S} s'_p$
        add s'_p to Q
      end if
    end for
  end for
  mark s as processed
end while

Fig. 1. Computing an abstract graph of services

by the algorithm presents all the possible scenarios for transforming the initial world into the final world using at most k services.

For a better readability we present a basic version of the algorithm, without optimisations (Fig. 1). For simplicity of the description, we distinguish in the query a part referring to an initial world (φ_S) and a part referring to a final world (φ_E).

7 Implementation

The abstract planner (see Fig. 2) is a tool which performs the first phase of web services composition, i.e., abstract planning (planning in types). Our tool can be accessed via a graphical user interface, or from a Java API.

As it has been described before, a state of a world is given by a set of objects of classes defined in an ontology, and of attributes set in an appropriate way. A composition of services consists in searching for sequences of services which can transform a world which (together with its state) is specified as an initial one, into a world satisfying some conditions requested. In our implementation:

— ontologies are modelled using the OWL language and the Protege environment;

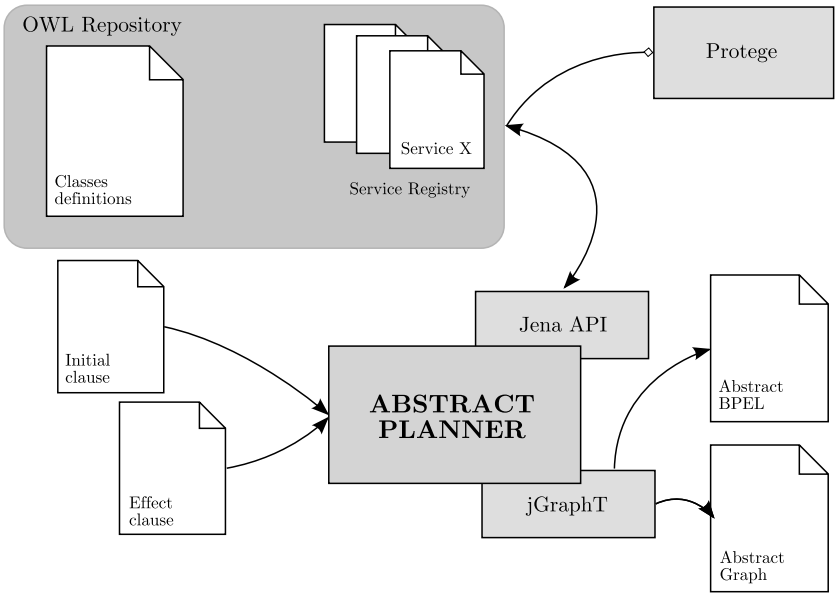

Fig. 2. A structure of the application

— OWL ensures a convenient way to define a hierarchy of types which are either objects of classes derived from the class `Service` (representing services), or objects which do not inherit from the above class (modelling items to be processed by services);
— conditions on the input and on the output world of each service are specified in the AtLa[8] language using attributes of services;
— the ontology contains both types of services and concrete instances of services.

An input to the composition system is a query, in which the user specifies an initial world Wb and a final world Wf, using the QLa[9] language.

The composition is a two-stage process. The first stage builds an abstract graph which describes possible solutions using types of services (and not their concrete instances). We distinguish two worlds (together with their states): a start world (at which the procedure starts) and an end world (a termination condition which means that the user request is satisfied). Although the algorithms operate in BFS mode, they can also work as a forward search (in such a case the start state is given by Wb, and the termination condition is given by Wf) or as a backward search (the start state is then given by Wf, and the termination condition — by Wb).

The program is implemented in Java, using the following components:

— the graph library jgraphT (representation of graphs and operations on them);
— the module jgraph (graphs visualisation);
— the parser ANTLR of the AtLa and QLa languages;
— Jena library (accessing OWL files generated by the Protege tool).

In order to provide a nice presentation of the results, both abstract and concrete graphs (i.e., graphs with concrete instances of services, not described in this work) are optionally represented in BPEL. This enables using visualisation and processing tools designed for graphs described in that language. As the abstract graph models directly the elements represented in BPEL (service calls, concurrent executions of services), transforming it to the BPEL format involves only simple translations of its structures into XML descriptions of BPEL constructions.

The approach presented enables also an integration with the Entish project [1]. In the project mentioned, the composition of services is based on matching their input and output data types. A service is understood as a function which creates a set of objects (documents) of precisely specified types, starting from another set of documents. A service can be executed if it is provided with the objects of the appropriate types. The above objects can be obtained in the following ways:

[8] *ATtribute LAnguage.*
[9] *Query LAnguage.*

— an object of the appropriate type "is in the user interface" (was either placed there by the executions of other services, or introduced there by hand);
— such an object was created by a service executed just before the current one.

If some objects required to execute a service are missing, then a user has to create them "by hand".

The above composition model can be transformed to the one proposed in the chapter in an easy way. The mechanisms of hierarchy of types, validation formulas and modification lists are not used, as there is no such mechanism in Entish types space. The services read from the Entish repository are added to our ontology, augmented with "our" processing lists (the input types of a service are used to build the list `requires`, the output types — the list `produces`, the list `consumes` is not used). The system of types of objects not being services is simplified (Entish inheritance is not taken into account). The problem of the missing documents which are to be created by an interaction with the user can be solved by using the services which produce these documents (of the empty lists `consumes` and `requires`).

The transformation described above is implemented in the current version of our tool.

7.1 Using the Abstract Planner

The current version of the application requires installing the Java Runtime Environment (in a version at least 1.6). After running the planner, one must load a file containing an ontology, or to import an Entish repository (menu *File*). One can also use a built-in generator of examples (menu *Examples*). The ontology loaded is shown in both a graphical and text form in the bookmark *ClassHierarchy* (Fig. 3). The bookmark *Services* enables to see the details of the descriptions and the internal representations of services (Fig. 4).

The main part of the tool is the bookmark *Composition* presented in Figure 5. The panel in the left-hand side enables specifying the parts of a query which describe an initial and a final world, setting the options required, and then running the composition process.

The result of composition (a graph) is displayed in the window on the right. The *to BPEL* button enables exporting the graph to the BPEL format. The tool generates one or more BPEL files describing the results, depending on the ontology loaded, on the query and on the options chosen.

Figure 6 shows a part of the graphical interface of the tool with the results for the example of Section 5. The left-hand side of the screen contains a query describing the initial world (empty) and the final world. The right part presents a visualisation of the abstract graph generated by the tool using a backward-search algorithm. The nodes marked in blue (dark grey in Fig. 3, 5, and 6) correspond to

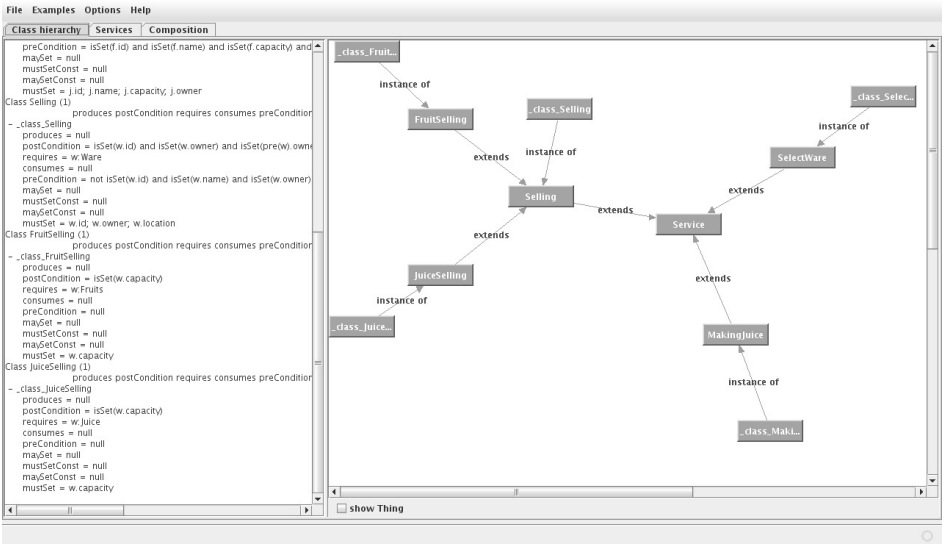

Fig. 3. The main screen of the tool. A diagram on the right shows the ontology, i.e., hierarchy of classes read from an OWL file for the example in Section 5

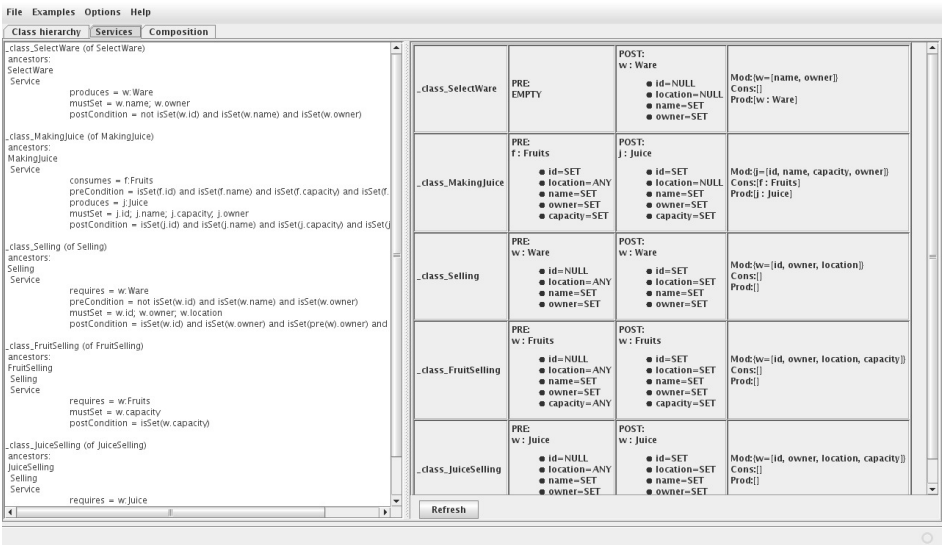

Fig. 4. Detailed information about services for the example in Section 5

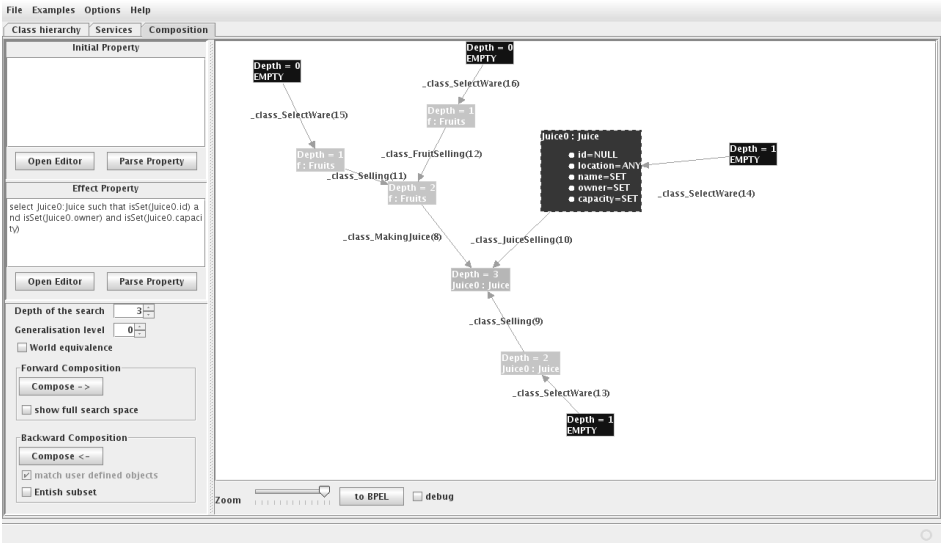

Fig. 5. Specifying a query for the example in Section 5

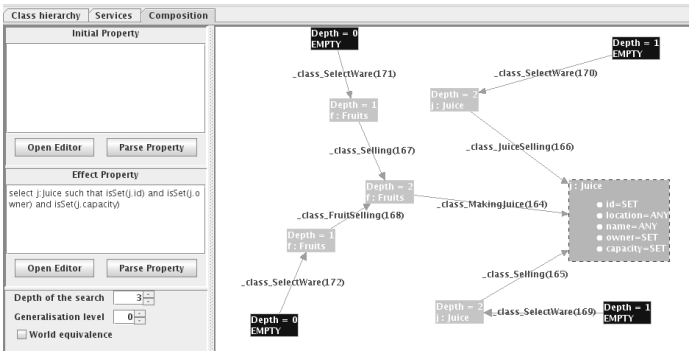

Fig. 6. The result for the example of Section 5

the initial domain (these are empty worlds). The green node (grey in the figures) represents the final world, satisfying user's request. Each path from an initial world to the final world represents a sequence of services whose execution results in an effect which satisfies the user's request. The sequences obtained are:

— `SelectWare`, then `FruitSelling` and then `MakingJuice`;
— `SelectWare` and then `JuiceSelling`;

— `SelectWare`, then `Selling` and then `MakingJuice`;
— `SelectWare` and then `Selling`.

It should be noticed that the planner generates all the possible paths, not only these involving the services specified in a "most detailed" way.

8 Final Remarks

The system presented in this chapter is at an early stage of development. Our aim is to prepare an easy mechanism for automatic composition, which can be applied to various domains. Our areas of interest are e-commerce and web support for medical services.

Directions of our future research involve a complete specification of the grammars for the validation formulas of concrete services, and the problem of building proxies connecting the composition system with real-world web services, together with mechanisms enabling service registration. It seems also necessary to extend the languages of formulas to a complete first-order language (with quantification). In particular, the modification lists should become elements of validation formulas, which would enable specifying optional modifications of a world. Moreover, solving further, bigger examples modelling real-world buisness processes will allow to evaluate the practical applicability of the approach. As operating on very complex ontologies can result in a state explosion, we are working on narrowing the searching sets by various methods, including symbolic processing and reasoning in ontologies.

The current version of our approach does not cover several aspects of web services such as management of them or ensuring security with federation mechanisms. We also do not use any ideas originated from research on a web services decomposition. Our aim in the near future is to integrate these ideas with our solutions.

References

1. Ambroszkiewicz S. enTish: an approach to service description and composition. ICS PAS, 2003.
2. Web Services Description Language (WSDL) 1.1. 2001, `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`.
3. SOAP version 1.2. 2007, `http://www.w3.org/TR/soap`.

4. Web Services Business Process Execution Language v2.0. 2007, `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`.
5. Universal Description, Discovery and Integration v3.0.2 (UDDI). 2005, `http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm`.
6. Klusch M., Geber A., Schmidt M. Semantic web service composition planning with OWLS-XPlan. Proc. 1st Int. AAAI Fall Symp. on Agents and the Semantic Web, AAAI Press, Arlington VA, USA, 2005.
7. Rao J. Semantic web service composition via logic-based program synthesis. Ph.D. diss., Dept. of Comp. and Inf. Sci., Norwegian University of Science and Technology, 2004.
8. Rao J., Küngas P., Matskin M. Logic-based web services composition: from service description to process model. Proc. IEEE Int. Conf. on Web Services ICWS'04, IEEE Computer Society, San Diego CA, USA, 2004.
9. Rao J., Su X. A survey of automated web service composition methods. Proc. 1st Int. Workshop on Semantic Web Services and Web Process Composition SWSWPC 2004, 43–54, San Diego CA, USA, 2004.
10. Redavid D., Iannone L., Payne T. OWL-S atomic services composition with SWRL rules. Proc. 4th Italian Semantic Web Workshop: Semantic Web Applications and Perspectives SWAP 2007, Bari, Italy, 2007.
11. Srivastava B., Koehler J. Web service composition — current solutions and open problems. Proc. Int. Workshop on Planning for Web Services, ICAPS 2003, Trento, Italy, 2003.
12. McDermott D., Ghallab M., Howe A., Knoblock C., Ram A., Veloso M., Weld D., Wilkins D. PDDL — the Planning Domain Definition Language — version 1.2. tech. rep., Yale Center for Computational Vision and Control, TR-98-003, 1998.
13. Ponnekanti S.R., Fox A. SWORD: a developer toolkit for web service composition. Proc. 11st Int. World Wide Web Conference WWW'02, Honolulu HA, USA, 2002.
14. DAML-S (and OWL-S) 0.9 draft release. 2003, `http://www.daml.org/services/daml-s/0.9/`.
15. Sirin E., Hendler J., Parsia B. Semi-automatic compositions of web services using semantic description. Proc. Int. Workshop Web Services: Modeling, Architecture and Infrastructure, ICEIS 2003, Angers, France, 2003.

Virtualized Infrastructure for SOA

Marcin Jarząb, Jacek Kosiński, Joanna Kosińska, Krzysztof Zieliński, and Sławomir Zieliński

Department of Computer Science, DSRG,
Al. Mickiewicza 30, 30-059 Kraków, Poland
{marcin.jarzab,jacek.kosinski,joanna.kosinka,krzysztof.zielinski, slawomir.zielinski}@agh.edu.pl
http://www.soa.edu.pl

Abstract. This chapter describes an innovative execution environment for SOA applications that consists in a set of infrastructure services supporting the service life cycle. These services are essential for distributed application services provisioning, discovery, monitoring, and management and operate over virtualized execution resources deployed over physical infrastructure. This chapter describes the concept of a service-oriented infrastructure (SOI) and explains its role in the context of SOA application deployment and execution. The system architectural elements and their fundamental functionalities which have to be implemented to support the managed, predictive and adaptive behavior of SOI, are specified.

Keywords: service oriented infrastructure, virtualized execution infrastructure, virtualized infrastructure monitoring

1 Introduction

Modern execution environments for SOA (Service Oriented Architecture) applications represent a set of infrastructure services supporting different stages of service lifecycle. These services are essential for application provisioning, discovery, monitoring, and management. They operate over virtualized execution resources deployed in a physical infrastructure. This leads to a system with three basic layers: physical, virtualization, and infrastructure, which together constitute an execution infrastructure for SOA applications.

Each of these layers can be structured with service orientation principles in mind, resulting in a so-called service oriented infrastructure (SOI). In modern SOI, each layer can be split into several sub-layers. The physical layer typically

includes communication, computational and data storage sublayers. The virtualization layer covers operating systems, server virtualization and operating system virtualization. Virtualization technologies can also be applied in the infrastructure services layer where typically containers supporting application service execution are instantiated. These containers depend on implementation technologies and provide a suitable virtualization of operating system processes, e.g. JVM (Java Virtual Machine) with the OSGi (Open Services Gateway initiative) container. They also support SOA middleware execution platforms such as ESB (Enterprise Service), application servers or MOMs. On top of this infrastructure business services can be installed.

From the provider's point of view each level of this highly complex infrastructure should be provisioned, monitored and managed. Software tools supporting this activity deal with very sophisticated mapping of higher-layer functionality onto lower-layer resources and mechanisms. In its most basic form, this process includes configuration selection and QoS (Quality of Service) tuning. A more advanced model involves transforming the managed platform into a predictive system and then into an adaptive system. Finally the most mature level autonomic system behavior can be considered.

Adaptive (or autonomic) behavior of the system requires specification of the goal that should be achieved as a result of adaptation or autonomic process realization. Such a goal is usually expressed as a collection of QoS or QoE (Quality of Experience) values to be attained. The adaptation/autonomic process is typically performed as a set of actions fired by management rules.

This chapter introduces the concept of SOI and explains its role in the context of SOA application deployment and execution. The architectural elements of the system are specified, along with their fundamental functionalities which need to be implemented to support managed, predictive and adaptive SOI behavior.

The structure of this chapter is as follows. First, the SOI model is introduced in Section 2. Subsequently, manageability levels are briefly elaborated upon. The concept and architecture of adaptive systems is discussed in detail and the basic building blocks of such a system are explained in Section 3. Section 4 presents the virtualization layer with particular attention to operating system and server virtualization. In Section 5 an adaptive system prototype using Solaris Containers is presented. This system relies on JIMS (Java Infrastructure Management System) to manage Solaris Containers and Projects in an adaptive way. Section 6 presents a similar concept for Xen VMs. A prototype called VEI (Virtual Execution Infrastructure) is described. Both prototypes follow similar architectural principles but offer complementary features. Finally the role of the metadata registry is discussed in the context of SOI management. This registry contains "glue-like" information, supporting integration of the SOI multilayer architecture into one system. Data

stored in this repository describes all categories of resources, in addition to configuration and service types which are supported by SOI. Access to this information is required to support dynamic composition and interoperability of SOI elements. The chapter ends with conclusions.

2 Service Oriented Infrastructure

The Service Oriented Infrastructure is based upon the SOA concepts of service providers and service consumers as applied to the computing, networking and storage infrastructure.

SOI describes IT infrastructures in terms of services. The architecture encompasses all phases of the SOI lifecycle: design, provisioning, operation, decommissioning and management, of services. It is relevant to the discovery of SOA-enabled applications and mapping business processes onto the underlying infrastructure and IT assets. Infrastructure Services typically use and/or provide a virtualized pool of shared resources (servers, network, storage and infrastructure software) which are deployed and managed in a highly automated way. SOI provides a foundation support for Service Oriented Application Architecture and other application architectures [1].

When demand for computing resources changes — for instance an application requires more CPU, RAM, network or I/O bandwidth — a SOA-style request is delegated to the SOI management framework. The IT infrastructure is treated as a set of services that can be called just like any other service. To achieve such automation, infrastructure services are exposed by a monitoring and management system that must be orchestrated by a coordinator, using business rules. Such rules are mapped from higher-level business applications down to infrastructure services, based on application requirements [2]. SOI mechanisms deliver core infrastructure services to the SOA applications, maintaining consistent service levels. This architecture can be used for the automatic allocation of resources in real time as application workload increases. Service Oriented Infrastructure guidelines provide support for the following essential elements of SOI solutions [3]:

— lifecycle support to manage the deployment of SOI components;
— virtualization of infrastructure resources to provide a service interface for SOI users;
— service management to ensure that the SOI solution provides the required service characteristics.

2.1 SOI Reference Model

Figure 1 presents the SOI reference model [1], as defined by The Open Group[1] that merges infrastructure services with applications and consumers, thus capturing all activities related to management and monitoring of IT infrastructures. The model defines the conceptual building blocks that must be provided in order to use a service (or other element) in a SOI, when such an architecture is implemented.

The proposed architecture consists of the following elements:

Infrastructure Management Framework — a collection of software tools that enable the management of IT infrastructures in an adaptive way, driven by bussines rules.

Infrastructure Services — a collection of services that expose interfaces for the management of specific infrastructure resources. Those services are implemented as combinations of virtualized and physical services.

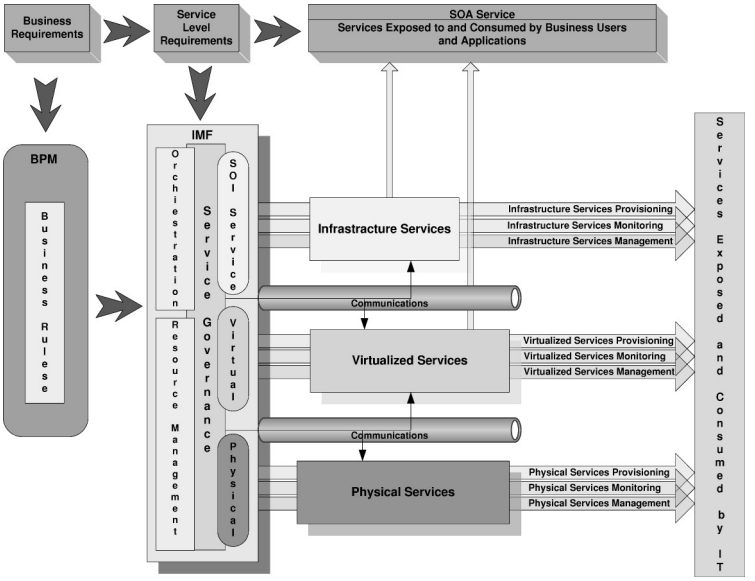

Fig. 1. SOI Reference Model

[1]The Open Group (`http://www.opengroup.org/`) is a vendor- and technology-neutral consortium whose vision of Boundaryless Information Flows enables access to integrated information within and between enterprises based on open standards and global interoperability.

Virtualized Services — a collection of tools that are part of the software provided by a specific virtualization solution.

Physical Services — represent hardware, storage and operating system resources. They can also be mapped to low-level services related to identity management (role-based configuration) or Single Sign-On (SSO).

Successful implementation of SOI involves a fusion of technologies, business models and infrastructure management techniques. Specifically, SOI accomplishes the goal of providing greater IT resource flexibility, efficiency and value generation by intelligently matching physical services to meet business demands on a pay-per-use basis. Intelligent matching refers to the combination of technologies, business processes, applications and services that solve a specific corporate IT problem. Merging provisioning and virtualization techniques is an effective approach to providing adaptive SOI, meeting the business requirements of SOA application users.

The provisioning process makes hardware and software available on an as-needed basis. It has the ability to allocate or reallocate a broad variety of computing resources, including servers, middleware, applications, storage systems and network interfaces, to the applications and systems that need them. Provisioning [4] addresses the tasks needed to deploy and activate a service. Whether dealing with a new operating system or an application upgrade, provisioning shifts, adds and changes components in a structured, repeatable process. It is an optimized form of change management and capacity planning.

Virtualization refers to resource management by manipulating devices, storage address spaces and data objects. Virtualization makes such resources more usable and effective through aggregation, subdivision and substitution. A good example of virtualization is an operating system, virtualizing the resources of a single computer and providing the foundation for higher-level virtualization. The same idea can be applied to computational resources such as servers, computer networks, storage space and desktop terminals.

The interfaces provided by the virtualized services for servers, storage, network and desktops enable management but also introduce their own challenges: additional complexity that calls for more advanced and flexible management capabilities. The provided tools must simplify administrative tasks and operate in a heterogeneous environment comprised of physical servers that are virtualized using different operating systems, all of which must be provisioned, updated and managed. The key requirements are:

Provisioning — there is a need for a centralized control of installation processes for physical and virtual nodes running specific operating systems. Such processes are also performed using pre-defined VM (Virtual Machine)/OS-Container appliances that contain operating system "images" with preinstalled software. A remote installation mechanism is required for new application

services or updates without the need to physically log into guest VM/OS-Containers.

Discovery — each node, whether physical or virtual, must be automatically discovered, identified and described, irrespective of its state. Such information must be stored in a global repository that is accessible for provisioning tools, to deploy application services which require specific computational resources and OS platforms.

Management — in a complex heterogonous environment, an important aspect is the ability to manage all systems remotely from a centralized location in terms of physical or virtual nodes, active services, disk utilization and system performance.

Monitoring — accurate information must be provided about performance metrics related to computational resource utilization. These metrics must be available for system adminstrators through an integrated console and presented in a uniform way.

Provisioning SOA applications over virtualized resources is a sophisticated process, due to the large number of logical elements that have to be managed as well as the complexity of installing modern applications. The deployment process is dependent on many parameters which must be customized for each computational element in the system.

2.2 Overview of Virtualization Technologies

Hardware and software advances introduce concepts of virtualization, which provides mechanisms for advanced management of resources e.g. devices, storage address spaces and OS instances. There are several different virtualization technologies (Fig. 2), each provides resources consumption granularity, some degrees of isolation and flexibility. They support ability that allows multiple OS and application instances to run on the same server, having the illusion that is running within its own hardware. It makes hardware resources more usable and more effective by aggregation and ability for partitioning them together.

When comparing domains to virtual machines, they are a hardware or firmware function that provides stronger isolation comparing to virtual machines. Booth of them enable server virtualization and subdivide a computer's physical resource (CPU, RAM, Network), but domains typically have lower overhead. Disadvantage of domains is a fixed number of hosted OS instances. Virtual machines are also managed by a hypervisor which itself is an operating system implemented in software and do not have fixed limits on number of running OS instances. Disadvantage of domains is architectural limit because they are only capable of running on Sun T1/T2 or IBM hardware platforms. Domains and virtual machines host

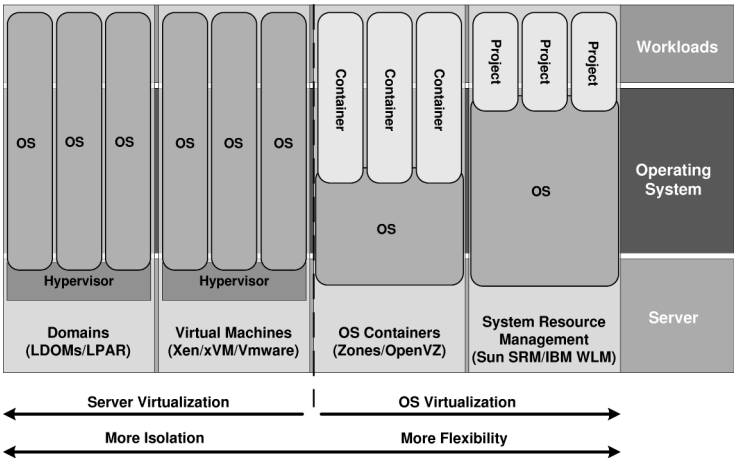

Fig. 2. Virtualization technologies

OS instances that also can be virtualized through OS virtualization techniques and founding so-called multilevel virtualization.

OS containers can be running in VMs, but have minimal overhead and isolation is accomplished by restricting the scope of system calls to the container from which they are invoked, thus there is no need for CPU-intensive emulation performed in hypervisiors. Solaris [5] or OpenVZ [6] containers can be created in large numbers, even on machine with a single CPU and provide more granular resource consumption control. OS Containers are available for many platforms including UltraSPARC, Intel or AMD architectures, but because they are running in single OS kernel it is not possible to mix and match software that requires conflicting OS versions. When multiple users or software application workloads are sharing the same domain, VM or OS container, it is possible that one workload would monopolize access to computational resources. To prevent such scenarios, facilities implemented in modern system resource management tools for instance by Sun SRM (Solaris Resource Manager) or IBM WLM (Workload Manager), are used. Workloads that are more important can be prioritorized along with other, less cirtical workloads, and ensure that bussines crtitical applications have access to required resources.

2.3 SOI and S3 Reference Model

When analyzing the relationship between SOI and the S3 Reference Model for SOA, the *Operating systems layer*, which includes all application assets running

in a virtualized IT infrastructure, is the main point of interest. SOI tools should enable effective provisioning of databases, transaction processing systems, middleware (e.g. J2EE or .NET) and SOA services running in a virtualized infrastructure. In fact, the SOI architecture tends to deliver services that can be described as a *Platform as a Service* (PaaS) and *Infrastructure as a Service* (IaaS).

PaaS exposes services on top of specific software platforms used to build higher-level services. In the context of the S3 model, such a platform would enable provisioning of SOA services at every stage of software development, testing, and regular maintenance. PaaS might produce a platform that contains properly configured OS, databases and application servers with pre-deployed SOA applications.

IaaS enables monitoring of SLA, resource utilization and capacity, as well as controlling CPU assignments, network and I/O resources through the use of appropriate standardized services with uniform access interfaces. Servers, storage systems, switches, routers, and other components are pooled and made available to handle workloads that range from application components to high-performance computing applications.

Such an approach to exposing software and computational resources (both physical and virtual) would also enable dynamic provisioning, reflecting the applications workload. There is a need for tools to monitor conditions and make adjustments automatically. Construction of such tools involves merging many software technologies and analytical methods which together form an SOI that enables greater IT automation. These tools would expose the functionality required by the *QoS layer* to monitor, log, and signal non-compliance with nonfunctional requirements defined in the *Governance and policies layer*.

2.4 Adaptive and Autonomic SOI

Adaptive and autonomic management of computing resources is not a new problem in computer science. For decades, system components and software elements have been evolving to deal with the increased complexity of system control, resource sharing and operational management [7].

Introducing autonomic computing architectures in the scope of SOI provisioning would reduce manual overhead related to administrative functions through the use of predefined policies. Developing components responsible for self-management of IT infrastructures adds the following characteristics [8]:

Self-configuring — dynamic on-the-fly configuration abilities mean that an IT infrastructure can adapt immediately, and with minimal human intervention, to the deployment of new components or changes in the IT environment.

Self-healing — any improper operations caused by system or application failures are detected automatically and corrective actions are initiated without disrupting application services. Corrective actions can alter the state of a product or influence changes in other elements of the environment.

Self-optimizing — an IT environment is able to efficiently maximize resource utilization to meet SLA with minimal human intervention. In the near term, self-optimization primarily addresses the complexity of system performance management. In the long run, self-optimizing components may learn from experience, then automatically and proactively tune themselves in the context of an overall business objective.

Self-protecting — the goal is to provide the right information to the right users at the right time through actions that grant access based on the users' role and pre-established policies. A self-protecting IT environment can detect hostile or intrusive behavior as it occurs and take autonomic actions to make itself less vulnerable to unauthorized access, viruses, denial-of-service attacks and general failures.

The evolution of IT lifecycle management (Table 1) in the scope of autonomic computing solutions can be divided into five levels, as explained in [9, 10]:

Basic Level — on this level IT professionals manage each system element. Activities related to configuration, optimization, healing and protection of IT infrastructure components are performed manually.

Managed Level — on this level system management technologies can be used to collect information from different systems. They assist administrators in collecting and analyzing information. In this scenario, IT professionals perform most of the analysis. This is the starting point for automation of IT tasks.

Predictive Level — on this level individual components monitor themselves, analyze changes, and offer advices. Therefore, dependency on administrators is reduced and decision making is improved.

Adaptive Level — on this level IT components can individually or collectively monitor operations and offer advice with minimal human involvement.

Autonomic Level — ot this level system operations are managed by business policies established by the administrator. In fact, business policies drive overall IT management, while the adaptive level, interaction's between humans and the system.

The architecture of Autonomic Computing Systems (ACS) proposed by IBM is presented in Figure 3 [10]. The basic elements of ACS are Autonomic Elements (AEs), i.e. interacting software agents which ensure self-management. Each AE consists of two types of building blocks: an Autonomic Manager (AM) and Managed Elements (ME) which are system components (hardware or software applications). Autonomic managers execute according to defined policies and are

Table 1. Division of IT management maturity levels according to processes, tools, skills and benchmarks [8]

	Basic	Managed	Predictive	Adaptive	Autonomic
Process	Informal, reactive, manual	Documented, improved over time, leveraging best practices, manual IT performance reviews	Proactive, shorter approval cycle	Automation of best practices in resource management and transaction management, driven by service-level agreements	All IT service management and IT resource management automated according to best practices
Tools	Local, platform and product-specific	Consolidated resource management consoles, problem management system, automated software installation, intrusion detection, load balancing	Role-based consoles with analysis and recommendations; product configuration advisors; realtime view of current and future IT performance; automation of repetitive tasks; common knowledge base for inventory and dependency management	Policy management tools drive dynamic changes, based on resource-specific policies	Cost/financial analysis tools, business and IT modeling tools, tradeoff analysis; automation of e-business management roles
Skills	Platform-specific, geographically dispersed technology	Multiple platform skills, multiple management tool skills	Cross-platform system knowledge, IT workload management skills, some business process knowledge	Service objectives and delivery per resource, analysis of impact on business objectives	E-business cost and benefit analysis, performance modeling, advanced use of financial tools for IT
Benchmarks	Time to fix problems and finish tasks	System availability, time to close issue tickets and work requests	Business system availability, service-level agreement fulfillment, customer satisfaction	Business system response time, service-level agreement fulfillment, customer satisfaction, IT contribution to business success	Business success, competitiveness of service-level agreement metrics, business responsiveness

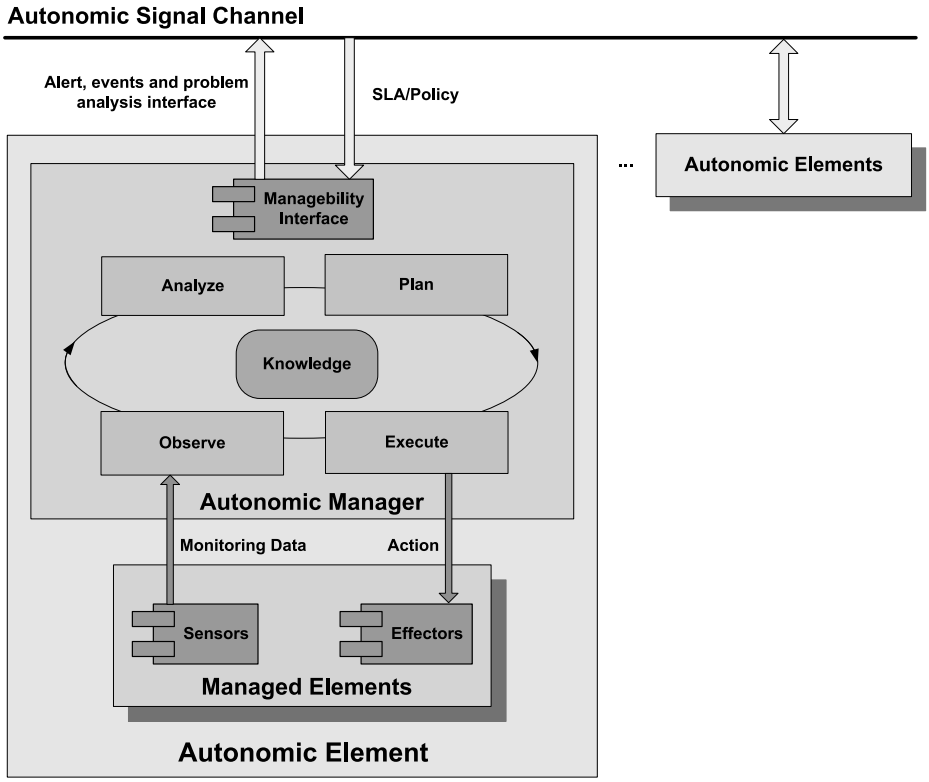

Fig. 3. Autonomic Computing System architecture

responsible for management of specific MEs that must expose sensor and effector interfaces. Sensors are used by AM to acquire monitoring data and then to compare it with specified goals. Corrective actions are performed on selected MEs through the use of effectors.

3 SOI Representation

Service-oriented infrastructure is defined as a set of technology-agnostic, standardized services delivered from a pool of resources controlled by a centralized management system. Reuse of services is of particular importance for many enterprises. In order to make services reusable, developers need to publish their descriptions. For small deployments, a documentation-based description can satisfy service users,

but in case of large deployments, the introduction of a service repository is necessary.

From applications point of view, a service repository is a catalog of available services containing descriptions of their capabilities. In a way, a service-oriented infrastructure provides an execution environment for (service-oriented) applications. The runtime environment can be reused by multiple applications at the same time. Moreover, the configuration of services provided to an application can be dynamically changed, especially in terms of quality of service and non-functional properties.

In order to provide maximum flexibility, the service-oriented infrastructure is typically decomposed into three layers:

— the physical layer, which represents physical resources related to computation, communication and data storage;
— the virtualization layer, which refers to hardware operating systems and mechanisms for server virtualization;
— the infrastructure services layer, which includes containers supporting application services.

The goal of SOI provider is to maximize the usage of infrastructure services while not jeopardizing the requirements specified in application contracts. Therefore, from the provider perspective, support for provisioning and management of resources are the key characteristics that a well-designed SOI repository should provide. Of course, when regarding infrastructure management, the provider is sure to rely both on the SOI repository and (maybe to an even greater extent) a monitoring subsystem. However, the repository by keeping track of available resources, could help with selecting appropriate options for application deployment.

Service repository is a part of a larger compound — the IT Repository, which forms a base of information regarding all IT components, services and their relations. It could be described as a federated view of information bases for domains such as:

— application management;
— configuration management;
— operational management;
— architecture repository.

The SOI repository described in this section is focused mainly on infrastructure configuration management, although the application management (in its deployment-related aspects) and operational management (especially user management) are also important sources of information to be stored in the repository (Fig. 4).

From the infrastructure point of view, deployment of an application is based on satisfying a set of requirements regarding the infrastructure services. The set

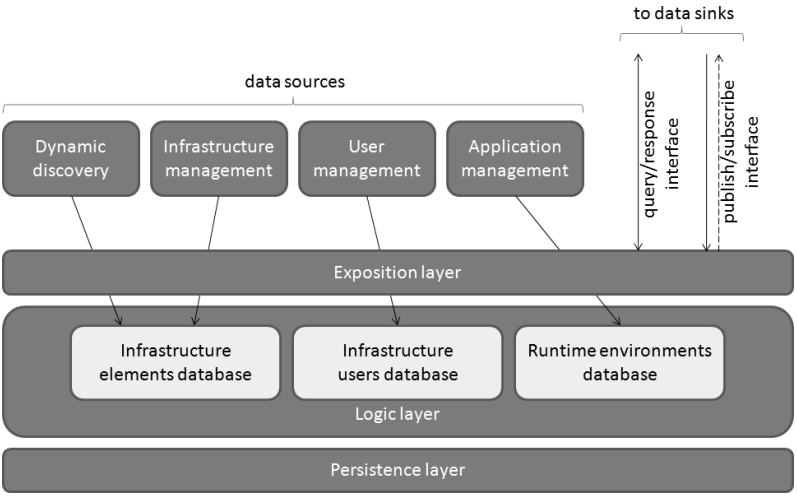

Fig. 4. The sources of information to be stored in the SOI repository

of requirements (both functional and non-functional) defines a runtime environment for a particular application. In the deployment process, the description of the environment is mapped to the virtualized execution infrastructure, which in turn is mapped to the physical resources. The introduction of two stages mapping allows for more flexibility in describing the runtime environment. As mentioned before, many dynamically deployed applications are expected to reuse the same shared virtualized infrastructure. Therefore, an authentication, authorization and accounting scheme should be implemented inside the infrastructure.

The infrastructure elements database is expected to be constructed based on management tools as well as dynamic discovery mechanisms. Many middleware technologies provide registries of services that could be used as basis of service repository inputs. Created and updated in the technology-specific service discovery process, the registries can hold a lot of information regarding interfaces of consumable services, which is particularly useful for automated searches. As examples of such registries, consider the OSGi Service Registry, Service Location Protocol Directory Agent, etc. The challenge is to unify these descriptions to provide a basis for technology-independent searches. A number of physical infrastructure discovery technologies (e.g. Universal Plug and Play) also exist. Again, the challenge is to unify these descriptions using common terminology and to aggregate the metadata describing the functionalities, interfaces, configuration options and requirements of particular elements.

A simplified model for SOI repository contents, which refer to application deployment, is presented in Figure 5.

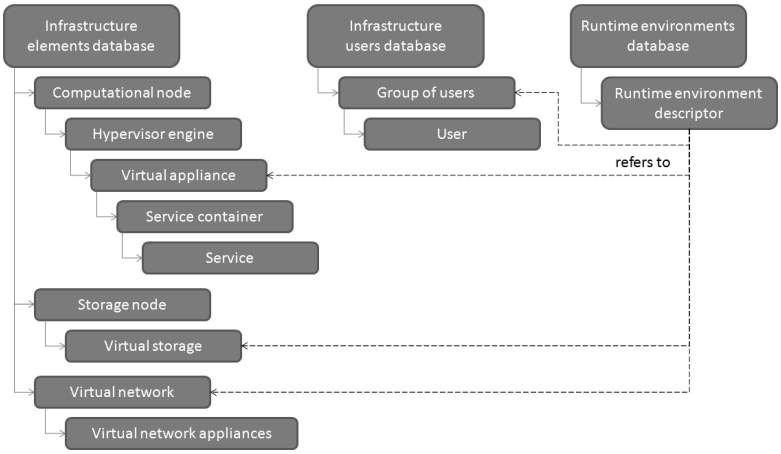

Fig. 5. Application deployment-related contents of the SOI repository

The application requirements are represented as an execution environment to be mapped to available resources. Because the application is going to be executed in a virtualized environment, the contents of the environment descriptor do not refer directly to physical resources, but rather to virtualized entities. Even in case of services provided by specialized hardware, it seems reasonable to wrap their functionalities in the form of services (sometimes called pure-play services) in order — for example — to change the hardware element without having the application notice the change.

The contents of the repository need to be supervised by a group of administrators. Therefore, the repository contents need to be presented to the administrator based either on the query/response or publish/subscribe patterns, although it is expected that in order not to overwhelm the administrator with large amounts of data, the publish/subscribe pattern will be used only with regard to the most important events, such as the discovery of new infrastructure elements, additions of users, installations of new software, etc. In response to the events reported by either the repository itself or other sources, the administrator will need to modify the infrastructure. In such case, the repository can be used for providing centralized control of the process of installing physical and virtual nodes. To accomplish the task, the repository should hold not only the descriptions of active entities, but of inactive (ready for instantiation/deployment) as well. Moreover, to support the process, the repository should cooperate with other tools, designed for performing specific operations on particular systems, with regard to e.g. dynamic allocation of resources.

4 Virtual Execution Environment

Architectures based on the SOI concept assume a mapping between a typical IT infrastructure and a set of services [1]. These services are usually based on the usage of shared virtualized resources (servers, communication infrastructure, data storage and applications). To ensure flexible mechanisms for creation of SOA environments that exploit the SOI architecture concept, suitable mechanisms for regulating resource access should exist. These mechanisms are called Resource Management Systems (RMS for short) and are particularly important in situations where resources are shared.

The usage of virtualization techniques is a natural evolution of existing RMS in distributed environments. Hence, the solution presented in this section can be treated as the answer to the question whether the virtualization offered by the available technologies is mature enough to develop virtual computers and virtual communication infrastructures to effectively organize processing in distributed environments. A key point point to address is whether efficient management of on-demand resources with virtualization techniques is possible using mechanisms such as dynamic infrastructure instantiation and resource allocation according to the requirements of applications or changing environmental conditions. Such mechanisms should be able to fully utilize the available infrastructure and decrease the operational costs through resource usage reduction, with no impact on the service QoS level.

A Virtual Execution Environment model that takes this approach into account has been proposed and analyzed in the context of SOA-related technologies. The novelty of the proposed approach can be expressed in terms of several important aspects [11]:

— VEI integrates virtualized computing and networking resources;
— VEI can be deployed on demand, according to application requirements, on the available physical resources in a shared infrastructure;
— once deployed, VEI can be modified during runtime so the running application can obtain or release access to physical resources during execution;
— VEI runtime management can be performed manually or by a policy engine, executing policy rules defined by the system administrator;
— VEI deployment and runtime management should be neutral from the perspective of SOA middleware usage.

The hierarchical management approach (Fig. 6) avoids many problems related to scalability and decreases the global complexity of management systems. In the proposed system specific management hierarchy levels correspond to the process of mapping an application onto virtualized resources, which is performed in two stages. The first binding is realized between physical and virtual resources, while

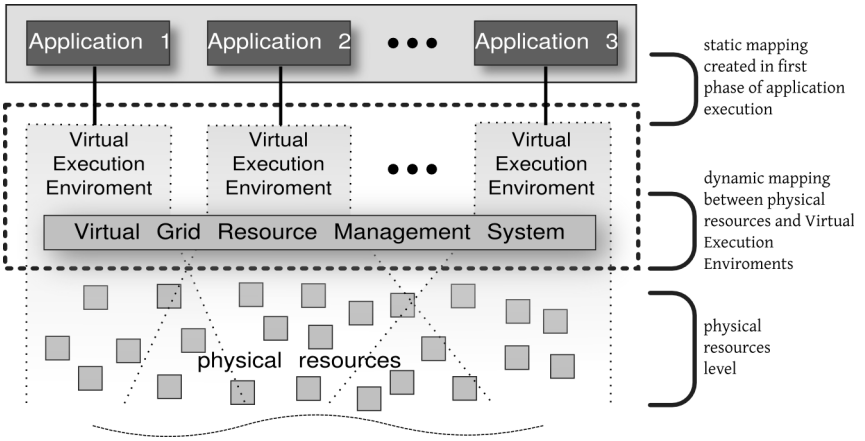

Fig. 6. Hierarchical resource management in VEI

the second involves allocation of virtual resources to an application. The advantage of this two-stage mapping is the simplification of resource allocation, since operating on virtual resources rather than on physical ones allows us to express application requirements at a higher level of abstraction.

The main task of VEI components is to control a number of virtual environments operating upon a shared physical infrastructure [12]. Control is executed by a rule engine, which processes rules that govern the usage and sharing of resources specified by users, based on application requirements. The other goal is to achieve a tradeoff between a guaranteed level of QoS for applications executing within VEIs and maximizing the utilization of shared resources.

4.1 VEI Architecture

The architecture of the VEI system is constructed according to current trends in building distributed systems. One of the most important assumptions is the decomposition of the environment into smaller components, each of which exposes the functionality of a single service.

The designed VEI resource management system architecture consists of five layers (Fig. 7). These layers result from the established concepts of using virtualization as a management technique. The layers are as follows:

— the physical resource layer — the heterogeneous resources constituting the infrastructure that typical computing centers are equipped with. Examples include physical computers, such as mainframes or PCs, communication infras-

tructures, specialized devices used in computing environments for application processing (such as disk arrays), etc.;

— the virtualization layer — this layer creates an abstraction of physical resources through proper mechanisms. In order for this to be possible, the resources must be properly instrumented. Instrumentation relies on provision of dynamic resource grouping functionality. Therefore, the virtualization layer components are responsible for:
  - VM creation;
  - enabling dynamic modification of virtual machine configuration (changing the mapping and level of physical resource usage). This is accomplished by modifying the dynamic VM parameters;
  - virtual network creation;
  - enabling dynamic modification of virtual network parameters.

  The presented virtualization layer is also responsible for provisioning an infrastructure, which enables distribution of the operating system over VM instances. This infrastructure must ensure access to disk images provided by users and containing, among others, the operating system software, computing libraries, and applications, along with their required input data;

— the exposition layer — components added to adjust and unify the virtualization layer mechanisms to simplify the interfaces for basic VEI resource management functions:
  - representing monitoring data and system configuration using facts;

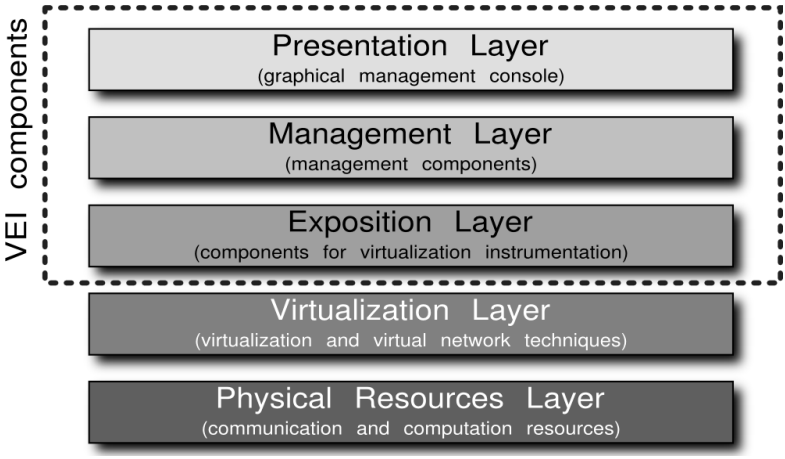

Fig. 7. Layered architecture of VEI components

- provisioning of services implementing localization of resources (specifically, VEI system components realizing resource management); registration and retrieval of other components;
- resource management policy realization through exposed effectors.

— the management layer — the main task of this layer is to process information about the environment state and operating procedures related to resource allocation for distributed applications. During runtime, information about the state of system components is collected as facts. This information is then processed by a rule engine in order to take appropriate actions;

— the presentation layer — this layer consists of components enabling the end user to affect the state of other system elements. It is implemented as a graphical interface, in the form of VEI system console. Each console can be connected with one or more component instances representing and managing VEIs. Presentation layer components can gain access to other VEI components through lookup services located in the exposition layer.

The VEI components can be located at various points in the physical infrastructure as they communicate over the network. The presented layers do not impose any restrictions on component allocation.

5 JMX-Based Infrastructure Management System for SOI

SOA applications that run on SOI must be managed by a monitoring and management environment that understands the health of the infrastructure supported by IaaS and PaaS services located in the operating system layer.

JIMS (JMX-based Infrastructure Management System) was initially developed as a grid infrastructure monitoring tool [13, 14], but its flexibility and extensibility allows its adaptation to different environments and scenarios. JIMS can monitor multiple machines in the network, but a JIMS agent must be installed on each of them. One machine (gateway) collects information from all other nodes and exposes it for clients. This node can effectively communicate with other nodes and can be accessed by clients using various protocols and technologies (including Web Services).

The JIMS communication model depicted in Figure 8 can be split into four architectural layers:

Instrumentation Layer. It provides infrastructure and Java application monitoring information using specially designed sensor and effectors modules installed in the JIMS Monitoring Agent (JIMS MA).

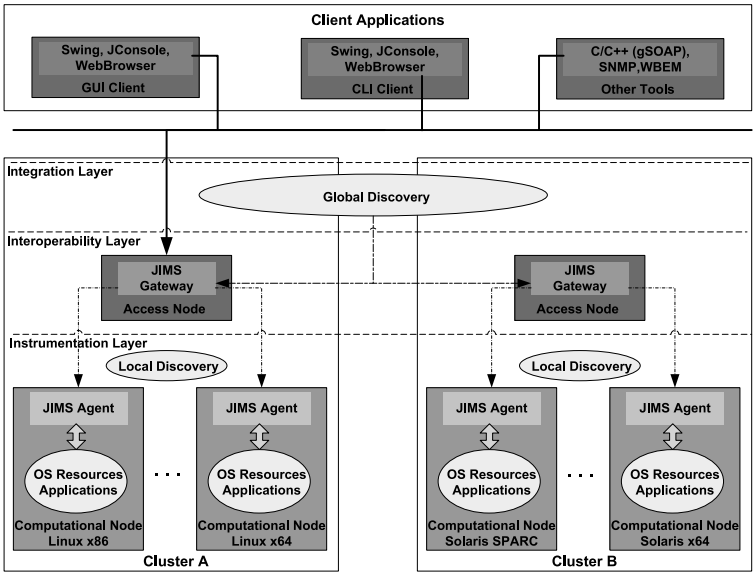

Fig. 8. JIMS layered architecture

Interoperability Layer. It provides a common point of communication with computing nodes (also called Worker Nodes or WNs) in clusters through dedicated access nodes (Access Node — AN) on which the JIMS Gateway in installed.

Integration Layer. It enables discovery of all accessible clusters and provides an overall view of the Grid. This layer consists of Global Discovery modules installed in the JIMS Gateway agent running on an Access Node in a specific cluster.

Client Application Layer. It consists of applications connected to the JIMS system, which consume information produced by the system.

The main features of JIMS which make it an attractive platform for monitoring and management are:

Extensibility — it is very easy to extend JIMS functionality thanks to the JMX technology. All that is needed is to create proper MBeans (Managed MBeans) and to load them (JIMS can dynamically download classes from network locations);

Auto-configuration — JIMS agents (installed on monitored computers) automatically discover each other. Almost no configuration is required. Moreover, installing new modules (MBeans) on nodes is easy: they only need to be made

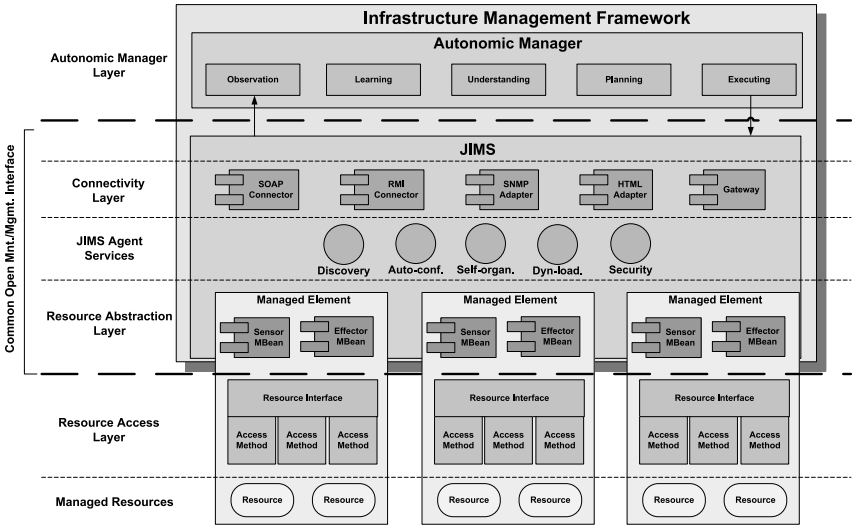

Fig. 9. JIMS management environment for SOI

available in a networked repository and then nodes can use them. This architecture makes module updates very easy, since they only have to be updated in the repository, then nodes will download them automatically;

Platform independence — JIMS is written in Java, therefore it can be run on any platform. JVM availability is the only requirement. Although some monitoring modules parts are platform-dependent, they are still easy to develop (only the system-dependent parts must be re-implemented);

Uniform monitoring and management interface — each resource monitored by JIMS is represented as an MBean and can be accessed by any JMX-compliant client, whether it is a graphical user interface or a Java application with an embedded management logic.

JIMS also offers control functions for managing the underlying resources and executing operations. These can be used as executors and effectors in an autonomic element of the ACS architecture.

The management environment installed with the JIMS system can be divided into six logical layers, shown in Figure 9.

The first two layers represent managed resources (hardware and software infrastructure, network, etc.) as well as access methods forming the interfaces for control and monitoring. The second layer, i.e. the resource access layer, uses vendor-specific resource interfaces and various methods for communicating with the operating system and other managed components. The third layer, the resource abstraction layer, is the de-facto JMX instrumentation layer, with JIMS agents installed on

each managed element. This layer contains managed beans (MBeans) representing underlying resources in a uniform way. It also provides a set of interfaces used for monitoring and management. Because all these interfaces are coherent and conform to the JMX specification regardless of the access methods in the second layer, they constitute a common and open resource interface (API), which can be used to build an Autonomic Element. In order to fulfill the AC requirements, the abstraction layer underlies another dedicated layer, providing a set of additional services in support of autonomous management. These functions include discovery mechanisms for finding and organizing managed elements found in local [13] and wide-area networks [15], system auto-configuration, management functions for controlling the JIMS system itself as well as the functionality necessary for loading and installing additional modules during runtime. The next layer, i.e. the connectivity layer, provides remote interfaces for accessing resources using various protocols and tools. Among those protocols the most important ones include connectors for RMI and SOAP communication, where clients can connect using the Dynamic Invocation Interface (DII) or dynamic proxies. The SOAP connector can also act as a secure and stateful Web Service for communication with clients written in popular programming languages (Perl, C/C++). An important feature of all these communication components is the common connector interface, implemented by all protocol-specific connectors. This fact makes the JIMS API coherent in all possible protocols and ready for future extensions. Protocol adaptors can in turn be used as front-ends extending JIMS functionality for use with certain types of standard management applications. Well-known adaptors for JMX include SNMP and HTTP/HTML protocol adaptors. They can be used in situations where specialized clients or JMX consoles are not suitable and cannot be applied.

In order to be monitored by the JIMS system, each computational node (physical or virtual) must have a generic JIMS agent installed, as depicted in Figure 10.

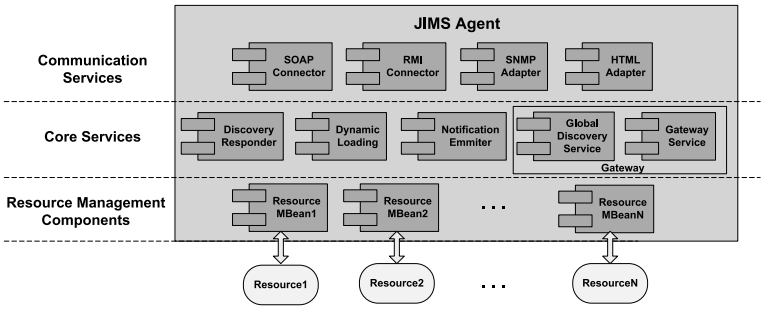

Fig. 10. JIMS Agent components

Three groups of MBean components can be distinguished: resource components, core services components and communication components. The first group has already been described and is responsible for translation of particular-vendor specific interfaces into a common API. Service components are responsible for autoconfiguration, dynamic loading and self-organization. Another interesting feature of the JIMS services layer is its self-organizing facility which relies on Discovery Responders installed in each generic JIMS agent, set up to listen to a well-known multicast address and responding to the JIMS gateway with the address of an RMI connector, which is a mandatory element of each agent. When the generic JIMS agent discovers that it is installed on an access node (a node exposed in the public network for communication purposes, acting as a gateway between the public wide area network and the monitored resources, hidden behind firewalls and deployed in a private address space), it automatically loads the JMX Gateway module. The Gateway module is responsible for organizing generic agents into a federation and acts as a mediator between remote clients and these agents. The JIMS Gateway running in the interoperability layer discovers generic agents. Gateway agents act as a proxy, exposing the management functionality of the underlying agents in the form of Web Services and provide access to the available resources using a single unified interface. This interface makes extensive use of reflection (based on the Java Reflection API), i.e. it does not define particular management methods and attributes, but is generic for the purposes of accessing any type of interface operations and attributes in existing and future managed objects (MBeans). The self-organizing facility is also responsible for organizing federations in a wide area network. The module that carries out the work of federation registration and heartbeating is the Global Discovery Service (GDS), responsible for organizing the JIMS system in the WAN. This aspect is elaborated in detail in [15, 16]. Autoconfiguration elements (not shown in the figure) enable JIMS to learn the features of the surrounding environment and load proper modules with sensors and effectors using the dynamic loading JMX functionality (MLet service). The main idea behind dynamic loading of components using the MLet service is based on reading component descriptors and downloading the actual components (JAR files). The types of loaded modules depend on the discovered features of the environment where the agent is started.

By implementing a discovery service, JIMS provides a very simple yet powerful mechanism for the auto-discovery of Java applications to be automatically monitored and managed by JIMS. It relies on the instrumentation mechanism first made available in the Java 1.5 release, which allows instrumentation of Java applications running within JVM with the Java agent option (module) enabled. Such an agent must provide a class with a premain method, which is called before the main method of the application class. Our implementation of this agent initializes the JIMS discovery mechanism, enabling automatic discovery of applications within a running JVM instance.

6 Summary

The SOI concept results from applying service orientation principles to SOA application execution infrastructures. The manageability of computational and communication resources has significantly increased due to substantial progress in the area of multilayer virtualization technologies offered by operating systems and specialized middleware. Virtualization technologies, providing flexible sharing of computer resources with QoS guarantees, are being developed to improve the utilization of high-performance hardware. The monitoring and control mechanisms offered by these technologies make it possible to expose computer resources as services, managed via well-defined interfaces.

Exposing computer resources as services enables us to apply policy-driven control mechanisms to their adaptive management. Services can be exposed in a standard way using the JMX technology. This approach seamlessly integrates system and application levels, as both can be programmed in Java.

It is necessary to point out that dynamic control of virtualized physical resources exposed as services is the only effective way to ensure application service QoS. Business-level requirements referring to QoS should be mapped onto control decisions affecting the operation of the SOI.

For more complex applications deployed as clusters of cooperating services, exposing each category of resources (e.g. CPU power, memory, network bandwidth, etc.) separately might not be enough. Such scenarios require integrated, synchronized control of several resource categories at the same time, leading to the concept of a virtual grid or networked cloud computing.

There is no doubt that SOI is going to become the most fundamental S3 layer, with substantial impact on the performance of SOA applications. Its increasing role reflects the progress in the development of virtualization technologies.

References

1. Service Oriented Infrastructure reference framework. (Draft Technical Standard), The Open Group, SOA Working Group, 2008, `https://www.opengroup.org/projects/soa-soi/uploads/40/19218/soi-V1-5-P1.pdf`.
2. A white paper from BEA and VMware, SOA and virtualization: how do they fit together. BEA, VMWare, 2007, `http://www.vmware.com/solutions/partners/alliances/oracle-middleware-whitepapers.html`.
3. Service Oriented Infrastructure project description — from spaghetti to lasagne. The Open Group, SOA Working Group, 2007, `http://www.opengroup.org/projects/soa-soi/uploads/40/14171/OG-SOI-Charter-v1-1.pdf`.

4. Gassman B. Provisioning IT services. Gartner Group, 2002, http://www.bus.umich.edu/KresgePublic/Journals/Gartner/research/110500/110501/110501.pdf.
5. Lageman M. Solaris containers what they are and how to use them. Oracle Corporation, Santa Clara, CA, USA, 2005.
6. OpenVZ User's Guide Version 2.7.0-8. SWsoft Corporation, 2005. http://tp.openvz.org/doc/OpenVZ-Users-Guide.pdf.
7. Alan G Ganek T.A.C. The dawning of the autonomic computing era. IBM Corporation, 2003.
8. Autonomic computing concepts: E-bussines white paper. IBM corporation, 2001.
9. Murch R. Autonomic Computing. IBM Press, 2004.
10. Nami M.R., Bertels K. A survey of autonomic computing systems. Proc. 3rd Int. Conf. on Autonomic and Autonomous Systems ICAS'07, Washington, DC, 26, IEEE Computer Society, 2007.
11. Kosińska J., Kosiński J., Zieliński K. Virtual grid resource management system with virtualization technology. Proc. 2nd Conf. High Performance Computers' Users KU KDM'09. Zakopane, Poland, 2009.
12. Kosiński J., Zieliński K. Enhancing grid computing with virtual grid concept model. Cracow Grid Workshop 2007, M. Bubak, M. Turala, K. Wiatr (eds.), Kraków, 291–298, ACC CYFRONET AGH, Kraków, 2007.
13. Padee A., Nawrocki K. Postprocessing monitoring system, report on the results of the WP3 2nd and 3rd prototype. ALGO, CSIC, CYFRONET, DATAMAT, ICM, PSNC, TCD, UAB, UCY, 2004, http://www.eu-crossgrid.org/Deliverables/M24pdf/CG3.0-D3.5-v1.2-PSNC010-Proto2Status.pdf.
14. Balos K., Zieliński K. JIMS — the uniform approach to grid infrastructure and application monitoring. Workshop Proc. CGW'04, Kraków, 2004.
15. Balos K., Zieliński K. WS-based discovery service for grid computing elements. Advances in Grid Computing EGC 2005, Amsterdam, 2005.
16. Balos K., Zieliński K. Discovery service for JMX-enabled monitoring system — JIMS case study. Workshop Proc. CGW'05, Kraków, 2005.

Ontology Scripting Language to Represent and Interpret Conglomerates of IoT Devices Accessed by SOA Services

Jarogniew Rykowski, Przemysław Hanicki, and Mirosław Stawniak

Department of Information Technologies
Poznań University of Economics,
Mansfelda 4, 60-854 Poznań, Poland
{rykowski,hanicki,stawniak}@kti.ue.poznan.pl

Abstract. The idea of dynamic, searchable conglomerates of Internet of Things (IoT) devices to be accessed by SOA services has been proposed and implemented. The main goal of this idea is to provide a single, consistent solution for the following problems: (1) efficient searching for IoT devices to be used "at the place", (2) dynamic (re)configuration of device groups to fulfill extended needs by providing complex functionality, (3) dynamic way of programming device behavior, and (4) efficient support for IoT devices with handicapped access and/or functionality. The solution is based, on the one hand, on application of a dedicated ontology related to a classification of IoT devices and their capabilities, and on the other hand, on the required functionality of the system from the point of view of SOA services. The ontology is to be interpreted at run-time, enabling the contextual choice of optimal devices according to dynamically changing circumstances, place of interaction, user/SOA service grants and restrictions, etc.

Keywords: ontologies, calm computing, Internet of Things (IoT), SOA

1 Introduction

Nowadays, we observe a boom of small computer devices helping people in their everyday activities. Personal-Area networking (PANs), body networks (BANs), calm-computing environments, or more generally Internet of Things (IoT) — there are plenty of devices everywhere waiting to assist humans, able to cooperate to observe different goals, and exchanging data with each other, as well as the environment. Such a diversity of IoT devices makes them hard to control, to synchronize and cooperate, to organize some complex services and functionality, etc. Moreover,

as humans carry some personal devices with them, these devices are ready to interact with at-the-place counterparts in ad-hoc mode. This mode of usage provokes several problems, mainly related with interfacing, functionality equivalence and replacement, local and global searching and accessibility, mutual cooperation to observe common goals, and many more.

In parallel to IoT development, a trend is observed to massively apply SOA services [1] implementing business models, and cooperating with humans to achieve common business goals [2]. Currently, such a cooperation tends towards the automation, i.e., replacement of traditional human-related activities by device functionality, with respect to the above-mentioned trend of mass application of different computer devices in different contexts and for diverse business purposes. As a result, the emergence of contextual usage of IoT devices controlled by SOA services is observed.

So far SOA services were targeted on interaction with humans. Such services were usually implemented for human-related purposes and business goals. While some system functionality, so far reserved for humans, is now being shifted to devices for automatic treatment, a need for new interfacing methods arose among these devices and SOA core.

While discussing a possible implementation of the above idea of connecting IoT devices and SOA services, one may enumerate several problems of different nature, both business-related and technical. Business-related problems are to be solved by SOA functionality. As such, this is not the main goal to be discussed in this chapter. Several technical problems are already solved as well, such as efficient and reliable radio communication, routing and addressability, even anonymity vs. security and tracking. However, the following issues are still open: (1) the standardized classification and representation of device groups (classes) enabling cooperation of different devices with the required functionality, (2) unified representation of static ("I am able to open doors") and dynamic ("I am related with the doors you are currently looking at") device capabilities, (3) definition of searchable, contextual catalogues of devices and device capabilities, dynamic run-time determination of "the right device at the right place", and (4) dynamic grouping of some devices into bigger conglomerates with extended functionality ("I will open the doors once you prove your access rights using any of your BAN devices").

In this chapter a single technology is proposed to solve all these problems. The idea is based on ontological representation of devices, their capabilities, and of possible cooperation towards programmable device conglomerates. The ontology [3] may be parsed at run-time, enabling a contextual choice of optimal devices according to dynamically changing circumstances, place of interaction, user grants and restrictions, etc. The remainder of this chapter is organized as follows. First, preliminary assumptions are given about possible ways of representation of functionality and state of IoT devices. Second, the design and usage of device conglomerates is introduced and discussed, taking into account different ways of programming targeted and observed device functionality, and proxying access to

some communication-limited devices, such as simple switches and locks. Finally, application areas of device conglomerates are enumerated, related with dynamic searching for choosing and using particular devices in the mode "the right device at the right place in the right moment".

2 Representation of IoT Devices — Preliminary Assumptions

The design and implementation of the idea of searchable, dynamic device conglomerates is based on several assumptions which are enumerated below.

First, the definition of a *IoT device* (Internet-of-Things device [4]), called sometimes a *calm-computing* device, has to be specified. An IoT device is a small, specialized electronic device capable to provide certain functionality and operating at certain geo-location, waiting for a possibility of an interaction with a user or his/her personal device. A way of interaction, i.e., device functionality and interfacing, is based on: (1) a device group — a conglomerate of certain devices operating at the same geo-location and cooperating to observe a common goal, (2) a dynamic context (some independent parameters determined by the environment, external requests, etc., and (3) some preferences and requirements about the interacting entities, mainly humans. IoT devices may differ by functionality, specialization, communication possibilities, data-processing and data-storing capabilities, as well as interaction methods with humans, animals, and other devices.

It is assumed that usually IoT devices are not to be called (identified) by name. Instead, an automatic searching procedure is provided for certain geo-localization and required device functionality, with the dynamic allocation of given named devices at run-time. The device cataloging, linking, and searching processes should be independent on business logic implemented as SOA services. As a consequence, SOA services may use the full potential of the devices at-the-place without a need for detailed knowledge on device location, capabilities, communication methods, etc.

As the possible set of IoT devices is huge, moreover, evolving over time, it was assumed that the devices are grouped into generic device classes. The classes are modeled as a graph, with a hierarchical structure enabling single- and multi-inheritance. Each class is described by certain parameters (some parameters are inherited from the super-classes in the class hierarchy, similarly to object-oriented programming paradigm). The class instances, with strictly declared values of the parameters, represent the given devices.

Both device classes and instances are represented as an ontology. To store and manipulate the ontology, OWL standard is used, and Protégé environment is applied. An interaction of devices and their classes, on one hand, and SOA services,

on the other hand, is based on a dynamic comparison of device capabilities (determined on the base of device descriptions and current state of device instances), and service requirements (invocation parameters). It is assumed that both device capabilities and service requirements are represented by the ontology.

Usually, a coherent group of heterogeneous devices is contacted rather than a single one. Thus, conglomerates of devices are provided. A conglomerate is a graph of statically linked devices, with an internal structure, to be interpreted at run-time. During the interpretation, late binding is applied to link device classes, directly pointed by the conglomerate, with the given device instances. This binding depends on (1) the invocation context, (2) the invocation parameters, such as keywords and description, and (3) the service requirements (direct parameters). Similarly to previous system elements, all the conglomerates are represented by the ontology.

For each device class, a Java code may be provided to implement the detailed behavior (contact) of any device belonging to this class. This code must be prepared according to certain rules, e.g., must observe given Java interface and must implement certain Java methods. A Java implementation of a device substantially extends device capabilities, especially for those devices with no software representation (such as switches, sensors, and similar hardware devices), devices with no geo-location, etc.

In the next sections of this chapter, proxying access and representing the state of IoT devices (Section 3), as well as basic way of design (Section 4) and interpretation (Section 5) of device conglomerates is described.

3 Proxying Access and Functionality of IoT Devices

The significant differences of devices produced to achieve different functionality and operating in different environments cause a lack of standardization, followed by the situation when a connection between a computer and an IoT device is defined by the producer of the IoT device, with no respect to other solutions.

To bypass the lack-of-standardization problem mentioned above, we introduced an interconnection layer among computer systems and IoT devices, as a proxy an access to selected devices. Our approach aims in providing a proxy service for each IoT device. Proxy functionality is twofold. First, a proxy standardizes on-line access to the device, from the system point of view, hiding some technical details related with a communication to/from a device. Second, the proxy possibly enables caching device state, making it possible to provide some details about the device without the need access this device (off-line access), and thus limiting some device restrictions such as a trade-off between a need for energy savings and an amount of information exchanged.

In the following subsections, first, categories of calm-computing devices are presented and discussed towards a standardized, generic access method. Second, sample implementation of a test system is described.

3.1 Categories of IoT Devices

IoT devices may be classified according to several criteria, categories and types, which are enumerated below.

First, these devices may be divided into the following categories, with respect to the observed device functionality:

— Passive InfraRed motion sensors (PIR);
— Photo-Electric Beams;
— Ultrasonic sensors;
— Microwave sensors;
— Vibration sensors;
— Smoke, heat and carbon monoxide detectors;
— Weather stations;
— Identification devices — i.e. Radio Frequency IDentification (RFID) or Near Field Communications (NFC), and
— Motion detection.

Second, with respect to communication type, IoT devices may be divided into input and output devices. A mixed mode of communication is also possible, however, quite rarely applied — most of IoT devices are specialized towards either output-only, or input-only communication.

Third, IoT devices may be of either analog or digital nature, depending on the type of equipment used. If the device provides analogic input/output only, a hardware converter must be used to convert the analogue signal to a digital counterpart. We assume that such a converter is provided for each analogue device, and possibly signal range is well defined in the terms of digital conversion.

Devices with digital input/output must be accompanied with some additional information about the format of the data to be exchanged with the device. Due to the nature of this formatting information, as well as the physical way of the communication with a device, the following device types are distinguished:

— binary input/output — information exchanged is binary; it points out only two states ("on" and "off"), to be represented in a computer system by logical values "true" and "false";
— three-way input/output — beside binary signal, a "pre-alert" signal is sent to warn about a possible abnormal functioning of a device (sometimes this subtype is also called a "parameterized" line);

Table 1. Summary of classifications of IoT devices

Type of IoT device	Signal encryption				Signal type	
	2	3	n	data frame	analogue	digital
Passive InfraRed	+	−	−	−	+	−
Photo-Electric Beams	+	−	−	−	+	−
Ultrasonic sensors	+	+	+	−	+	−
Microwave sensors	+	−	−	−	+	−
Vibration sensors	+	−	−	−	+	−
Smoke detectors	+	+	−	−	+	−
Heat detectors	+	+	−	−	+	−
Weather stations	−	−	+	+	−	+
Identification devices	+	−	−	+	−	+
Motion detection	+	−	−	+	−	+

— continuous values — possibly input/output value is only limited by its domain, such as the possible temperature range for a thermometer;
— digital data frame — some datagrams are defined to be exchanged with a device in order to precisely describe the data. Datagram format and content is usually producer-dependent.

A summary of all the above-mentioned classifications and IoT device types are presented in Table 1.

To meet different requirements and expectations related to the use of IoT devices, a proposed communication proxy must be generic enough to deal with all the above mentioned classifications and device types. In the proposed implementation, we decided to introduce software proxies, residential at given computer hosts, to be programmed at-request according to the required device functionality and the way of its use.

To define proxy functionality, Java [5] was chosen as a programming language. Each proxy service is implemented using Apache Axis 2 framework [6], as an independent Web service. Web service communication is based on SOAP protocol [7]. The main goal of a proxy it to: (1) hide details related with physical communication with a given IoT device, (2) track device addressability and connectivity, and (3) cache the current device state and update this information according to a certain caching strategy.

A proxy stands for a programmable broker between an IoT device and a computer-based system core. As such, this broker requires a proposal/adaptation of two communication methods: one for the physical communication of a proxy and a corresponding IoT device, and one for the communication with the core system. Below, a discussion is provided about the possible choices of some communication standards for both communication types mentioned above, with respect to possible device diversity.

3.2 Physical Communication of a Proxy and Corresponding IoT Device

Physical communication between a proxy service and a IoT device may be currently realized by means of the following channels:

— traditional COM (serial) and LPT (parallel) interfaces;
— wireless connections (IrDA, Bluetooth, WiFi);
— serial-transmission cables, such as USB, FireWire, SCSI bus, etc.;
— Local Area Network (LAN) connections (Fig. 1).

Each proxy service provides a separate Web service equipped with a given communication method, for a single device it is connected with (Fig. 2).

Fig. 1. A sample IoT device with LAN connection

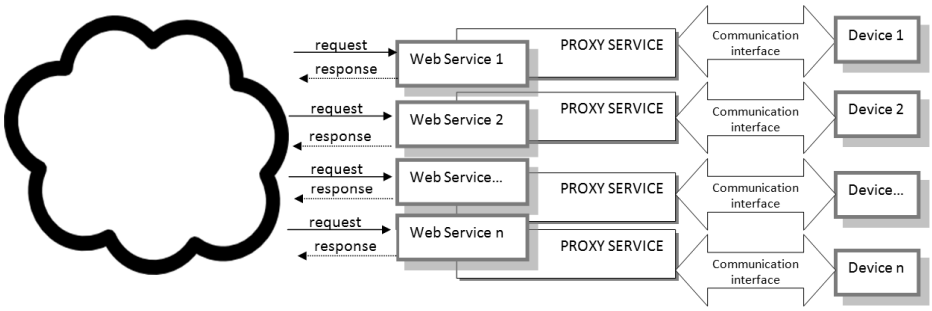

Fig. 2. Basic proxy architecture

In our current implementation, the COM interface has been chosen as a basis for proxy connections with real devices, such as manual switches and LEDs. For the purposes of the testbed, the following devices were connected:

— two two-state LEDs as output devices;
— two two-state monostable manual switches as input devices;
— one manual switch as an additional input device.

Beside on-line access to IoT devices (reading device state for input devices, and setting such state for output devices), the current proxy implementation stores and provides at-request a history of past invocations. This feature is not used directly by the ontology-based tested implementation; however, it was found quite useful for early-testing purposes.

3.3 Communication Proxy — Core Computer System

As was mentioned at the beginning of this section, on-line access to an IoT device via its proxy makes it possible to check/set its device state, and store/fetch its invocation history. Such information is to be accessed by a computer system by means of a proxy URL and a dedicated language implemented as a set of certain text-based commands. The commands are to be sent via HTTP PUT method to the proxy service identified by the URL proxy address, interpreted there, and changed to the appropriate invocations of real devices. A result of an interaction with a device is in turn sent as proxy HTTP response. For current testing purposes, a fixed set of commands, their parameters and possible responses have been defined, precisely the following four commands:

— ON — indicates that the device is turned on;
— OFF — indicates that the device is turned off;
— STATUS — returns the status of the device, the device may be turned ON or OFF;
— HISTORY — returns the history of the queries addressed in the past to the proxy.

To facilitate a definition of different proxies with similar functionality, a generic Java interface was defined to describe the communication protocol between this proxy and IoT device interconnected. This interface includes some methods used to establish a connection, further send and receive data, and finally close this connection. Each communication channel must implement this interface to separate the communication standard with the core computer system, and the one related to the physical communication with selected IoT device. Each time one plan to develop a new proxy, either an existing communication interface implementation

is chosen, or a designer is forced to develop certain Java code. Such an approach makes it possible to semi-automatize the process of development of new proxies — once a communication method, that was already used in the past, is chosen, there is no need to write a new separate Java code — one has to select and register only the code that is already available.

4 Design of Device Conglomerates

4.1 Overview of OSL

Device conglomerates are represented as ontology sub-trees. To facilitate the task of design and further interpretation of a conglomerate, the specialized Ontology Scripting Language OSL has been developed. This language, being a mixture of ontology-related entries, arithmetical and logical expressions, and links with Java code, is a unique, generic tool for the preparation and execution of any conglomerate of arbitrary complexity, and late-binding for device instances (i.e., given device implementation). It is worth to note that OSL descriptions are fully represented within a single ontology, thus it is not needed to apply any other programming language (except for Java code for device instances) during the process of design and execution (interpretation) of the conglomerates.

OSL has been developed to provide a generic way of definition of a behavior of complex sets of inter-connected devices. OSL conglomerates, however, are not just complex objects, with both a static structure and an interpretation flow. Instead, OSL conglomerates are characterized by a static composition, based on static definitions of device classes, and a dynamic interpretation flow, with lazy binding of device instances according to the current interpretation context.

The above mentioned device classes are divided into two basic groups: the classes related with given generic devices (so called *device classes*), and some pre-defined classes related with non-existing, artificial devices called *virtual-device* classes. Both groups are not fixed, i.e., they may be extended at any time by system designers. While the first group is used to contact and use real, existing devices of any type, the latter group is devoted to help the organization of a conglomerate by providing "composition" functionality similar to a composition of a program (a script): conditional choice, loops, variables, etc.

Both groups of classes may have instances, also represented as ontology entries. An instance of a device class is responsible for a contact with a given, single device, while an instance of a virtual-device class is a representative of a part of conglomerate composition (a counterpart of a program statement). The instances operate with fixed values of parameters, based on the current context (conglomerate invocation parameters).

The interpretation of a conglomerate starts with a dedicated virtual-device class names *Activity*. Each activity may be composed of several *Actions*. A single Action represents either a device class, or a virtual-device class. In the first case, a dedicated Java code is executed statically linked with the class, however, parameterized with some values of context parameters. In the latter case, according to the current context, several Actions may be involved linked with the currently examined *Action* by actual values of context parameters.

In the next sections, several OSL issues are described, starting from naming and value space, via arithmetical and logical expressions, to pre-defined virtual devices representing typical code statements such as a choice, a loop, a variable definition, a method call, etc.

4.2 OSL Naming Space

As described above, three basic kinds of entities are represented in OSL: values and their types, optionally linked to parameters, in turn providing a context of invocation, classes, and their instances.

OSL value stands for a single value of a given type. To represent values and their types, the following dedicated ontology classes are provided.

A TYPE ontology class is used to represent the optional types of values. Currently, the following predefined types are included in the ontology:

— Type_BOOLEAN (logical value TRUE or FALSE);
— Type_NUMBER (numerical value of Java double domain);
— Type_STRING (any string of characters, could be empty);
— Type_UNTYPED (representing any other type that the ones presented above).

This list is not closed, in particular new types may be included such as Type_GPS (geo-location), Type_DATE and Type_Time. The final content of the type list depends on the application area.

A VALUE is a pair of string-based representation of a given value, and suggested value type (as described above). A value may be empty. In such case, for some types (such as a number) the value is undefined (to be treated as non-existing).

A NAME is a string-based, non-empty name. Names are always of type Type_STRING. Names may be composed of any characters.

A PARAMETER stands for a pair composed of a name and a typed value. Parameters may be of a constant value (fixed parameters, such as PI=3.14... and E=2.71....), and with a value evolving over time (variables).

4.3 OSL Expressions

The OSL language enables the use of variables. Variables are to be searched by name, with values (represented by the ontology class VALUE) interpreted according to their types (ontology class TYPE). A variable value may be accessed inside an expression. The full expression grammar is provided in Appendix A, in BNF notation, being an input for Cup-of-Java YACC compiler [8].

Expressions are stored as variable values. An expression is interpreted, according to the above-presented grammar rules, each time a variable value containing this expression is to be accessed. For example, accessing a variable called "Addition" with its value equal to "2 + 2" results in a final expression value equal to "4".

To enable accessing current values of variables inside expressions, a variable name should be accompanied with parentheses "{" and "}". Current variable values are always fetched just before an expression evaluation. For example, the following variable set (XML-like syntax has been used for the clarity of the example):

```
<Variable name = ''TWO'', value = ''2''                      >
<Variable name = ''THREE'', value = ''3''                    >
<Variable name = ''EXPRESSION'' value = ``{TWO} + {THREE}''  >
```

results in value "5" once variable "EXPRESSION" is accessed. Note that the current values of variables composing the expression may change, providing different results for different computations.

To create a new variable, and to update its value, dedicated virtual-classes are used, these will be described in details later in this section. More examples of variable usage will be given as well.

4.4 OSL Entry Point

OSL ontology tree is going to be browsed starting from certain class of virtual devices called *Activity*. Similar to *Main* procedure in Java/C programming language, named *Activity* stands for an entry point for a conglomerate. Each Activity is composed of four elements:

— Activity name, being conglomerate identifier at the same time;
— Activity description, represented as a set of keywords;
— list of Activity formal parameters;
— an OSL Action (cf. next section) element starting an interpretation of the conglomerate.

As for current implementation, no dedicated possibilities of a conglomerate are stated as invocation (*Activity*) parameters, such functionality will be provided for the next version of OSL interpreter.

Activities may be invoked by:

— providing activity name (upper- and lower-case characters are not distinguished);
— providing a list of keywords, each of such keywords must be present in activity description, either as a whole word, or a part of a word.

Once invoked, either as a result of providing activity name as one of the invocation parameters, or comparing the sets of keywords and detecting all the activities fulfilling certain criteria, an *Activity* inspect the list of formal parameters. For each parameter of a given name, and existence of its counterpart of the same name is inspected in the invocation context. If found, a named variable is created, with context value standing for current variable value. If not found, and a value is declared for an activity formal parameter, this value is used to establish a variable, in a similar was as in the previous case. However, if a value is neither provided in the context of invocation, nor in the list of formal parameters, an error is signaled and the *Activity* is not interpreted at all.

After successful completion of all invocation parameters, an OSL element is determined declared as an *Action* of this *Activity*. An *Action* is a representative of a device class, or a virtual-device class. Predefined *Actions* are described in the next section. However, this list is not closed, as each designer is entitled to provide his/her own Actions for both real and virtual devices.

4.5 OSL Actions

Overview. Each OSL Action is composed of a certain Java code, and this fact is pointed out in the ontology by a link of each Action with certain name of a Java class implementing the predefined OSLStatement interface. The detailed definition of this interface is given in Appendix B. To facilitate the creation of different *Actions*, it is assumed that each class implementing OSLStatement interface is also inherited from the class OSLStatementEmpty described below (empty-statement implementation). This basic implementation class is equipped with several methods to access the ontology, manipulate variables, compute expression results, etc.

The predefined OSL *Actions* related to virtual devices are based on a typical shell programming language, interpreting the following statements:

— Empty statement;
— Variable declaration and updating statement;
— Choice (counterpart of IF-THEN-ELSE) statement;

— Loop (counterpart of DO-WHILE and REPEAT_UNTIL loops);
— Sequence (counterpart of a compound statement);
— Method-call (counterpart of method/procedure call) statement to call another Activity;
— Device statement, to implement an interaction with real devices.

The above presented sample virtual-device classes are described in details in the next subsections. Note, however, that the detailed behavior of any virtual-device class may be redefined at any time, once a Java code related with the class is changed. Thus, the below description should be treated as a reference only.

To facilitate reading the description of predefined virtual-device classes, all these descriptions are grouped into tables containing: a class name, related OSL statements, required and optional parameters, and fixed values of the parameters, if any (to be treated as constants or default values for variables).

Empty statement

Java class name:
`OSLStatementEmpty`
Statement purpose:
An empty statement (for doing nothing), being a base of implementation for all the other classes
Note: if a class name is omitted in the ontology definition of an *Action*, the empty statement is assumed by default. However, if the name of a non-existing class is provided in the ontology for a given *Action*, an error is indicated while interpreting this *Action*.
Required parameters:
—
Optional parameters:
—
Default values:
—

Assigning values to variables

Java class name:
`OSLStatementVariable`
Statement purpose:
Assigning or updating a value of a variable during the interpretation process of the ontology *Activity*.

Required parameters:

Ontology link `actionVariable_is_composed_of_variable_name`: variable name (any non-empty string of characters beginning with a letter), case is not matched

Ontology link `actionVariable_is_composed_of_variable_expression`: expression describing a variable value, composed according to the above-defined expression grammar. If any error occurs during expression parsing, the non-parsed result is assumed as an expression result. Empty value means removing the variable from the list of current variables of an interpretation process.

An access to a non-defined or removed variable is not treated as an error — instead, an empty variable value is assumed.

Optional parameters:

—

Default values:

—

Sequence statements

Java class name:

`OSLSequenceSerialized`
`OSLSequenceParallel`

Statement purpose:

Implementing a sequence of several *Actions*, either one-by-one (a serialized sequence, counterpart of a compound statement) or in parallel

Required parameters:

Ontology link `actionSequence_is_composed_of_activityN`: a pointer to an *Action* being interpreted as the next activity in the sequence; if empty, no activity is performed. N stands for any digit, representing an order of the sequence — up to ten *Actions* may be defined for a single sequence.

Optional parameters: Ontology link

`actionSequence_is_composed_of_next_action_sequence`:

a pointer to another sequence of up to ten *Actions*; if empty, no activity is performed

Default values:

—

Conditional statements

Java class name:

`OSLStatementConditional`

Statement purpose:

Implementing a choice between two children *Actions*, according to current

value of a condition to evaluate — counterpart of typical IF-THEN-ELSE program statement

Required parameters:

Ontology link `actionChoice_is_composed_of_condition`: condition expression, to be evaluated towards a Boolean value; any non-empty string represents the logical *true*, while an empty string — the logical *false*

Ontology link `actionChoice_is_composed_of_IF_statement`: a pointer to an *Action* being interpreted while the condition evaluates to logical *true*; if empty, no activity is performed

Ontology link `actionChoice_is_composed_of_ELSE_statement`: a pointer to an *Action* being interpreted while the condition evaluates to logical *false*; if empty, no activity is performed

Optional parameters:

—

Default values:

—

Loops

Java class name:

`OSLStatementLoopWhileDo`

`OSLStatementLoopRepeatUntil`

Statement purpose:

Implementing a loop in the interpretation process, with a looping condition evaluated either below entering the loop (a counterpart of a *WHILE-DO* program statement) or after completing the loop (*REPEAT-UNTIL*).

Required parameters:

Ontology link `actionLoop_is_composed_of_condition`: Condition expression, to be evaluated towards a Boolean value; any non-empty string represents logical *true*, while an empty string — logical *false*

Ontology link `actionLoop_is_composed_of_activity`: a pointer to an Action being interpreted while the condition evaluates to logical *true*; if empty, no activity is performed (the interpretation is looped, however, no *Action* is called)

Optional parameters:

—

Default values:

—

4.6 Device-related Statements

Java class name:
`OSLStatementFindDevice`
Statement purpose:
Implementation of the procedure of determination of certain devices, based on device name, keywords, description, etc. (cf. Section 3 devoted to define a proxy for device access and accessing/caching device state).
Required parameters:
There is no required parameter for this *Action*, however, at least one of the optional parameters must be provided in order to determine a device[1]. Once no parameter is provided, no device is found, however, an error is not signaled (just empty device list).
If several optional parameters are provided, all of them are treated as a filtering mechanism — all the searching conditions are verified parameter by parameter (the order of verification is not defined).
Optional parameters:
Ontology link `actionDevice_is_composed_of_device_name`: direct searching mode for a device — if any device of a given name is defined for the ontology, this device is provided as a result of searching
Ontology link `actionDevice_is_composed_of_keywords`: description-based searching mode for a device — all the devices with keyword sets containing all keywords from the parameter value are identified (case is not matched, keywords to compare may be partial words)
Ontology link `actionDevice_is_composed_of_location`: direct searching mode for a device — all the devices near certain geo-location are identified. The "near" function is defined for the expression evaluator
Default values:
For current implantation it is assumed that "near" means "at a distance of no more than 3 meters"

4.7 Modularization of OSL

One of the fundamental concepts in the design of OSL is related with language extensibility. New language elements — either implementing some generic operations or operations specific to a particular application domain — can be added to

[1]In the reality a device proxy is to be accessed rather than a real device (cf. Section 2), and URL of this proxy is kept in the ontology only tohether with the details about proxy configureation. However, to clarify the text, no distinction will be made between a device proxy an its corresponding real device.

the system, by providing certain ontology entries and Java-based implementation classes. The above-presented sample implementation classes are just examples of the system capacity and flexibility.

In particular, it is possible to define several cataloging, searching and determination methods for the calm-computing devices of a certain type and functionality. It is also possible to define complex device conglomerates, similar to the way of programming shell scripts for some operating systems and Web pages.

5 Interpretation of Device Conglomerates

A prototype implementation has been implemented to evaluate the behavior and basic functionality of the proposed system. This implementation is based on four main elements (Fig. 3):

— the ontology of device classes and instances, device conglomerates, service requirements, and conglomerate possibilities;
— the conglomerate interpreter, based on dynamic searching for device instances according to the current invocation context;
— a Java-code interpreter for the program code related to conglomerate classes, as described in the previous section;
— Web services acting as inter-mediators between the system core and SOA services.

The ontology was implemented with Protégé 3.4.1 [9] ontology editor and viewer, using OWL [3] description standard (Fig. 4).

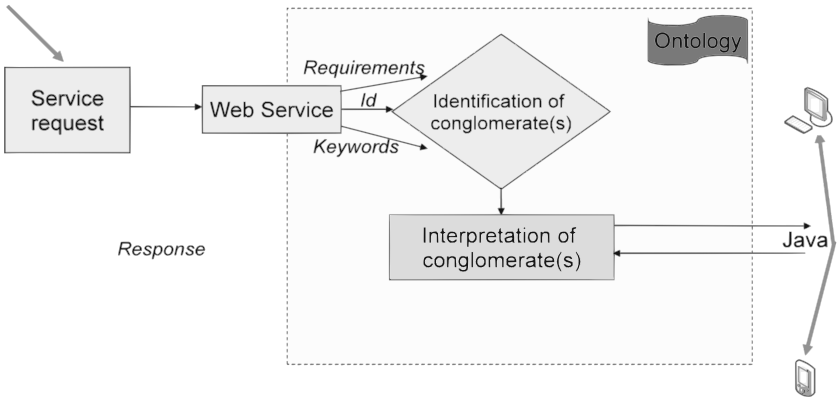

Fig. 3. Basic system architecture

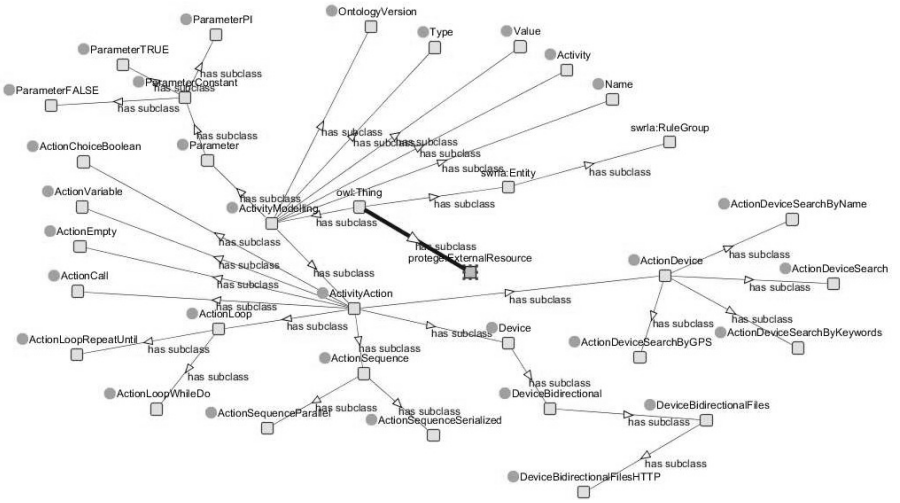

Fig. 4. Ontology of IoT devices for restricted device classes

For the test implementation bi-stable devices, accessible by RS232/485 and USB interfaces, were chosen. Such devices, as magnetic switches, PIR sensors, LED diodes, door-bold, etc. were tested. To extend some programming possibilities related to such simple, non-programmable devices, a proxy enhancement has been provided based on data representation in files and programmable proxies for certain devices, operating as Web Services. With this enhancement, it was possible to extend device descriptions by e.g., geo-location and keyword-based descriptions.

The ontology, as well as above-presented Java-related implementation classes, together with the simulator and device proxies, were used to implement a basic test environment (cf. Fig. 3). The basic system usage is the following. First, a SOA request is determined, extended by an invocation context (e.g., a geo-location of a person/device, access grants, etc.). The request is sent to a specialized Web Service, being an entry point to the system. The service is responsible for determining the invocation parameters, and further an Activity (or several Activities) fulfilling the request — based on activity name, keywords, description, etc., to be extracted from the invocation parameters and the context. For each determined Activity, its conglomerate is fetched, and the required parameters are verified. The conglomerate is interpreted, Action by Action, with necessary Java code implementing these Actions (as described earlier). During the interpretation, the given IoT devices are searched and contacted, based on the conglomerate structure and context parameters (e.g., a geo-location). To this goal, a device proxy and/or simulator may be applied to extend the device possibilities. A response from a device/proxy/simulator

is formatted and sent as request response. If more than one conglomerate/device is contacted for a single request, all the responses are concatenated.

6 Conclusions and Future Work

The idea of dynamic, searchable conglomerates of IoT devices to be accessed by SOA services has been proposed and implemented. These conglomerates are supposed to be used in ad-hoc and evolving environments, such as the ones related to body-networks, personal-area networking, and applications of private mobile devices in a public environment (offices, supermarkets, etc.). Our ontology-based approach made it possible to solve several problems observed in such application areas, including but not restricted to:

— the representation of technical details, the classification of possibilities of (possibly all) IoT devices in a unique and consistent manner;
— the efficient searching for the optimal device according to the requirements and restrictions of a context: a place, time, user, business model and SOA environment, etc.;
— the dynamic grouping of device conglomerates in JIT (just-in-time) mode; the conglomerates of several devices of different nature, communication and functionality making it possible to introduce a new business quality by means of programming of complex behavior according to contextual needs;
— the *ad hoc* and JIT programming of conglomerate (and device) behavior according to evolving situation;
— enhanced access to some handicapped IoT devices, usually unconnectable directly to any computer system, such as manual switches, lamps, locks, and other sensors and effectors traditionally operated manually, i.e., directly by humans or at least by specialized hardware-based systems.

A sample implementation of the system has been implemented as a proof-of-concept covering an ontology-based representation of mono- and bi-stable input/output devices, such as a LED lamp or a manual switch, enhanced by GPS localization and contextual access, with several business scenarios operated on different device conglomerates.

A future work is related to an evolution of the tested implementation towards a semi-commercial system covering more complex cooperation of IoT devices to fulfill sophisticated business needs of real users. It is expected that the following new problems will be addressed:

— the hardware connections and software representations of programmable IoT devices, including RFID readers, NFC communicators, movement- and presence-detectors, physico-chemical sensors of different environment parameters, etc. Particular proxy functionality, related to device possibilities, will be proposed and implemented as Java code. There is a clear need to design a way of ontology-based representation of such programmable-device capabilities;
— the class generalization for the above mentioned devices and their conglomerates, to be applied for certain application areas. An ontology should be extended towards more sophisticated methods of searching for and mutually connecting the devices;
— the hierarchical ontology description of device (conglomerate) capabilities;
— the automatic generation of the code of software representation (a proxy) of a device, for both real and virtual devices. It will be checked if the Java code related to a particular proxy may be automatically built (partially or totally) according to some device characteristics and access-method parameters, such as URL location. Once such an automatic proxy preparation is possible, it will be included as basic system-core functionality, with the necessary parameters and the pieces of Java code represented at the ontology level as part of device-type descriptions. Such an automatization will make it possible to use the system in highly evolving and dynamic environments, also in ad-hoc mode, and for incidental usage;
— the automatic cataloging, searching, and invoking of certain devices and their conglomerates, according to service requirements.

Technical enhancements towards a semi-commercial system include also the design of a specialized GUI interface for ontology management, as well as a subsystem for the management of device proxies, with a dedicated textual/GUI interface, connected with a dedicated interface for device observer/administrator.

Appendix A Expression Grammar in BNF-Like Notation

```
// JavaCup specification for a simple expression evaluator

package pl.pue.dit.OSLInterpreter;

import java_cup.runtime.*;
parser code
        {: public java.util.ArrayList allResults=new java.util.ArrayList(50); :}

// Terminals (tokens returned by the scanner).
        terminal SEMI, PLUS, MINUS, TIMES, DIVIDE, MOD, POW, GREATER, LESS, EQUAL,
                 GREATEREQUAL, LESSEQUAL, NOTEQUAL, NEAR, SEPARATOR, EMPTY;
        terminal UMINUS, LPAREN, RPAREN;
```

```
        terminal FUNCTION_E, FUNCTION_PI, FUNCTION_GPS,
                 FUNCTION_ABS, FUNCTION_EXP,
                 FUNCTION_LOG, FUNCTION_SQRT;
        terminal FUNCTION_ACOS, FUNCTION_ASIN, FUNCTION_ATAN, FUNCTION_COS,
                 FUNCTION_SIN, FUNCTION_TAN;
        terminal FUNCTION_ACOSR, FUNCTION_ASINR, FUNCTION_ATANR, FUNCTION_COSR,
                 FUNCTION_SINR, FUNCTION_TANR;
        terminal FUNCTION_TODEGREES, FUNCTION_TORADIANS;
        terminal         OSLParameter NUMBER;
        terminal         OSLParameter STRING;

        // Non terminals
        non terminal Object       expr_list, expr_part;
        non terminal OSLParameter expr;

        // Precedence
        precedence left EMPTY, SEPARATOR ;
        precedence left GREATER, LESS, EQUAL, NOTEQUAL,
                        GREATEREQUAL, LESSEQUAL, NEAR ;
        precedence left PLUS, MINUS;
        precedence left TIMES, DIVIDE, MOD, POW;
        precedence left FUNCTION_E, FUNCTION_PI, FUNCTION_GPS, FUNCTION_ABS,
                        FUNCTION_EXP, FUNCTION_LOG, FUNCTION_SQRT,
                        FUNCTION_ACOS, FUNCTION_ASIN,
                        FUNCTION_ATAN, FUNCTION_COS, FUNCTION_SIN, FUNCTION_TAN,
                        FUNCTION_ACOSR, FUNCTION_ASINR, FUNCTION_ATANR,
FUNCTION_COSR,
                        FUNCTION_SINR, FUNCTION_TANR;
        precedence left FUNCTION_TODEGREES, FUNCTION_TORADIANS;
        precedence left UMINUS, LPAREN;

        // The grammar
        expr_list ::= expr_list expr_part
                      |
                      expr_part ;
        expr_part ::= expr:e
                      {: parser.allResults.add(e); :}
                      SEMI
                      ;

        expr      ::= expr:e1 PLUS expr:e2
                      {: RESULT = new OSLParameter
                         (null, (new Double (e1.doubleValue() +
                         e2.doubleValue())).toString(), OSLParameter.NUMBER); :}
                      |
                            expr:e1 MINUS expr:e2
                          {: RESULT = new OSLParameter
                                 (null, (new Double (e1.doubleValue() -
                                 e2.doubleValue())).toString(),
                         OSLParameter.NUMBER); :}
                      |
                      expr:e1 TIMES expr:e2
                          {: RESULT = new OSLParameter
```

```
            (null, (new Double (e1.doubleValue() *
            e2.doubleValue())).toString(),
           OSLParameter.NUMBER); :}
|
expr:e1 DIVIDE expr:e2
    {: RESULT = new OSLParameter
            (null, (new Double (e1.doubleValue() /
            e2.doubleValue())).toString(),
   OSLParameter.NUMBER); :}
|
expr:e1 MOD expr:e2
    {: RESULT = new OSLParameter
            (null, (new Double (e1.doubleValue() %
            e2.doubleValue())).toString(),
   OSLParameter.NUMBER); :}
|
expr:e1 POW expr:e2
           {: RESULT = new OSLParameter
       (null,
       (new Double (java.lang.Math.pow(e1.doubleValue(),
       e2.doubleValue()))).toString(), OSLParameter.NUMBER);
     :}
|
expr:e1 GREATER expr:e2
    {: RESULT = (e1.compare(e2,
    OSLParameter.FUNCTION_GREATER)); :}
|
expr:e1 LESS expr:e2
    {: RESULT = (e1.compare(e2,
           OSLParameter.FUNCTION_LESS)); :}
|
expr:e1 EQUAL expr:e2
    {: RESULT = (e1.compare(e2,
           OSLParameter.FUNCTION_EQUAL)); :}
|
expr:e1 NOTEQUAL expr:e2
    {: RESULT = (e1.compare(e2,
           OSLParameter.FUNCTION_NOTEQUAL)); :}
|
expr:e1 NEAR expr:e2
    {: RESULT = (e1.compare(e2,
           OSLParameter.FUNCTION_NEAR)); :}
|
expr:e1 GREATEREQUAL expr:e2
    {: RESULT = (e1.compare(e2,
           OSLParameter.FUNCTION_GREATEREQUAL)); :}
|
expr:e1 LESSEQUAL expr:e2
    {: RESULT = (e1.compare(e2,
           OSLParameter.FUNCTION_LESSEQUAL)); :}
|
NUMBER:n
    {: RESULT = n; :}
```

```
                    |
                    MINUS expr:e
                        {: RESULT = new OSLParameter
                           (null,
                           (new Double (0 - e.doubleValue())).toString(),
                                        OSLParameter.NUMBER); :}
                    %prec UMINUS
                    |
                    FUNCTION_E
                        {: RESULT = new OSLParameter
                           (null,
                           (new Double (java.lang.Math.E)).toString(),
                       OSLParameter.NUMBER); :}
                    |
                    FUNCTION_PI
                        {: RESULT = new OSLParameter
                       (null, (new Double (java.lang.Math.PI)).toString(),
                       OSLParameter.NUMBER); :}
                    |
                    FUNCTION_ABS expr:e
                        {: RESULT = new OSLParameter
                       (null,
                       (new Double
                          (java.lang.Math.abs(e.doubleValue()))).toString(),
                       OSLParameter.NUMBER); :}
                    |
                    FUNCTION_GPS
                  LPAREN expr:e1
                          SEPARATOR expr:e2 SEPARATOR expr:e3 SEPARATOR expr:e4
RPAREN
                      {: RESULT = new OSLParameter
                       (new OSLGeoLocation
                               (e1.doubleValue(), e2.doubleValue(),
                       e3.doubleValue(), e4.toString())); :}
                    |
                    FUNCTION_GPS
                  LPAREN expr:e1 SEPARATOR expr:e2 SEPARATOR expr:e3 RPAREN
                      {: RESULT = new OSLParameter
                       (new OSLGeoLocation
                               (e1.doubleValue(), e2.doubleValue(),
                       e3.doubleValue())); :}
                    |
                    FUNCTION_GPS expr:e
                      {: RESULT = new OSLParameter(new
                       OSLGeoLocation(e.toString())); :}
                    |
                    FUNCTION_EXP expr:e
                        {: RESULT = new OSLParameter
                       (null,
                       (new Double
                          (java.lang.Math.exp(e.doubleValue()))).toString(),
                               OSLParameter.NUMBER); :}
                    |
```

```
FUNCTION_LOG expr:e
    {: RESULT = new OSLParameter
   (null,
   (new Double
        (java.lang.Math.log(e.doubleValue()))).toString(),
           OSLParameter.NUMBER); :}
|
FUNCTION_SQRT expr:e
    {: RESULT = new OSLParameter
   (null,
   (new Double
      (java.lang.Math.sqrt(e.doubleValue()))).toString(),
          OSLParameter.NUMBER); :}
|
FUNCTION_ACOS expr:e
            {: RESULT = new OSLParameter
(null,
   (new Double
   (java.lang.Math.toDegrees(java.lang.Math.acos(
   e.doubleValue()))))).toString(),
   OSLParameter.NUMBER); :}
|
FUNCTION_ACOSR expr:e
    {: RESULT = new OSLParameter
   (null,
   (new Double
      (java.lang.Math.acos(e.doubleValue()))).toString(),
           OSLParameter.NUMBER); :}
|
FUNCTION_ASIN expr:e
    {: RESULT = new OSLParameter
   (null,
   (new Double
   (java.lang.Math.toDegrees(java.lang.Math.asin(
   e.doubleValue()))))).toString(),
   OSLParameter.NUMBER); :}
|
FUNCTION_ASINR expr:e
    {: RESULT = new OSLParameter
   (null,
   (new Double
      (java.lang.Math.asin(e.doubleValue()))).toString(),
           OSLParameter.NUMBER); :}
|
FUNCTION_ATAN expr:e
    {: RESULT = new OSLParameter
   (null,
   (new Double
   (java.lang.Math.toDegrees(java.lang.Math.atan(
   e.doubleValue()))))).toString(),
   OSLParameter.NUMBER); :}
|
FUNCTION_ATANR expr:e
```

```
   {: RESULT = new OSLParameter
   (null,
   (new Double
      (java.lang.Math.atan(e.doubleValue()))).toString(),
           OSLParameter.NUMBER); :}
|
FUNCTION_COS expr:e
    {: RESULT = new OSLParameter
   (null,
   (new Double
      (java.lang.Math.cos(java.lang.Math.toRadians(
   e.doubleValue()))))).toString(),
   OSLParameter.NUMBER); :}
|
FUNCTION_COSR expr:e
    {: RESULT = new OSLParameter
   (null,
   (new Double
      (java.lang.Math.cos(e.doubleValue()))).toString(),
           OSLParameter.NUMBER); :}
|
FUNCTION_SIN expr:e
    {: RESULT = new OSLParameter
   (null,
   (new Double
   (java.lang.Math.sin(java.lang.Math.toRadians(
   e.doubleValue()))))).toString(),
   OSLParameter.NUMBER); :}
|
FUNCTION_SINR expr:e
    {: RESULT = new OSLParameter
   (null,
   (new Double
      (java.lang.Math.sin(e.doubleValue()))).toString(),
           OSLParameter.NUMBER); :}
|
FUNCTION_TAN expr:e
    {: RESULT = new OSLParameter
   (null,
   (new Double
           (java.lang.Math.tan(java.lang.Math.toRadians(
   e.doubleValue()))))).toString(),
           OSLParameter.NUMBER); :}
|
FUNCTION_TANR expr:e
    {: RESULT = new OSLParameter
       (null,
     (new Double
         (java.lang.Math.tan(e.doubleValue()))).toString(),
           OSLParameter.NUMBER); :}
|
FUNCTION_TODEGREES expr:e
    {: RESULT = new OSLParameter
```

```
                    (null,
                    (new Double (java.lang.Math.toDegrees(
                e.doubleValue()))).toString(),
                    OSLParameter.NUMBER); :}
            |
            FUNCTION_TORADIANS expr:e
                {: RESULT = new OSLParameter
                   (null,
                   (new Double (java.lang.Math.toRadians(
                e.doubleValue()))).toString(),
                   OSLParameter.NUMBER); :}
            |
            STRING:n
                {: RESULT = n; :}
            |
            EMPTY:n
                {: RESULT = new OSLParameter(""); :}
            |
            LPAREN expr:e RPAREN
                {: RESULT = e; :}
            ;
```

Appendix B Class Interface for OSL Statement Definitions

```
/**
 * Ontology Shell Language OSL - statement definitions
 */
package pl.pue.dit.OSLInterpreter;

import java.util.Map;
import pl.pue.dit.ontologyReader.OntologyParser;
import com.hp.hpl.jena.ontology.OntResource;

public interface OSLStatementInterface {
        public static final String LOG_PREFIX="    ";
        public static final String LOG_EMPTY="";
        public static final String
                LOG_NODE_NAME_ENTRY_POINT="LOG_ENTRY";
        public static final String LOG_NODE_NAME_END_POINT="LOG_EXIT";
        public static final String LOG_NEWLINE="\n";

        public static final String RESULT_PREFIX="    ";
        public static final String RESULT_EMPTY="";
        public static final String
                RESULT_NODE_NAME_ENTRY_POINT="RESULT_ENTRY";
        public static final String
                RESULT_NODE_NAME_END_POINT="RESULT_EXIT";
        public static final String RESULT_NEWLINE="\n";
```

```
        public static final String RESULT_VARIABLE_NAME="_result";
        public static final String
                RESULT_PREFIX_VARIABLE_NAME="_resultPrefix";
        public static final String
                LASTRESULT_VARIABLE_NAME="_lastresult";
        public static final String LOG_VARIABLE_NAME="_log";
        public static final String
                LOG_PREFIX_VARIABLE_NAME="_logPrefix";
        public void execute(
                 Map<String, String> actualParameters,
                 OntResource currentClassInstance, OntologyParser defaultParser,
                 long maxMomentInTime) throws OSLException;
        public void addToLog(
                 String newItem, boolean produceXMLStartingNode,
                boolean produceXMLEndingNode, String nodeName,
                boolean includePrefix, Map<String, String> actualParameters);
        public String getLog(
                 Map<String, String> actualParameters);
        public void addToResult(
                 String newItem, boolean produceXMLStartingNode,
                boolean produceXMLEndingNode, String nodeName,
                boolean includePrefix, Map<String, String> actualParameters);
        public void setLastResult
                (Map<String, String> actualParameters, String newItem);
        public String getResult (Map<String, String> actualParameters);
        public String getLastResult (Map<String, String> actualParameters);
        public void increaseLogPrefix (Map<String, String> actualParameters);
        public void decreaseLogPrefix (Map<String, String> actualParameters);
        public void increaseResultPrefix (Map<String, String> actualParameters);
        public void decreaseResultPrefix (Map<String, String> actualParameters);
}
```

Appendix C Abbreviations

GPS	Global Positioning System
GUI	Graphical User Interface
IoT	Internet of Things
LED	Light-Emitting Diode
NFC	Near Field Communication
OWL	Ontology Web Language
PDA	Personal Digital Assistant
PDF	Portable Document Format
PIR	Passive Infra-Red
RFID	Radio Frequency Identification
USB	Universal Serial Bus
XML	eXtensible Markup Language

References

1. Service Oriented Architecture — SOA. 2009. http://www-01.ibm.com/software/solutions/soa/.
2. Bonnet M. Personalization of web services: opportunities and challenges. Ariadne 28, 2001.
3. OWL Web Ontology Language overview. 2009, http://www.w3.org/TR/owl-features/.
4. The Internet of things. A critique of ambient technology and the all-seeing network of RFID. 2009, http://www.theinternetofthings.eu/what-is-the-internet-of-things.
5. JSEE Platform. 2009, http://java.sun.com/java.
6. Web services — Axis. 2009, http://ws.apache.org/axis/.
7. SOAP specifications. 2009, http://www.w3.org/TR/soap/.
8. CUP parser generator for Java. 2009, http://www.cs.princeton.edu/~appel/modern/java/CUP/.
9. Protégé ontology editor overview. 2009, http://protege.stanford.edu/.
10. RFID J. 2009, http://www.rfidjournal.com/.
11. GPS. 2009, http://en.wikipedia.org/wiki/Global_Positioning_System.

Design of SOA-Based Distribution System

Jan Kwiatkowski, Mariusz Fras, Marcin Pawlik, Dariusz Konieczny, and Adam Wasilewski

Institute of Informatics
Wrocław University of Technology,
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland
{jan.kwiatkowski, mariusz.fras, marcin.pawlik, dariusz.konieczny, adam.wasilewski}@pwr.wroc.pl

Abstract. In recent years the evolution of software architectures led to the rising prominence of the Service Oriented Architecture (SOA) concept. This architecture paradigm facilitates building flexible service systems. The services can be deployed in distributed environments, executed on different hardware and software platforms, reused and composed into complex services. In this chapter an efficient computational and communication resources allocation for complex service requests is proposed. The requests are examined in accordance to the SOA request description model. The functional and non-functional request requirements in conjunction with the monitoring of execution and communication links performance data are used to distribute requests and allocate services and resources.

Keywords: Service Oriented Architecture, service request distribution, resource allocation

1 Introduction

The Service Oriented Architecture (SOA) paradigm facilitates building flexible service systems that can be deployed in distributed environments, executed on different hardware and software platforms, reused and composed into complex services. The resulting flexibility of systems based on the SOA concept comes with the cost of rising complexity. New computational architectures are built utilizing the power of classical processors together with new developments. Classical processors, used individually or grouped in clusters and grids, are cooperating with novel architectures: general-purpose graphics processing units (GPGPU), filed-programmable gate arrays (FPGA) or general-purpose multiprocessor hybrid ar-

chitectures [6, 4]. Heterogeneous resources can be connected to form a distributed system offering a cloud of services, cooperating or competing for the possibility to fulfil user demands. Different network topologies and architectures, from simple local installations to complicated global scale systems can be utilized to provide the backbone for the transfer of necessary data.

The growth of the Internet network have caused that it is as an integrated part of many current information systems. The communication resources of the systems which rely on the Internet links are hard to analyze and control with respect to quality parameters. The design of the distribution system proposed in this chapter assumes utilization of just such communication resources. In order to guarantee proper control of distribution of the client requests, the architecture of measurement, modeling and estimation of values of communication links parameters is introduced in the section 3.

An efficient system architecture has to be designed to ensure users satisfaction and the system components efficient utilization. In this chapter a two-tier architecture for computational and communication resources is proposed. The system is built as a federation of independent execution systems. The execution systems hide their internal complexity by offering a common SOA interface to their internal resources. The service requests distribution system analyses each of the execution systems in the context of the incoming user request requirements. Utilizing the measurements of the systems and communication links performance, the request broker selects the execution system capable of performing the user task. The user request is sent to the matchmaking module inside the execution system, which selects the resources to be allocated to fulfil the request. Each of the execution systems works in an autonomous manner, ensuring efficient local resources utilization. This structure is designed to offer a SOA service abstraction on the highest level with efficient resources utilization performed inside the execution systems.

The chapter is organized as follows. Section two describes the top-level service request distribution system design. In the third section, we provide the description of the request broker system designed to analyze incoming requests and select an execution system appropriate for the task. In the next section the execution monitoring system is described — working in each of the execution systems to gather the data necessary to ensure an efficient resource utilization. The section five describes the architecture of a matching module, implemented in each of the execution locations to select the computational resources for the incoming request. The final section concludes the paper and describes the future work plans.

2 Service Request Distribution System

2.1 General Infrastructure of the System

The information systems designed with the SOA paradigm are working both in local and wide area networks, particularly built with Internet and GRID resources. For a client that requests a service there is no need to care about the localization and execution meanings of the service he wants to use. However, the services are well described in the system and their localizations are known. The set of available services can dynamically change, however this process is not frequent, and usually the set of the locations of a given service is constant for some period of time [9].

The paradigm of SOA says that a given service can be achieved by different service providers. Because the parameters (especially effectiveness) of service provider's servers as well as communication links to them (communication costs) can be different, the quality of service delivery (especially the time of delivery) can be significantly different too [13]. Delivery of services according to client's demands as well as satisfying formulated system's requirements is a key issue for current network service systems. Taking into account the distribution of services and the characteristic of the Internet network, the proper architecture of service delivery may significantly increase the quality of services built with the SOA paradigm.

The general goal of the presented system is the management of request and service execution by allocating the proper resources in order to fulfil formulated demands for execution of services. The idea of the proposed service request distribution system is to manage clients' requests at two levels. Near the client at the edge of the Internet, the service brokers B_n (Fig. 1) distribute the service requests to the known execution systems of service providers sp_m. Knowing the demands of a service execution, they distribute the corresponding requests to the proper execution system. It is assumed that an execution system can be a complex heterogeneous computation center. At the service provider layer, an execution system delivers the desirable information to brokers and guarantees the requested service execution parameters for allocating the proper resources.

The three possible scenarios for the functioning of the system are the following:

— each broker works separately serving nearest clients, and each execution system works separately having no information about the others;
— all brokers cooperate each other in order to globally serve all clients distributed in the Internet network;
— all brokers and servers work as a complex, cooperative system managing resources from a global point of view.

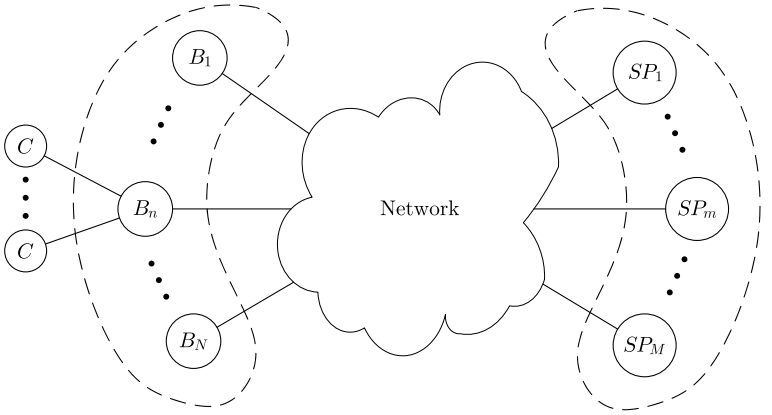

Fig. 1. General infrastructure of service request distribution system

According to these scenarios different goals can be formulated and different solutions can be obtained. In general, the goals can be divided into two areas: the control of the service execution according to the demands from the client point of view (e.g. guaranteeing the quality of the requested services specified by the client), and the control of service execution according to demands from the execution infrastructure point of view (e.g. guaranteeing load balancing demanded by the system).

In this chapter, the case of the first scenario i.e. an architecture of the system with a service request distribution by non-cooperative brokers to non-cooperative execution systems, is considered.

The assumptions for proposed requests distribution system are the following:

— the client of the broker is a system delivering complex services $cs(i) \in CS$, where CS is a set of possible complex services, $i \in \langle 1, ..., I \rangle$, aggregated from atomic services $as(j)$, $j \in \langle 1, ..., J \rangle$, available in execution systems distributed in the Internet network (actually the broker is a part of the complex service delivery system);
— the set $SP = \{sp_1, sp_2 ... sp_m, ... sp_M\}$ of execution systems is known and it constant for some considered period;
— the set of atomic services $AS = \{as(1), as(2), ... as(j), ... as(J)\}$ is known, can be requested by the client, and is available in the execution systems $sp_m \in SP$;
— delivered services are considered as the same if they offer the same functionality;
— a given atomic service $as(j)$ can be available in several different localizations (execution systems) — a concrete atomic service at a given localization is called an instance of atomic service;

— the considered resources (from the broker point of view) that affect service execution parameters, are the communication links and the execution systems.

The goal of the broker is to distribute the requests for atomic services, which constitute the requested complex services (to allocate communication resources) in order to fulfil the formulated demands for service execution. The goal of the execution system is to manage the execution of requested atomic services in order to fulfil the same demands.

2.2 Problem Formulation

A complex service cs is composed with more basic services, especially atomic services $as(k)$,$k \in \langle 1, ..., K \rangle$, which cannot be partitioned thereafter. An execution of a complex service is an execution of all its components (atomic services). Execution dependences exist between atomic services and – some atomic services cannot be executed before some others. Basically, a complex service can be described as a directed graph of atomic services called its implementation graph $GA\ \langle V, A \rangle$, where $V = \{as(1), as(2), ...as(k), ..., as(K)\}$, and $A \subset V \times V$ is a set of pairs defining edges representing execution dependencies between atomic services (vertices), as the example in Figure 2. In this example the execution of a complex service starts with the atomic service $as(1)$, followed by $as(2)$ and $as(3)$ executed independently. The atomic service $as(4)$ can be executed only after completion of $as(2)$ and $as(3)$. The execution terminates with the service $as(6)$. It is assumed that the graph of a complex service is simple and acyclic.

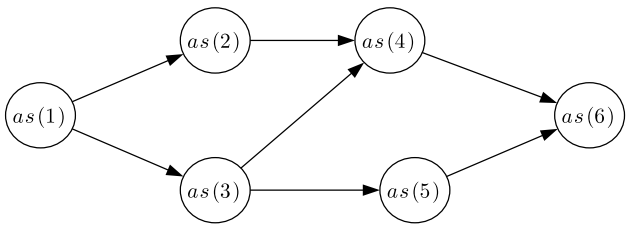

Fig. 2. An Example of implementation graph of complex service

Let $CS = \{cs(1), cs(2), ...cs(i), ..., cs(I)\}$, $i \in \langle 1, ..., I \rangle$, be the set of possible complex services supplied by the complex service delivery system, and $AS = \{as(1), as(2), ...as(j), ..., as(J)\}$, $j \in \langle 1, ..., J \rangle$, be the set of atomic services from

which complex services can be aggregated. The atomic services are available (localized) in the constant set $SP = \{sp_1, sp_2, ...sp_m, ..., sp_M\}$, $m \in \langle 1, ..., M\rangle$ of execution systems (service providers). The execution system sp_m delivers the set $AS(sp_m) = \{as(m_1), as(m_2), ..., as(m_p), ..., as(m_P)\}$ of atomic services: $as(m_p) \in AS$, i.e. the set $IS(sp_m) = \{is(m_1), is(m_2), ..., is(m_p), ..., is(m_p)\}$ of instances of atomic services. IS is the set of all instances of atomic services $IS(sp_m) \subset IS$.

The instance of service $is(m_p)$ is characterized by the functional parameters $\varphi(is(m_p))$, which correspond to the specific atomic services implemented by the instance $(\varphi(is(m_p)) = \varphi(as(m_p)))$, and the set of non-functional parameters $\psi(is(m_p)) = \left[\psi^1_{m_p}, \psi^2_{m_p}, ..., \psi^f_{m_p}, ..., \psi^F_{m_p}\right]$, that may be different for two different instances of the same atomic service. Examples of non-functional parameters are: the financial cost, the level of security, the completion time of service execution, etc. Finally, the given instance $is(m_p)$ is defined with the triple $\langle sp_m, \varphi(is(m_p)), \psi(is(m_p))\rangle$.

Let $GA_n\langle AS_n, ES_n\rangle$ be the implementation graph of a complex service requested at the moment n. The set AS_n is a subset of the set AS containing all atomic services necessary to accomplish the complex service cs_n requested at the moment n: $AS_n = \{as(n_1), as(n_2), ..., as(n_k), ..., as(n_{K_n})\} \subset AS$, $k \in \langle 1, ..., K_n\rangle$, where K_n is the number of atomic services (number of vertices in the implementation graph). For each implementation graph, there is an equivalent graph $G_n\langle IS_n, ES_n\rangle$, with $IS_n = \{IS_{n1}, IS_{n2}, ..., IS_{nk}, ..., IS_{nK_n}\} \subset IS$. The set IS_{nk} is the set of all instances of atomic service $as(n_k)$, i.e. $IS_{nk} = \{is(n, k, 1)$, $is(n, k, 2)$, ..., $is(n, k, l)$, ..., $is(n, k, L)\}$, and it corresponds to the set $SP(as(n_k)) = \{sp_m : as(n_k) \in AS(sp_m)\}$, i.e. the set of all execution systems that delivers instances of atomic service $as(n_k)$. The graph $G_n\langle IS_n, ES_n\rangle$ defines different execution graphs for the implementation graph $GA_n\langle AS_n, ES_n\rangle$.

Suppose, the implementation graph of the complex service cs_n presented in Figure 3, and $IS_{n1} = \{is(n, 1, 1), is(n, 1, 2)\}$, meaning that the atomic service $as(1)$ is delivered in two different locations (has two instances), and $IS_{n2} = \{is(n, 2, 1), is(n, 2, 2), is(n, 2, 3)\}$, meaning that atomic service $as(2)$ is delivered in three different locations (has three instances). The complex service can be executed in six different ways as shown in Figure 4 (index n is omitted).

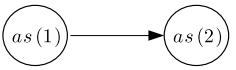

Fig. 3. A simple example of implementation graph of complex service

The choice of a specific execution path is a problem that is involved when specifying demands on non-functional parameters of the services. Let $\psi(cs(i))$ be the non-functional parameters of a complex service, and X_n be the request for the complex service cs_n at the moment n. It is assumed that the request X_n contains the implementation graph of the requested complex service cs_n and the formulated demands on non-functional parameters of the service cs_n: $X_n = \langle GA_n\langle AS_n, ES_n\rangle, SLA_{nf,n}\rangle$, where $SLA_{nf,n} = \psi(cs_n)$.

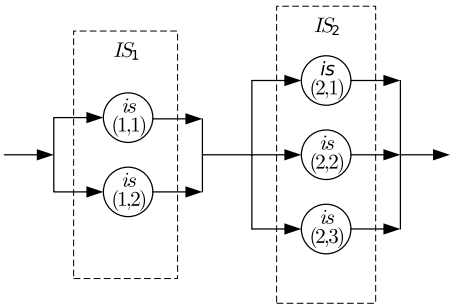

Fig. 4. Possible paths of execution of an example complex service

The non-functional parameters of the complex service $\psi(cs_n)$ correspond to the non-functional parameters of the component atomic services $\Psi(cs_n) = \{\psi(as(n_k)\}$ and to non-functional parameters of the instances of atomic services $\{\psi(is(n_k)\}$.

Non-functional parameters of an atomic service instance can be static (for assumed period of time — e.g. the level of security) or dynamic (variable in time — e.g. completion time of execution). Knowing the parameters $\{\psi(is(n_k)\}$ it is possible to perform different procedures for fulfiling $\psi(cs_n)$.

Taking into account that considered information systems are distributed systems, two components have an impact on the non-functional parameters: communication environment and execution systems. Thus, one of the main goal of a service request distribution system is to determine the execution graphs of complex services, that needs to take into account the localizations of the requested atomic services, i.e. instances of atomic services. This goal is formulated as:

$$GA_n\langle AS_n, ES_n\rangle \rightarrow G_n^*\langle IS_n^*, ES_n\rangle$$

where: $IS_n^* \underset{min}{\leftarrow}_{k=1\ldots K_n} h(\psi(is(n_k)))$, what denotes the problem of finding the appropriate instances of atomic services that should be requested in order to fulfil the formulated demands on non-functional parameters. $G_n^*\langle IS_n^*, ES_n\rangle$ is the optimal execution graph of complex service, and $h(\psi(is(n_k)))$ is a formulated criterion for fulfiling $SLA_{nf,n}$.

The proposed two-layered schema for the management of complex services execution is presented in Figure 5. The task of communication resources allocation is performed at the upper layer. It is done taking into consideration not only communication parameters but also service execution parameters. At this level, the execution graph $G_n^*\langle IS_n^*, ES_n\rangle$ of complex service is determined. Then, requests for instance of atomic service $x_{nk} = (is(n_k), \psi(is(n_k))$ (containing demands on non-functional parameters) are sent to the lower layer. The goal of the lower layer is the

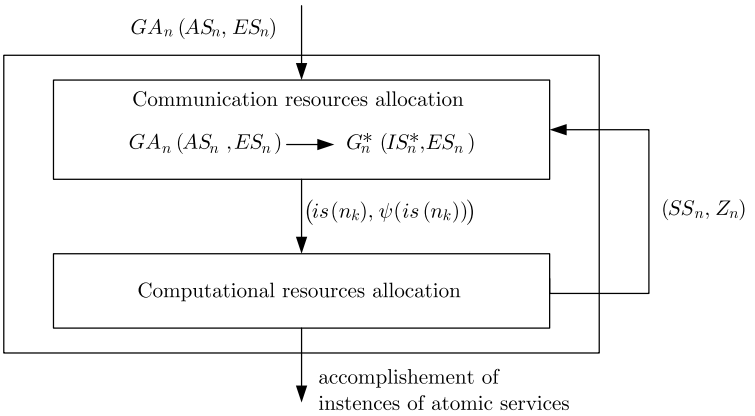

Fig. 5. Two-layered of management of execution of complex services

proper execution of atomic service instances in order to guarantee the formulated non-functional parameters of the requested service instance.

The upper layer is supplied with the execution systems status data $SS_n = \{SS_n(sp_1),\ SS_n(sp_2),\ ...,\ SS_n(sp_m),\ ...,\ SS_n(sp_M)\}$ at the moment n, and the actual non-functional parameters of executed atomic services instances $Z_n(cs_n) = \{Z_n(is(n_1)),\ Z_n(is(n_2)),\ ...,\ Z_n(is(n_k)),\ ...,\ Z_n(is(n_{K_n}))$.

The depicted architecture of our service request distribution system introduces the following problems:

— the estimation/forecasting of the dynamic non-functional parameters of atomic service instances;
— the determination of the execution graphs of complex services fulfiling $SLA_{nf,n}$ or more global criteria;
— the estimation of a service execution time by the execution systems that implement this service;
— determining the execution system that fulfils the non-functional requirements from $SLA_{nf,n}$ with an efficient utilization of resources.

A typical, and one of most important, non-functional parameter is the completion time of a service execution. In this case, the execution graph must be built on the basis of estimated or forecasted completion times. The lower layer receives requests with a specified execution time that must be secured with a proper management of the computational resources of heterogeneous execution systems. Next sub-chapters describe the proposed solutions to different aspects of this problem.

3 The Request Broker Architecture

Our general model of service requests distribution system assumes that between the client and the edge of the Internet there is a special node called the service request broker which manages distribution of requests for atomic services. For a given atomic service $as(n_k)$ there is a known set $SP(as(n_k)) = \{sp_m : as(n_k) \in AS(sp_m)\}$ of M execution systems that delivers instances $is(n, k, m)$ of atomic service $as(n_k)$ (Fig. 6). The services are delivered through M communication links s_m. It is assumed that the distance between the client and the broker is negligible — basically the broker is a part of the system that delivers complex services.

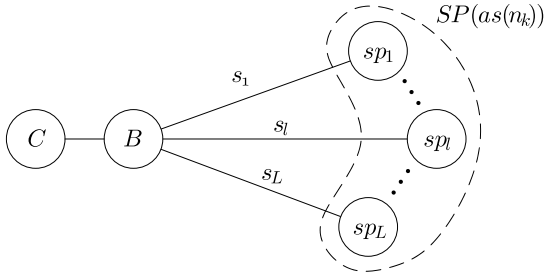

Fig. 6. Distribution model for atomic service $as(n_k)$

The service delivery time consists of two components: the data transfer time t_T between the client request and the server reply, and the service request execution time t_O (preparation of the reply data by the execution system processing the request). In general, even for a given atomic service, these two values are not constant. They depend on the dynamic change of communication link parameters as well as the current state of execution system (e.g. its load).

The aim is to distribute a client request to such servers that the service delivery time is the best according to formulated criteria (e.g. the smallest delivery time). It requires the estimation of the completion times of the atomic services, which consists of the completion times of communication and service execution at execution system. Hence, the broker has to support the following functionalities:

— the monitoring of client requests serving;
— the prediction of selected communication link parameters;
— the estimation of the transfer times and execution times of all instances of atomic services, on the basis of adaptive models of communication links and execution systems;

— the decision making where to distribute requests according to time based criteria.

In the Figure 7 we propose the general architecture of the broker that fulfils all functionalities necessary to determine, for every request arriving at the moment n, the destination server which should serve this request.

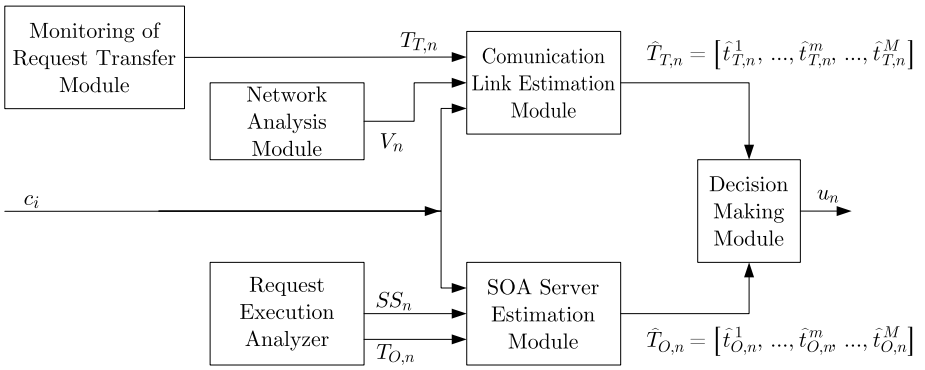

Fig. 7. General architecture of the SOA request broker

The decision u_n of the choice of one out of M execution systems (hereafter index k is omitted) is performed on the basis of the vector $\hat{T}_{T,n}$ i.e. the vector of the estimated transfer times $\hat{t}^m_{T,n}$ to each server m, and vector $\hat{T}_{O,n}$ i.e. the vector of estimated execution times $\hat{t}^m_{O,n}$ at each execution system m, $m \in \langle 1, ..., M \rangle$. These times are estimated with the aid of models of execution systems and communication links built for every atomic service $as(i)$. The term c_i denotes the identifier of the service $as(i)$. If the client of the broker is not a service system that specifies the component atomic services in the request, some classification procedure ought to be applied.

The key components of the broker are the estimation modules of communication links and execution systems. They estimate the corresponding times according to fuzzy-neural models of links and execution systems similar to [2]. Modules are built of M models of links or execution systems respectively, which work as controllers.

The input of the communication link model (Fig. 8) is an identifier of the atomic service c_i that occurs at the moment n, the vector V_n of communication link parameters and measured real transfer times $\hat{t}^m_{T,n}$ of prior requests (before the moment n). The vector V_n consists of two parameters: forecasted link latency $\hat{t}_{TCPC,n}$

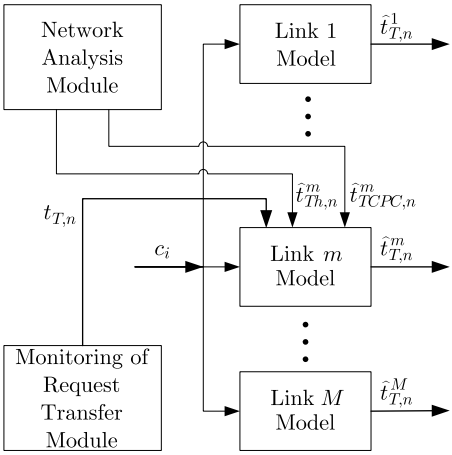

Fig. 8. Communication Link Estimation Module

(namely TCP Connect Time) and forecasted link throughput. These two parameters are derived from the periodic measurements of latency and link throughput $\hat{t}_{Th,n}$, and using time series analysis based prediction algorithms.

The execution system model is very similar. The input of the model is the identifier of atomic service c_i that occurs at the moment n, the execution system state parameters vector $SS_n(sp_m)$ of the server m, known at the moment n, and the completion times of requests executions till the moment n. The vector $SS_n(sp_m)$ may consists of different parameters characterizing the current state of the system. We have taken into account two: the load of the server and its efficiency. The output of each model block of the module is the estimated execution time of the request at each execution system.

Each model (link and execution system) is built for each destination and for each atomic service — namely for each instance of atomic service. It is designed as a fuzzy-neural controller (Fig. 9) which consists of a 3-layered neural network similar to [2, 10, 5]. We describe shortly the communication link model (the execution system model is analogous). The first layer of the network constitutes the fuzzyfication block. For the link model, the inputs are the forecasted link latency $\hat{t}_{TCPC,n}$ at the moment n, and the forecasted link throughput $\hat{t}_{Th,n}$. The outputs of the fuzzyfication block are the grades of membership of latency $\mu_{t_{TCPC}}$ in defined fuzzy sets for latency, and the grades of membership of throughput $\mu_{t_{Th}}$ in defined fuzzy sets for throughput. The logic block computes the grades of membership μ_R with the rules that are in the format proposed in [2]:

$$\text{IF } t_{Th} = Z_{Th,a} \text{ AND } t_{TCPC} = Z_{TCPC,b} \text{ THEN } y = t_{T,c}$$

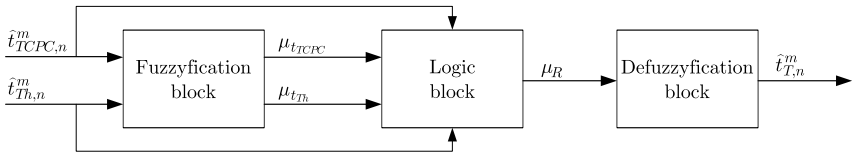

Fig. 9. Communication link model as a fuzzy-neural controller

where: Z — fuzzy sets for throughput (Th) and latency ($TCPC$), t_T — estimated transfer time (output y), a, b, c — indexes of appropriate fuzzy sets and output.

The rule is fired when $\mu_R = \mu_{t_{TCPC}} * \mu_{th} > 0$. The parameters of fuzzy sets, as well as the output y are tuned during a learning process using the back propagation method [7] on the basis of the measured actual times of request transfer time and service execution time.

After the defuzzyfication (using weight method), the estimated transfer times of the request are derived:

$$\hat{t}^m_{T,n} = \sum y_{n,c} * \mu_{R_{n.c}}$$

where: c — index of rule, C — number of rules (output fuzzy sets), $y_{n,c}$ — output (the transfer time according to rule c), $\mu_{R_{n.c}}$ — grade of membership of rule c.

4 Execution Performance Measurements

The service distribution system and the request broker described in the previous sections for correct functioning require information about parameters important for the fulfilment of the user requirements and the efficient utilization of resources. A monitoring system called *rsus* [11] was designed to perform the necessary measurements. The set of measured values can be extended to include additional parameters. At present, the two most important factors are analyzed — the execution time and the resource utilization.

The data gathered by the rsus monitoring environment is utilized by the service request distribution system. The wallclock time, representing the aggregated time of a request execution, and the memory utilization measurements can be processed directly by the request distribution system. The resource utilization efficiency is interpreted locally, in each of the execution environments, to provide the information necessary to ensure the appropriate resource utilization. For each of the requests coming to the execution environment, the matching module selects which resources (*atomic service concrete realization*) are to be used for processing. First

the selection is made based on the non-functional properties of the incoming request. If the cardinality of the resulting set is larger than 1, the available concrete realizations is selected based on the efficiency of previous executions. In the next subsection the monitoring system architecture is described. The method designed to perform efficiency execution evaluation is described in the following section, together with the experimental results of the performed tests.

4.1 Monitoring System Architecture

The rsus monitoring system is a distributed environment dedicated to perform the measurements of the execution parameters of sequential and parallel applications. The measurements are performed in a non-invasive, low-overhead, external manner, not requiring any modifications to the service being measured. The monitoring system can work off-line, gathering the data after the application execution is over, or in an on-line fashion, with the measurement data periodically sent to the control application during the monitored application lifetime. An independent rsus environment installation is present inside every execution system.

The rsus environment is designed to work with the linux operating system - the most popular OS in the HPC applications. The modular design makes it possible to port some or all of the rsus components to other operating systems or environments. To keep the measurement overhead as low as possible, rsus utilizes the data about the CPU time and memory consumption gathered by the operating system (and accessible through the /proc directory hierarchy). The architecture of the rsus system is depicted in the Figure 10.

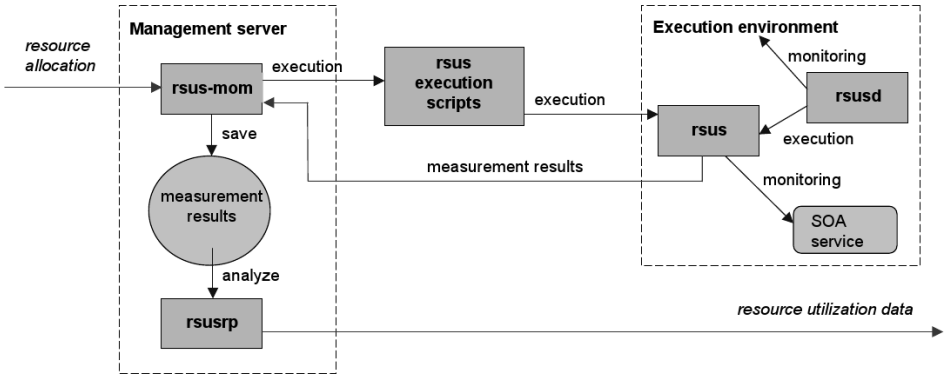

Fig. 10. Rsus system architecture

The rsus environment is composed of a set of applications. The rsus-mom control application controls the rsus environment and gathers the measurement results from the distributed measurement applications. The information acquired is saved on the hard drive for further processing.

The rsus measurement application is present in every monitored execution environment. It is capable of starting a monitored application by itself, when the appropriate command is issued by the rsus-mom application. It can also monitor an already started application utilizing appropriate system calls and the data accessible from the operating system. The rsusd application is employed to monitor the system activity and start the rsus application when an application to be monitored appears in the system.

The rsus environment can be integrated with various execution environments. Currently a set of appropriate execution scripts was prepared for environments compatible with the Portable Batch System (PBS).

4.2 Parallel Execution Efficiency Evaluation

The functioning of the resources allocation system presented in the chapter requires the knowledge of the application execution efficiency. Traditionally in many cases the execution relative efficiency evaluation is performed, comparing the execution time of the sequential and parallel execution of the application. Being a relative measure, the relative efficiency requires that the application is run on a single processor to compare its runtime with the parallel execution. Oftentimes it would be impractical or impossible to perform such additional executions of parallel applications in the designed SOA system. A new measurement technique, called Execution Time Decomposition (ETD) analysis [12], had to be designed to overcome this problem. Basing on the decomposition of the execution time between the time devoted to the computations and the overhead time, the ETD analysis can be carried out, providing not only insight into the application execution but also the possibility to perform fast and accurate performance estimations.

The time of a parallel algorithm execution can be defined as composed of the computations time, communication time and the time processors spend idle waiting for the others to finish their computations. In the case of MPI-based applications, two additional time periods can be introduced — the time before the MPI_Init function call is finished, and the time from the start of the call until the application end.

For the sake of the following analysis the concept of a *fundamental computation time* is defined as the execution time of the computational operations present in the algorithm when it is executed on a single processor. This value describes the time of operations always present in the parallel algorithm, no matter how many processors are used.

The communication time is defined as the time processors spend sending the data to other processors or (if the synchronous transfer is used) waiting for other parties to receive the data sent. The idle time is the time processors spend waiting for the needed data to arrive.

Two different methods of efficiency measurement and approximation are used and described below — the method based on an external measurement of the CPU time, and the method based on the communication time data gathered from the MPI profiling. Using both methods on every processor the wall-clock time is measured and the measurements are performed to estimate the amount of fundamental time.

In the case of popular MPI profilers (e.g. mpiP), the wall-clock time is measured with the exclusion of the initialization and finalization times, and the fundamental time is approximated utilizing the difference between the wall-clock time and the time spent on MPI-related operations. CPU-time based measurements utilize the wall-clock time measurements and the fundamental time is approximated by the CPU-time measured.

An MPI-based application performs a call to the MPI_Init function before any other MPI-related functions. At the end of the MPI-related section the application calls the MPI_Finalize function. Usually this call is also the last one in the application but it is possible to perform an arbitrary number of non-MPI operations after this call. In the case of the MPI profiler the measurements start after the MPI_Init call and terminates before the MPI_Finalize call resulting in the inability to measure the initialization and finalization times. The measurements erroneously include in overhead time all the computational operations that may be present in MPI functions and should be attributed to the fundamental time. On the other hand additional computations induced by parallelization and performed outside the MPI functions may be erroneously considered to be a part of the fundamental time.

In the case of CPU-time based measurements the wall-clock time is measured correctly. The fundamental time measurement erroneously includes additional operations such as additional computations induced by parallelization, a part of operations needed during communication and computational operations occurring before MPI_Init call and after MPI_Finalize call.

The wall-clock time and fundamental time measurements as performed by ETD methods are depicted in the Figure 11. The first two lines denote the actual time values, while the next lines represent the values measured by MPI profilers and CPU-time based measurement tools accordingly.

The formalization of the concepts discussed above can be performed as follows. Let τ_{wall}, τ_{comp} and τ_{comm} denote respectively the application wallclock, computation and communication times. In the contemporary operating systems the τ_{wall} and τ_{comp} time values are continuously monitored (e.g. to perform task scheduling).

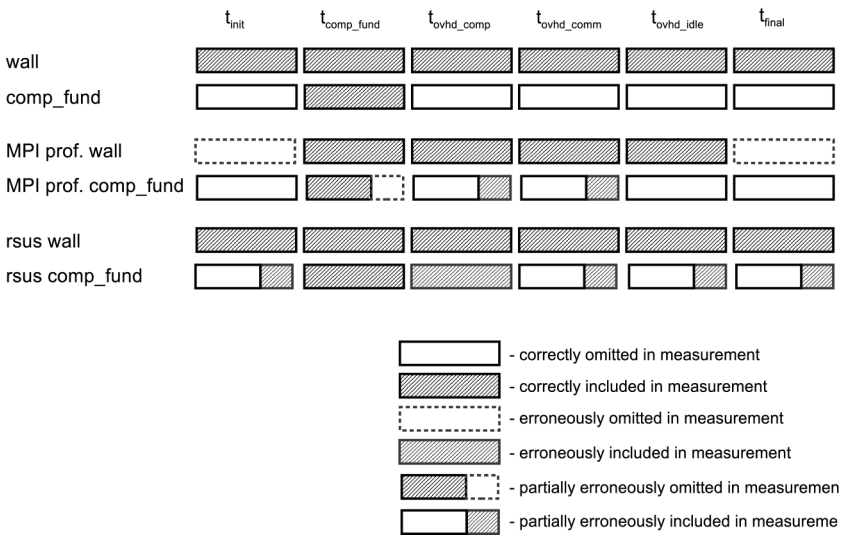

Fig. 11. Time measurements in the ETD methods

Let $X_y^{(P)}$ denote the value of the measured parameter X for the execution performed on P processors. Let $X_y^{(P,k)}$ denote denote the value of the parameter X measured on the processor with index k from the P utilized processors. The efficiency analysis can be performed utilizing the wallclock and the communication time measurements. It is frequently the case that the processors allocated to execute the request have to wait for all the other processors requested to be present to start the computations. The value of this time, called the phantom time, has also to be taken into consideration. This leads to the four basic methods of performing the measurements.

Utilizing computation time as a base, the efficiency value $\eta_{\tau b}^{(P)}$ can be defined:

$$\eta_{\tau b}^{(P)} = \frac{\sum_{k=1}^{P} \tau_{comp}^{(P,k)}}{\tau_{wall}^{(P)}}$$

When the phantom times are not considered:

$$t_{wall}^{(P)} = \arg\max_k t_{wall}^{(P,k)} \tag{1}$$

and thus:

$$\eta_{\tau b}^{(P)} = \frac{\sum_{k=1}^{P} \tau_{comp}^{(P,k)}}{P \cdot \arg\max_k t_{wall}^{(P,k)}} \tag{2}$$

For the communication time based measurements, the fundamental time can be computed as the difference between the wallclock time and the communication time. For a single processor k this leads to the efficiency value formulation:

$$\eta_{\tau c}^{(P,k)} = \frac{\tau_{wall}^{(P,k)} - \tau_{comm}^{(P,k)}}{\tau_{wall}^{(P,k)}}$$

Utilizing the equation 1, the efficiency value of the execution on P processors can be computed as:

$$\eta_{\tau c}^{(P)} = \frac{\arg\max_k t_{wall}^{(P,k)} - \sum_{k=1}^{P} \tau_{comm}^{(P,k)}}{P \cdot \arg\max_k t_{wall}^{(P,k)}} \tag{3}$$

When the phantom times are analysed it is necessary to perform the appropriate wallclock time corrections for each of the utilized processors, replacing the $t_{wall}^{(P,k)}$ value with the corrected $\tau_{wall\_phant}^{(P,k)}$ value.

For a single processor, the computation based efficiency value is given by:

$$\eta_{\Psi\tau f}^{(P,k)} = \frac{\tau_{comp}^{(P,k)}}{\tau_{wall\_phant}^{(P,k)}}$$

The inclusion of phantom operations lets us assume that all the processors start and finish the computations at the same time (with the phantom times filling the "gaps"):

$$t_{wall}^{(P)} = \frac{\sum_{k=1}^{P} t_{wall}^{(P,k)}}{P}$$

The computation based efficiency evaluation utilizing the phantom times can thus be defined as:

$$\eta_{\Psi\tau f}^{(P)} = \frac{\sum_{k=1}^{P} \tau_{comp}^{(P,k)}}{P t_{wall}^{(P)}} = \frac{\sum_{k=1}^{P} \tau_{comp}^{(P,k)}}{\sum_{k=1}^{P} \tau_{wall\_phant}^{(P,k)}} \tag{4}$$

Analogously for the communication time based measurements, the efficiency of execution on a single processor is described by:

$$\eta_{\Psi_{\tau cf}^{(P,k)}} = \frac{\tau_{wall\_phant}^{(P,k)} - \tau_{comm}^{(P,k)}}{\tau_{wall\_phant}^{(P,k)}}$$

For the full execution on P processors, the communication time based efficiency value is given by:

$$\eta_{\Psi_{\tau cf}^{(P)}} = \frac{\sum_{k=1}^{P} \tau_{wall\_phant}^{(P,k)} - \sum_{k=1}^{P} \tau_{comm}^{(P,k)}}{\sum_{k=1}^{P} \tau_{wall\_phant}^{(P,k)}} \tag{5}$$

The above four methods of the efficiency evaluations were utilized during the experiments described below.

4.3 Experimental Results

The evaluation of the proposed ETD methods was performed by the comparison of the values measured with a particular measurement method to the value of the relative efficiency measured in a classical fashion. After the analysis of typical production runs it was decided to perform the measurements for the programs from the NAS Parallel Benchmark suite [1] together with a test implementations of the Cannon parallel matrix multiplication algorithm. The measurements were also performed for the Hydsol explicit solver for compressible flows developed at the Stuttgart University [3].

The tests were executed on two clusters — the Calvus cluster located at the Wrocław University of Technology and the Cacau cluster in the HLRS supercomputing center in Stuttgart. In the following discussion, the computation and communication based method of determining the efficiency (equations 2 and 3) together with their counterparts using the phantom times (equations 4 and 5) were analyzed.

Over eighty thousand of application executions were measured for more than one hundred combinations of input data size and execution environment. On every processor involved in computations the CPU time consumption was recorded with 10 times per second frequency, leading to the acquisition of more than twenty two million individual probes. Due to the lack of space, in the following analysis only the most important results will be discussed. The included illustrations present the efficiency measurement in relation to the number of used processors. The chart line *efficiency* represents the relative efficiency. The line *efficiency-orig* represents the efficiency measuremed without the phantom times. The *cpu, comm, cpu_phant* and *comm_phant* lines represent appropriate efficiency estimations. The omission of the

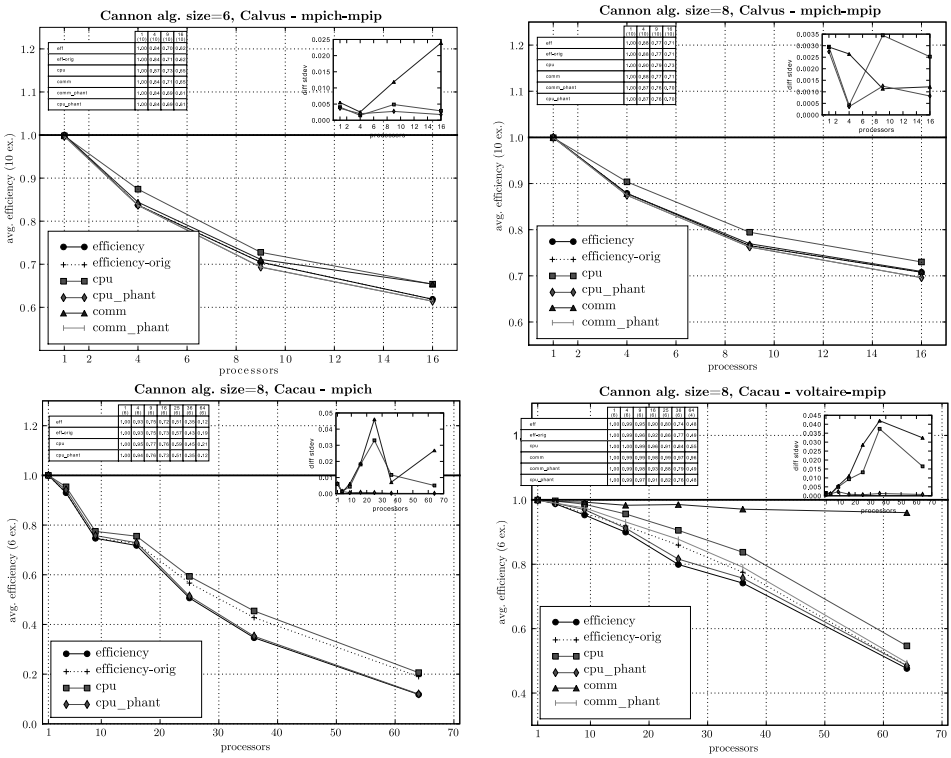

Fig. 12. Efficiency measurements — Cannon algorithm

values for the communication based measurements indicates that for a particular experiment the communication time could not be measured. The sub-chart in the upper right corner presents the standard deviation of the results represented as the percentage of the mean value.

During the tests performed for the Fast Ethernet environment on the Calvus cluster all the tested methods behaved in a similar manner, offering an estimation closely matching the actual efficiency. The typical example of the obtained results can be seen on the upper two graphs of the Figure 12. The results obtained with the cpu time measurements and the communication time measurements profiler almost ideally match the shape of the efficiency curve.

The tests performed on the Cacau cluster brought different results for the profiler-based estimation (lower two graphs on Fig. 12). Because of much larger initialization time in this environment, the initialization and finalization times ignored in this method have considerable impact on the final wall-clock time and the efficiency. When the phantom times are utilized, the efficiency estimation becomes accurate.

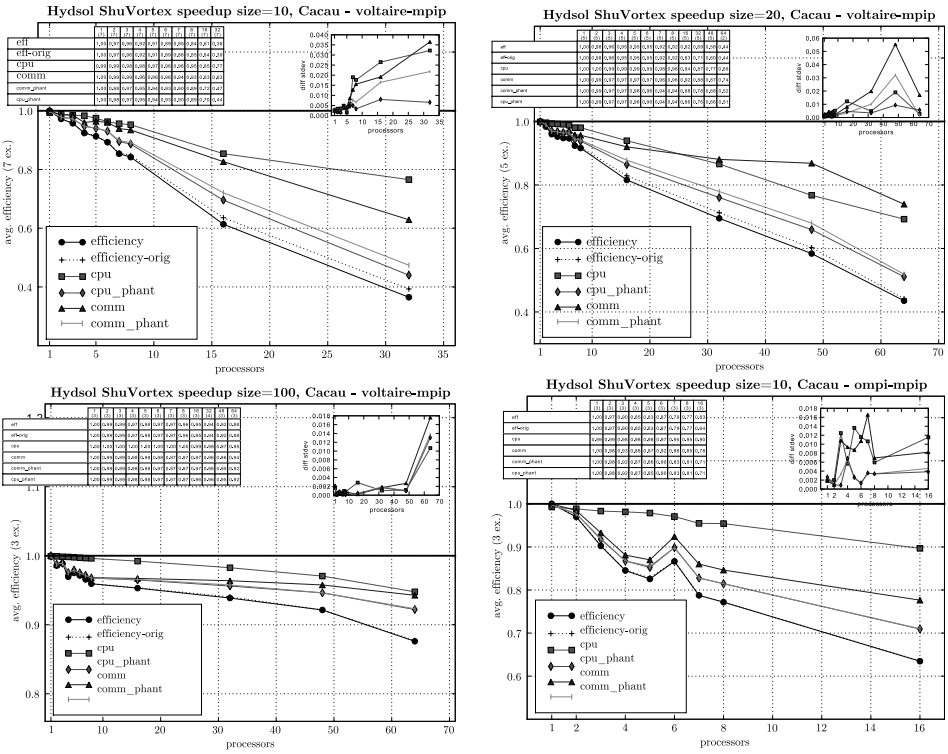

Fig. 13. Efficiency measurements — Hydsol application

In the case of fast interconnects (e.g. Infiniband), another source of estimation inexactness surfaces. The CPU time based estimation would assume that the computational overhead is a part of the fundamental computations time. That assumption leads to the serious overestimation of the application efficiency. In the case of the Cacau cluster, the erroneous measurement of startup and finalization time in the profiler based methods results in the similar efficiency overestimation as in the case of the Ethternet interconnect. The utilization of communication based methods eliminates this problem.

In the case of the NAS Parallel benchmarks and Hydsol application (Fig. 13 and 14), a similar behavior of estimation methods can be seen although a different usage of communication routines leads to a different relation between the CPU time based and communication time based measurement methods. In the last graph of the Figure 14, an example of inability of the ETD-based methods to correctly estimate execution anomalies [8] is depicted. In this case a situation is presented when the measured application achieved a superlinear speedup — a behavior that

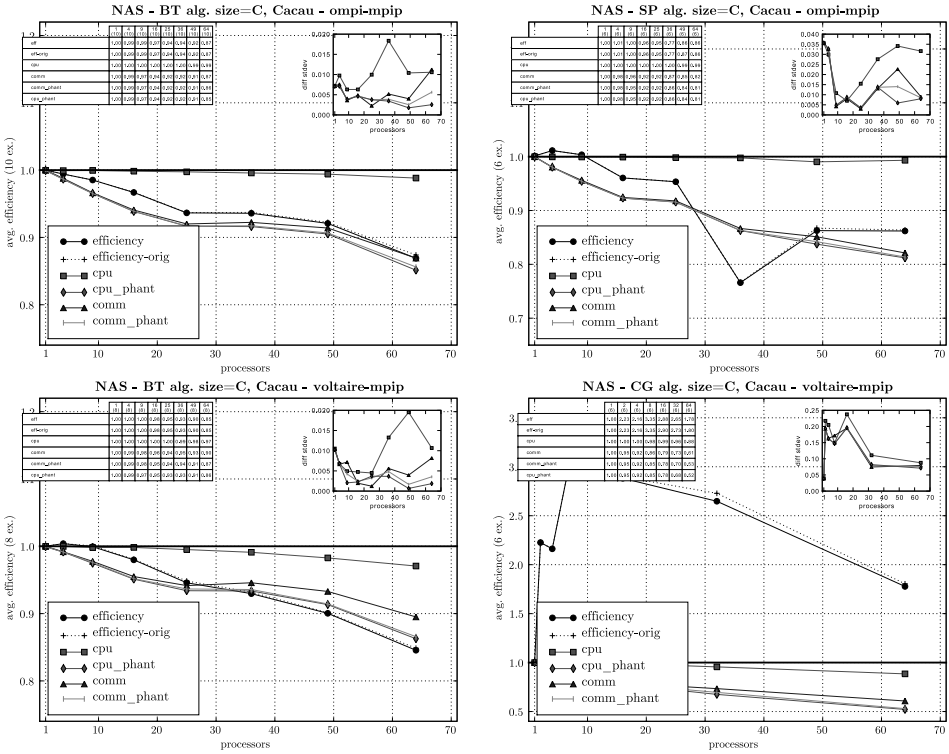

Fig. 14. Efficiency measurements — NAS benchmarks

lies out of the scope of the current ETD model.

The experimental results confirm that the ETD method is a valuable tool for the accurate measurements of the resource utilization efficiency. The ETD method can be used when it is impractical or infeasible to employ the relative efficiency measurements.

5 Matching Module

As it has been assumed at the highest level of abstraction, any SOA service can be seen as a complex one that is composed with some atomic services. During its decomposition into a set of determined atomic services, functional requirements are mainly considered. Therefore when resources are allocated to requested services only execution systems, which implement these atomic services and pre-serve the

expected non-functional properties can be taken into account. It means that for every version of an atomic service implemented at different locations only some instances of these versions can be used as a response to the request. On the other hand, registration means only that a chosen execution system can deliver an atomic service with the specified functional and not functional requirements, and it is not enough because, it can be a number of different service instances running on different hardware platforms included in the execution system. It causes that as a response to a request, a realization of an instance of atomic service has to be chosen, and this is the main functionality of the Matching Module.

To do this in the most efficient way, methods of semantic description of services, available resources, and requirements, was implemented. Ontology-based dependence models of the infrastructure, services, and execution requirements, has been designed. Algorithms were developed to transform execution and requirements information into an execution flow knowledge. A recommendation and description method was proposed for selected SOA services utilization in SOA applications. Different aspects related to systems quality assurance, reliability, availability, safety and security were taken into consideration during this analysis process.

5.1 Architecture of the Matching Module

The aim of the Matching Module is to find the most suitable execution system for a SOA request. Each execution system consists of different hardware platforms, including hybrid ones that offer different performance and different accuracy. Moreover different virtualizers can be used to simulate the needed hardware platforms and/or operation systems. This means that to take an efficient decision, the description of the available hardware platforms, virtualizer, operating system (with libraries) and the realizations of the instances of atomic services, should be used. Basing on the above specified information, the Matching Module which architecture is presented in Figure 15, a set of couples that consists of a selected realization of the instances of atomic service and an allocated execution environment, is generated.

To generate the above defined couples the module first check what hardware platforms are available. Then the possibility of using the proper virtualizers which can be executed on that hardware platforms is checked. In a description of registered virtualizers, there are some rules which define its requirements (amount of memory, type of processor, etc.) Such a pair constitutes a virtual hardware (VH) platform. On any of such virtual hardware platform, it can be run an operating system (called a virtual operating systems - VOS). In description of a VOS, there are rules about the requirement on the virtual hardware needed for running the VOS. A pair of VOS and VH constitutes a possible execution environment (EE). Then the connection between the service realizations with the

desirable non-functional properties, and the adequate execution environments is established. In the description of a service realization, there are rules which define the non-functional properties depending on the properties of the virtual operating system. At this moment a set of possible concrete realization (hardware, virtualizer, virtual operating system, service realization) is created. Then depending on the current EE load and the analysis of the previous execution of the service realization, the module choose one of the concrete realization and propagate it to the execution module with all information needed for its execution.

Because not all hardware has its virtualizer, there is also a dummy virtualizer. This virtualizer vitrualizes nothing and is interpreted in a special way by the execution module. This means that the rules for virtualization will change nothing, but these virtualizers are also marked to inform the execution module to not run a virtualizer. It can also mean that an operating system is still running on such processor and it is only needed to execute a service realization. The system of information about hardware, virtualizers, virtual systems and service realizations is very flexible. Depending on registering the information of afore mentioned elements the whole system can be viewed as a one instance of service or not. The information can be registered in such a way that an executor can distinguish between different types of hardware but without distinguishing computation units, or in an other

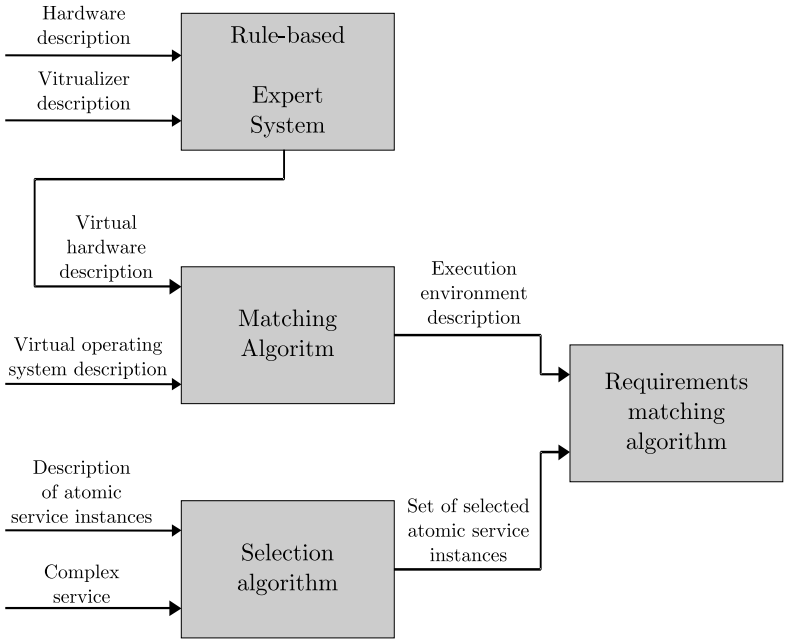

Fig. 15. Matching module architecture

way, that service will be interchangeably assigned to exactly one unit of hardware. It allows to use different policy of hardware, facilitating the execution of service requests.

5.2 An Example of Using Matching Module

Suppose three workstations with processors of type PA (PA1, PA2, PA3), and two with processors of type PB (PB1, PB2) in the system. To simplify considerations, workstations will be later identified with their processor. There are also a number of virtualizers: two formal that do not change their processor type, and one that changes processor of type PA into type PB. Moreover there are two virtual operating systems, OS1 and OS2, which are equipped with different matrix operations libraries: OS1 operates on processor of type PA using a single-precision real numbers representation, when OS2 operates on the same processor using double-precision real numbers representation. For processors PB only one operating system OS3 is available. Each of these operating systems require 1GB of RAM memory. In the system a service UA1 is registered. This service can be executed using the operating system OS1 or OS2, however accuracy of computations depends on the used matrix operations library. Another service UA2 is also registered, which operates in the system OS3, and used only double-precision real number representation. Of course, other services are also registered, but these two implementations are functionally compatible with the service request.

Currently on the computer PA1, there are 0.7GB of free memory, at PA2 — 2GB, PA3 — 1.5 GB, PB1 — 3GB and at PB2 — 0.5GB, respectively. Firstly the possible virtual hardware configurations are generated. After applying all virtualizers we can generate a 3 processor of type PA, and 5 CPU of type PB (2 real and 3 through virtualization PA to PB). Virtual processor PB built with the processor PA1 is denoted as PB(PA1), etc. Then it is determined what operating systems can be run. Since all operating systems require at least 1GB of memory, and therefore none can be run on processors PA1, PB2 and PB(PA1). On processors PA2 and PA3, it can be run any operating systems OS1 or OS2, while on the PB1, PB(PA2) and PB (PA3), it can be run only the operating system OS3. Therefore, 7 possible runtime environments are possible.

Suppose that a request comes for the atomic service UA, with some non-functional properties. One of these features is to calculate with double-precision real numbers. After reviewing the services of registered projects, of course UA1 and UA2 are possible. It has to be checked which runtime environment provide that specific requirement. Since UA1 can run in any of the systems with the OS1 or OS2, we can create 4 concrete realizations. By contrast, UA2 can run on the remaining 3 runtime environments. However, two of the realization which use UA1 do not meet the requirement of calculations precision (the results are generated

with single precision), so eventually we can create the following outputs for the requested service AU:

— UA1, OS2, PA2
— UA1, OS2, PA3
— UA2, OS3, PB1
— UA2, OS3, PB(PA2)
— UA2, OS3, PB(PA3)

Then, based on data from previous runs, a decision for which specific implementation to choose can be done. Such a choice may be directed by other non-functional characteristics such as the duration of action, the cost of the service execution, etc.

6 Conclusions and Future Work

The architecture of a newly designed service requests distribution system has been proposed. The aim of the system is the efficient allocation of execution and communication resources to SOA-based services. The resource allocation decision is taken in two steps. In the first, communication resources are allocated using implemented Request Broker and then the Matching Module using the historical data stored in the system repository and information from the Execution Monitoring System, takes a decision about the execution resources allocation. The virtualization used by the system makes it possible to utilize a variety of hardware resources from the Execution systems (SP), dynamically selected to ensure the efficient SOA service execution performed in accordance with the requirements. The project is under the development and we have just completed the implementation of the prototype first version. However, the current prototype is still missing some important features common in such systems, the first performed tests confirmed the correct functioning of all designed modules. Currently the system is integrated and in the next step, a set of performance tests will be conducted to an ensure efficient, with low overhead operations, of the final system implementation. Later on the opportunity of using the developed system as a cloud computing platform will be considered.

Acknowledgments. The research presented in this paper has been partially supported by the European Union within the European Regional Development Fund program no. POIG.01.03.01-00-008/08.

References

1. Bailey D., et al. The NAS parallel benchmarks. RNR Technical Report RNR-94-007, 1994.
2. Borzemski L., Zatwarnicka A., Zatwarnicki K. Global distribution of HTTP requests using the fuzzy-neural decision-making mechanism. Proc. 1st Int. Conf. on Comp. Collective Intelligence. Notes in AI, Springer, Berlin, 2009.
3. Dumbser M., Munz C.D. Building Blocks for arbitrary high order discontinuous Galerkin schemes. J. Sci. Comp. 27, 1–3, 2006.
4. Halfhill T. Parallel processing with CUDA. Microprocessor Report, 2008.
5. Jain L.C., Martin N.M. Fusion of neural networks, fuzzy sets, and genetic algorithms industrial applications. CRC Press LLC, London, 1999.
6. Kahle J., Day M., Hofstee, Johns C., Maeurer T., Shippy D. Introduction to the cell multiprocessor. IBM J. Res. Dev. 49, 4/5, 589–604, 2005.
7. Keong-Myung L., Dong-Hoon K., Hyung L. Tuning of fuzzy models by fuzzy neural networks. Fuzzy Sets Syst. 76, 1, 47–61, 1995.
8. Kwiatkowski J., Pawlik M., Konieczny D. Parallel program execution anomalies. Proc. 1st Int. Multiconf. on Computer Science and Information, Wisła, Poland, 2006.
9. Mabrouk M. SOA fundamentals in a nutshell. IBM Corp., 2008, available at IBM Corp. Website.
10. Mamdani E.H. Application of fuzzy logic to approximate reasoning using linguistic synthesis. IEEE Trans. Comp. C-26, 12, 1182–1191, 1977.
11. Pawlik M. Execution time decomposition method in performance evaluation. Science and supercomputing in Europe — report 2007, CINECA, 2007.
12. Pawlik M., Kwiatkowski J., Konieczny D. Parallel program performance evaluation with execution time decomposition. Proc. 16th Int. Conf. Systems Science, Wrocław, Poland, 2007.
13. Williams W., Herman R., Lopez L.A., Ebbers M. Implementing CICS Web Services. IBM Readbook, 2007.

QoS-Aware Complex Service Composition in SOA-Based Systems

Piotr Rygielski, Paweł Świątek, and Adam Grzech

Institute of Computer Science
Wrocław University of Technology,
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland
{Piotr.Rygielski,Pawel.Swiatek,Adam.Grzech}@pwr.wroc.pl

Abstract. In this work, a method for the QoS-aware complex service composition in SOA-based systems is presented. The considered complex service composition process consists of three stages: complex service structure, scenario and plan composition. Complex service structure contains a set of required functionalities and precedence relations between them, and is derived directly from a service level agreement. Service scenario is derived from service structure by choosing the optimal (in the sense of certain quality criterion) order of execution of required functionalities. Finally, a service plan emerges from a scenario by choosing the best versions of atomic services for the delivery of required functionalities. For such distinguished composition process methods for complex service scenario and execution plan optimization are proposed. Optimization of a complex service scenario is based on the analysis of possible parallel executions of required functionalities. In order to deliver required level of the quality of service, well known QoS assurance models (i.e.: best effort, integrated services and differentiated services) are applied in the process of complex service execution plan optimization.

Keywords: Service Oriented Architecture (SOA), Quality of Service (QoS), complex services composition

1 Introduction

Systems based on the Service Oriented Architecture (SOA) paradigm offer services (complex services) which are often delivered as a composition of atomic services [12, 13]. The main feature of such an attempt is that the required complex services may be efficiently and flexibly composed of available atomic services providing certain, well defined, required and personalized functionalities. Requested complex services

are characterized by a set of parameters specifying both functional and nonfunctional requirements; the former define the exact data processing procedures, while the latter describe various aspects of required service quality. The set of parameters describing requested complex service form SLA (*Service Level Agreement*) [1, 14].

Functionalities of the requested complex service are available as a sum of atomic services functionalities. In order to deliver a complex service with requested functional and non-functional properties appropriate atomic services must be chosen in the process of complex service composition [10]. Required functionality, precisely defined in the SLA, determines a set of required atomic services as well as a plan according to which atomic services are performed in the distributed environment. Properties of a requested complex service, which are mainly related to QoS (*Quality of Service*) may be in most cases assured or obtained by proper resources (processing and communication) and tasks (atomic services) allocation [4, 5, 7, 17].

Discussed complex services delivery approach is available only in a distributed environment; possible parallel executions of distinguishable atomic services requires the allocation of a proper amount of processing and communication resources in a parallel manner. The distributed environment may be obtained both by the allocation of separated or virtualized resources.

In order to obtain various required quality of service levels in distributed environment well-known QoS strategies, i.e., best-effort, integrated services and differentiated services concepts may be applied. Usefulness of the mentioned concepts strongly depends on the formulation of the non-functional part of the entire SLA. The application of the best-effort concept, based on common resources sharing, leads to a solution where the same, high enough, average quality of service is delivered to all performed services. The next two previously mentioned concepts offer differentiated quality of service for requested services (also guarantees) and are mainly based on resources reservation for individual requests (integrated services concept) or for classes of requests (differentiated services concept) [4, 15].

In general, the task of complex service composition consists of finding, for given ordered set of required functionalities (stated in the SLA), an order of atomic services execution such that non-functional requirements are met. The task of complex service composition can be decomposed into three subtasks, each of which providing an input for subsequent subtask:

— Complex service structure composition — transformation of the SLA into a set of required functionalities and the precedence relations between them. The result of this task is a complex service structure represented as a directed graph (not necessarily connected) of required functionalities.
— Complex service scenario composition — transformation of a complex service structure graph into a single and consistent graph of required functionalities

with precisely defined order of execution of all atomic functionalities. The determination of the complex service scenario is composed of two stages. The goal of the first is to propose for a given complex service structure graph a set of consistent graphs, each representing certain order of execution of atomic functionalities. The aim of the second stage is to select the best (for assumed criteria) connected graph (scenario). Among others this task consists in making the decision on whether to apply possible parallel executions to functionalities, which are not bound by precedence relations in the complex service structure. Since it is possible that a single functionality is delivered by more than one atomic service (different versions of atomic service), the scenario graph represents in fact a family of execution graphs where member graphs differ in the atomic service versions applied to deliver the required atomic functionality.

— Complex service execution plan composition — the choice of particular atomic services in a complex service scenario graph such that non-functional requirements of complex service are met. This stage is composed of three sub-stages. In the first one, the nodes (functionalities) of the corresponding optimal scenario graph are replaced by subsets of atomic services providing respective functionalities. In the second substage particular atomic services from atomic services subsets are chosen. In the last substage particular versions of chosen atomic services are picked.

The main advantage of SOA-based systems is that atomic services which deliver certain functionalities may be provided by different service providers, what allows users to chose the required complex services from many, functionally and non-functionally equivalent, available alternatives. In order to facilitate such capabilities service providers have to describe delivered services semantically in a unified manner. In order to deliver to the user a complex service with a requested functionality, semantic matching of service request with the available complex services has to be performed. This very important step of complex service composition is performed as a part of the first stage of the complex service structure composition task which was described above.

Since service requests, besides required functionality, include required non-functional parameters, they must be taken into account during service composition procedure. These parameters however concern, in general, the quality of required service, which vary in time and depends mostly on current state of computer communication system used as a backbone for complex service delivery. The aim of this paper is to provide framework for QoS-aware complex service composition, which allows to make decisions based on the network state of the SOA-based system.

In order to deliver complex service with requested functionality and non-functional properties various optimization tasks need to be solved on consecutive stages of complex service composition task (i.e.: stages of complex service structure, scenario and execution plan composition).

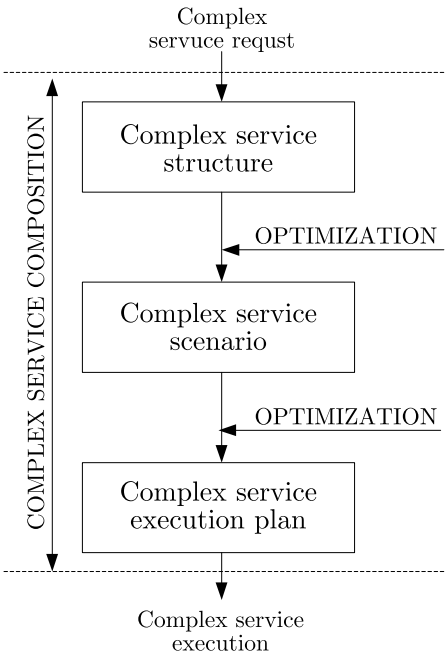

Fig. 1. Decomposition of the process of complex service composition

In this chapter a model of complex service, which facilitates decomposition of complex service composition task and allows to formulate and solve various complex service optimization tasks, is proposed. Decomposition of the complex service composition task is presented on Figure 1. This work is organized as follows. In Section 2 model of complex service and complex service structure composition task is presented. In section 3 the task of complex service scenario composition is presented. Section 4 describes three approaches to the task of complex service execution plan composition, which are based on well-known QoS strategies. Some numerical examples of proposed solutions are presented in section 5. Section 6 summarizes presented work and gives directions for future research.

2 Complex Service Model

Let $CS = \{cs(1), \ldots, cs(i), \ldots, cs(I)\}$ denote the set of all complex services delivered in the considered system. It is assumed, that a functionality $\varphi(cs(i))$ of

a complex service $cs(i)$ is delivered by the execution of a certain number of atomic services $as(j)$ from the set $AS = \{as(1), \ldots, as(j), \ldots, as(J)\}$ of atomic services available in the system. The functionality $\varphi(as(j))$ of complex service $as(j)$ is an aggregation of the functionalities $\varphi(as(j))$ of the atomic services $as(j)$, which compose the complex service $cs(i)$. Similarly non-functional properties $\psi(cs(i))$ of complex service $cs(i)$ are the aggregation of the non-functional properties $\psi(as(j))$ of the applied atomic services $as(j)$.

The i-th ($i = 1, 2, \ldots, I$) complex service $cs(i)$ is a response to a call for a service fully described by the appropriate service level agreement denoted by $SLA_l = (SLA_{fl}, SLA_{nfl})$, which contains information about the functionality (SLA_{fl}) required by the user, and the nonfunctional requirements (SLA_{nfl}) determining the required level of the quality of service. The functional part of $SLA_{fl} = (\Phi_l, R_l)$ is a subject of the structure composition process which delivers the set of atomic functionalities $\Phi = \{\varphi_{l1}, \ldots, \varphi_{ln_l}\}$ present in the system. It defines the allowed order of execution of the required functionalities with use of the precedence relations $\prec$, given in matrix R_l.

The discussed precedence matrix, describing the set of precedence relations may be — in the simplest case — described by a square binary order constraints matrix R_l of size $(n_l + 2) \times (n_l + 2)$. The R_l matrix defines which functionalities are bound with the relation of precedence.

Exemplary order constraints (precedence) matrix is presented below:

$$R_l = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{1}$$

The matrix R_l of dimension 5×5 corresponds to three atomic functionalities with the addition of abstract starting and ending functionalities $\Phi_l = \{\varphi_{ls}, \varphi_{l1}, \varphi_{l2}, \varphi_{l3}, \varphi_{le}\}$. The matrix should be understood as follows:

— column 1 — Functionality φ_{ls} (abstract start functionality) cannot be preceded by any other functionality;
— column 2 — functionality φ_{l1} can be preceded by functionality φ_{ls} or φ_{l3};
— column 3 — functionality φ_{l2} can be preceded by functionalities φ_{l1} or φ_{l3};
— column 4 — functionality φ_{l3} can be preceded by functionalities φ_{ls}, φ_{l1} or φ_{l2};
— column 5 — functionality φ_{le} (abstract end functionality) can be preceded by functionalities φ_{l2} or φ_{l3}.

The ordering constraints given with by matrix R_l can be transformed into description using the precedence relation $\prec$ as follows:

$$\begin{array}{l} \varphi_{ls} \prec \{\varphi_{l1}, \varphi_{l3}\}; \\ \varphi_{l1} \prec \{\varphi_{l2}, \varphi_{l3}\}; \\ \varphi_{l2} \prec \{\varphi_{l3}, \varphi_{le}\}; \\ \varphi_{l3} \prec \{\varphi_{l1}, \varphi_{l2}, \varphi_{le}\}; \\ \varphi_{le} \prec \emptyset \end{array} \tag{2}$$

In general, the binary 1 in i-th row and in j-th column ($r_{lij} = 1$) means that the functionality φ_j can be preceded by the functionality φ_{li}. The zero value in i-th row and in j-th column ($r_{lij} = 0$) means that the functionality φ_{lj} cannot be preceded by the functionality φ_{li}. Abstract start and abstract end is guaranteed by zeros in column $j = 0$ and row $i = n_l + 1$.

$$r_{lij} = \begin{cases} 1 & \text{if } \varphi_{li} \prec \varphi_{lj} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Moreover each row (except for row $i = n_l + 1$) has to have at least one 1 value which guarantees the presence of exactly one end functionality. Additionally guarantee of having exactly one start functionality is determined by having at least one 1 in each column (except for $j = 0$ column). Above assumptions can be summarized with the following formulas:

$$\forall i \in \{0, \ldots, n_l\} \quad \sum_{j=1}^{n_l+1} r_{lij} \geq 1 \tag{4}$$

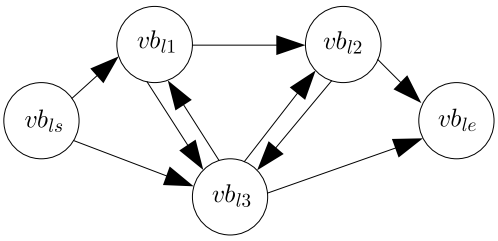

Fig. 2. Graph GB_l representation for matrix R_l (eq. 1). Each binary "1" value in the matrix corresponds to an edge in the structure graph GB_l. Redundant edges make the graph cyclic

$$\forall j \in \{0, \ldots, n_l + 1\} \quad \sum_{i=0}^{n_l} r_{lij} \geq 1 \tag{5}$$

Having the functionalities set $\Phi_l = \{\varphi_{l1}, \ldots, \varphi_{ln_l}\}$ and the order constraints matrix R_l gives the ability to build a base graph denoted by GB_l. $GB_l = GB(SLA_{fl}) = GB(\{\Phi_l, R_l\}) = GB(VB_l, EB_l)$ is a graph defining the structure of a complex service, where $VB_l = \{vb_{l1}, vb_{l2}, \ldots, vb_{lk}, \ldots, vb_{ln}\}$ is the set of vertex of a base graph (each vertex vb_{lk} corresponds to a proper functionality φ_{lk}) and EB_l is set of edges corresponding to the precedence relations defined by matrix R_l.

The example graph for R_l matrix (eq. 1) and the functionalities set $\Phi_l = \{\varphi_{ls}, \varphi_{l1}, \varphi_{l2}, \varphi_{l3}, \varphi_{le}\}$, is presented on Figure 2. Each binary value "1" represents an edge between the functionalities in graph GB_l.

3 Complex Service Scenario Composition

The structure of a complex service determines which atomic functionalities are delivered within it and what are the order bounds. Such a service can be an entry of the optimization process concerning the determination of the exact order of functionalities delivery and parallel execution. The result of processing a complex service structure is called a l-th complex service execution scenario.

A complex service execution scenario is a graph GC_1 defined by the scenario determination process. The scenario $GC_1 = GC(SLA_{fl}) = GC(\{\Phi_l, R_l\}) = GC(VC_l, EC_l)$ is a graph containing the vertices set $VC_l = VB_l = \{vb_{l1}, vb_{l2}, \ldots, vb_{lk}, \ldots, vb_{ln}\}$ the same as in service structure, and the edge set $EC_l = EB_l \setminus EA_l$ which is a subset of the structure edge set EB_l:

$$EC_l \subseteq EB_l$$
$$EC_l \cup EA_l = EB_l$$

The problem of finding an optimal scenario can be formulated as follows:
Given:

— the l-th complex service request given with SLA_l;
— the graph GB_l with set of vertices VB_l and set of edges EB_l;
— the order constraints matrix R_l;
— the other parameters vector a_l.

Find:

An adjacency matrix R_{GC_l} (as a representation of the graph GC_l) from the set of binary matrices of size $(n_l + 2) \times (n_l + 2)$ which minimizes the function f:

$$R^*_{GC_l} = \arg\min_{R_{GC_l}} f(R_{GC_l}; a_l) \tag{6}$$

with respect to constraints:

— the graph GC_l represented by the matrix R_{GC_l} should be acyclic;
— the graph GC_l represented by the matrix R_{GC_l} should have exactly one abstract start and an abstract end functionality:

$$\forall i \in \{0, \ldots, n_l\} \quad \sum_{j=1}^{n_l+1} r_{CG_{ijl}} \geq 1$$
$$\forall j \in \{1, \ldots, n_l+1\} \quad \sum_{i=0}^{n_l} r_{CG_{ijl}} \geq 1$$

— the matrix R_{GC_l} should comply with the order constraints matrix R_l: $R_{GC_l} \otimes R_l = R_{GC_l}$.

The satisfaction of ordering constraints can be determined via the logical multiplication of each binary element of matrix. The operation $\otimes$ is defined as follows:

$$A \otimes B = C$$

$$\forall i \in \{0, \ldots, n_l+1\} \ \forall j \in \{0, \ldots, n_l+1\} : \ c_{ij} = \begin{cases} 1 & \text{if } a_{ij} = 1 \text{ and } b_{ij} = 1 \\ 0 & \text{otherwise} \end{cases}$$

The equality $R_{GC_l} \otimes R_l = R_G C$ determines the satisfaction of the ordering constraints. The function $f(R_G C; a_l)$ can be any function determining the quality of service level e.g. execution time, cost, security level, reliability, etc. Depending on the function relationship with quality (a function value growth can mean quality level increase or decrease), the optimization task might has to be changed to maximization or minimization.

The determination of the optimal scenario consists in removing a subset of edges EA_l from the EB_l set in such a way, that each vertex in the result graph belongs to some path connecting the start vertex and the end vertex in such a way that the input and output degrees of vertices $\{\varphi_{l1}, \varphi_{l2}, \ldots, \varphi_{ln_l}\}$ are at least equal to 1. Moreover the input degree of start vertex and the output degree of end vertex must be equal to 0. Additionally, the result graph GC_l representing a scenario must be acyclic.

The set of edges that are subject to be removed is uniquely defined by the subtraction of the adjacency matrices $R^*_{A_l} = R_l - R^*_{GC_l}$. The remaining binary "1" values in the adjacency matrix $R^*_{A_l}$ define the edges that are subject to be removed from the graph GB_l to obtain an optimal complex service execution scenario GC^*_l.

For example with the matrix R_l (eq. 1), there are six possibilities of scenario graphs GC_l. All are presented in Figure 3.

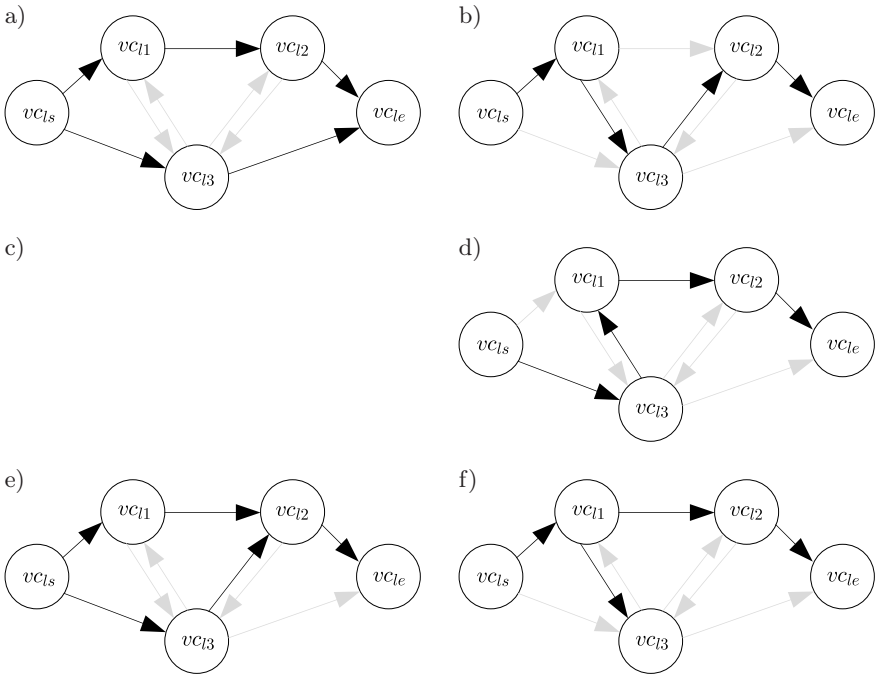

Fig. 3. Six possible scenario graphs GC_l obtained for the structure graph GB_l with the ordering constraints given in matrix R_l (eq. 1). Black edges are edges from EC_l, grey ones are from EA_l

The scenario presented in Figure 3a was obtained after the substraction $R_{GC_{ln}} = R_l - R_{GA_{ln}}$:

$$R_{GC_{ln}} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

One can notice that the scenarios presented in figure 3 differ in the serial and parallel execution cases. All functionalities from scenarios b), c) and d) are executed serially (one-by-one), thus one can suspect that quality expressed e.g. the execution time, may be worse than in scenarios a), e) and f) wich execute in parallel. But if a parallel execution of functionalities shortens a complex service execution time, it uses more system resources than a serial execution.

In the case of a serial scenario, we can estimate the time of execution of a complex service using the following equation:

$$t(GC_{lserial}) = \sum_{k=1}^{n_l+2} t(vb_{lk}) + t(EC_l)$$

where $t(EC_l)$ is the overall time of transferring the requests between functionalities in a complex service scenario.

If a transfer delay request between the i-th and the $(i+1)$-th functionality is denoted by $t(ec_{li,i+1})$, the calculation of the transfer times in a whole complex service within a serial scenario is as follows:

$$EC_l = \{ec_{lij} : r_{CG_{ijl}} = 1\}$$

$$t(EC_l) = \sum_{i=1}^{n_l+1} t(ec_{li,i+1}).$$

In the extreme parallel scenario, above calculations are slightly different. The time of execution of a complex service is calculated as follows:

$$t(GC_{lparallel}) = \max_{k \in \{1,2,\ldots,n_l+2\}} \{t(vb_{lk}) + t(ec_{l1k}) + t(ec_{lkn+2})\}$$

Two extreme scenarios — by mean of best and worst execution time — are presented on Figure 4.

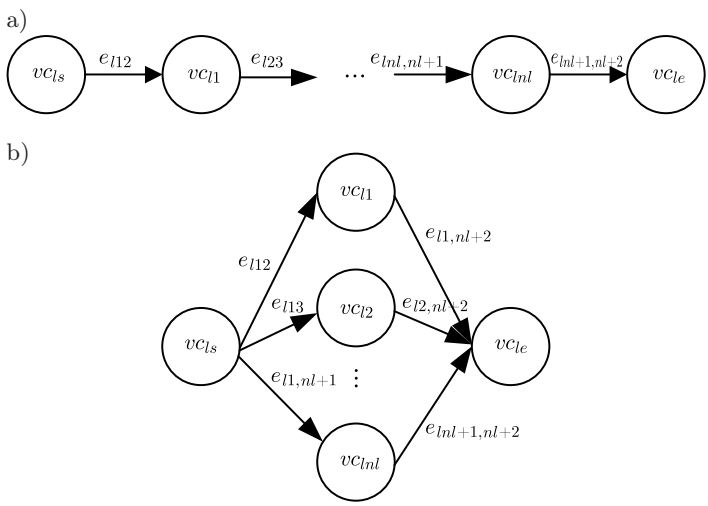

Fig. 4. Two extreme scenario graphs. Serial scenario: a — gives the worst execution time, extreme parallel scenario, b — gives the best execution time

Situation where the maximum number of parallel functionalities delivery is limited must be considered. A need for the introduction of such a limitation may arise in the case where the utilization of a larger number of parallel functionalities delivery can decrease the end-user quality e.g. increases the cost of the service to such a level which violates user requirement.

In order to determine the parallelism in an execution scenario, it is needed to introduce a measure to determine parallelism level. In the literature [16] the most popular parallelism level is the ILP — Instruction Level Parallelism — which measures the number of operations that can be done simultaneously divided by the number of operations (eq. 7):

$$l_{p_{ILP}} = \frac{n_{l_{par}}}{n_l} \tag{7}$$

where $l_{p_{ILP}}$ is the ILP measure and $n_{l_{par}}$ is the number of operation that can be executed in parallel manner.

For scenarios given by graphs, other parallelism measures can be used, e.g. a measure that uses weight path length in a graph, where weights correspond to time needed to deliver a proper functionality φ_{lk}. The proposed measure l_{pMP} can be defined as follows:

$$l_{pMP}(GC_l) = \left(\frac{1}{n_p} \sum_{i=1}^{n_p} p_l(i) \right)^{-1}$$

where n_p is the number of paths in the graph GC_l and $p_l(i)$ is a weight of i-th path length. Other parallelism measures can be found e.g. in [11].

In order to optimize end-user quality there is need to introduce constraints concerning the parallelism level in an execution scenario graph GC_l. When there are no constraints in SLA_{nfl} concerning QoS parameters like e.g. cost, the formerly formulated problem can be used without change; otherwise one should add a constraint concerning the parallelism level of a scenario:

$$l_p(GC_l) = l_p(R_{GC_l}) \leq l_{p\,\max}$$

where the l_p function returns a parallelism level (with respect to the chosen measure) for a graph or for an adjacency matrix which is a representation of the graph. This function should be designed in such a way that a higher value means the delivery of a larger number of functionalities simultaneously, when a smaller value means that the scenario graph is mainly executed like the extreme serial scenario.

4 Complex Service Execution Plan

The result of the scenario composition stage is an optimal scenario graph $GC_l^* = (VC_l, EC_l^*)$, where $VC_l \subset AS$ is a set of atomic services, which provide the functionality requested in SLA_{fl}, and EC_l^* is a set of edges describing the optimal order of atomic services execution. The main feature of systems based on the SOA paradigm is that each atomic service may be available in many versions which differ in non-functional properties. Therefore, a scenario graph GC_l^* must be treated as a set of graphs, which differ in the versions of atomic services.

$AS(\varphi_{lk}) = \{as_{lk1}, \ldots, as_{lkn_{lk}}\} \subset AS$ denotes a set of versions of atomic services, which provide functionality φ_{lk}, where n_{lk} is the number of versions:

$$AS(\varphi_{lk}) = \{as_{lkm_{lk}} \in AS : \varphi(as_{lkm_{lk}}) = \varphi_{lk}, m_{lk} = 1, \ldots, n_{lk}\}.$$

As an example, consider graph GC_l^* presented in Figure 3a, which consists of three functionalities. Functionality φ_{l1} is provided by three different versions of atomic services, and functionalities φ_{l2} and φ_{l3} are provided by two versions. The number m_l of all possible complex service execution plans is equal to the product of the numbers n_{lk} of the atomic service versions providing the respective

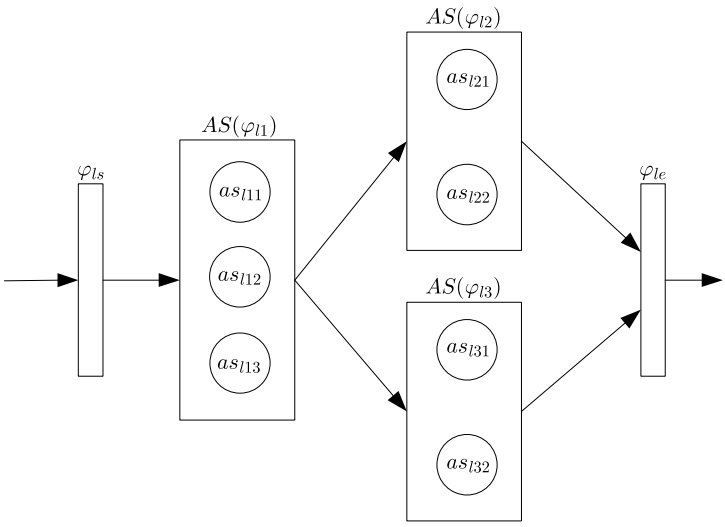

Fig. 5. Different versions of atomic services providing the respective functionalities resulting in twelve possible complex service execution plans

functionalities φ_{lk}; $m_l = \prod_{k=1}^{n_l} n_{lk}$. The twelve possible execution plans of complex service represented by graph GC_l^* are presented on Figure 5.

5 Problem Formulation

The task of complex service execution plan composition consists in choosing for each functionality φ_{lk} a version of atomic service, such that the non-functional properties $\varphi(cs(i))$ of the composed complex service $cs(i)$ meet the requirements Ψ_l stated in the non-functional part of the service level agreement SLA_{nfl}. Formally this task can be formulated as follows:
Given:

— non-functional requirements for complex service Ψ_l;
— optimal complex service execution scenario $GC_l^* = (VS_l, ES_l^*)$;
— complex service quality criterion Q.

Find:
A set of atomic services versions that such the following quality criterion is minimized:

$$\begin{aligned} & as_{l1m_{l1}}, \ldots, as_{ln_lm_{lnj}} = \\ & = \arg \min_{as_{l1m_{l1}} \in AS(\varphi_{l1}), \ldots, as_{ln_lm_{ln_l}}} Q\left(G(\{as_{l1m_{l1}}, \ldots, as_{ln_lm_{lnj}}\}, E_l)\right) \end{aligned} \tag{8}$$

where the graph $G(\{as_{l1m_{l1}}, \ldots, as_{ln_lm_{ln_l}}\}, E_l) = G(A_l, E_l)$ originates from the graph $GC_l^* = (VS_l, ES_l^*)$ in such a way, that the nodes A_l are particular atomic services delivering the same functionalities than nodes VS_l, and $E_l = ES_l^*$.

6 QoS Assurance Models

Depending on specific non-functional requirements and on the assumed model of the quality of service assurance various complex service optimization tasks can be formulated basing on the above general task of complex service execution plan composition. In the literature, there are three main concepts of QoS assurance i.e.: best effort (BE), integrated services (IntServ) and differentiated services (DiffServ). In the best effort approach no quality of service is guaranteed. Incoming request are serviced according to their arrival order and for each of them the best possible execution plan is chosen. In this model, only average values of QoS parameters may be guaranteed with the use of proper admission control mechanism.

In the IntServ model necessary resources are reserved for each incoming request, what allows to deliver strict guarantees on the values of quality of service parameters. Besides resources reservation, this model requires the utilization of other QoS mechanisms such as: admission control, request scheduling, etc.

In the DiffServ model, incoming requests are separated into classes and for each class, required resources are reserved. Requests in different classes may be served according to the best effort or the IntServ model. Application of the best effort model to single DiffServ class, results in delivering guaranties on average values of QoS parameters for requests in this class. On the other hand, the application of the IntServ model to the single DiffServ class allows to provide strict QoS guarantees for each single request belonging to this class.

In order to present the application of aforementioned models of QoS assurance in the considered task of complex service execution plan optimization, let us assume that the optimal complex service scenario requires that the functionalities Φ_l are executed in serial order, and that the non-functional requirements Ψ_l of the requested complex service, apply to complex service response time. Above assumptions are made in order to clarify the presented approaches, and do not affect the generality of further considerations.

The optimal complex service execution scenario considered in this example is presented on Figure 6. This scenario requires n_l atomic services to be executed in serial order. Moreover, there are n_{lk} available versions of atomic services for each functionality $\varphi_{lk}(k = 1, \ldots, n_l)$ what results in $m_l = n_1 \cdot n_2 \cdot \ldots \cdot n_l$ possible complex service execution plans. For simplicity, we denote the m-th complex service execution plan as the sequence of indices of chosen versions of the consecutive atomic services $ep_{lm} = (m_{l1}, \ldots, m_{ln_1})$, where $n_{l1} = 1, \ldots, n_{l1}; \ldots; m_{ln_l} = 1, \ldots, n_{ln_l}$.

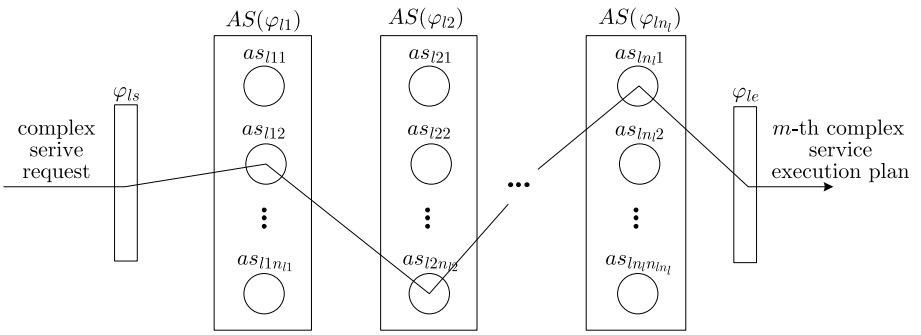

Fig. 6. Example scenario and plan of complex service execution with a serial execution order of atomic services

In the considered example, the non-functional property $\Psi(as_{lkm_{lk}})$ of the atomic service $as_{lkm_{lk}}$ is interpreted as an atomic service response time. Property $\psi(as_{lkm_{lk}},\ as_{l(k+1)m_{l(k+1)}})$ denotes a communication delay on link between the atomic services $as_{lkm_{lk}}$ and $as_{l(k+1)m_{l(k+1)}}$. The response time $\psi(ep_{lm})$ of complex service executed according to the m-th execution plan $ep_{lm} = (m_{l1},\ \ldots,\ m_{ln_l})$ is equal to the sum of response times of the applied atomic services and communication delays between them:

$$\psi(ep_{lm}) = \sum_{m_{lk} \in ep_{lm}} \left[\psi(as_{lkm_{lk}}) + \psi(as_{lkm_{lk}}, as_{l(k+1)m_{l(k+1)}}) \right] \tag{9}$$

The average response time $\overline{d_l}$ experienced by the service requests in the whole system can be calculated as the weighted average over the response times of each execution plan ep_{lm} $(m = 1,\ \ldots,\ m_l)$):

$$\overline{d_l} = \sum_{m=1}^{m_l} p_m \cdot \psi(ep_{lm}) \tag{10}$$

where p_m is the probability, that certain service request will be served according to the m-th execution path ep_{lm}.

7 Average Complex Service Response Time Minimization (best effort)

In such a system, the task of minimization of the average complex service response time (best effort model) can be formulated as a task of finding such a vector $\mathbf{p} = [p_1,\ \ldots,\ p_{m_l}]$ of probabilities to chose a particular execution plan with minimized average response time:

$$\mathbf{p}^* = \arg\min_{\mathbf{p}} \sum_{m=1}^{m_l} p_m \cdot \psi(ep_{lm}) \tag{11}$$

with respect to constraints on probabilities $\mathbf{p}$:

$$\sum_{m=1}^{m_l} p_m = 1 \text{ and } p_m \geq 0 \text{ for } m = 1, \ldots, m_l \tag{12}$$

In general the average response time $\psi(ep_{lm})$ of each execution plan ep_{lm} depends on request arrival intensity and probabilities $\mathbf{p}$, which change over time. Therefore the optimization task (11) has to be solved iteratively in consecutive

time steps. The execution of complex services consists in assigning to incoming request such execution plans that the number of requests executed according to each execution plan is proportional to calculated probabilities $\mathbf{p}^*$. For a large number m_l of possible execution plans this approach may be inefficient due to the high computational complexity of the optimization task (11). In such a case, one can approximate optimal solution by the application of greedy approach, which for each new service request chooses the execution plan ep_{lm}, with the lowest average delay $\psi(ep_{lm^*})$:

$$m^* = \arg \min_{m=1,\ldots,m_l} \psi(ep_{lm}) \tag{13}$$

8 Service Response Time Guarantees (IntServ)

$S(t_l)$ denotes the state of the system at the moment t_l of arrival of the new request SLA_l. State $S(t_l)$ contains information concerning the moments of arrival, the assigned execution plans, and the location of all service requests present in the system at moment t_i. Given the system state $S(t_l)$, it is possible to calculate exact service response time $\psi(ep_{lm})$ for the request SLA_l for each execution plan ep_{lm} ($m = 1, \ldots, m_l$):

$$\psi(ep_{lm}) = d(S(t_l), SLA_l, m) \tag{14}$$

where function $d(S(t_l), SLA_l, m)$ (presented in [6]) represents an iterative algorithm for the calculation of the response time of the service request SLA_l delivered according to the m-th execution plan.

In the the quality of service delivery task it is assumed that each incoming request SLA_l contains a set $SLA_{nfl} = \Psi_l$ of requirements concerning the values of various parameters describing quality of service such as: response time, security, cost, availability, etc. For the purpose of this example, it is assumed that the set $\Psi_l = \{\Psi_{l1}\}$ contains only one requirement concerning complex service response time.

The aim of the task of guarantying service response time is to find such a execution plan ep_{lm} for which the service response time requirements are satisfied:

$$m^* = \arg \min_{m=1,\ldots,m_l} \psi(ep_{lm}) = \arg \max_{m=1,\ldots,m_l} d(S(t_l), SLA_l, m) \tag{15}$$

with respect to:

$$\psi(ep_{lm}) \leq \psi_{l1}$$

It is possible that it does not exist such an execution plan for which the response time requirements are met. In this case requirements can be renegotiated, for example by suggesting a minimal possible service response time ψ_{l1}^*:

$$\psi_{l1}^* = \min_{m=1,\ldots,m_l} \psi(ep_{lm}) \tag{16}$$

When the required execution plan ep_{lm^*} is found (by solving either task (15) or (16) in order to be able to guarantee the requested service response time, resources in execution plan ep_{lm} have to be reserved.

9 Average Service Response Time Guaranties (DiffServ)

Assume that each incoming service requests SLA_l belongs to a certain class c_l ($c_l = 1, \ldots, C$). Each class c ($c = 1, \ldots, C$) is characterized by the probability q_c, that the response time requirements of requests from this class are met:

$$P\{\psi(ep_{lm}) \leq \psi_{l1}\} = q_{c_1} \tag{17}$$

where $\psi(ep_{lm})$ and ψ_{l1} denote respectively the response time of the request SLA_l executed according to the execution plan ep_{lm}, and the response time requirement of request SLA_l.

The aim of the task delivering the average service response time guaranties is to assign each incoming service request SLA_l to such an execution plan ep_{lm^*} for which equation (17) holds. Since the probability $P\{\psi(ep_{lm}) \leq \psi_{l1}\}$ for each service execution plan can be calculated by means of the cumulative distribution function $F_{lm}(\psi_{l1})$ [8], the task of delivering the average service response time guaranties can be formulated as follows:

$$m^* = \arg \min_{m=1,\ldots,m_l} \{F_{lm}(\psi_{l1})\} \tag{18}$$

with respect to:

$$F_{lm}(\psi_{l1}) \geq q_{c_l}$$

Similarly to the task of delivering strict guaranties, it is possible that none of the execution plans allows to obtain the required probability for the response time requirement. In such a case, the execution plan with the highest probability for response time requirement enforcement may be suggested:

$$m^* = \arg \max_{m=1,\ldots,m_l} \{F_{lm}(\psi_{l1})\} \tag{19}$$

10 Numerical Example

To illustrate the presented tasks of optimal execution plan determination a simulation study was carried out. Therefore a simulation environment has been designed and developed to allow the execution of various experiments concerning quality of service aspects in a system based on service oriented architecture paradigm [9].

The simulation environment has been configured for the needs of experiment in the following way. Simulated system consisted of three serially ordered functionalities each having atomic services in three versions: $AS(\varphi_{l1}) = \{as_{l11}, as_{l12}, as_{l13}\}$, $AS(\varphi_{l2}) = \{as_{l21}, as_{l22}, as_{l23}\}$ and $AS(\varphi_{l3}) = \{as_{l31}, as_{l32}, as_{l33}\}$. Moreover, in the simulation system, three classes of requests were considered; request belonging to the first class were served according to the best effort model in which the average complex service response time is minimized; requests from the second class were served according to IntServ model which allows to deliver strict guaranties for complex service maximal response time; requests from the third class were served according to the DiffServ model and has been divided into four subclasses, each class having different average response time guarantee. Subclasses from the DiffServ class have different average guarantee level set. Each request was required to be served in $\psi(ep_{lm}) = 0,5$ second although the first subclass with probability 0.8, the second with 0.7, the third with 0.6 and the fourth with a probability equal to 0.5.

A stream of requests following a Poisson law was connected to the input system, characterized with an average stream intensity $\lambda_0 = 50$. The share of each request class in the overall stream was as follows: best effort: 50%; IntServ: 10% and DiffServ: 40%. Each subclass of DiffServ request had 10% share in overall stream of requests. The ratio of the number of requests from different requests classes was chosen to be similar to the ratio of traffic volume in real computer communication networks.

The aim of the simulation was to evaluate the performance of the proposed approaches to a deliver quality, meant in this experiment as the response time guaranties delivered to the distinguished traffic classes for increasing the value of request arrival intensity. The results of the performed simulation are presented on Figures 7 and 8.

Figure 7 depicts the influence of an increasing request arrival rate on the average service response time for three main classes. Requests from both — best effort and IntServ — classes are served according to a plan that minimizes the average response time. There are two main differences between these classes: for IntServ, the computational and communication resources are reserved to provide strict guaranties. Moreover, the IntServ traffic is treated in prioritized way in comparison to the best effort traffic. Due to the lowest priority of the best effort class, all computational resources of atomic services are assigned to the traffic with higher

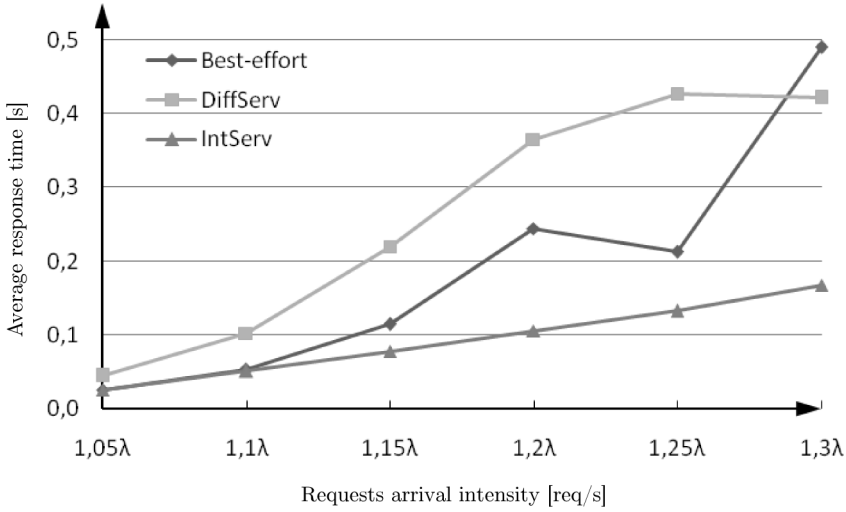

Fig. 7. Influence of increasing request arrival intensity λ on average service response time for three main requests classes: best effort, IntServ, DiffServ

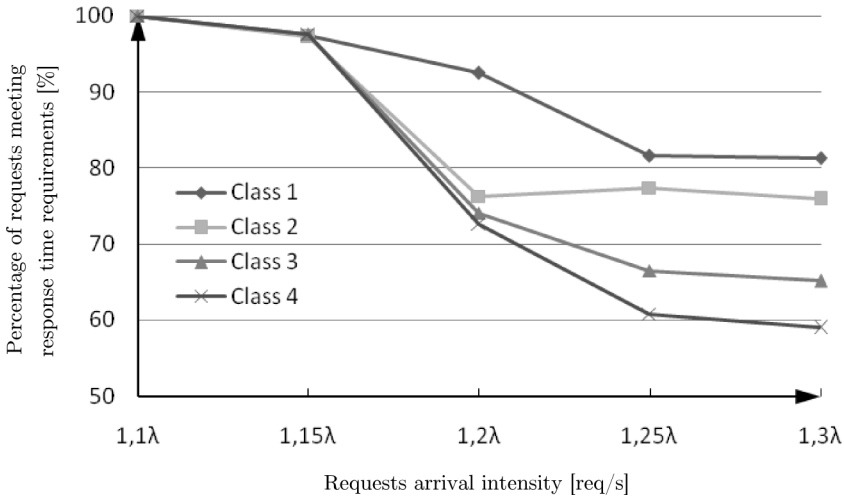

Fig. 8. Influence of the increasing request arrival intensity on the percentage of requests from each subclass of DiffServ meeting its response time requirements

priority and the best effort traffic is provided with only of the resources that are not consumed by other classes.

It is predictable that, for increasing the request arrival intensity, average service response time should grow for all traffic classes. An interesting situation takes place when intensity reaches $\lambda = 1.25\lambda_0$. Average response time of requests from the DiffServ class approaches its required response time at half a second delay, and stops increasing. At the same time, the response time of the best effort class slightly decreases and after a short moment begins to increase rapidly. This situation is caused by the fact, that when the DiffServ class reaches its requirement it does not need as much resources as earlier. Excess resources were assigned to the best effort class, what resulted in a decreased response time. When the request intensity increased, the DiffServ class needed more resources to provide the required response time guaranties. Necessary, resources were taken from the best effort class, what caused a rapid growth of the best effort average response time.

Each subclass of the DiffServ class have different requirements on the percentage of requests meeting response time requirements. The results of quantitative analysis of the influence of increasing the request arrival intensity on the percentage of requests from each subclass of DiffServ, meeting then response time requirements, is presented on Figure 8. One can notice, that as the request arrival rate grows, the percentage of requests not violating the response time guaranties approaches to the required values, which in the presented study were set to 0.8, 0.7, 0.6, and 0.5 for corresponding subclasses.

11 Related Work

The problem of QoS-aware complex service composition in the systems based on service-oriented architecture is being addresses as one of the main research field by both academy and industry [22]. Most of research analyzes the problem of service candidate selection as an optimization problem assuming that the business workflow is given [29]. In general it has been shown that the complex service composition problem is NP-hard when considered as a global optimization task [18, 19]. There are few factors which have an influence to the problem complexity. The main is the exponential growth of the solutions space with increasing number of service candidates, but the multidimensional QoS requirements along with the variety of connecting atomic services possibilities can also cause the problem to be more complicated [26, 27, 28]. The problem's complexity grows exponentialy so only heuristic algorithms can be proposed to obtain a feasible solution in runtime. The metaheuristics like simulated annealing, tabu search or genetic algorithms seems to be a logical idea to use due to intractable nature of the problem. Moreover a local optimization problem has been investigated in order to obtain a

suboptimal solution [24, 25]. Most of researchers used the Integer or Linear programming in order to determine optimal QoS-aware service composition, but the proper problem formulation allows to use the multidimensional multichoice knapsack problem (MMKP) solutions in order to obtain desired composition satisfying all of the QoS constraints. However the MMKP problem has been also shown to be NP-hard [21]. [20] proposes a branch and bound algorithm (BBLP) to find the optimal solution for MMKP. In [30] author proposed an approach to optimize both service semantic functionality fit and the quality of service in single optimization stage. The aggregated task was complex, that is why author used metaheuristics to obtain only a suboptimal solution.

To cope with the complexity of the composition problem we propose to divide the composition process into three stages as shown in the paper. First stage, where the complex service structure is determined, allows to use semantic composition methods in order to satisfy end-user's functional requirements. In the second stage complex service execution scenario is determined, which addresses both aspects of composition process — functional and non-functional requirements satisfaction. The last stage — determination of complex service execution plan — is purely the quality of service optimization stage, which has been mostly investigated in the literature. The presented division of the composition process allows to perform pre-optimization tasks in order to reduce the possible solutions space. As shown in [23] the scenario graph optimization allows to estimate the obtainable complex service execution time before determining the execution plan which is the last optimization stage in our approach.

Besides the pre-optimization advantages, the composition process decomposition allows to investigate various optimization tasks concerning purely functionalities — with use of domain ontologies — in the first stage, and purely QoS-awareness in the last stage. The middle stage — where the complex service execution scenario is determined — allows to find a tradeoff between functional and non-functional requirements satisfaction. The presented three stage process can be also understood as a two layers of optimization (functional and non-functional) with the middle layer not considered in the literature before.

12 Final Remarks

In this work we presented the general method for QoS-aware complex service composition which consists of three stages: complex service structure, execution scenario and execution plan composition. On illustrative examples, we showed how to enhance the quality of complex service by solving certain optimization task in each stage of complex service composition. In particular we showed that complex service scenario optimization, analysing of complex service parallelism degree, may prove useful in the task of quality of service assurance.

Moreover, we showed that it is possible to deliver a required level of quality of service and differentiate it between distinguished request classes by the application of commonly known quality of service assurance approaches in the process of composition of complex service execution plan. It is worth noting, that the presented method uses only few mechanisms (resource reservation, request scheduling) from classical QoS assurance models. The application of all QoS mechanisms (e.g.: traffic shaping and conditioning, request classification, contract renegotiation, congestion control, etc.) as well as knowledge engineering methods [5] (e.g.: prediction of client behavior, adaptive scheduling, atomic services load prediction, etc.) to the management of systems resources may allow to significantly improve delivered quality of service.

The main advantage of our proposed method for complex service composition is that it is not necessary to go through all stages during service composition. In the case, when a new service request arrives in the system, it is possible to make use of partial solutions derived for similar requests which were served earlier. For example a single execution scenario may be used for functionally similar requests. Quality differentiation between these requests is achieved by the application of different execution plans for each request. The determination of semantic similarity between different complex service requests is one of the tasks for our future research.

References

1. Anderson S., Grau A., Hughes C. Specification and satisfaction of SLAs in service oriented architectures. 5th Annual DIRC Research Conference, 141–150, 2005.
2. Garey M., Johnson D., Sethi R. The complexity of flowshop and jobshop scheduling. Math. Oper. Res. 1, 117–129, 1976.
3. Graham R.L., Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G. Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann. Discr. Math. 3, 287–326, 1979.
4. Grzech A. Teletraffic control in the computer communication networks. Wrocław University of Technology, 2002.
5. Grzech A., Świątek P. Parallel processing of connection streams in nodes of packet-switched computer communication networks. Cybernet. Syst. 39, 2, 155–170, 2008.
6. Grzech A., Świątek P. Modeling and optimization of complex services in service-based systems. Cybernet. Syst. 40, 08, 706–723, 2009.
7. Grzech A., Świątek P. The influence of load prediction methods on the quality of service of connections in the multiprocessor environment. Syst. Sci. 35, 3, 7–14, 2009.

8. Grzech A., Rygielski P., Świątek P. QoS-aware infrastructure resources allocation in systems based on service-oriented architecture paradigm. Proc. 6th Working Conf. on Performance Modeling and Evaluation of Heterogeneous Networks, Zakopane, Poland, 35–48, 2010.
9. Grzech A., Rygielski P., Świątek P. Simulation environment for delivering quality of service in systems based on service-oriented architecture paradigm. Proc. 6th Working Conf. on Performance Modeling and Evaluation of Heterogeneous Networks, Zakopane, Poland, 89–98, 2010.
10. Jaeger M.C., Rojec-Goldmann G., Muhl G. QoS aggregation in web service compositions. IEEE Int. Conf. on e-Technology, e-Commerce and e-Service, 181–185, Hong Kong, China, 2005.
11. Jain K.K., Rajaraman V. Parallelism measures of task graphs for multiprocessors. Microproc. Microprogr. 40, 4, 249–259.
12. Johnson R., Gamma E., Helm R., Vlisides J. Design patterns; elements of reusable object-oriented software. Addison-Wesley, New York, 1995.
13. Milanovic N., Malek M. Current solutions for web service composition. IEEE Internet Comp. 8, 6, 51–59, 2004.
14. Narayanan S., McIlraith S. Analysis and simulation of web services. Comp. Networks 42, 5, 675–693, 2003.
15. Rajan R., Verma D., Kamat S., Felstaine E., Herzog S. A policy framework for integrated and differentiated services in the Internet. IEEE Network, 34–41, 1999.
16. Rau B.R., Fisher J.A. Instruction-level parallel processing: history, overview, and perspective. Readings in computer architecture. M.D. Hill, N.P. Jouppi, G.S. Sohi (eds.), Morgan Kaufmann, San Francisco.
17. Wang Z. Internet QoS: architecture and mechanisms for Quality of Service. Academic Press, London, 2001.
18. Ardagna D., Pernici B. Global and local qos guarantee in web service selection. Business Process Management Workshops, LNCS 3812, Nancy, France, 32–46, 2005.
19. Bonatti P.A., Festa P. On optimal service selection. In WWW '05: Proc. 14th int. Conf. on World Wide Web, 530–538, New York, 2005. ACM.
20. Khan S. Quality adaptation in a multisession multimedia system: model, algorithms and architecture. Ph.D. diss. Department of ECE, University of Victoria, Canada, 1998.
21. Martello S., Toth P. Algorithms for Knapsack Problems. Ann. Discr. Math. 31, 70–79, April 1987.
22. Papazoglou M.P., Traverso P., Dustdar S., Leymann F. Service-oriented computing: State of the art and research challenges. IEEE Comp. 40(11), 38–45, 2007.
23. Grzech A., Rygielski P. Translations of service level agreement in systems based on Service Oriented Architecture. Knowledge-Based and Intelligent Information and Engineering Systems, 14th Int. Conf. KES2010, Part II, LNAI 6277, Cardiff, UK, 523–532, 2010.

24. Kwiatkowski J., Pawlik M. Budowa struktur gridowych współpracujących komputerów i agregacji zasobów w strukturach organizacyjnych. Przegl. Elektrotech., 221–225, 2009.
25. Kwiatkowski J., Pawlik M., Fraś M., Konieczny D. Request distribution in hybrid processing environments. Int. Conf. on Parallel Processing and Applied Mathematics, Wrocław, Poland, 2009 (abstract).
26. Kołaczek G., Juszczyszyn K. Smart security assessment of composed Web services. Cyber. Syst. 41, 1, 46–61, 2010.
27. Kołaczek G. Multiagent security evaluation framework for Service Oriented Architecture Systems. Int. Conf. on Knowledge Engineering Systems KES2009, LNAI, Santiago, Chile, 30–37, 2009.
28. Kołaczek G. A logical-based security level evaluation method for service oriented systems. Int. J. Inform. Technol. Secur. 2, 29–42, 2009.
29. Yu T., Zhang Y., Lin K.-J Efficient algorithms for Web services selection with end-to-end QoS constraints. ACM Trans. Web 1, 1, 6, 2007.
30. Lecue F. Optimizing QoS-Aware semantic Web service composition. The Semantic Web - ISWC 2009, LNCS 5823, 375–391, 2009.

Security Assessment of Composed Web Services in a Layered SOA Security Architecture

Krzysztof Juszczyszyn, Grzegorz Kołaczek, and Agnieszka Prusiewicz

Institute of Computer Science
Wrocław University of Technology,
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland
{Krzysztof.Juszczyszyn,Grzegorz.Kolaczek,Agnieszka.Prusiewicz}@pwr.wroc.pl

Abstract. In this work a method for the assessment and optimization of security level of composed Web services, assuming layered security architecture and multiagent approach is presented. As the security evaluation requires the precise definition of the set of evaluation criteria the key criteria for each functional layer of SOA have been proposed. An information fusion scheme, based on Subjective Logic formalism, was used to join information coming from different layers and agents. The framework assumes also that opinions about the credibility of the agents may be build and used jointly with the security assessment.

Keywords: Service Oriented Architecture (SOA), security, complex services, Subjective Logic

1 Introduction

Most organizations deliver their business processes using information technology (IT) applications. Many different software tools are used to capture, transform or report business data. Their role may be for example to structure, define and transform data or to enable or simplify communication. Each such interaction with an IT asset can be defined as a service. The set of delivered from the business processes services provide the incremental building blocks around which business flexibility revolves. In this context, Service Oriented Architecture (SOA) is the application framework that enables organizations to build, deploy and integrate these services independent of the technology systems on which they run [8]. In SOA, applications and infrastructure can be managed as a set of reusable assets and services. The main idea about this architecture was that businesses that use

SOA can respond faster to market opportunities and get more value from their existing technology assets [9].

The final success of the SOA concept can be obtained if many groups, both internal and external to the organization, contribute to the execution of a business process. Because in most cases the most valuable and also sensible part of each organization is information, a business partner is much more willing to share information and data assets, if it knows that these assets will be protected and their integrity maintained. Business partners will also be more likely to use a service or process from another group if it has assurance of that assets integrity and security, as well as reliability and performance. Therefore ensuring security is a one of the most crucial elements while putting SOA approach into practice. Security issues become crucial when complex processes are being composed of atomic services which may have different security properties.

The composition of Web services allows building complex workflows and applications on the top of the SOA model. Besides the obvious software and message compatibility issues a good service composition should be done with respect to the Quality of Service (QoS) requirements. From the point of view of the client preserving the non-functional requirements (availability, performance and security — to name just the most important) is a key factor, which importance rapidly grows in distributed environments where complex services are composed from atomic components [8, 18, 19, 44].

QoS parameters may be defined as observable properties connected with non-functional aspects of the Services. In order to fulfil the non-functional requirements Service Level Agreements (SLAs) are defined or negotiated between service providers and their clients. The SLAs must be precise and unambiguous, they should be also obeyed regardless the current system state and the complexity of services being provided [17, 46].

The QoS characteristics may be expressed in many ways, there are also several QoS specification languages including QML [21] and WSLA [22]. However, the assessment of the actual QoS properties of services which may be composed of atomic components, available in multiple versions which differ in their properties, is rather complex task and there are several optimization problems involved [20, 24, 25]. In some cases sophisticated frameworks are used to deal with the QoS of complex services [23].

In this work a formal model that supports reasoning about security properties of complex services with respect to predefined SOA layers and inherently distributed architecture have been proposed. The chapter is structured as follows. The second section presents the general motivation and works related to the problems of security level evaluation and service oriented architecture. The following section introduces SOA security governance model and defines the layers of the SOA security architecture. The following, fourth, section gives the description of multiagent framework for security evaluation in SOA systems. Subjective Logic — as a model for information fusion and reasoning about trust level of composed

services and software agents is also presented. The fifth section defines the methods for building opinions about the security levels of consecutive SOA layers. The sixth section discusses the properties of SOA security assessments in the form of Subjective Logic's opinions. The last section consists of the conclusion and the direction of future research.

2 Motivation and Related Work

A mobile agent is a composition of computer software and data which is able to move from one host to another autonomously and continue its execution on the destination host. Mobile agent technology can reduce the bandwidth requirement and tolerate the network faults. As the security evaluation process must be accurate and efficient, these basic features relevant to agent and multiagent systems are the main motivation for many researchers to apply multiagent approach to the tasks related to system security. The second premise in this case is the correspondence of the multiagent environment to SOA-based systems. Multiagent systems are composed from the number of autonomous and mobile entities that are able to act both cooperatively and separately. The fundamental concept for SOA-based system is service – entity that could be evoked individually as well as in cooperation with other services. And at last, both multiagent and SOA systems tend to act in heterogenic and highly distributed environment [45].

As the number of SOA-based systems implementation grows, the concerns about security also increases. From its inherited nature, the service oriented systems are distributed. Ensuring the fundamental security proprieties such as integrity, confidentiality, availability of the services in distributed, dynamic and heterogeneous environment is critical challenge for all system administrators. In consequence, apart from providing all the security capabilities expected from enterprise application platforms, service oriented security requires to address the key challenge of providing widely distributed security domains and enabling interoperability across all participants within the system. This includes especially such features as trust-based interactions, credentials and tokens passing across the systems, non-repudiation, authentication and authorization. There is still no fully matured security infrastructure for service-oriented application environments [2, 10]. However, the present SOA infrastructure need to provide security, so most frequently the security is provided by the base infrastructure on which services are provided [4, 9]. The literature related to the security of SOA focuses on problems with threat assessment, techniques and functions for authentication, encryption, and verification of services [1, 2, 6]. Some other works focus on high level modeling processes for engineering secure SOA [4, 9] with trust modeling [7], identity management and access control [12, 10]. Many studies focus on secure software design

practices for SOA, with special interest in architectural or engineering methodologies as the means to create secure services [3, 5].

From the other hand, the rapid development of semantic technologies which assume the use of domain ontologies and semantic similarity assessment methods for service composition and orchestration can be observed. The example approaches to the composition of semantically described services are presented in [26, 27, 28]. The key issue resulting from the use of semantics is the automation of many service management's tasks, and especially service composition [33]. Automation in distributed environments implies the use of agent technologies which are extensively used for service discovery, selection and composition [[29, 30].

In most cases, building complex services converts into a constraint satisfaction problem - there can be many candidates (atomic services) as building blocks of a complex service (process) and it is necessary to select the optimal execution plan. The required composition is expected to satisfy chosen QoS parameters [31]. Several approaches to the assessment of QoS parameters of composed services have been proposed so far but there is no general approach to the estimation of the security level of complex service [32, 34, 37]. The solution proposed in this work fills this gap by proposing information fusion-based model of service security level evaluation.

Moreover, to our best knowledge, the framework proposed in this paper is the first that introduces a multiagent approach to the composed services security level evaluation problem. The motivation for this is rather obvious estimation of security-related service properties involves processing of information coming from different layers and locations of a distributed system. The other important and novel issues addressed in this work are the personalization of the security level evaluation process, multilevel security evaluation, and support for information fusion applied to service security assessment.

3 SOA Security Governance

The consequence of the SOA paradigm are problems related to information assurance in a distributed environment which is designed to provide interoperability and availability. A service-oriented system assumes a scenario with multiple service consumers interacting with multiple providers, all potentially without prior human confirmation of the validity of either the consumer request or the provider response. This concept is strictly related to another key idea for the future Internet — the "Semantic Web". In this model it is no longer necessary for direct human intervention or interpretation to seek or acquire "valid" information. Instead, it is based on the automatic processing of well-defined resource description standards and the high availability of such resources.

The specific problems in a service-oriented environment related to the system environment are as follows [35]:

— identity management; the identity need to be decoupled from the services; all entities like users, services, and so may have their own identities; these identities need to be properly identified so that appropriate security controls can be applied;
— seamless connection to other organizations on a real-time;
— proper security controls management; in SOA, there is a need to ensure that, for composite services, proper security controls are enacted for each atomic service, and for service combinations;
— security management sovereignty; SOA needs to manage security across a range of systems and services that are implemented in a diverse mix of new and old technologies;
— protection of data in transit and at rest;
— compliance with a growing set of corporate, industry, and regulatory standards.

These important security issues arise in the context of the following SOA characteristic features:

— high heterogeneity of resources (both software and hardware);
— distribution of resources;
— intensity and variety of communication;
— different level of resources and tasks granularity;
— resource sharing;
— competition for resources;
— the dynamics of resources;
— the business level requirements related to security.

There are several standards and mechanisms that have been elaborated to provide and to maintain a high security level of SOA-based systems. The basic solutions address the problems of confidentiality and integrity of data processed by SOA-based system. Because of the network context of SOA and the multilevel security risk related to the ISO/OSI network model layers, there are several solutions that offer data protection mechanisms at the corresponding level of each network layer.

The most commonly used and described are standards and protocols from the application layer that are maintained by the OASIS consortium [12]. These solutions have been worked out to support the development of web services and thus SOA-based systems. The other type of protection methods, mechanisms and protocols, like for example IPv6, are common for all network applications and can be used in SOA-based systems as well in any other type of software.

As a SOA-based system can be defined by its five functional layers (Tab. 1) the corresponding definition of SOA security requirements for a security evaluation

Table 1. The security evaluation requirements for each of the SOA functional layers (selection)

SOA Layer	Evaluate/verify/test
Policy and Business Processes	Policy consistency Policy completeness Trust management Identity management
Service	Identification Authentication Security management of the complex services
Service Description	Description completeness Availability Protection from attacks
Service Communication Protocol	Confidentiality Authentication Norms compliance
Transport	Availability Protection from attacks Integrity

process should address the specific security problems within each layer. A subset of security requirements for the SOA layers has been presented in Table 1. The complete list can be found in [5].

4 Multiagent Framework for SOA Security Evaluation

There are several different problems considering the SOA security level evaluation process. The most crucial, as it has been stated in the earlier sections are: the complexity of the architecture, the multilevel relationships between the system components and the heterogenity of the environment. Each security level evaluation method and tool must take into account all these factors and apply the appropriate solutions for them to provide the accurate final results of the security evaluation process.

4.1 Architecture of a Multiagent System for SOA Security Level Evaluation

This section presents the main assumptions about a multiagent SOA-based system security evaluation framework. The main idea about this framework is the application of a multiagent approach. As systems implementing a Service Oriented

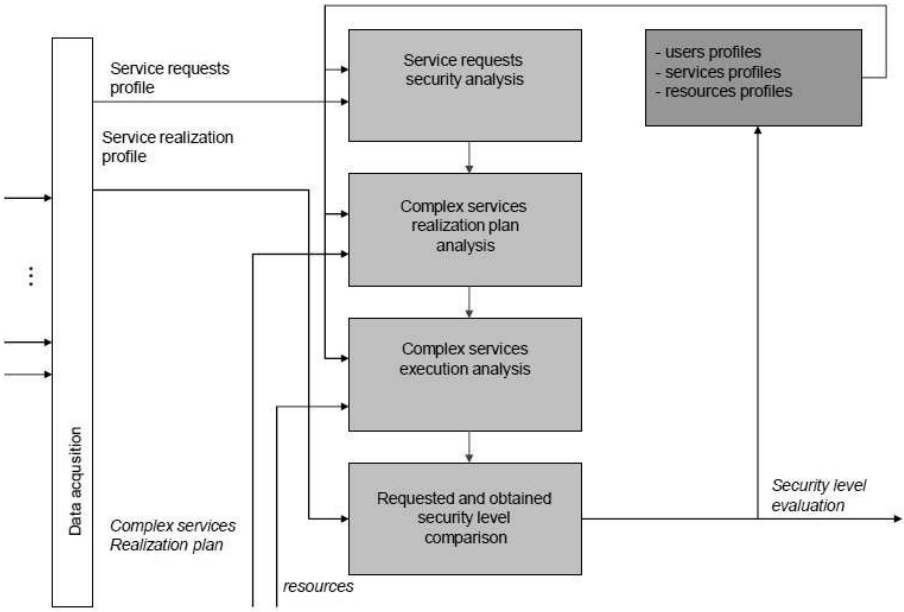

Fig. 1. General architecture for service-based systems security level evaluation

Architecture are often geographically and logically dispersed, an appropriate tool for monitoring and controlling all components is necessary [39]. The multiagent approach offer all the relevant mechanisms and concepts and seems to be the best solution in the described situation [40].

The idea of the SOA-based systems security level evaluation presented in this chapter is built on the assumption that the security level of any complex service is related to the following set of characteristics [36]:

— the profiles of services execution requests generated by system users and other services and the way how these requests are handled;
— the system interface characteristics;
— the complexity of service composition;
— the utilization profile of the system and telecommunication resources.

According to this, the general architecture for security level evaluation we propose, has the following functional components (Fig. 1):

— services requests security analysis;
— complex services realization plan analysis;
— complex services execution analysis;
— requested and obtained security level comparison;

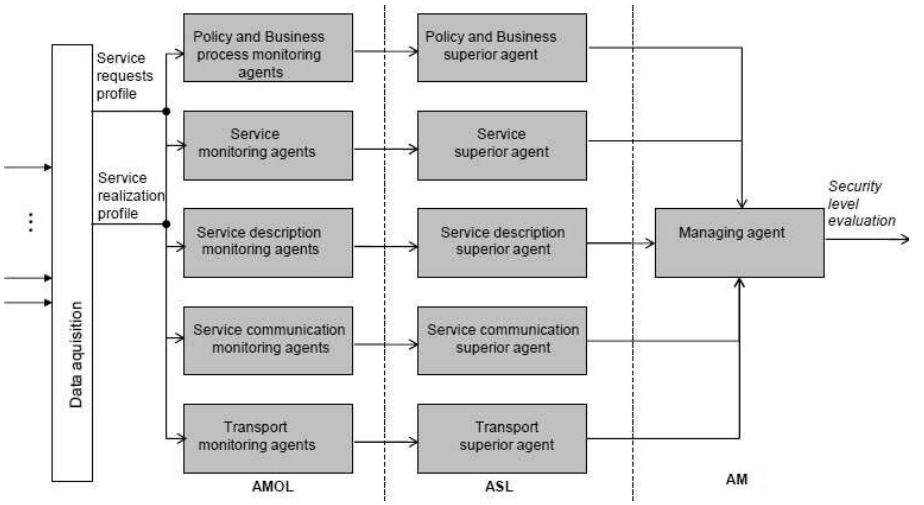

Fig. 2. Detailed architecture of the multi-agent system for service-based system security level evaluation

The aim of the component responsible for service requests security analysis is to evaluate the requested user (or the service) security level taking into account Business Processes Layer variables such as Service Level Agreement (SLA), security policy, etc. The next component "Complex services realization plan analysis" is responsible for atomic and complex services security level evaluation according to the available service composition plan. The component "Complex services execution analysis" evaluates the security level using the information about the computational and communicational resources utilization. The last component is responsible for the evaluation of the correspondence between the requested and the obtained security level.

Described above components use information provided by the data acquisition module and central repository. The data acquisition module supports the security evaluation by delivering up to date information about service requests and service realization profiles (e.g. which services have been requested and what resources have been used to enable the service realization). The central repository brings historical records describing user activity (users profiles), services activity (services profiles) and system resources (resources profiles).

The process of security level evaluation managing all the mentioned components is performed by a multi-agent system (Fig. 2).

This multiagent system architecture for SOA security evaluation introduces the following agents classes (their functionalities has been explained in Table 2):

1. AMOL — Agents for MOnitoring Layer security levels — monitoring agents;

2. ASL — Agents for Supervising the monitoring agents of the corresponding Layer — superior agents;
3. AM — Agents for data fusion and Management — managing agents;
4. AC — Agents representing Clients — services consumers agents.

Monitoring agents AMOLs observe the security-related properties of the services associated with a given layer and generate partial opinions about their security level. Superior agents (ASLs) acquire partial opinions and use them to compute opinions about the security of each level of the complex Web service under consideration. The result is a set of opinions which is then processed by the monitoring agent AM. Its role is to perform information fusion by joining the opinions originating from different levels and generate a global opinion about the security of a service as a whole.

The most important functionality related to the SOA security level evaluation architecture is the description of all the components, mechanisms and relations that are necessary to precisely evaluate the security level of the particular SOA system. As it was described in 3, the problem of security evaluation is complex and there exist more than one solution that could be acceptable within a context of a particular system and its environment. This part of the paper describes some general ideas about SOA security level evaluation in relation to the requirements listed in the Table 1 and multiagent architecture presented in the Figure 2.

Below two tasks and the corresponding algorithms for security level evaluation using information related to functional SOA layer are defined.

Task 1. Security level evaluation for a separate SOA functional layer.

Given:

— N — ordinal number of SOA layer
— ac_k — k-th client of the SOA-based system
— $p_1, p_2, \ldots, p_l$ — details or preferences related to ac_k request
— asl — agent superior layer
— $amol\_n_1, \ldots, amol\_n_m$ — set of specialized agents that perform security evaluation using appropriate tests, mechanisms, etc. related to the *n-th* SOA layer

Result:

— L_n — security level value for *n-th* layer

Algorithm 1

BEGIN
— ac_k prepare and send to am a request concerning the security level of the *n-th* layer of the SOA system;
— am find all the monitoring agents related to *n-th* layer ($amol\_n_1, \ldots, amol\_n_m$), prepare and send the appropriate requests to them;

Table 2. The characteristic of the agent classes

AMOL_1= $\{amol\_1_1, amol\_1_1, \ldots, amol\_1_n\}$	• Set of autonomous agents which perform the security level evaluation related to the tasks defined in Table 1 • For example for the first functional layer of SOA — the transport layer $amol\_1_1$ may be an agent that evaluates the confidentiality of the transport layer, $amol\_1_2$ may be an agent that evaluates data integrity at the transport layer level, etc.
AMOL_2= $\{amol\_2_1, \ldots, amol\_2_m\}$ AMOL_3= $\{amol\_3_1, \ldots, amol\_3_l\}$ AMOL_4= $\{amol\_4_1, \ldots, amol\_4_o\}$ AMOL_5= $\{amol\_5_1, \ldots, amol\_5_p\}$	• Sets of autonomous agents for corresponding four SOA layers (transport, ..., business processes) that perform specific security evaluation tasks related to each particular layer, as described in Table 1
ASL=$\{asl\_1, \ldots, asl\_5\}$	• For each SOA functional layer there has been defined one *superior agent* • The *superior agents* range of responsibility is to coordinates all the tasks related to the security evaluation process for the particular SOA functional level, to collects the results provided by *amol* agents, to interpret the results provided by *amol* agents and finally to present the results of the security level to *managing* agent and to *client agents*
AM=$\{am\}$	• The *managing agent* is responsible for the most top-level security evaluation; it coordinate the activity of asl agents, collect the results of the SOA layer evaluation, combines all security level related information and produces the general SOA security level value, serves the *consumer* agents requests
AC=$\{ac_1, ac_2, \ldots, ac_q\}$	• SOA *services consumers* agents collect the information about security level of provided by SOA systems services and evaluates the security level of composite services

— Monitoring agents ($amol\_n_1,\ldots, amol\_n_m$) perform the security evaluation tasks using all tools, methods, algorithms, etc. available to them;
— *asl* collects the results obtained by all monitoring agents and using the specific algorithm (data fusion, consensus operator, etc.) and taking into account the list $p_1, p_2,\ldots, p_l$ of ac_k preferences evaluates the final security level value of the *n-th* layer of this SOA system;
— *am* returns L_n to the ac_k.

END

The role of the managing agent AM is to integrate the security assessments coming from different layers and to provide a top-leves security estimate for given service. This is done with the following algorithm:
Task 2. SOA security level evaluation.
Given:

— ac_k — k-th client of the SOA-based system;
— *am* — managing agent;
— $p_1, p_2,\ldots, p_l$ — details or preferences related to ack request.

Result:

— L_{soa} — SOA system security level.

Algorithm 2

BEGIN
— ac_k prepare and send to *am* a request concerning the security level of the SOA system;
— Using *Algorithm 1* managing agent *am* evaluates $L_1,\ldots, L_n$ — security levels for all SOA system's layers;
— Managing agent *am* evaluate L_{soa} the final security level value of the SOA system using selected data fusion methods and taking into account the list $p_1, p_2,\ldots, p_l$ of ac_k preferences;
— *am* returns L_{soa} to the ac_k.

END

Regardless of the approach for opinion fusion (a specific solution of this important problem will be proposed in the following sections), a final opinion about the security level of the service under consideration can be evaluated. It takes into account all the layers defined for the SOA architecture and reflects the measured properties of the service.

However, in order to perform an information fusion process integrating information from different layers (which are differently grounded and typically has

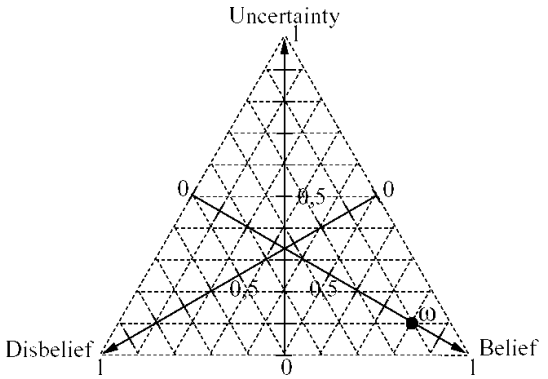

Fig. 3. The Opinion triangle [14]

even different physical interpretation) a formal method to underline the information exchange between the above defined agents is required. Such a method, the Subjective Logic, will be introduced in the following section.

4.2 Subjective Logic

Subjective Logic was proposed by Josang as a model for reasoning about trust in secure information systems [15]. It is compatible with Dempster-Shafer's theory of evidence [14] and binary logic [16]. Subjective Logic includes standard logic operators and additionally two special operators for combining beliefs – consensus and recommendation. The definitions in this section introduce the fundamental notions of Subjective Logic and come from [14] and [15].

When expressing belief about a statement (predicate) it is assumed that it is either true or false, but one is not certain about it — one is only able to have an opinion about it (because of imperfect knowledge). Let's denote *belief*, *disbelief* and *uncertainty* as b, d and u respectively.

Definition 1 (Subjective Logic's opinion). A tuple $\omega = \langle b, d, u\rangle$ where $\langle b, d, u\rangle \in [0,1]^3$ and b + d + u =1 is called an *opinion*.

From Definition 1 we can graphically express an opinion as a point belonging to *opinion triangle* (Fig 3. — point ω marks opinion $\langle 0.8, 0.1, 0.1\rangle$).

Opinions have always assigned membership (are expressed by certain agents) and are not inherent qualities of objects but *judgments* about them. For any opinions $\omega_p = \langle b_p, d_p, u_p\rangle$ and $\omega_q = \langle b_q, d_q, u_q\rangle$ about predicates p and q the following operators may be defined (proofs and in-depth discussion are to be found in [14]):

Definition 2 (Conjunction)

$$\omega_{p\wedge q} = \langle b_p b_q, d_p + d_q - d_p d_q, b_p u_q + u_p b_q + u_p u_q \rangle \tag{1}$$

Definition 3 (Disjunction)

$$\omega_{p\vee q} = \langle b_p + b_q - b_p b_q, d_p d_q, d_p u_q + u_p d_q + u_p u_q \rangle \tag{2}$$

Definition 4 (Negation)

$$\omega_{\neg p} = \langle d_p, b_p, u_p \rangle \tag{3}$$

Now assume two agents, A and B, where A has an opinion about B. An opinion about other agent is interpreted as an opinion about proposition "B's opinion is reliable". We will denote an opinion expressed by agent B about a given predicate p and agent's A opinion about B as ω_p^B and ω_p^A respectively. Then the opinion of agent A about p is given by *the discounting operator* (a.k.a *reputation* operator):

Definition 5 (Recommendation, denoted by $\otimes$)

$$\omega_p^{AB} = \omega_p^A \otimes \omega_p^B = \langle b_B^A b_p^B, b_B^A d_p^B, d_B^A + u_B^A + b_B^A u_p^B \rangle \tag{4}$$

The joint opinion of two agents A and B about a given predicate is computed by *the consensus* operator (ω_B^A and ω_p^B are opinions of A about B and B's about p):

Definition 6 (Consensus, denoted by $\oplus$)

$$\omega_p^{AB} = \omega_B^A \oplus \omega_p^B = \left\langle (b_p^A u_p^B + b_p^B u_p^A)/\ k, (d_p^A u_p^B + d_p^B u_p^A)/\ k, u_p^A u_p^B\ /\ k \right\rangle \tag{5}$$

where $k = u_p^A + u_p^B - u_p^A u_p^B$

The consensus operator is commutative and associative thus allowing to combine more opinions. Note that $\oplus$ is undefined for so-called *dogmatic opinions* (containing b_p=1 or d_p=1), reflecting that there is no joint opinion, if one is absolutely certain about a given fact. Opinions about binary events can be projected onto a 1-dimensional probability space resulting in *probability expectation* $E(\omega_p)$ value for a given opinion:

Definition 7 (Probability expectation)

$$E(\omega_{\ p} = E(\langle b, d, u \rangle) = b + \frac{u}{2} \tag{6}$$

When ordering opinions the following rules (listed by priority) hold:

1. The opinion with the greatest probability expectation E is the greatest,

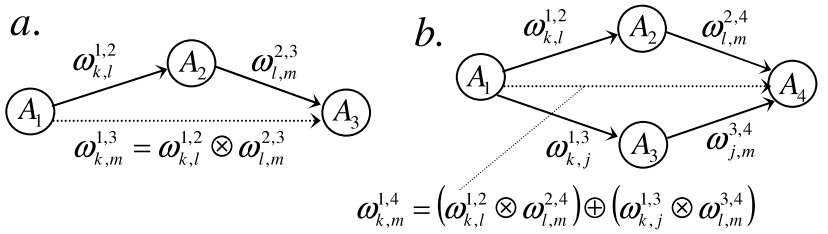

Fig. 4. The use of recommendation and consensus operators

2. The opinion with the smallest uncertainty is the greatest,

Thus, for instance when three opinions are considered: $\omega_1 = \langle 0.4, 0.2, 0.4 \rangle$, $\omega_2 = \langle 0.5, 0, 0.5 \rangle$, $\omega_3 = \langle 0.2, 0, 0.8 \rangle$ the proper order of them, according to equitation 6 is as follows: $\omega_2 > \omega_1 > \omega_3$. What can be interpreted as the situation where ω_2 denotes 'best opinion' (formally, probability expectation E of opinion ω_2 is the greatest) while ω_3 the 'worst one' (formally, probability expectation E of opinion ω_3 is the smallest).

Figure 4 shows a possible joint application of recommendation and consensus operators are shown.

If opinions (Fig. 4a) $\omega_{k,l}^{1,2}$ and $\omega_{l,m}^{2,3}$ are known (say $\omega_{k,l}^{1,2} = \langle 0.8, 0.1, 0.1 \rangle$ and $\omega_{l,m}^{2,3} = \langle 0.7, 0.1, 0.2 \rangle$), any agent of ASL or AM type is able to compute — with recommendation operator — the value of $\omega_{k,m}^{1,3}$ On the Fig.4b the application of the consensus operator (5) is shown to build the opinion $\omega_{k,m}^{1,4}$ with the help of joined opinions of agents A_2 and A_3.

4.3 Security Level Evaluation

At this point two different scenarios for generating a general opinion about the security level can be proposed (the architecture of the system is presented on Figure 5):

Strict security assessment. a Subjective's Logic conjunction operator is used to fuse opinions and and resulting service security assessment ω_{SS} for the service has form: $\omega_S S = \omega_{pbp}$ ^ω_{srv} ^ω_{srvd} ^ω_{scp} ^ω_{trl}. According to the definition of conjunction operator, the belief component of the resulting opinion is close to the lowest security level measured for service layers — it is a case where the layers of the service architecture are treated as dependent from each other in the context of security assessment.

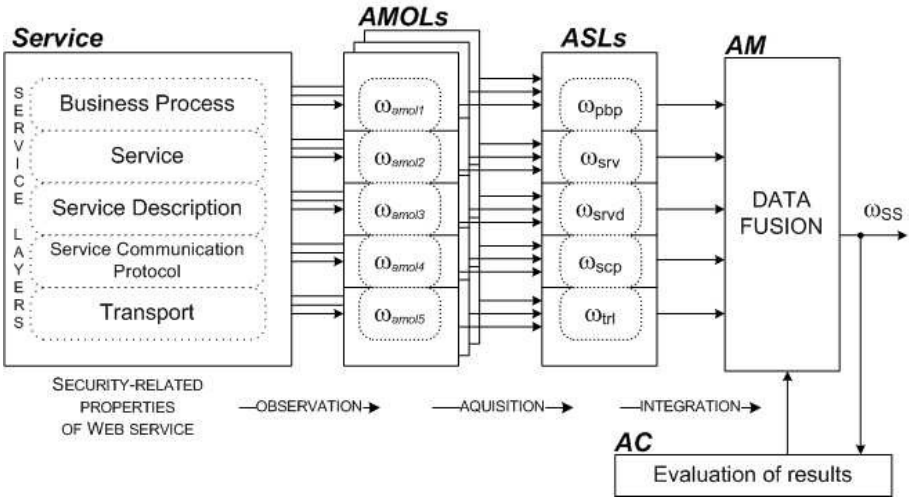

Fig. 5. The application of Subjective Logic opinions in the multiagent system for SOA security level evaluation

General security assessment. Subjective's Logic consensus operator is used to fuse the opinions and the resulting service security assessment ω_{SS} for the service has form: $\omega_{SS} = \omega_{pbp} \oplus \omega_{srv} \oplus \omega_{srvd} \oplus \omega_{scp} \oplus \omega_{trl}$. In the case of the consensus operator conflicting opinions (with clearly different belief and disbelief components), the final values are averaged,and so the final opinion reflects the average security assessment of the service. In this case the layers of the service architecture are independent.

The last part of the proposed architecture is a consumer agent AC. Its role is to provide the possibility of tuning the framework and take into consideration the feedback of users. In general, the AC uses the possibility provided by Subjective Logic formalism which allows to express opinions not only about facts but also about the subjects who formulate opinions. Then the recommendation operator ($\otimes$, Def. 5) is used to evaluate the opinion with the trust level granted to the subject who expressed it. In the case of high value of the disbelief component of the opinion about an agent, the uncertainty of the resulting opinion grows, reflecting the fact that the subject is not trusted. In the context of the presented framework this feature will be used to differentiate the opinions about the SOA layers and react to the cases in which the users will perceive the opinions as contradictory with their experience (i.e. services with high security assessments will cause security accidents).

Another scenario (not addressed in this chapter) assumes the use of opinions about the agents to express the knowledge about their effectiveness — if an agent

has no access to complete information about the service under monitoring (for example: due to network failure) this may be reflected in the opinion associated with him.

4.4 Aggregation of Security Properties for Service Composition Plans

Service composition assume serial, AND-parallel or XOR-parallel execution plans (Fig. 6). After having assessed the security of atomic services (including levels), the security of a composed service may be considered. In general, it is assumed that security of aserial execution plan, where a chain of services executed one-by-one, is defined by the security of the "weakest point" in the chain (indicating the possibility that any security breach may occur during execution of a complex service). The same concerns an AND-parallel execution plan. In the case of XOR-parallel plan, where one actually does not know which service will be executed, the security level of both services must be taken into account.

The above considerations result in the following rules for generating an opinion about execution plan's security (given that opinions about security of any component service are already computed according to the rules defined in the preceding sections):

1. Serial execution plan: opinion is computed by applying Subjective Logic's AND operator ($\hat{}$, Def. 2) to all opinions in the plan.
2. AND-parallel execution plan: opinion is computed by applying Subjective Logic's AND operator ($\hat{}$, Def. 2) to all opinions in the plan.
3. XOR-parallel execution plan: opinion is computed by applying Subjective Logic's Consensus operator ($\otimes$, Def. 6) to all opinions in the plan.

The resulting opinion gives us the security assessment (opinion) of given execution plan of the composed service.

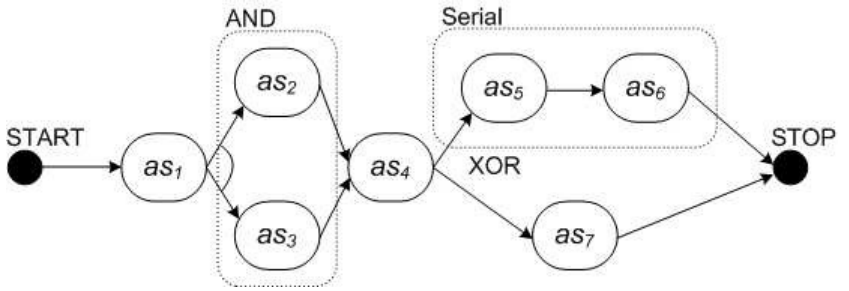

Fig. 6. Sample execution plan complex service

An illustrative example is given on Figure 6. For this simple execution plan (and assuming that the managing agent AM has generated opinions ω_{as1}, ..., ω_{as7} about the security of component atomic services as_1, ..., as_7), the final security assessment will be an opinion: $\omega = \omega_{as1} \hat{} (\omega_{as2} \hat{} \omega_{as3}) \hat{} \omega_{as4} \hat{} ((\omega_{as5} \hat{} \omega_{as6}) \otimes \omega_{as7})$.

It should be noted, that the information used for security assessment comes from the sources which are fundamentally different by origin and technical nature (which also concerns the layers defined in sec. 4). For that reason two general strategies for generating opinions about composed services have been proposed:

1. General security evaluation — based on the general opinions generated by managing agent AM in result of the information fusion process.
2. Layer dependant security evaluation — based on layer-specific opinions produced by layer's superior agents ASL. In this case the abovementioned rules for generating opinions about the security of execution plan apply separately to opinions concerning each specific layer of any component service involved in given execution plan.

This flexible strategy allows the generation of opinions with the required granularity and more in-depth security analysis of a composed service. The next section introduces a set of methods for generating opinions about all layers of SOA security architecture.

5 Evaluation of Subjective Logic Opinions for SOA Layers

5.1 Subjective's Logic Opinions — the Interpretation

This section describes how the security level of the particular SOA layer can be expressed using Subjective Logic opinions and operators. The aim is to assign quantitative values to the measured security of the SOA architectural levels. Opinions of Subjective Logic are tuples composed of three values ($\omega = \langle b,d,u \rangle$), for which the following interpretation have been defined:

1. Belief component of the Subjective Logic's opinion reflects the trust in the security level of the service under consideration. Its value close to one literally means that one perceives the service as safe.
2. Disbelief component reflects the opinion that a service is not safe and may cause security breaches.
3. Uncertainty component is to express that the knowledge is partial or incomplete and the security assessment does not give a definite result.

In order to have an easy interpretation to the Subjective Logic's opinions about the security of services, a quantitative value associated with the security level of the service will be given by probability expectation (Def. 7). The services with the resulting probability expectation close to one will be treated as safe, while the value approaching zero will suggest that the service is unsecure.

5.2 Policy and Business Processes Layer

The Subjective Logic opinion about the Policy and Business Processes layer will be calculated as the conjunction of following opinions:

$$\omega_{\mathrm{pbp}} = \omega_{\mathrm{pcon}} \hat{} \omega_{\mathrm{pcom}} \hat{} \omega_{\mathrm{trm}} \hat{} \omega_{\mathrm{idm}}$$

where: ω_{pcon} — Subjective Logic opinion about policy consistency, ω_{pcom} — Subjective Logic opinion about policy completeness, ω_{trm} — Subjective Logic opinion about trust management, ω_{idm} — Subjective Logic opinion about identity management.

These opinions can be evaluated in the following way:

$$\omega_{\mathrm{pcon}} =< b_{\mathrm{pcon}}, d_{\mathrm{pcon}}, u_{\mathrm{pcon}} >$$

where: $b_{\mathrm{pcon}} = verified * satisfaction$, $u_{\mathrm{pcon}} = 1 - verified$

$$verified = \begin{cases} 0 & iff\ the\ policy\ consistency\ has\ not\ been\ verified \\ 1 & iff\ the\ policy\ consistency\ has\ been\ verified \end{cases}$$

$$\omega_{\mathrm{pcom}} =< b_{\mathrm{pcom}}, d_{\mathrm{pcom}}, u_{\mathrm{pcom}} >$$

where: $b_{\mathrm{pcom}} = verified * satisfaction$, $u_{\mathrm{pcom}} = 1 - verified$

$$verified = \begin{cases} 0 & iff\ the\ policy\ completness\ has\ not\ been\ verified \\ 1 & iff\ the\ policy\ completness\ has\ been\ verified \end{cases}$$

$$\omega_{\mathrm{trm}} =< b_{\mathrm{trm}}, d_{\mathrm{trm}}, u_{\mathrm{trm}} >$$

where: $b_{\mathrm{trm}} = exist * satisfaction\_level$, $u_{\mathrm{pcon}} = 1 - exist$

$$exist = \begin{cases} 0 & iff\ the\ trust\ management\ has\ not\ been\ applied\ at\ this \\ & SOA\ layer \\ 1 & iff\ the\ trust\ management\ has\ been\ applied\ at\ this\ SOA\ layer \end{cases}$$

$satisfaction \in < 0, 1 >$ — represents user satisfaction level regarding trust management method

$$\omega_{\text{idm}} =< b_{\text{idm}}, d_{\text{idm}}, u_{\text{idm}} >$$

where

$$\begin{aligned} b_{\text{idm}} = exist * (&user.control + minimal.disclosure + directed.identity \\ &+decentralized + bidirectional + single.logout \\ &+scalability)/7 \end{aligned}$$

All other variables used in b_{idm} evaluation have values taken from the binary set [0,1] where value 1 means that the identity management implements the particular functionality (e.g. directed identity management, user control, etc.).

The evaluation of the Subjective Logic opinion for all subsequent SOA layers is performed in similar way, with respect to the specific nature of security properties of the given layer.

5.3 Service Layer

The Subjective Logic opinion about the Service layer is the conjunction of following opinions:

$$\omega_{\text{srv}} = \omega_{\text{auth}} \hat{} \omega_{\text{cmpx}} \hat{} \omega_{\text{id}}$$

where ω_{auth} — Subjective Logic opinion about service authentication, ω_{cmpx} — Subjective Logic opinion about security of complex services composition, ω_{id} — Subjective Logic opinion about service identity.

These opinions can be evaluated in the following way:

$$\omega_{\text{auth}} =< \ b_{\text{auth}}, d_{\text{auth}}, u_{\text{auth}} \ >$$

where $b_{\text{auth}} = (bio * s_1 + pass * s_2 + tok * s_3 + hidden * s_4)$ $u_{\text{auth}} = 0$, bio, pass, tok, hidden $\in$ [0,1] — a particular variable which value equals 1 to indicate that the authentication method that has been applied (biometric, password, token, hidden) $s_1, ..., s_4$ weights characterizing the quality of the authentication method, $(s_1 + s_2 + s_3 + s_4) = 1$

$$\omega_{\text{cmpx}} =< b_{\text{cmpx}}, d_{\text{cmpx}}, u_{\text{cmpx}} >$$

where $b_{\text{trm}} = exist * satisfaction\_level$ $u_{\text{trm}} = 1 - exist$

$$exist = \begin{cases} 0 & iff \ no \ specialized \ security \ mechanisms \ has \ not \ been \ applied \\ 1 & iff \ some \ specialized \ security \ mechanisms \ has \ not \ been \ applied \end{cases}$$

$satisfaction \in < 0, 1 >$ — represents the user satisfaction level regarding security of complex services composition.

$$\omega_{\mathrm{id}} =< b_{\mathrm{id}}, d_{\mathrm{id}}, u_{\mathrm{id}} >$$

where: $b_{\mathrm{id}} = exist * (unique + universal + friendly\ )/3\ u_{\mathrm{trm}} = 1 - exist$

$$exist = \begin{cases} 0 & iff\ the\ identification\ has\ not\ been\ applied\ at\ this\ SOA\ layer \\ 1 & iff\ the\ identification\ has\ been\ applied\ at\ this\ SOA\ layer \end{cases}$$

All other variables used in b_{idm} evaluation have values from the binary set [0,1] where value 1 means that the identity management implements the particular functionality (e.g. directed identity management, user control, etc.).

5.4 Service Description Layer

The Subjective Logic opinion about the Service description layer is the conjunction of the following opinions:

$$\omega_{\mathrm{srvd}} = \omega_{\mathrm{sdcom}} \hat{} \omega_{\mathrm{av}} \hat{} \omega_{\mathrm{aprot}}$$

where ω_{sdcom} — Subjective Logic opinion about service description completeness, ω_{av} — Subjective Logic opinion about availability of services description, ω_{aprot} — Subjective Logic opinion about system harndess.

$$\omega_{\mathrm{sdcom}} =< b_{\mathrm{sdcom}}, d_{\mathrm{sdcom}}, u_{\mathrm{sdcom}}$$

where $b_{\mathrm{sdcon}} = verified\ u_{\mathrm{sdcon}} = 1 - verified$

$$verified = \begin{cases} 0 & iff\ the\ service\ description\ completness\ has\ not\ been \\ & verified \\ 1 & iff\ the\ service\ description\ completness\ has\ been\ verified \end{cases}$$

$$\omega_{\mathrm{av}} =< b_{\mathrm{av}}, d_{\mathrm{av}}, u_{\mathrm{av}} >$$

where $b_{\mathrm{av}} = served/requested\ u_{\mathrm{av}} = 1 - active\_time/total\_time$, served — variable value denotes the number of request that have been served successfully, request — variable value denotes the number of total request addressed to the service description layer, active_time — the period of time when service description has been available, total_time — the global period when the availability of service description has been observed

$$\omega_{\mathrm{aprot}} =< b_{\mathrm{aprot}}, d_{\mathrm{aprot}}, u_{\mathrm{aprot}} >$$

where:

$$b_{\text{aprot}} = \frac{\sum_{i=1}^{n} WP_i + \sum_{j=1}^{m} OPN_j}{(n+m)}$$

$u_{\text{aprot}} = 1-(n+m)/total$, WP_i — binary variable value which denotes the result of a particular penetration test, WP_i= 1 iff the test was successful, 0 otherwise, OPN_j — binary variable value which denotes the result of a vulnerability search, WP_i= 1 iff the vulnerability has not been found, 0 otherwise, n — number of performed penetration tests, m — number of vulnerabilities that has been tested, *total* — the total number of all know penetration tests and vulnerabilities

5.5 Service Communication Protocol Layer

Subjective Logic opinion about Service Communication Protocol Layer is the conjunction of following opinions:

$$\omega_{\text{scp}} = \omega_{\text{conf}} \hat{} \omega_{\text{auth}} = \hat{} \omega_{\text{norm}}$$

where ω_{conf} — Subjective Logic opinion about communication protocol confidentiality, ω_{auth} — Subjective Logic opinion about communication protocol authentication, ω_{norm} — Subjective Logic opinion about communication protocol norm compliance

$$\omega_{\text{conf}} =< \ b_{\text{conf}}, d_{\text{conf}}, u_{\text{conf}} \ >$$

where

$$b_{\text{conf}} = exist * \left(\min\left(1, symmetric * \frac{keylength}{2048}\right) + \min\left(asymmetric * \frac{keylength}{256}\right)\right)$$

$u_{\text{conf}} = 1 - exist$

$$exist = \begin{cases} 0 & iff\ the\ data\ has\ not\ been\ enciphered\ at\ this\ SOA\ layer \\ 1 & iff\ the\ data\ has\ been\ enciphered\ at\ this\ SOA\ layer \end{cases}$$

keylenght — cryptographic key length, symmetric, asymmetric — according to the applied cryptographic protocol, variable value is 0 or 1

$$\omega_{\text{auth}} =< b_{\text{auth}}, d_{\text{auth}}, u_{\text{auth}} >$$

All Subjective Logic opinion components of the authentication at this SOA layer has been defined in the same way as it has been shown for the Service layer.

$$\omega_{\text{norm}} =< b_{\text{norm}}, d_{\text{norm}}, u_{\text{norm}} >$$

where $b_{\text{norm}} = verified * result$, $u_{\text{norm}} = 1 - verified$,

$$verified = \begin{cases} 0 & iff\ the\ policy\ consistency\ has\ not\ been\ verified \\ 1 & iff\ the\ policy\ consistency\ has\ been\ verified \end{cases}$$

result $\in \langle 0,1 \rangle$ — variable which denotes compliance to norms

5.6 Transport Layer

The Subjective Logic opinion about the Transport Layer security is the conjunction of the following opinions:

$$\omega_{\text{trl}} = \omega_{\text{av}} \hat{} \omega_{\text{int}} \hat{} \omega_{\text{aprot}}$$

where ω_{ar} — Subjective Logic opinion about transport layer availability, ω_{in} — Subjective Logic opinion about transport layer integrity, ω_{ar} — Subjective Logic opinion about transport layer protection from attacks

$$\omega_{\text{av}} =< b_{\text{av}}, d_{\text{av}}, u_{\text{av}} >$$

All Subjective Logic opinion components of the availability at this SOA layer has been defined in the same way for the Service Description layer.

$$\omega_{\text{aprot}} =< b_{\text{aprot}}, d_{\text{aprot}}, u_{\text{aprot}} >$$

All Subjective Logic opinion components of the protection from the attacks at this SOA layer has been defined in the same way as at Service Description layer.

$$\omega_{\text{int}} =< b_{\text{int}}, d_{\text{int}}, u_{\text{int}}$$

where $b_{\text{int}} = exist * strength$, $u_{\text{trm}} = 1 - exist$

$$exist = \begin{cases} 0 & iff\ the\ trust\ management\ has\ not\ been\ applied\ at\ this \\ & SOA\ layer \\ 1 & iff\ the\ trust\ management\ has\ been\ applied\ at\ this\ SOA\ layer \end{cases}$$

strength $\in < 0, 1 >$ — represents the confidence about the strength of the applied integrity protection mechanisms.

6 The Properties of the Security Assessments

This section presents the basic properties of security assessments in the form of opinions combined by Subjective Logic's operators. It shows how to obtain the aggregated security level for a SOA-based system.

The first example demonstrates the situation when all superior agents (ASL) returned the same opinions describing the security levels related to all distinguished SOA layers (Table 3). In the presented approach it means that $\omega_{pbp} = \omega_{srv} = \omega_{srvd} = \omega_{scp} = \omega_{trl}$.

Table 3. Various values of opinions

	ω_{pbp}	ω_{srv}	ω_{srvd}	ω_{scp}	ω_{trl}	ω_{ss}
op_1	<1,0,0>	<1,0,0>	<1,0,0>	<1,0,0>	<1,0,0>	1
op_2	<0.9,0.1,0>	<0.9,0.1,0>	<0.9,0.1,0>	<0.9,0.1,0>	<0.9,0.1,0>	0,6561
op_3	<0.8,0.2,0>	<0.8,0.2,0>	<0.8,0.2,0>	<0.8,0.2,0>	<0.8,0.2,0>	0,4096
op_4	<0.7,0.3,0>	<0.7,0.3,0>	<0.7,0.3,0>	<0.7,0.3,0>	<0.7,0.3,0>	0,2401
op_5	<0.6,0.4,0>	<0.6,0.4,0>	<0.6,0.4,0>	<0.6,0.4,0>	<0.6,0.4,0>	0,1296
op_6	<0.5,0.5,0>	<0.5,0.5,0>	<0.5,0.5,0>	<0.5,0.5,0>	<0.5,0.5,0>	0,0625
op_7	<0.4,0.6,0>	<0.4,0.6,0>	<0.4,0.6,0>	<0.4,0.6,0>	<0.4,0.6,0>	0,0256
op_8	<0.3,0.7,0>	<0.3,0.7,0>	<0.3,0.7,0>	<0.3,0.7,0>	<0.3,0.7,0>	0,0081
op_9	<0.2,0.8,0>	<0.2,0.8,0>	<0.2,0.8,0>	<0.2,0.8,0>	<0.2,0.8,0>	0,0016
op_10	<0.1,0.9,0>	<0.1,0.9,0>	<0.1,0.9,0>	<0.1,0.9,0>	<0.1,0.9,0>	0,0001
op_11	<0,1,0>	<0,1,0>	<0,1,0>	<0,1,0>	<0,1,0>	0

Figure 7 presents the graphical interpretation of the relation between ASL opinions (illustrative values taken from Table 3) and aggregated security level (aggregated security level has been denoted in the Table 3 as ω_{SS}).

The value of aggregated security levels has been calculated using *the Strict security assessment* method described in Section 4.3. It means that that SS = ω_{pbp} ˆω_{srv} ˆω_{srvd} ˆω_{scp} ˝E ω_{trl}.

In situation where the uncertainty level equals zero, the agregated security level decreases more quickly than the layers security. This means that the proposed method of security level aggregation preserves the fundamental requirement of security — the paradigm of the weakest link. The observed characteristic of aggregated security also corresponds with the description at the beginning of this section. The high dynamicity of this type of systems allows us to assume that assigning as the security level of the whole SOA-based system simply the minimum value of the set of layers security levels is not sufficient.

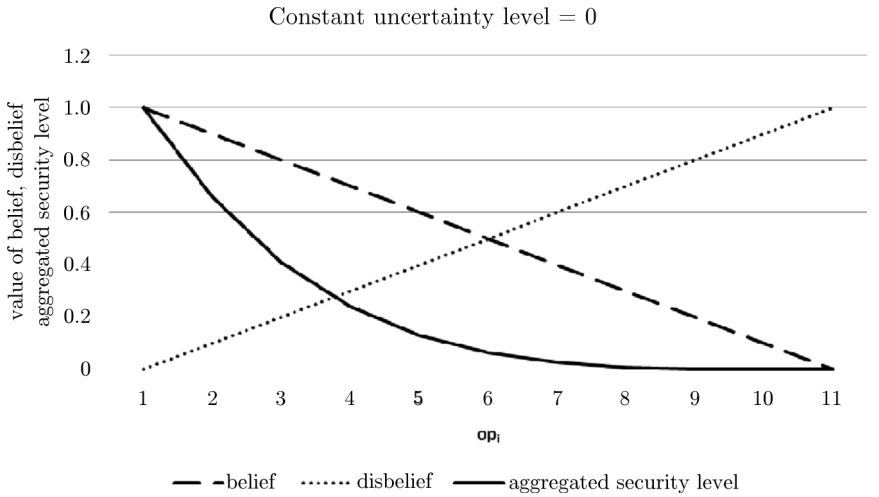

Fig. 7. Aggregated security level. Opinions about security of all SOA layers are equal and the opinions values are from the set {belief ∈ [0,1], disbelief ∈ [0,1], uncertainty = 0}

The two subsequent examples (Fig. 8 and Fig. 9) show us another important property of the proposed method for security level aggregation. The obtained results characterize the different role of the uncertainty and disbelief in the process of aggregated security level evaluation. These charts illustrate the difference between a situation where no evidence that the system is unsecure has been detected — Figure 8, and where one does not know exactly if the security of the system is guaranteed — Figure 9.

Figure 9 illustrates the situation where there is no possitive evidence of system security. In this case the aggregated security level of SOA system is significantly worse then it has been observed in the two earlier examples. This behavior also corresponds with common sense understanding of security relations.

The next three figures ilustrates the impact of the security level of a particular SOA layer for the aggregated value of SOA security level. The assumption is that all but one SOA layers are secure. It means that the opinions about secure layers are in the form of {belief = 1, disbelief = 0, uncertainty = 0}.

The first Figure 10 presents situation when one layer from SOA model has opinions with belief component equal 0 and disbelief values from 1 till 0.

It can be noticed that the presence of only one unsecure layer (it means with opinion with belief component equal 0 and disbelief component equal 1) makes the whole system unsecure (aggregated opinion has the same value — belief component equals 0 and disbelief component equal 1). In the figure below (11) another effect related to an unsecure layer presence has been demonstrated. If the unsecure layer

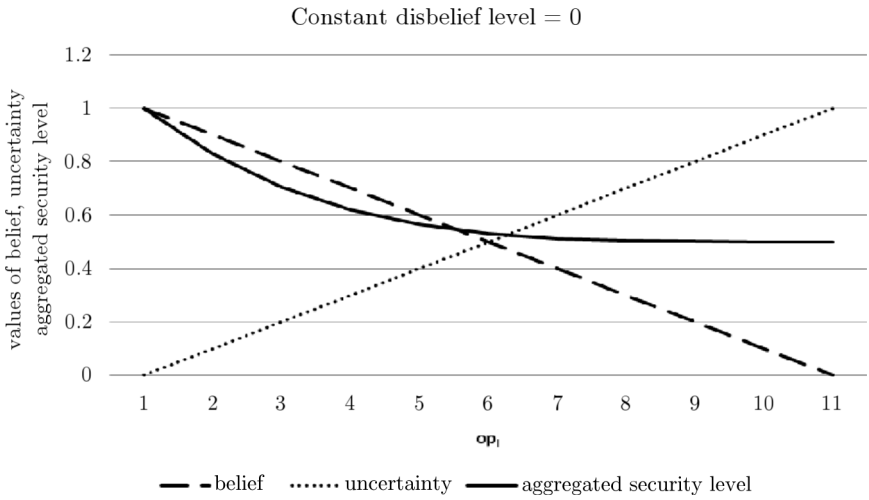

Fig. 8. Aggregated security level. Opinions about security of all SOA layers are equal and the opinions values are from the set {belief ∈ [0,1], disbelief = 0, uncertainty ∈ [0,1]}

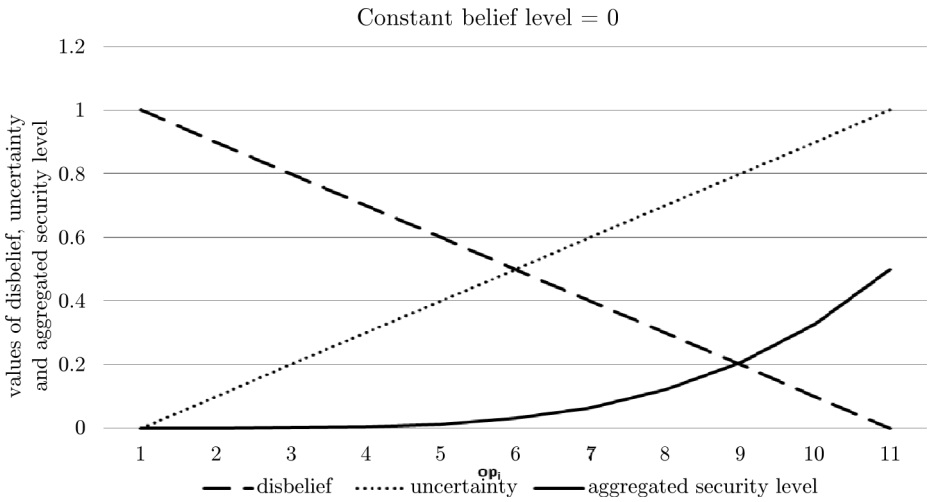

Fig. 9. Aggregated security level. Opinions about security of all SOA layers are equal and the opinions values are from the set {belief = 0, disbelief ∈ [0,1], uncertainty ∈ [0,1]}

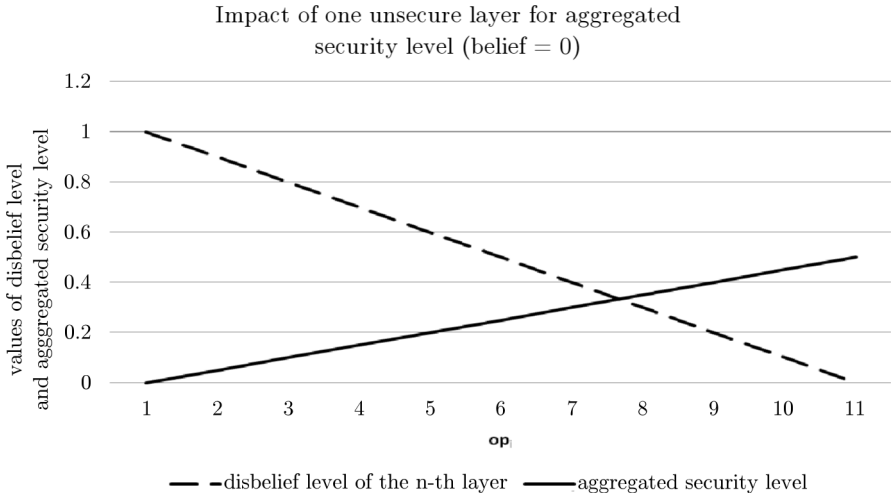

Fig. 10. Impact of one unsecure layer for aggregated security level. All but one layers are secure. The opinions about 'unsecure SOA layer' are from the set {belief = 0, disbelief ∈ [0,1], uncertainty ∈ [0,1]}

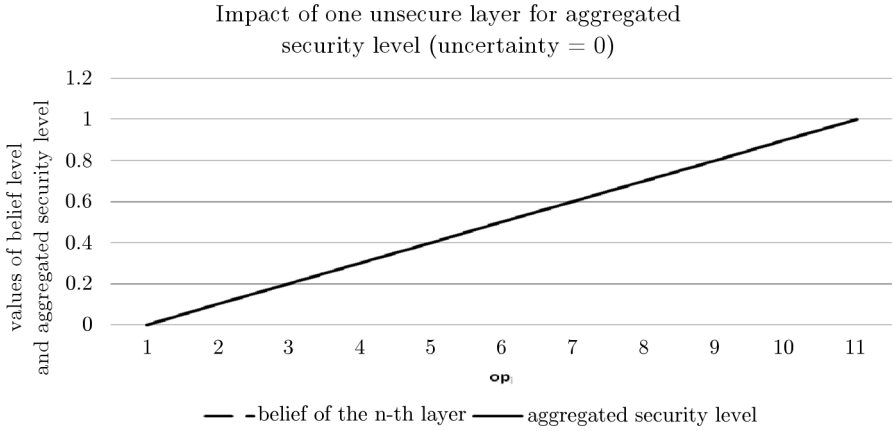

Fig. 11. Impact of one unsecure layer for aggregated security level. All but one layers are secure. The opinions about 'unsecure SOA layer' are from the set {belief ∈ [0,1], disbelief ∈ [0,1], uncertainty = 0}

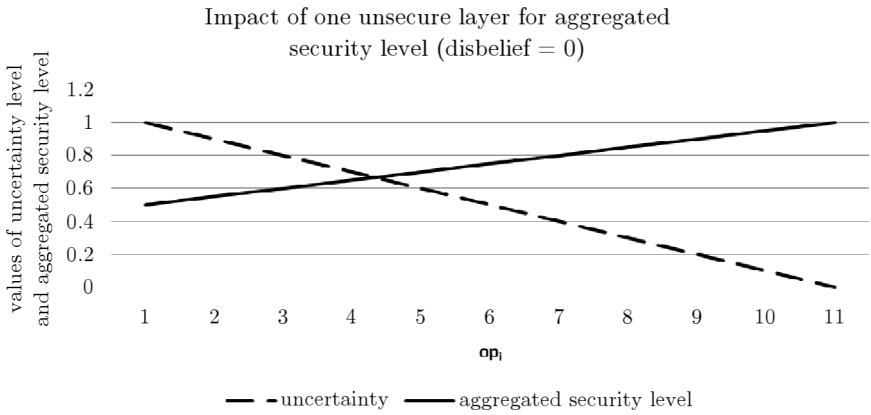

Fig. 12. Impact of one unsecure layer for aggregated security level. All but one layers are secure. The opinions about 'unsecure SOA layer' are from the set $\{\text{belief} \in [0,1], \text{disbelief} = 0, \text{uncertainty} \in [0,1]\}$

has no uncertainty component (uncertainty = 0) then the aggregated security level is this of this unsecure layer. In this case the lines describing the belief of the n-th layer and the aggregated security level cover each other. This result obtained in an absence of uncertainty corresponds with the weakest-link paradigm.

The last example (Fig. 12) presents the situation with limited trust to the security provided by some particular layer but with no clear evidence that this layer is unsecure (disbelief equals 0). The aggregated level of security is relatively high in this situation.

The presented examples, illustrating the behavior of the aggregated security level for SOA-based systems, show that the proposed method using Subjective Logic formalism is sound. It integrates the main classical security paradigms as e.g. the weakest-link property, but extends the expressiveness of security level by the incorporation of the uncertainty factor.

The analysis also has shown that aggregated values of SOA system security level corresponds with the common sense understanding of a complex systems security.

7 Validation and Analysis

As it has been stated at the beginning of this paper, to the best knowledge of the authors, the framework proposed in this paper is the first that introduces a multiagent approach to the composed services security level evaluation problem.

The other new and important outcome of the presented approach and related to the SOA security level evaluation architecture is the description of all the components, mechanisms and relations that are necessary to precisely evaluate the security level of the particular SOA system. The well-defined software agents (sec. 4.1) acting within the service oriented system performing defined during the research algorithms (sec. 4.1) allows to integrate the security assessments coming from different layers and to provide a top-level security estimate for given service. This is an unique approach, because the most of the existing standards and mechanisms have their focus on the selected technological level. The security agents are able to generate two different types of opinions about a security level of the service oriented systems. The defined procedures of evaluation Strict security assessment and General security assessment allow the agents to produce security level information which corresponds to the lowest security level measured (when the layers of the service architecture are treated as dependent from each other in the context of security assessment) or to the average security assessment of the service (when the layers of the service architecture are independent) (sec. 4.3). As the managing agents of the presented system calculate their opinions about security level using Subjective Logic the proposition how the security level of the particular SOA layer can be expressed using Subjective Logic opinions and operator has been presented in section 5.1. Subjective Logic formalism has been applied to the presented multiagent framework as the widely used method for expressing opinions and trust levels in computer security domain. The applicability of the proposed method of security evaluation has been proved in section 6 where the basic properties of security assessments in the form of opinions combined by Subjective Logic operators has been analyzed. One of the most important result of the analysis indicates that in dynamic environments of the service oriented systems assumption that the security level of the whole SOA-based system is simply the minimum value of the set of layers security levels is not sufficient. In this way the proposed approach proves the applicability and defines a new method for for SOA-based system security level evaluation.

8 Conclusions and Future Work

A novel framework using a multiagent system for composed SOA services security evaluation has been proposed in this chapter. The multiagent architecture is composed of the three types of agents: monitoring agents that tests the various security parameters related to a particular SOA layer, superior agents that manage the activity of monitoring agents, managing agents that are responsible for all superior agents and for the communication with service consumer agents. In result the framework is flexible and allows observation of all security-related issues of the

layered service architecture on different levels of abstraction. Two information processing algorithms used by monitoring agents and superior agents have been also proposed and discussed. The presented framework assumes also the evaluation of the agents themselves which may be assigned opinions expressing their trust level. Security assessment framework uses the Subjective Logic-based information fusion approach which allows to generate opinions concerning the security of both atomic and complex services – in general or within the context of an arbitrarily chosen security layer.

The important part of the future work related to the problems described in this chapter will be a series of experiments which will relate security issues with the performance of the composed services. The result of these experiments should enhance the knowledge on how strong security mechanisms impact the performance and what is the trade-off between the security the effectiveness of a service execution. In this context, the proposed framework, offering quantitative description of security issues, could be used to propose effective strategies for the optimization of service composition.

As a further research the comparison of the presented approach with the other existing methods of SOA-based system security assessment has been planed [41, 42]. There exist some similar approaches [43] but — taking into account the clear distinction between layers, the agent approach and an effective information fusion method, the presented framework is more general and flexible

Acknowledgments. The research presented in this paper has been partially supported by the European Union within the European Regional Development Fund program no. POIG.01.03.01-00-008/08.

References

1. CERT Retrieved March 20. 2009, `http://www.cert.org/`.
2. National Vulnerability Database of the National Cybersecurity Division. Department of Homeland Security, 2009, `from:http://nvd.nist.gov/`.
3. Epstein J., Matsumoto S., McGraw G. Software security and SOA. IEEE Secur. Priv. 4, 1, 80–83, 2006.
4. Fernandez E.B., Delessy N. Using patterns to understand and compare web services security products and standards. 2006.
5. Kolaczek G. Opracowanie koncepcji specyfikacji metod i modeli szacowania poziomu bezpieczeństwa systemów SOA i SOKU. Wroc. Univ. Technol., 2009.

6. Nakamura Y., Tatsubori M., Imamura T., Ono K. Model-driven security based on web services security architecture. IEEE Int. Conf. Services Computing 1, Orlando, USA, 7–15, 2005.
7. SANS Institute. 2006, http://www.sans.org/.
8. Skalka C., Wang X. Trust by verify: Authorization for web services. ACM Workshop on Secure Web Services, 47–55, 2004.
9. A reference model for Service Oriented Architecture. SOA Reference Model Technical Committee, OASIS, 2006.
10. Steel C., Nagappan R., Lai R. Core security patterns: best practices and strategies for J2EE, web services, and identity management. Pearson, Upper Saddle River, New Jersey, 2006.
11. Tari Z., Bertok P., Simic D. A dynamic label checking approach for information flow control in web services. Int. J. Web Serv. Res. 3, 1, 1–28, 2006.
12. WS-security policy 1.2. OASIS, 2009.
13. Yuan E., Tong J. Attributed based access control (ABAC) for web services. IEEE Int. Conf. on Web Services, 561–569, 2005.
14. Jøsang A. A logic for uncertain probabilities. Int. J. Uncert. 9, 3, 279–311, 2001.
15. Jøsang A. A metric for trusted systems. Proc. 21st National Security Conf., Louvain-la-Neuve, Belgium, 68–77, NSA, 1998.
16. Jøsang A. Conditional inference in subjective logic. Proc. 6th Int. Conf. on Information Fusion, Cairns, 279–311, 2003.
17. Anderson S., Grau A., Hughes C. Specification and satisfaction of SLAs in service oriented architectures. Proc. 5th Annual DIRC Res. Conf., Edinburgh, UK, 141–150, 2005.
18. Rao J., Su X. A survey of automated web service composition methods. Semantic Web Services and Web Process Composition, 1st Int. Workshop SWSWPC 2004, San Diego, California, 43–54.
19. Milanović N., Malek M. Current solutions for web service composition. IEEE Internet Comp. 8, 6, 51–59, 2004.
20. Blanco E., Cardinale Y., Vidal M.-E., Graterol J. Techniques to produce optimal web service compositions. IEEE Congr. on Services, Honolulu, USA, 553–558, 2008.
21. Frolund S., Koisten J. QML: A language for quality of service specification. 1998, http://www.hpl.hp.com/techreports/98/HPL-98-10.html.
22. Keller A., Ludwig H. The WSLA framework: specifying and monitoring service level agreements for web services. IBM Research Report, May 2002.
23. Hughes C., Hillman J. QoS Explorer: a tool for exploring QoS in composed services. Int. Conf. on Web Services ICWS'06, 797–806, 2006.
24. Jaeger M.C., Rojec-Goldmann G., Muhl G. QoS aggregation in web service compositions. Proc. 2005 IEEE Int. Conf. on e-Technology, e-Commerce and e-Service EEE'05, Hong Kong, China, 181–185, 2005.
25. Ardagna D., Pernici B. Global and local QoS constraints guarantee in web service selection. IEEE Int. Conf. on Web Services ICWS'05, Orlando, USA, 805–806, 2005.

26. Maamara Z., Narendrab N.C., Sattanathan S. Towards an ontology-based approach for specifying and securing Web services. Inf. Software Technol. 48, 441–455, 2006.
27. Karakoc E., Senkul P. Composing semantic Web services under constraints. Exp. Syst. Appl. 36, 11021–11029, 2009.
28. Klusch M., Fries B., Sycara K. OWLS-MX: A hybrid SemanticWeb service matchmaker for OWL-S services. Web Semantics: Science, Services and Agents on theWorldWideWeb 7, 121–133, 2009.
29. Gibbins N., Harris S., Shadbolt N. Agent-based semantic web services. Web Semantics: Science, Services and Agents on the World Wide Web 1, 141–154, 2004.
30. Raghuram M., Sreenath A., Munindar P. Singh, Agent-based service selection. Web Semantics: Science, Services and Agents on the World Wide Web 1, 261-279, 2004.
31. Angus F.M., Huang, Ci-Wei Lan, Stephen, Yang J.H. An optimal QoS-based Web service selection scheme. Inf. Sci. 179, 3309–3322, 2009.
32. Cardoso J., Miller J., Sheth A., Arnold J. Modeling quality-of-service for workflows and Web service processes. tech. rep., LSDIS Lab, Computer Science, University of Georgia, #02-002, 2002.
33. Charif Y., Sabouret N. An overview of semantic web services composition approaches. Electr. Notes Theor. Comp. Sci. 146, 33–41, 2006.
34. Ko J.M., Kim C.O., Kwon I.-H. Quality-of-service oriented web service composition algorithm and planning architecture. J. Syst. Software 81, 2079–2090, 2008.
35. Grzech A., Świątek P. Modeling and optimization of complex services in service-based systems. Cyber. Syst. 40, 8, 706–723, 2009.
36. Kołaczek G. Opracowanie koncepcji specyfikacji metod i modeli szacowania poziomu bezpieczeństwa systemów SOA i SOKU. Wroc. Univ. Technol., 2009.
37. Kołaczek G. Mobile agent system for trust evaluation and evolution modeling in service oriented architecture and service oriented knowledge utility environment. Intelligent decision making systems. Proc. 4th Int. ISKE Conf., 577–582, Hasselt, Belgium, 2010.
38. Juszczyszyn K., Kołaczek G. Subjective Logic-based framework for the evaluation of Web services' security. Information systems architecture and technology: service oriented distributed systems: concepts and infrastructure, 219–228, 2009.
39. Kołaczek G. Multiagent security evaluation framework for service oriented architecture systems. Lect. Notes Comp. Sci. 5711, 30–37, 2009.
40. Kołaczek G. A logical-based security level evaluation method for service oriented systems. Int. J. Inf. Technol. Secur. 1, 2, 29–42, 2009.
41. Voas J., Ghosh A., McGraw G., Charron F., Miller K. Defining an adaptive software security metric from a dynamic software failure tolerance measure. Proc. 11th Annual Conf. on Computer Assurance, Gaithersburg, USA, 1996.
42. Endrei M., et al. Patterns: service-oriented architecture and web services. Redbook, SG24-6303-00, 2004.
43. Liu Y., Traore I. Systematic security analysis for service-oriented software architectures. IEEE Int. Conf. on e-Business Engineering ICEBE'07, Hong Kong, China, 612-621, E-Business Engineering, 2007.

44. Kwiatkowski J., Pawlik M. Budowa struktur gridowych wspolpracujacych komputerow i agregacji zasobow w systemach organizacyjnych. Przegl. Elektrotech. 9, 221–225, 2009.
45. Kwiatkowski J., Pawlik M., Frea M., Konieczny D. Request distribution in hybrid processing environments. Int. Conf. Parallel Processing and Applied Mathematics, Wrocław, Poland, 185–197, 2009.
46. Grzech A., Rygielski P., Świątek P. Qo-S-aware infrastructure resources allocation in system based on service-oriented architecture paradigm. Proc. 6th Working Int. Conf. Performance Modelling and Evaluation of Heterogeneous Networks, Gliwice, 35–48, 2010.

SOA-Based Support for Dynamic Creation and Monitoring of Virtual Organization

Kornel Skałkowski, Jakub Sendor, Mikołaj Pastuszko, Bartłomiej Puzoń, Jakub Fibinger, Dariusz Król, Włodzimierz Funika, Bartosz Kryza, Renata Słota, and Jacek Kitowski

Department of Computer Science
AGH University of Science and Technology,
Mickiewicza 30, 30-059, Kraków, Poland
bkryza@agh.edu.pl

Abstract. Modern IT infrastructures provide technological means for supporting collaborations between various organizations through several virtualization solutions. These can include simple communication software, collaboration by means of shared data access or advanced teleconferencing. However due to the high level of heterogeneity of IT infrastructures between various organizations the problem of setting up such distributed collaborations, ensuring proper privacy and security as well as enforcing that the collaboration adheres to a set of proper rules is a key issue. This is where the concept of Virtual Organization comes in. In this chapter we present the Framework for Intelligent Virtual Organization (FiVO), which provides several features supporting the process of creating and managing a Virtual Organization. These include the semi-automatic adaptation of a legacy information source to a unified semantic description, collaborative contract negotiation, automatic Virtual Organization deployment, semantic based Virtual Organization monitoring, and contract enforcement.

Keywords: Virtual Organization, SOA, contract negotiation, SLA monitoring, semantic, security

1 Introduction

For several years, the problem of supporting collaboration between people and resources belonging to separate physical organizations or communities has been a goal approached in several ways. On the one hand, virtual organization as a business concept emerged along with the globalization due to the necessity of managing international distributed enterprises, either through branching or outsourcing. On

the other hand, virtual organizations have been used in scientific communities, especially those related to Grid technology, providing a framework for building international collaborative scientific teams working on complex problems. In both cases however, the common aspect of putting the VO concept into practice is the enabling IT technology.

Nowadays, the practices of sharing both hardware and software resources between companies and institutions are often based on the idea of Virtual Organizations. A virtual Organization can be understood as a partnership of entities (e.g. physical organizations), created in order to solve a particular task or to approach an emerging market opportunity, which has a temporary nature and is ruled by a properly defined agreement. In our approach the agreement is a formal contract, defined in an ontological language, using the semantic description of organizations resources, people, and knowledge that will be part of the new Virtual Organization [1]. Another important concept related to the VO is that of Virtual Organization Breeding Environment (VBE, sometimes referred to as VOBE). A VBE groups organizations, which are willing to collaborate together with other organizations and have proper pre-arrangements, which enable them to create and participate in Virtual Organizations. These include resources, knowledge and people as well as an appropriate IT infrastructure, which allows them to integrate their business processes with other organizations during the VO creation and execution.

The problem is that most modern VO tools and frameworks provide rather non-flexible means for defining, deploying, managing and controlling selected aspects of the Virtual Organization, which means that through its lifetime several often incompatible tools must be used. In order to foster, however, the adoption of the VO concept, it is necessary to provide the VO participants with unified and easy-to-use tools which allow for dynamic creation and control over the VO. These include such aspects as identification of the VO goal, searching for partners, negotiating the VO agreement, deploying the VO within the IT infrastructure of the participants, controlling the VO execution through proper monitoring, enabling the evolution of the VO in response to changing environment and eventually dissolution of the VO. A Virtual Organization comprises a set of individuals and resources of the participating organizations interconnected for collaborative problem-solving, business, research or any other purposes, where the cooperation of several parties needs to be established and supported in terms of IT infrastructures.

Another problem stems from the fact that the concept of Virtual Organization has been involving in parallel within two communities, i.e. Grid environments supporting scientific research, as well as business oriented ones, mainly through the Collaborative Networked Organizations Community (CNO). Although the essential problem was the same, the two tracks of research have stressed its different aspects. The Grid community focused mostly on the practical issues related to security in a VO and monitoring of the VO resources, while the CNO community concentrated on the problem of identifying the goals and the business models for the Virtual Organizations, and how to support the inception and management of

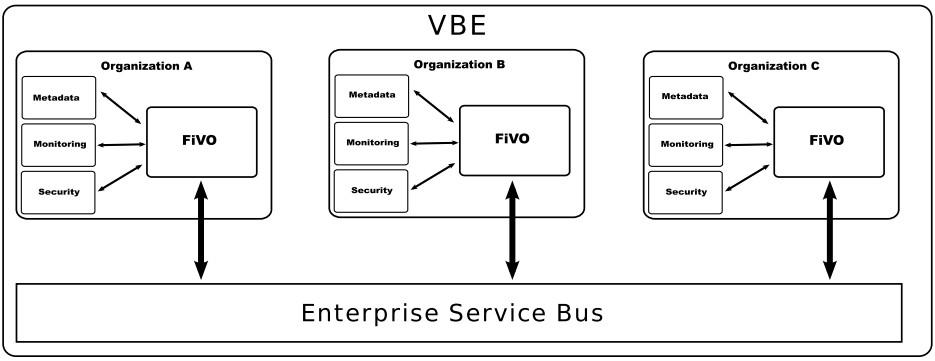

Fig. 1. FiVO enabled organizations within a VBE

a VO through the concept of virtual Organization Breeding Environment (VBE). Another main difference between the two approaches is that the Virtual Organizations on the Grid are based on a common infrastructure where users are integrated into communities of practice, while in CNO scenarios, different organization provide their own service and infrastructure to collaborate on a common goal.

The current version of the FiVO framework supports several of the above VO related issues, i.e. contract negotiation, VO deployment and monitoring of VO execution with respect to the negotiated contract statements. Other issues including partner identification and VBE management are discussed in Chapter 13. The negotiation phase is a distributed collaborative process between the potential VO participants, supported by a special GUI enabling the negotiators to edit the contract statements both directly using ontological concepts as well as in a natural language. VO deployment tools take the negotiated contract and translate it into proper configuration entries in the IT infrastructure components of the organizations participating in the VO. FiVO supports the configuration of the security and monitoring infrastructures. VO monitoring tools verify in real time whether the negotiated contract, especially with respect to QoS statements, is being fulfilled by VO participants. In the case where the contract is not respected, some predefined action can be triggered to correct it, depending on the statements of the contract. For instance the Virtual Organization can be stopped or additional service can be executed.

The FiVO framework, along with all its components should be deployed in all organizations participating in a particular Virtual Organization Breeding Environment as presented in Figure 1.

The organizations first register with some Virtual Organization Breeding Environment which gathers the organizations e.g. for sharing some common interests. The VBE can be a simple catalogue of organizations or some more advanced infrastructure supporting the organization communication and cooperation, for instance

based on the Enterprise Service Bus. The bottom line is that the organizations belonging to the VBE can discover each other, to choose partners for negotiating on a new VO.

The FiVO (Framework for intelligent Virtual Organizations) [2] framework that supports creation and management of Virtual Organizations, is presented in this chapter along with a semantic monitoring tool called SemMon [3]. The initial version of the FiVO framework has been previously applied to Grid based business scenarios including banking and rich media sectors [4].

The rest of the chapter is organized as follows. Section 2 contains an overview of existing related work and progress beyond concerning the topics addressed by the FiVO framework. Section 3 provides a general overview of the FiVO vision. Section 4 explains the issues related to contract negotiation in Virtual Organizations. Section 5 describes the components responsible for VO deployment based on a negotiated contract. Section 6 presents the semantic monitoring approach using the SemMon monitoring tool. Finally, Section 7 presents some plans for the future.

2 Progress Beyond the State of the Art

This section presents the progress beyond the state of the art in the areas related to the FiVO framework, i.e. Virtual Organization management, NLP support for contract negotiation, contract negotiation, security, and monitoring.

2.1 Virtual Organizations

The concept Virtual Organization is discussed in business and management literature for some years now [5]. However one of the first IT propositions of putting this concept into practice came with the vision of the Grid [6] and the authors of the Globus framework [7]. Although no single definition of VO exists, some general characteristics common to all of these definitions have been identified, including Dematerialization, Delocalization, Asynchronization, Integrative atomization, Temporalization, NonInstitutionalization and Individualization [8].

Significant work has been done on VOs not only in the Grid community, but also in the agent-oriented community. The latter has focused on algorithms for dynamic environments so that the VO and its participants can achieve their objectives. Another issue is related to VO constituents. These can include both tangible and intangible resources. The VO should be understood as a longer lasting and more complex collaboration of enterprises [9]. In this respect, it is important to remember that in order to have a chance to be established VOs need an environment usually called Virtual Breeding Environment (VBE) [10]. The authors

of [11] have reviewed several existing methodologies and projects. The paper [12] proposes a VO reference model, which was extrapolated from over 100 research papers on the subject. Its authors introduced three types of VO topologies that are the most common in practice: supply-chain, star and peer-to-peer, and claim that all of the analyzed projects could be categorized to one of these models.

The analysis done in [13] presents requirements and propose an architecture for a VBE management system for dynamic Virtual Organizations. The authors of [14] describe requirements for the automation of contract management in a VO. They identify three kinds of contracts in VO: business contract, ICT contract and ASP (Application Service Provider) contract. Also [15] tries to formalize a definition of contract based multi-agent Virtual Organization. Its authors define four key properties of VOs: Autonomy, Heterogeneity, Dynamism and Structure. Further, in [16] authors present a web-Pilarcos J2EE based agent framework for managing contract based Virtual Organizations.

Our work leverages the existing solutions by providing a unified framework supporting all the aspects of VO creation and execution. It provides a set of distributed components integrated through the Enterprise Service Bus technology for VO contract negotiation, deployment, and control of execution.

2.2 Natural Language Processing based Contract Analysis

The most important part of a contract from the IT infrastructure perspective are the Service Level Agreement (SLA) and the security assertions, as they enable VO technical administrators to configure middleware components so that they meet the negotiated contract requirements. However, the configuration process, often tedious and error prone, could be performed by a software application instead of human, which goal is targeted by FiVO. In order to represent a contract content, FiVO proposes the use of OWL ontologies. Most contracts are, however, written in a natural language. The most significant application that uses information extraction techniques for ontology learning is Text2Onto [17]. What makes it inapplicable in this case is that it focuses on learning entirely new ontologies rather than creating individuals based on the already existing ontologies. Moreover, Text2Onto uses the Probabilistic Ontology Model for knowledge representation purposes while the contract statements representation requires very high level of certainty, correctness, and exactness. Other two useful approaches are presented in [18] but neither of them covers the ontology as an output requirement. The authors of [19] present an approach to mapping natural language corpora to existing ontologies, and thus inferring the meaning of the sentences. This raises a need for a system that would process the contracts, analyze and translate them into the contract ontology. Some research on the identification of success in natural language based negotiations has been proposed in [20].

The concept of negotiation process in natural language is already implemented in some prototypes. For instance the Inspire [21] web-based negotiation system allows participants to negotiate simple statements like prices or deadlines for services. The natural text used during the negotiations is not, however, interpreted automatically and the system is limited to statements based on numerical values. A more advanced system is Negoisst [22], which uses the OWL language to annotate the textual statements to provide some understanding of the statements created by users in natural language. None of these solutions is however applicable to the case when the negotiated contract must be used to automatically deploy a Virtual Organization once the negotiations are complete, which can be only achieved when the full meaning of contract statements provided by the negotiating parties can be automatically translated to a formal model (ontology) and then processed by the appropriate components during the VO deployment.

2.3 SLA Monitoring

As mentioned in the previous subsection, the rules of cooperation between companies participating in a Virtual Organization are described by a contract. Some of them constitute the SLA requirements, which concern the *particular* companies. The SLA requirements could be defined on different levels of abstraction. These levels usually depend on the domain of a Virtual Organization. For instance the contracts among the network providers include low-level SLA requirements like mean network throughput, jitter, mean latency, etc., whilst the contracts among institutions like hospitals include high-level SLA requirements like the number of visits per month or the number of interventions carried out, etc. The subject of SLA monitoring and enforcement have been discussed to a large extent in the literature [23, 24, 25]. However, since the whole Virtual Organization should provide complex and sophisticated services for the end client, the fulfillment of these SLA requirements constitutes an essential part of the process of VO creation and functioning. This causes a need of continuous checking if each physical organization fulfills the SLA statements written in the contract. This is the main reason, why SLA monitoring constitutes a part of a VO management framework. The process of designing and implementing this subsystem is a challenge, because it has to cope with the following requirements:

— heterogeneous environment — the SLA Monitoring subsystem has to collaborate with different low-level and medium-level monitoring systems;
— lightweight deployment process — a part of this subsystem has to be deployed in each organization participating in a VO;
— distributed architecture — the subsystem has to join physically distributed organizations and companies;

— high scalability and robustness — this requirement is obvious since the subsystem constitutes the essential part of the virtual organizations management framework.

Our framework which provides several distinguishing features such as the use of generic ontologies as well as domain specific ones, allows for a unified description of various kinds of resources and requirements, the integration with legacy low level monitoring systems, and providing VO specific high level messages to the higher layers of the VO management framework.

2.4 Semantic Monitoring

One of the main issues in the monitoring of the fulfillment of an SLA contract by the organizations forming a VO is a high level of abstraction used for the creation of such contracts. Often, a contract contains a reference to the domain for which the VO is formed. Therefore, the monitoring process of the contract should be bound to a specific domain. Unfortunately, most of the existing monitoring systems are focused only on one particular area, which prevents them from being suitable to monitor a generic VO. There are also systems that try to be orthogonal to the monitored area, but they operate at lower levels of abstraction than typical SLA contracts, and cannot provide valuable information about contract fulfillment.

Therefore, a new approach is proposed which can be parameterized to monitor different domains. On the other hand, the monitoring process will still be performed on high level abstraction to avoid unnecessary data generation from the end-user point of view. The proposed solution is a system of attribution of measurement data, called SemMon2, configurable through a semantic description of the area (in form of a domain specific ontology), which is the subject of interest. It is not a monitoring system in the common meaning since it is oriented on exploiting other monitoring systems to obtain measurement data, which can be additionally transformed in order to obtain information at an appropriate level of abstraction. For this reason, it can be called a high level monitoring system in contrast to the systems of lower level, which can be any monitoring systems that provide their own measurement data to external customers.

2.5 Security

What poses one of the greatest difficulties when a new virtual organization deployment is considered is the heterogeneity of the underlying systems, specifically the heterogeneity of their authentication and authorization layers [26, 27, 28]. Several

established authentication and authorization frameworks are common and, especially in scientific institutions, various in-house security solutions are in use. Even when an established standard is used through a set of peers, serious inconsistencies may arise caused by framework version mismatch, framework misuse or heavy usage of homemade patches to the system. In order to enable various institutions to share their services, that is to provide access to their resources for users, some complex and precise steps are often to be taken. When working on enabling the security systems to communicate, within the development of our framework, we needed to address the following issues:

— the complexity of the task increases significantly with the growth of the number of participants;
— each change in security configuration can cause a hypothetical threat for the integrity of the organization's security systems.

Since the majority of the tasks are complex and require high awareness and analytical capabilities, we propose to automate the process of configuring authentication/authorization systems of a virtual organization to a reasonable extent. In our FiVO Deployment system we provide security administrators with a tool that will assist them in configuring the security layer for a VO while not forcing them to expose the finegrained details of the underlying security provider. Moreover, we strive not to impose a single policy language and interpreter (contrary to the work described in [29]), as this would be an extremely hard task in a heterogeneous system.

3 Overview of FiVO Architecture

The FiVO framework overall architecture is presented in Figure 2.

The FiVO framework is supported by the GOM (Grid Organization Memory) knowledge base [32] that stores all the necessary information about the organization and the VOs, which it belongs to in a unified semantic form. In order to allow the organizations to transform the information about the resources and capabilities they possess, an X2R [33] plug-in has been adapted to GOM, which allows the automation of the translation of information stored in LDAP, RDBMS or XML databases to an ontological form. The information stored in GOM can later be used during the negotiation process to present the relevant resources of each organization to the negotiators, which can be then directly used to create contract statements. Once the contract is negotiated, it is stored in GOM and can later be used by VO Deployment components to automatically create the Virtual Organization, i.e. to configure a needed middleware infrastructure within the organizations. The Negotiation Service ensures that the contract statements exchanged

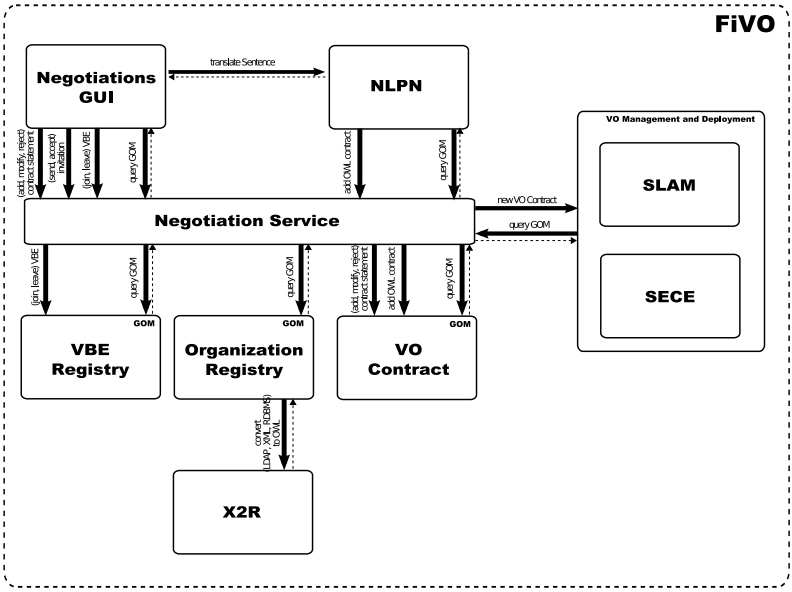

Fig. 2. The overall FiVO architecture

between partners are sent to the proper FiVO instances in other organizations, as well as intermediates in the communication between FiVO components and the GOM knowledge base.

4 Contract Negotiation for Virtual Organizations

4.1 Contract Ontology Design

A VO contract is defined in the Web Ontology Language (OWL), using the concepts defined within the Contract Ontology. The Contract Ontology is defined as a set of concepts representing possible contract statements. Each type of statements allows for the representation of different contract elements. An overview of the Contract Ontology and its dependencies are presented in Figure 3.

The core ontologies provided with FiVO framework are divided into 4 separate ontologies:

— **VO Ontology** — contains the concepts related to the structure of the VO as well as the roles and administrative issues of the VO;

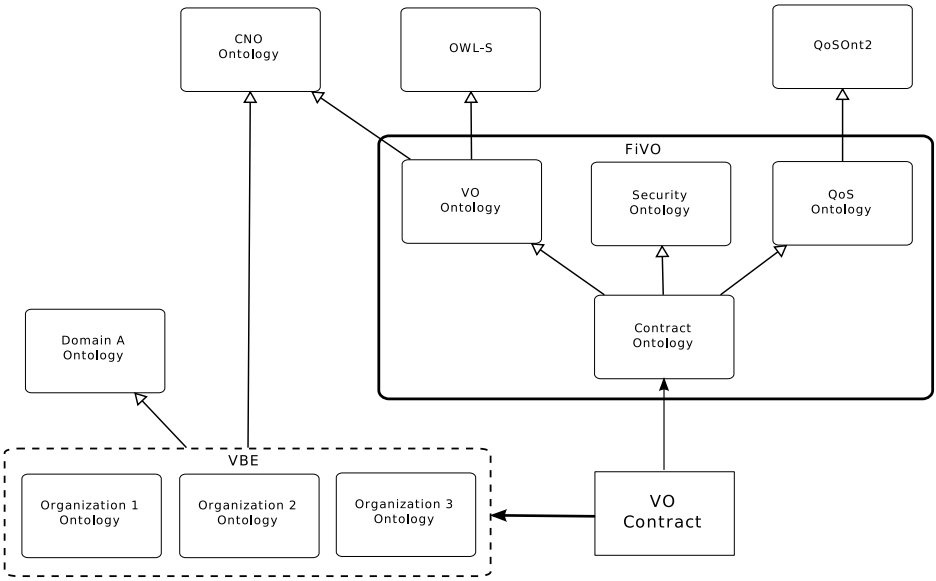

Fig. 3. Contract Ontology and its dependencies

— **Security Ontology** — allows to define the security policies and security requirements for the services and resources within the VO;
— **QoS Ontology** — this ontology delivers the concepts allowing for the definition of the QoS requirements for services in the VO;
— **Contract Ontology** — this ontology provides the concepts, which allow for an actual definition of the contract, i.e. the types of possible contract statements, which later can be automatically processed to deploy the VO.

The main dependencies of this ontology are:

— **CNO Ontology** — an ontology developed within the framework of the ECOLEAD project [34], which provides concepts allowing to define a structure of the VO, the resources belonging to particular organizations and their role in the VO as well as the capabilities and the competencies of the organizations;
— **OWL-S** — is an ontology whose purpose is to allow for semantic description of Web Services declared using the WSDL language;
— **QoSOnt2** — this ontology provides a comprehensive set of concepts for defining the QoS requirements for the resources such as the maximum time of response or average throughput.

Based on the concepts provided by these ontologies, as well as any domain specific ontologies that are available within a particular VBE, the VO Contract

can be defined by means of instantiating these concepts into ontology individuals, as presented in the example below:

```
<!- Example of role definition statement ->
<VOOntology:Role rdf:about="#VOTours1RegisteredCustomer">
 <ContractOntology:hasSuperRole
               rdf:resource="#VOTours1Customer"/>
 <ContractOntology:hasRoleID
               rdf:resource="#VOTours1RegisteredCustomer"/>
</VOOntology:Role>

<!- Example of resource provision statement ->
<ContractOntology:ResourceProvision

rdf:about="#VOTours1ResourceProvisionStatement0003">
 <ContractOntology:provider
rdf:resource="#ChataquaAirlines"/>
 <ContractOntology:resource
               rdf:resource="#ChataquaTicketBookingService"/>
 <ContractOntology:consumer rdf:resource="#VOTours1"/>
</ContractOntology:ResourceProvision>
```

The current version of the Contract Ontology contains a set of concepts representing the possible types of statements that can be understood by the FiVO framework, as presented in Figure 4.

As can be seen from Figure 4, the contract can contain the statements relating to the structure of the VO, its time span as well as SLA and security.

4.2 GOM

GOM (Grid Organizational Memory) is a distributed knowledge base component of the FiVO framework, developed previously within the framework of the EU IST K-Wf Grid project [35] and extended to support knowledge management for Virtual Organizations.

GOM provides means for storing ontologies and ontology individuals in a scalable fashion, allowing to use multiple available storage and reasoning mechanisms. Additionally, GOM provides several interfaces for querying and modifying the managed ontologies, allowing the developers of components using GOM to use it both as an information source as well as a registry, which can be updated when needed. The communication protocols supported by GOM currently include RMI and SOAP.

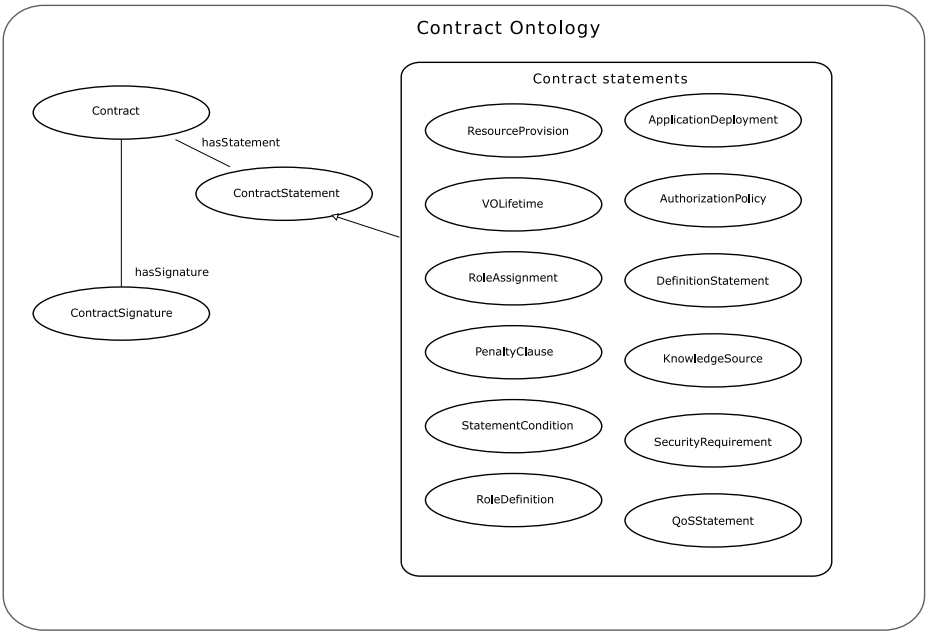

Fig. 4. The possible types of statements in the Contract Ontology

In order to use GOM as a knowledge base for FiVO, GOM has been extended by an X2R tool [33], which allows to automate the process of translating the existing information that describes organization's resources into the ontological form. Currently, GOM and X2R support the translation of information stored in relational databases, LDAP catalogues or XML repositories.

4.3 Contract Negotiation Process

The process of contract negotiation is presented in Figure 5.

During the negotiation phase two roles are distinguished for the partners participating in the negotiations:

— **VO Administrator** — the lead partner who defines the VO goal, invites partners, starts the negotiation process and concludes the negotiations by deploying the new VO;
— **VO Participant** — a regular partner invited by the VO administrator who can create and modify contract statements, as well as accept or reject statements regarding their organization.

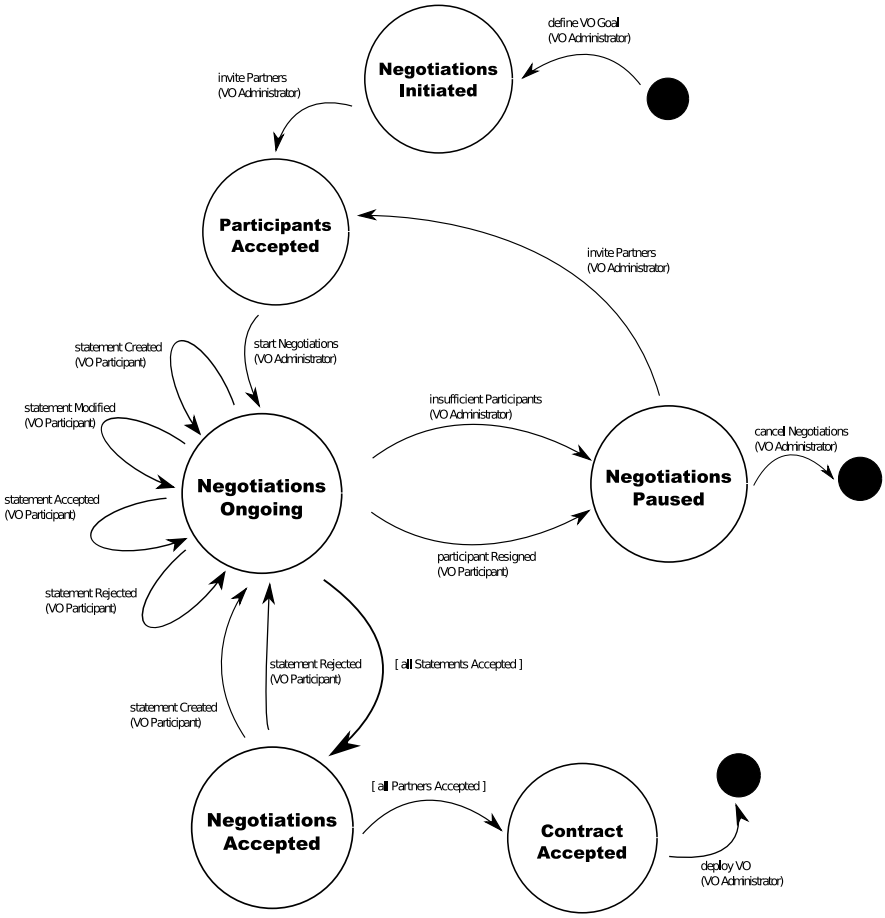

Fig. 5. Contract negotiation process states

The negotiation process is initiated by the VO Administrator who defines the overall VO goal. This definition has the form of an initial set of contract statements (e.g. "*Organization A will provide Service X*"). Then the administrator sends the invitations to the organizations within the VOBE in which the new VO should be created. When all partners accept the invitation, the VO Administrator starts the proper negotiation process. During this phase each partner can create and modify the statements present in the contract. Statements can be grouped into negotiation tables, which allow subsets of participants to negotiate parts of contract which do not need close attention from other participants. Next, each statement has to be accepted or rejected by all partners negotiating a particular statement or group of

statements. In the case where all statements in the contract are accepted, the negotiation state automatically switches to the accepted state. Only in this state the VO Administrator can ask each partner to accept the contract as a whole. If any of the partners has some additional issues, it is possible to create or modify any of the existing statements thus bringing the negotiation state back to "ongoing". During the negotiation phase two exceptions are possible. Some of the partners can resign the negotiation process or the VO Administrator can decide that the participants are not able to fulfill the predefined VO goal. In both cases the VO Administrator can either cancel the negotiation process or invite additional participants to the negotiation process. In the case a partner resignes, all the statements relating to his/her organization are invalidated, and when new partners join the negotiation process these statements will have to be negotiated from scratch.

4.4 Contract Negotiation Framework

In order to allow FiVO users to negotiate contracts for Virtual Organizations, a convenient front-end interface has been created. Its goal is to enable the negotiation process in real-time and to make its results immediately visible to the negotiators so that they have a consistent and up-to-date view of the current form of the contract. This assumption raises such requirements for a user interface as usability, extensibility, and portability (various platforms should be supported).

A technology, which has been selected to meet all those requirements, was Eclipse Rich Client Platform (Eclipse RCP)[36] as it provides components intended for rapid development of extensible applications that exploit the OSGi Framework and SWT graphic library [37]. As a result, it became a basis for the implementation of the FiVO graphical user interface.

The application consists of four perspectives:

— Knowledge Databases (Fig. 6);
— VBE Browser (Fig. 7);
— Organization Definition (Fig. 8);
— Negotiations (Fig. 9).

Each of them is intended for a different use case.

The Knowledge Databases perspective (Fig. 6) is aimed to connect the application to servers that store the semantic knowledge, which describes an environment where negotiations take place, as well as the services that distribute the negotiation process between the organizations within the VBE.

The VBE Browser perspective (Fig. 7) presents several views that group the semantic knowledge according to Virtual Organization Breeding Environments, Organizations or Virtual Organizations. Moreover, it is possible to alter properties and membership of the specific VBE. Each entity (e.g. organization's resources)

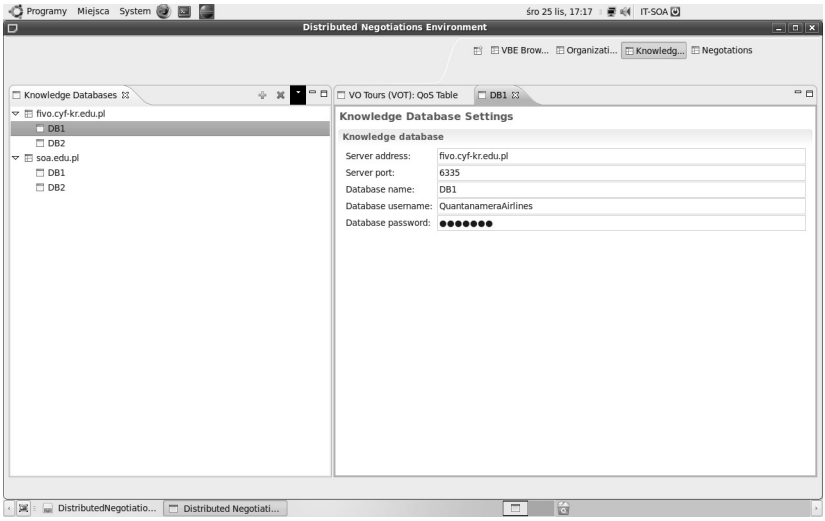

Fig. 6. Knowledge Database Perspective

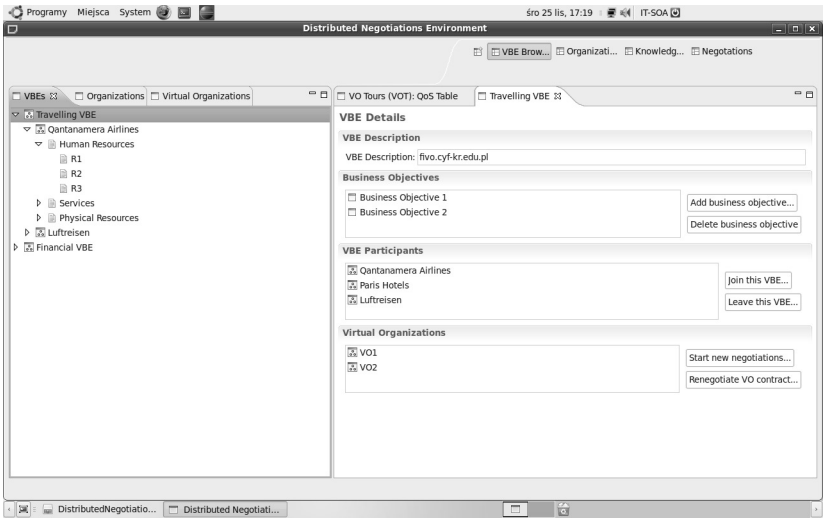

Fig. 7. VBE Browser Perspective

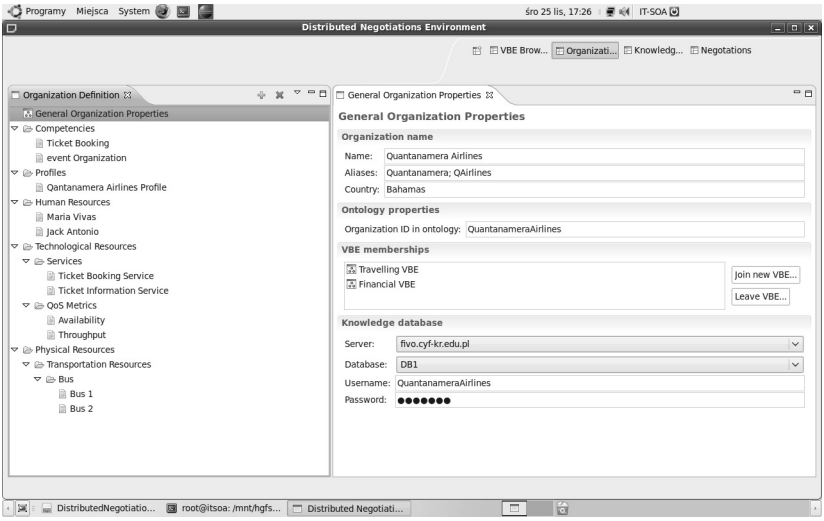

Fig. 8. Organization Definition Perspective

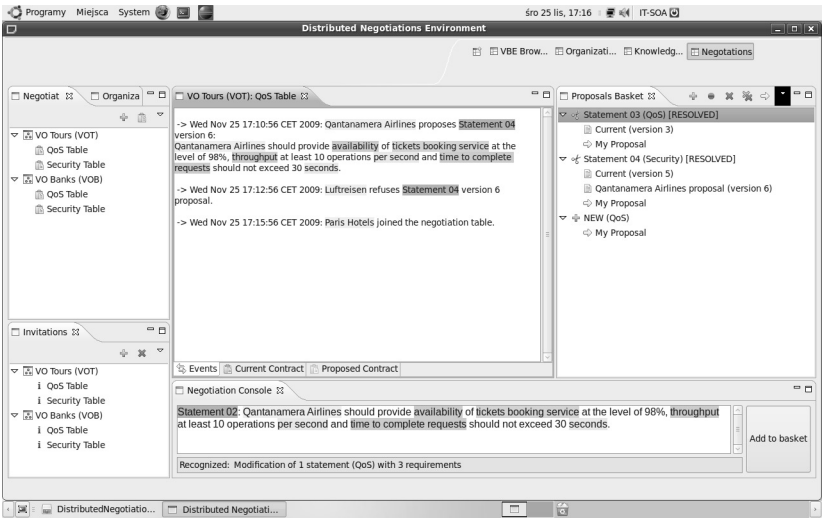

Fig. 9. Negotiation Perspective

may be displayed enabling the user to easily design a contract statement or its entire form.

The Organization Definition perspective (Fig. 8) is used by the user who wants to describe resources of his or her organization so that the other negotiation participants know what may act as a subject of contract statements. This perspective should be used before the negotiation process and every time when the user's organization resources change.

The Negotiations perspective (Fig. 9) is the most important one and actually the most interesting one. This part of the user interface is compatible with our previous negotiation model [1]. In this model the negotiations are performed by means of performing actions on contract statements such as creation, modification, approval and rejection. The negotiations are divided into negotiation tables, which allow for the distribution of the overall negotiation process into negotiations on some subsets of the whole contract. The negotiation tables can also be accepted or rejected as a whole. The perspective consists of several views:

— Negotiation Tables — the list of the negotiation tables concerning the Virtual Organization whose contracts are currently being negotiated;
— Invitations — the list of invitations to negotiation tables that may be accepted or refused by user;
— Proposals Basket — the list of the statements that have been altered or created by a user in the current session but not yet published (a detailed description is presented below);
— Negotiations Console — a text editor to compose statements using a natural language (detailed description is presented below);
— Negotiation Table outlook — presents the events that occur during the negotiation process as well as the current and proposed form of the contract (as a text in a natural language or in OWL/XML).

Two of the above views need to be described more precisely. The first one is the Proposals Basket. A user who takes part in the negotiation process may want to change a contract statement, remove or create a new one. This can be done either using a wizard that leads through several simple steps or expressing it with the natural language. Irrespective of the method, the proposal (of a change, removal or creation) is put in the basket. This means that it is not visible yet to other negotiators. In order to achieve this the basket has to be published. The second view is the Negotiation Console. Basically it is a text editor enriched with several useful helper tools that make it a powerful negotiation tool. In general, it enables the user to compose a contract statement in natural language. It is possible to refer to an already existing statement so that the user does not need to write it completely but can edit only its parts (e.g. metric values in case of QoS statements). Since the natural language sentence is translated into an ontological form before submitting it to other participants, the system will support multi-language negotiations.

Currently, only the English language is supported. More, a keyboard shortcut may be used at any time in order to display a context-based list of suggestions (e.g. a list of services that belong to the organization that the statement concerns). Each entity that has been recognized as a semantic subject (an instance defined in the ontology that describes some organization) is highlighted to improve readability as well as to indicate correct spelling and successful recognition. Below the text area, there is a status field, where the user is informed about the understanding of the written text by the NLP module. Finally, an understood statement may be added to the Proposals Basket and then published to other negotiators.

5 Automatic VO Deployment

5.1 Configuration of Security Infrastructure

When dealing with different security systems and protocols in different organizations, the main issue is to design an architecture, which is able to bring them all together. The solution is to build a system that fulfills the following requirements:

— based on few core components which can be common for all organizations;
— loosely coupled with other system elements that can be changed dynamically;
— enabling adding support for new security features (not mentioned in a core version).

As a result, an architecture based on the plug-in concept has been developed. It is depicted in Figure 10.

The main flow looks as follows:

— as the input, a FiVO contract is provided. It contains, in particular, information about organizations, their security systems, and the virtual organization being deployed;
— the OWL Parser is a component that reads in a contract written in the OWL language using the Jena library;
— the Builder creates a model which reasembles the situation described in the contract;
— it is possible to add support for many security systems, since the architecture is designed to allow contributors to attach plug-ins for the Analyzer module. They should implement the provided interface to support the analysis of custom fragments of a contract file. All attached analyzer plug-ins are served by Analyzer's registry;
— based on a given VO model, the Configuration Generator decides what to do to maintain the expected VO security configuration;

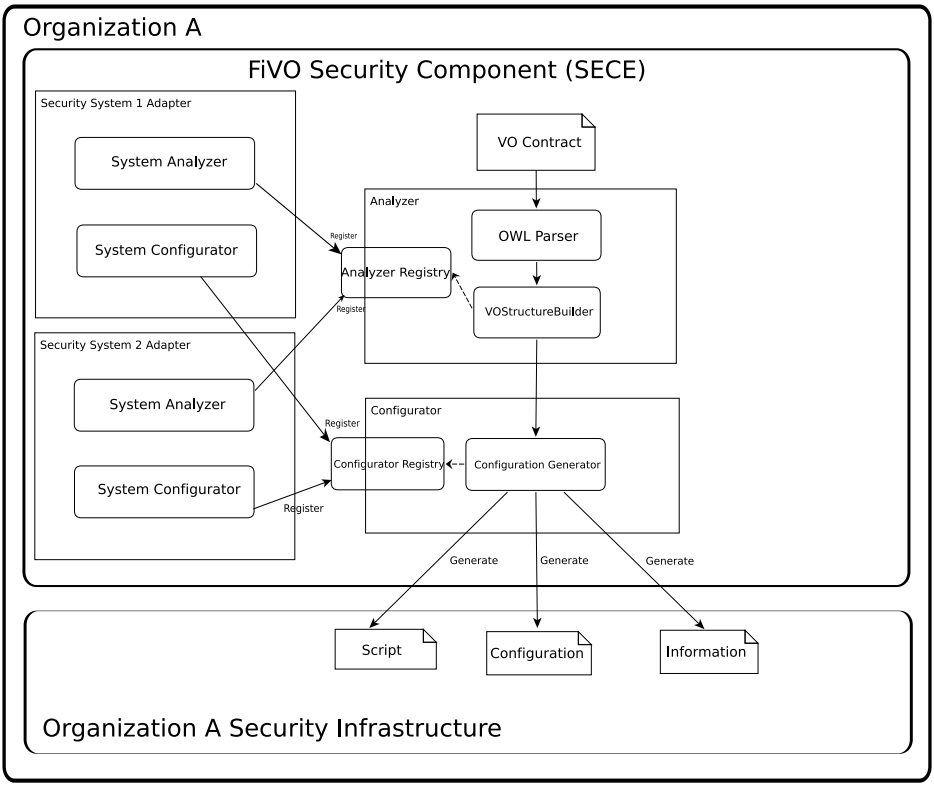

Fig. 10. FiVO security deployment system (SECE) architecture

— as above, the Configuration Generator may be extended by a set of plug-ins working as custom Configuration Generator fragments. They are served by the Configuration registry.

The security part of the contract format is designed to fulfil all the requirements: to define authentication/authorization issues and information about the existing and anticipated security systems. A simple diagram of these fragments of ontologies is presented in Figure 11. It defines the required authentication/authorization issues of most security systems.

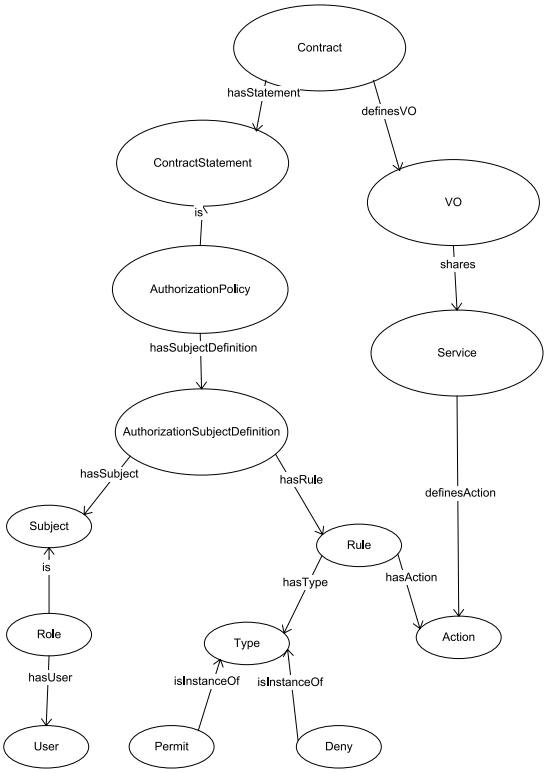

Fig. 11. SECE ontology schema

5.2 Monitoring the Contract Fulfillment

This phase of the virtual organization's life cycle follows the contract negotiation and preparing phase. As mentioned in Section 2.2, the contract contains two kinds of statements:

— SLA requirements and configuration statements of the monitoring infrastructure;
— configuration statements of the security infrastructure.

The security statements processing is described in section 5.1. This section explains how the monitoring infrastructure configuration statements are processed and how the SLA statements are monitored.

In Section 2.3 we mentioned that the SLA requirements could be defined at different levels of abstraction. Example SLA statements may look like the following:

— "The Web Service which belongs to company A should have a mean response time below 2 milliseconds."
— "Company B should provide a minimum of six tons of bricks per month."
— "Doctors from hospital C should participate in minimum 40 telemedical consultations per month."
— "Computing centre D should always provide 4 TB of free storage space."
— "Hotel E should provide six free class A rooms per day."
— "Company F has to process a maximum of 200 tons of concrete per month."

As we can notice the above SLA requirements belong to various domains, thus the ways of monitoring them differ. The SLA Monitoring subsystem has to cope with this heterogeneous environment. Of course, companies can be globally distributed what constitutes another challenge for the system design. Since the SLA Monitoring subsystem is a crucial part of the whole virtual organizations management framework, it has to work continuously, to be fault-tolerant, and, finally, to be scalable.

In order to cope with the above features, the following architecture assumptions were imposed:

— the system has to consist of loosely-coupled, independent services;
— the failure of one of these services should not affect the whole monitoring process;
— the services should be able to work on different physical machines.

The above architecture requirements can be fulfilled only by a very careful design and a very careful choice of the technologies used to implement the system. When working on the system architecture, we extracted six different services. An overview of the system architecture is presented in Figure 12.

At bottom, we see three different physical organizations. Each organization has its own specific monitoring system. The SLA Monitoring subsystem has six core components and, additionally, provides specific adapters which are dependent on the low level monitoring systems. The first component — Ontology Analyzer — translates the ontological form of the contract to the appropriate instances of Java classes. These instances are used by other components in further processing. The next components — Metrics Aggregation Service and Metrics Monitoring Service — are responsible for the main process of SLA fulfillment monitoring. Metrics Monitoring Service periodically sends requests about metric values, and compares the received values with the expected ones. If they are too low or too high, a proper notification is sent to the Reaction Service. The Metrics Aggregation Service is an additional service which aggregates and caches the results received from the medium-level monitoring system if this system does not do it itself. The Reaction Service is responsible for invoking some preventive actions if some services break the contract statements — these actions can be for example exposed as the

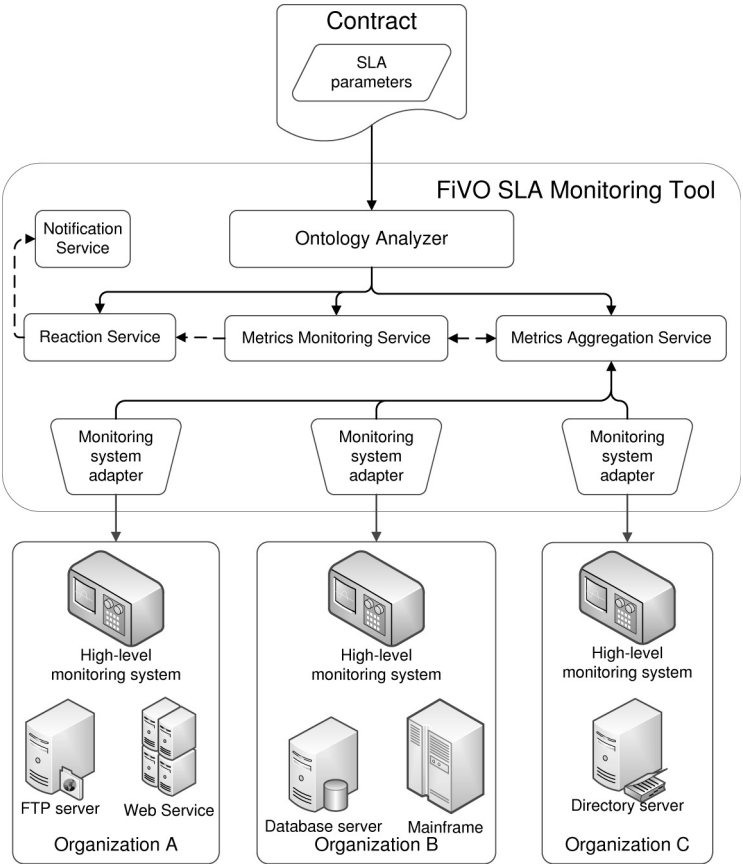

Fig. 12. The SLA Monitoring subsystem architecture

appropriate web services. This component also sends notifications about violation of the contract via different channels (e.g. RSS/Atom feed, instant messenger, etc.).

The implementation of the system is based on two main technologies:

— ServiceMix — it is an implementation of the ESB (Enterprise Service Bus) architecture. The main goal of this architecture is to integrate loosely-coupled services by the overlay messaging system (for a more detailed overview of the ESB architecture, please see [39]);
— Apache Camel — it is an implementation of the EIP (Enterprise Integration Patterns) specification. The main goal of this architecture is to provide universal patterns for exchanging messages between distributed services.

In the case of the SLA Monitoring subsystem, all services are implemented as ESB endpoints, and they communicate by exchanging messages (what is supported by the Apache Camel technology). This approach to the system architecture allowed us to accomplish the assumed system features. More detailed information about the SLA Monitoring system architecture can be found in [38].

6 Enhancing Virtual Organization Monitoring with Semantics

As mentioned in Section 3, one of the most important issues of VO management is the process of contract fulfillment monitoring. In Section 5.2 we described a holistic approach to the VO contract fulfillment monitoring. However, we did not discussed the problem of attribution of data coming from different organizations (included in a VO) which can use different technologies to develop resources (e.g. applications) that are exposed to other partners within the VO. In order to solve the problem, a data attribution system, called SemMon2, was developed. The main features of this tool distinguishing it from the existing monitoring systems include:

— the use of domain specific ontologies to describe a monitoring domain;
— the use of other low-level monitoring systems to collect partial measurement data;
— generating high-level messages in the case where the conditions defined by the user are fulfilled.

In the following subsections we describe the SemMon2‘s architecture, the structure of domain specific ontologies, and the way the system works.

6.1 Architecture

SemMon2 was designed as a distributed system in order to achieve high scalability. This is a very desirable feature, when enabling to connect to multiple external monitoring systems. Distributing computations between multiple physical machines is therefore an indispensable procedure in order to achieve optimal performance of computation.

The architecture of the system with external elements, e.g. lower level monitoring systems, is presented in Figure 13. The main elements of the architecture include:

— *Kernel* is the central point of the architecture. It is responsible for processing data from low-level monitoring systems on the basis of information available in

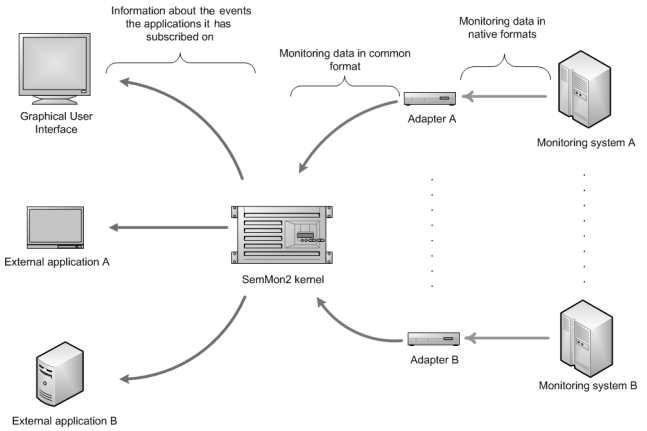

Fig. 13. Overview of the SemMon2 system architecture

domain specific ontologies. The main modules comprise: a module managing the semantic descriptions of the domain, a measurement data processing module, and a module that tracks the defined conditions and sends information about condition fulfillment to listener objects.

— *Adapters of the low-level monitoring systems* provide a bridge between SemMon2 and systems that provide information about the state of the monitored resources. Since SemMon2 is designed to co-operate with different monitoring systems, it is necessary to transform information from the latter into a common format. For this purpose, the well-known *Adapter design pattern* is exploited, to establish a common format for the communication between all stakeholders.

— *External applications* are event listeners of information from the kernel. For this purpose, the *Observer pattern* was used. External objects, such as a Graphical User Interface can subscribe to one or more types of messages defined. An appropriate module from the kernel, is responsible for providing information about events that occur during runtime to the listeners.

Each of the described parts of the system can run on a separate physical device. Also, the core modules of the system kernel have been designed in a way that they can be distributed on different devices if necessary.

6.2 Ontologies

Semantic descriptions in the form of ontologies are the key element of the SemMon2 system configuration. Their content exemplify the domain, which will be monitored. In order to clarify the meaning of the different concepts in the monitoring domain, the following division was adopted:

— `Resource` presenting objects from the domain of monitoring, such that a network service, a hospital department or a doctor ophthalmologist.
— `Capability` that reflects the features of resources that can be monitored, for example the amount of received requests, the number of patients treated the last month or the duration of the examination of the eyes. It is important that each feature was assigned to a resource, based on which it is possible to establish the "context" of a monitoring process. However, a single feature can be assigned to multiple resources simultaneously.
— `Metrics` are aimed to provide information on how to monitor a single characteristic or several interrelated features of the resource. In this way, it is possible to *combine* information about many different resources into a *single metric*, such as using information about the duration of the visit and the amount of work for each day to determine the number of patients admitted. It is also possible to create *multiple metrics* for measuring a single characteristic but with a different statistical approach, such as arithmetic mean, weighted average, maximum, or minimum value.

With this division and with the ability to connect multiple monitoring systems, it is possible to adapt the SemMon2 for monitoring different domains. As support for the user in building their own ontology, a extendable base ontology has been developed. It contains definitions of base classes and relationships, which ensure correct interpretation of a domain specific ontology that is passed to SemMon2. From the ontology developer's point of view, building a domain specific ontology must meet the following requirements:

— each resource class that describes an object in a given area must extend the `Resource` class from the ontology base;
— each class describing a resource monitorable feature must extend the `ResourceCapability` class. For instance, when an instance of the `ResourceCapability` class (X) is a representation of a feature of the resource instance (Y) then a relation `X hasResourceCapability Y` must be satisfied;
— each metric class must extend the class `AbstractMetric` from the base ontology. For instance, when we want to indicate that a metric instance (X) monitors a feature instance (Y) then the relation `X monitors Y` must be satisfied.

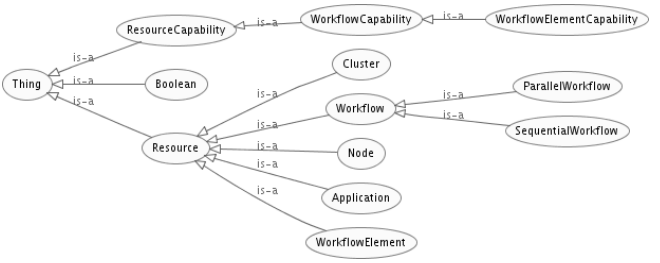

Fig. 14. Sample Resource ontology

A sample ontology which defines domain resources and meets these assumptions is presented in Figure 14. Currently, the ontology is loaded only once, at the tool boot time. The loaded data constitute the *model for monitoring* which remains constant throughout the system runtime. However, the ongoing work aims to enable dynamic modifications of the model domain, without having to restart the tool.

6.3 How SemMon2 Internal Mechanism Works

The SemMon2 kernel is the central point of the system architecture and is responsible for communication between external components, transforming the collected data, according to metric definitions, to more valuable information and tracking user-defined conditions. Immediately after starting the kernel, no monitored applications are connected. Thus, no information is transmitted between the elements of architecture needlessly. During boot time, an ontology is first loaded and a connection to monitoring systems through the mechanism of adapters is established. In order to start monitoring an application, this must be registered in the system through the *Register* tool. Once properly registered, the kernel queries the adapters to provide information about the instances of the resource classes defined in the loaded domain specific ontology. At this point, the user can send a request to start measurements. For each metric started, the kernel runs a new thread, which polls required information from an adapter of a "physical" monitoring system in a cyclic manner. The next step is the optional conversion of the information collected according to the metric definition. After obtaining the final value of the metric, an event is created which carries the value of the changed metric. When the kernel detects a new event occurring it sends information about it to each listener object, which has subscribed to this particular event type, according to the Observer pattern.

One of the event listeners is a metric value analyzer which checks each condition previously defined for compliance with the metric value. If there is a fulfilled condition, the analyzer creates an event and sends it to the kernel, from where it will be further distributed to the subscribed event listeners. The event-based approach is used due to its good scalability (the data are sent only when there is a demand for them), and to low effort needed for the application developers who aim to receive these events.

7 Conclusions

In this chapter we presented our results about the development of a Virtual Organization management framework called FiVO. The framework supports currently such aspects of VO management as contract negotiation, VO knowledge management, automatic VO deployment, and semantic based monitoring.

The impact of the presented framework for supporting dynamic creation and management of Virtual Organizations is in its versatility. First of all, this includes a distributed contract negotiation, using directly both ontological concepts and natural language support. Furthermore, automatic Virtual Organization configuration and deployment, based on a formal agreement within the IT infrastructure of the participating organizations is another strong point. Finally, the monitoring of the execution of the Virtual Organization and the verification whether the negotiated contract is respected by the parties, or not, must be underlined.

The performance evaluation of the SLA monitoring system was presented in [40]. Future work will include further evaluation of the framework within the envisioned applications related to medical teleconsultations.

References

1. Zuzek M., Talik M., Swierczynski T., Wisniewski C., Kryza B., Dutka L., Kitowski J. Formal model for contract negotiation in knowledge-based virtual organizations. M. Bubak, G.D. van Albada, J. Dongarra, P.M.A. Sloot (eds.). Proc. ICCS 2008, Kraków, III, LNCS 5103, 409–418, Springer, Berlin, 2008.
2. Kryza B., Dutka L., Słota R., Kitowski J. Security focused dynamic virtual organizations in the grid based on contracts. Collaboration and the knowledge economy, issuses, applications, case studies. P. Cunningham, M. Cunningham (eds.), IOS Press, Amsterdam, 2008.
3. Funika W., Godowski P., Pegiel P. A semantic-oriented platform for performance monitoring of distributed Java applications. M. Bubak, G.D. van Albada, J.

Dongarra, P.M.A. Sloot (eds.). Proc. ICCS 2008, Kraków, III, LNCS 5103, 233–242, Springer, Berlin, 2008.

4. Kryza B., Dutka L., Słota R., Kitowski J. Dynamic VO establishment in distributed heterogeneous business environment. G. Allen et al. (eds.). ICCS 2009, Baton Rouge, Los Angeles, LNCS 5545, 709–718, Springer, Berlin, 2009.
5. Warner M., Witzel M. Managing in virtual organizations. Thomson Learning, London, 2004.
6. Foster I., Kesselman C., Tuecke S. The anatomy of the grid: enabling scalable virtual organizations. LNCS 2150, 1–4, Springer, Berlin, 2001.
7. Foster I. Globus Toolkit version 4: software for service-oriented systems. J. Comput. Sci. Technol. 21, 513–520, Science Press, Beijing, 2006.
8. Kurumluoglu M., Nostdal R., Karvonen I. Base concepts. Camarinha-Matos et al. Virtual organizations systems and practices. Springer, Berlin, 2005.
9. Camarinha-Matos L.M., Afsarmanesh H. Brief historical perspective for Virtual Organizations. Camarinha-Matos et al. (eds.). Virtual organizations systems and practices, 3–10, Springer, Berlin, 2005.
10. Rezgui Y., Wilson, Olphert W., Damodaran L. Socio-organizational issues. Camarinha-Matos et al. (eds.) Virtual organizations systems and practices. Springer, Berlin, 2005.
11. Loh H., Zhang C., Katzy B. Modeling for virtual organizations. Camarinha-Matos et al. (eds.). Virtual organizations systems and practices. Springer, Berlin, 2005.
12. Katzy B., Zhang C., Loh H. Reference models for virtual organizations. Camarinha-Matos et al. (eds.). Virtual organizations systems and practices. Springer, Berlin, 2005.
13. Negretto U. Report on methodologies, processes and services for VO management. tech. rep., ECOLEAD Project Consortium, 2006.
14. Shelbourn M., Hassan T., Carter C. Legal and contractual framework for the VO. Camarinha-Matos et al. (eds.). Virtual organizations systems and practices. Springer, Berlin, 2005.
15. Udupi Y.B., Singh M.P. Contract enactment in virtual organizations: a commitment-based approach. Proc. 21st Nat. Conf. on Artificial Intelligence AAAI Ul Bostch AAAI Press, Menko Pauls, CA, 2006.
16. Metso J., Kutvonen L. Managing virtual organizations with contracts. Workshop on Contract Architectures and Languages CoALa2005, Enschede, 2005.
17. Cimiano P., Völker J. Text2Onto — a framework for ontology learning and data-driven change discovery. A. Montoyo et al. (eds.). Proc. 10th Int. Conf. on Applications of Natural Language to Information Systems NLDB, Alicante, LNCS 3513, 227–238, Springer, Berlin, 2005.
18. Milosevic Z., Geoff Dromey R. On expressing and monitoring behaviour in contracts. Proc. 6th Int. Conf. Enterprise Distributed Object Computing Conference, Lausanne, 2002.
19. Kuta M., Polak S., Palacz B., Milos T., Slota R., Kitowski J. TeToN — a Jena-based tool for text-to-ontology approach. M. Bubak, S. Unger (eds.). Proc.

Cracow'06 Grid Workshop The knowledge-based workflow system for grid applications, 2006, Kraków, Poland, ACC Cyfronet AGH, 2007.

20. Sokolova M., Szpakowicz S., Nastase V. Language patterns in text messages of electronic negotiations: a preliminary study. INR05/04, `http://interneg.concordia.ca/`.
21. Kersten M.J., Kersten G.E. Developing communication skills using INSPIRE negotiations. INR12/99, `http://interneg.concordia.ca/`.
22. Jertila A., Schoop M. Electronic contracts in negotiation support systems: challenges, design and implementation. Proc. 7th IEEE Int. Conf. on E-Commerce Technology CEC'05, IEEE Computer Society, Los Alamitos, CA, 2005.
23. Yuanming C., Wendong W., Xiangyang G., Xirong Q. Initiator-domain-based SLA negotiation for inter-domain QoS-service. Provisioning, K. Chin, C. Dini, L. Lehmann, D.C. Pheanis (eds.). IEEE Computer Society, Los Alamitos, 2008.
24. Kosinski J., Nawrocki P., Radziszowski D., Zielinski K., Zielinski S., Przybylski G., Wnek P. SLA monitoring and management framework for telecommunication services. J. Bi, K. Chin, C. Dini, L. Lehmann, D.C. Pheanis (eds.). IEEE Computer Society, Los Alamitos, CA, 2008.
25. Zulkernine F.H., Martin P., Craddock C., Wilson K. A policy-based middleware for web services SLA negotiation. ICWS'09 Los Angeles, CA, IEEE, Washington, 2009.
26. Alfieri R., Cecchini R., Ciaschini V., dell'Agnello L., Frohner A., Gianoli A., Lrentey K., Spataro F. VOMS: an authorization system for virtual organizations. European Across Grids Conf. LNCS 2970, 33–40, Springer, Berlin, 2003.
27. Chadwick D.W., Zhao G., Otenko S., Laborde R., Su L., Nguyen T.-A. PERMIS: a modular authorization infrastructure. Concurr. Comp. Practice Exp. 20, 1341–1357, Wiley, New York, 2008.
28. Sinnott R.O., Jiang J., Watt J.P., Ajayi O. Shibboleth-based access to and usage of grid resources. 'Grid'06' Barcelona, IEEE Computer Society, Washington, 2006.
29. Masuoka R., et al. Policy-based access control for task computing using Rei. Proc. Policy Management for the Web Workshop. Chiba, Japan, 2005.
30. Pahlevi Said M., Kojima I. S-MDS: Semantic monitoring and discovery system for the grid. J. Grid Comp. 7, 205–224, Springer, Netherlands, 2009.
31. Vaculin R., Sycara K. Semantic web services monitoring: an OWL-S based approach. Proc. Hawaii Int. Conf. on System Sciences, 2008.
32. Kryza B., Slota R., Majewska M., Pieczykolan J., Kitowski J. Grid organizational memory-provision of a high-level Grid abstraction layer supported by ontology alignment. Int. J. FGCS Grid Comp. Theory, Meth. Appl. 23, 3, 348–358, 2007.
33. Kryza B., Mylka A., Świderska A., Kitowski J. Application of unified metadata grid framework supporting resource discovery and matchmaking. Presented at KUKDM 2010, Zakopane, 2010 and submitted to Computing and Informatics.
34. Rabelo R.J., Gusmeroli S. The Ecolead collaborative business infrastructure for networked organizations. Virtual enterprises and collaborative networks. M. Luis, Camarinha-Matos, Willy Picard (eds.). Springer, Berlin, 2008.

35. Kryza B., Pieczykolan J., Bubak M., Majewska M., Slota R., Hluchy L., Kitowski J. Managing semantic metadata in K-Wf grid with Grid Organizational Memory. M. Bubak, M. Turała, K. Wiatr (eds.). Proc. of Cracow Grid Workshop — CGW'05, 2005, Kraków, Poland, 66–73, ACC-Cyfronet UST, 2006.
36. Eclipse RCP project site. `http://www.eclipse.org/home/categories/rcp.php`.
37. Eclipse RCP. `http://en.wikipedia.org/wiki/Eclipse_RCP#RCP`.
38. Skałkowski K., Sendor J., Król D., Funika W., Kryza B., Słota R., Kitowski J. Monitoring of SLA parameters within VO for the SOA paradigm. Proc. Int. Conf. on Parallel Processing and Applied Mathematics PPAM09, Springer, Berlin, 2010.
39. Chappell D.A. Enterprise Service Bus. O'Reilly, Sebastopol, 2004.
40. Skałkowski K., Sendor J., Slota R., and Kitowski J. Application of the ESB Architecture for Distributed Monitoring of the SLA Requirements. 2010 Ninth Int. Symp. on Parallel and Distributed Computing. ISPDC 2010, Istambul, IEEE Computer Society, 2010, DOI 10.11.09/ISPDC.2010.21.

Breeding Virtual Organizations in a Service-Oriented Architecture Environment

Willy Picard, Zbigniew Paszkiewicz, Piotr Gabryszak, Kamil Krysztofiak, and Wojciech Cellary

Department of Information Technology
Poznań University of Economics,
Mansfelda 4, 60-854 Poznań, Poland
{picard,zpasz,gabryszak,krysztofiak,cellary}@kti.ue.poznan.pl

Abstract. A Virtual Organization Breeding Environment (VOBE) is an accepted concept in the research area of collaborative networks. So far, existing VOBEs are based on an infrastructure providing only limited support for efficient integration of VOBE members, and virtual organization partners on both technical and business levels. Thus, the Service-Oriented Architecture (SOA) has been proposed in this chapter as an approach to implement VOBE. A VOBE implemented in this way is called a Service-Oriented Virtual Organization Breeding Environment (SOVOBE). A SOVOBE is systematically organized around the concept of services, which are not limited to Web services, but which encompass also services performed by humans (organizations). In this chapter a set of core services is specified provided by SOVOBE to support SOVOBE members and virtual organizations throughout their lifecycle. The core services include Competence Management Service, Social Network Service, VO Collaboration Service, VO Creation Service, and VO Monitoring Service.

Keywords: Virtual Organization, breeding environment, SOA

1 Introduction

A *Virtual Organization* (VO) is defined as an operational structure consisting of different organizational entities and created for a specific business purpose, to address a specific business opportunity. Based on the above concept of VO, the concept of *Virtual Organization Breeding Environment* (VOBE, sometimes shortened to VBE in the literature) has been proposed as "an association of organizations and their related supporting institutions, adhering to a base long term cooperation

agreement, and adoption of common operating principles and infrastructures, with the main goal of increasing their preparedness towards collaboration in potential Virtual Organizations (VO)" [1]. The main aims of VOBEs are: establishment of mutual trust among organizations to facilitate their collaboration in VOs, reduction of cost and time to find suitable partners for a particular VO, assistance in VO creation including reaching agreement between partners, and VO re-configuration aiming at adaptation to new business challenges and opportunities. To encompass general concepts of Virtual Enterprises, VOBEs, and VOs, the term Collaborative Network Organization (CNO) has been coined [1].

While the concept of VOBE is currently widely accepted in the CNO research community, there is still no final consensus about the architecture and implementation of VOBEs. So far existing VOBEs have been created in an ad hoc manner and have an infrastructure allowing limited support for efficient integration of VOBE members and VO partners on business and technical levels. An appropriate IT infrastructure of a VOBE should provide at least the functionality associated with: collaboration and negotiation, interoperability, discovery and distribution of knowledge and resources, and integration of business processes. A discussion of IT solutions to support collaboration among VOBE members is presented in [2].

In [2] the Service-Oriented Architecture (SOA) has been suggested as a valuable approach to VOBEs implementation. SOA has been defined by the OASIS group [3] as "a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. [...] In SOA, services are the mechanism by which needs and capabilities are brought together." The OASIS definition emphasizes some characteristics of SOA shared with CNOs: CNOs may be seen as structures aiming at "organizing and utilizing distributed capabilities under the control of different ownership domains" [3].

The above works provide general guidelines, recommendations and general specification of services that should be provided by a VOBE. A systematic approach to breeding of virtual organizations in a SOA environment is still to be proposed, as well as a detailed specification and implementation of core services of VOBEs.

In this chapter the concept of Service-Oriented Virtual Organization Breeding Environment (SOVOBE) is proposed. A SOVOBE is a VOBE organized systematically around the concept of a service. In SOVOBEs, services are not limited to Web services, but they encompass services performed by humans. Core services supporting VOs and SOVOBE members are proposed in this chapter.

This chapter is organized as follows. In Section 2, the concept of VOBE is described being a foundation for further analysis. In Section 3, the Service-Oriented Virtual Organization Breeding Environment (SOVOBE) is presented. In Section 4, core internal services related with SOVOBE members are detailed, i.e., organization competence services and social network services. Next, internal SOVOBE services associated with the lifecycle of VOs, i.e., VO creation, collaboration and monitoring, are presented. Finally, Section 5 concludes the chapter.

2 Virtual Organization Breeding Environments

2.1 Rationale of VOBEs

VO partners possess complementary skills and competences, and cooperate by the use of information and communication technology. While partners theoretically can be identified and recruited from the open universe of available organizations (path 2 in Fig. 1), such on approach meets a number of questions that need to be answered [4]:

— how to quickly define the agreements on the roles and responsibilities of each partner, to reflect sharing of tasks and the rights on the produced results?
— how to know about the mere existence of potential partners in the open universe and deal with incompatible sources of information?
— how to acquire basic profile information about organizations, when there is no common template or standard format?
— how to quickly and reliably establish an interoperable collaboration infrastructure, given the heterogeneity of organizations at multi-levels, and the diversity

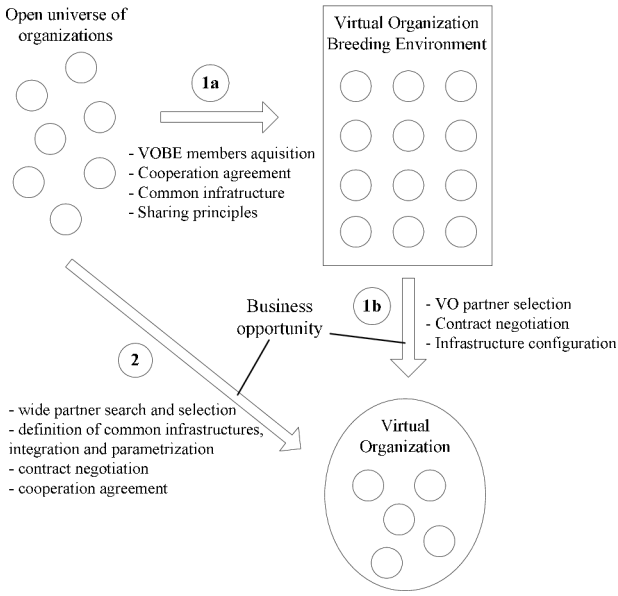

Fig. 1. Two approaches to virtual organizations creation [5]

of their interaction systems?
— how to build trust among organizations, which is the base for any collaboration?
— how to quickly develop and agree on common principles of sharing and working together?

These questions are difficult to be answered if the open universe of available organizations is assumed. As a consequence, in this case the process of VO creation is time-consuming and complex, discouraging organizations from taking advantage of a sudden business opportunity and from adapting to new business needs. To overcome the above problems, the concept of Virtual Organization Breeding Environment has been developed.

A Virtual Organization Breeding Environment (VOBE) (cf. Section 1) is forcing its members to share common principles, standards and solutions, and thus facilitates the process of VO creation (path 1 in Fig. 1).

2.2 Functionality of a VOBE

The following VOBE functions are supporting the establishment of VOs:

— providing access to information not available in the open universe, such as information about the past performance of VOBE members, providing a standardized description of partner profiles, competences and services;
— supporting the potential partner search and selection;
— providing methods and tools for analysis and evaluation of present and future cooperation performance;
— providing necessary information for trust building among selected members;
— providing software to support collaboration of partners within newly created VOs.

A VOBE facilitates the VO creation by standardizing required information and exchanged data, by creating an environment for the integration of partners, and by providing a set of information not available in the open universe. As a consequence, time and complexity of VO creation, as well as business risk associated with the participation in a VO, are reduced, therefore increasing the number of seized business opportunities.

A VOBE provides also support for VO operations in the other phases of the VO lifecycle [5]:

— in the operation phase: support for communication and exchange of documents, infrastructure for integration of heterogeneous information systems, management of common infrastructure, guidelines for standardized data formats, data storage facilities, information about changing environment (context) of collaboration, reuse of artifacts elaborated by other VOs (in particular business process models, best practices), information about new collaboration opportunities, etc;
— in the evolution phase: mechanisms supporting adaptation such as redefinition of business processes and goals, new partner search, support for negotiation, etc;
— in the dissolution phase: knowledge inheritance, i.e., the capturing and reusing of former experience gained during the operation of VOs.

The link between a VOBE and a VO throughout the VOBE lifecycle is presented in Figure 2. Once a VOBE is created, its main goal is to support VOs, in particular VO creation. VOBE evolves during its lifetime, modifying and adapting its infrastructure, and policies to new needs of its members. Finally, a VOBE can be a subject of metamorphosis leading to potentially new forms of organization. As VOBEs usually possess large amount of information, and knowledge, as well as infrastructure and social capital, it is unusual for VOBEs to dissolve.

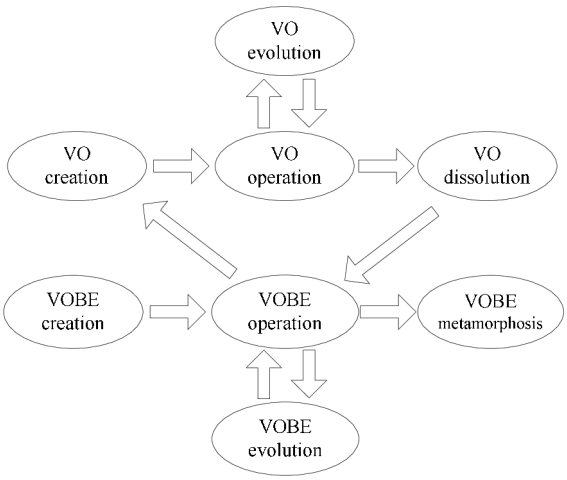

Fig. 2. VOBE and VO life cycles

3 Service-Oriented Virtual Organization Breeding Environments

3.1 Rationale of SOVOBEs

While VOBEs are implemented in an ad hoc manner, *Service-Oriented Virtual Organization Breeding Environments* (SOVOBEs) are organized systematically around the concept of a service. As a consequence, concepts underlying SOA may be applied at the coarser level to organizations within the context of SOVOBEs:

— *service reuse* — a given organization may provide the same service within many VOs;
— *service abstraction* — the details of the implementation of services offered by a given organization within a VO are usually hidden for other organizations, because the implementation of the core business services is associated with the know-how capital that gives the organization business advantage over competitive organizations;
— *service discoverability* — services provided by organizations in a SOVOBE are described so that both services and associated organizations may be identified as potential VO partners for a given business opportunity;
— *service composition* — a complex service provided by a VO is the result of composition of services provided by VO partners and eventually by the SOVOBE.

3.2 SOVOBE Architecture Overview

A SOVOBE is a VOBE organized in a systematic way around the concept of a service. Services may be Web services, potentially integrated by an Enterprise Service Bus (ESB), as well as services performed by humans (organizations). Depending on the type of service providers and consumers, the following classification of services is proposed (cf. Fig. 3):

— *business services* — services provided by SOVOBE members for chosen VO partners;
— *internal services* — services provided by the SOVOBE and consumed by its members. This set of services includes services for partner and business service selection, tools for social protocol modeling, social network modeling, performance estimation, and competence modeling;

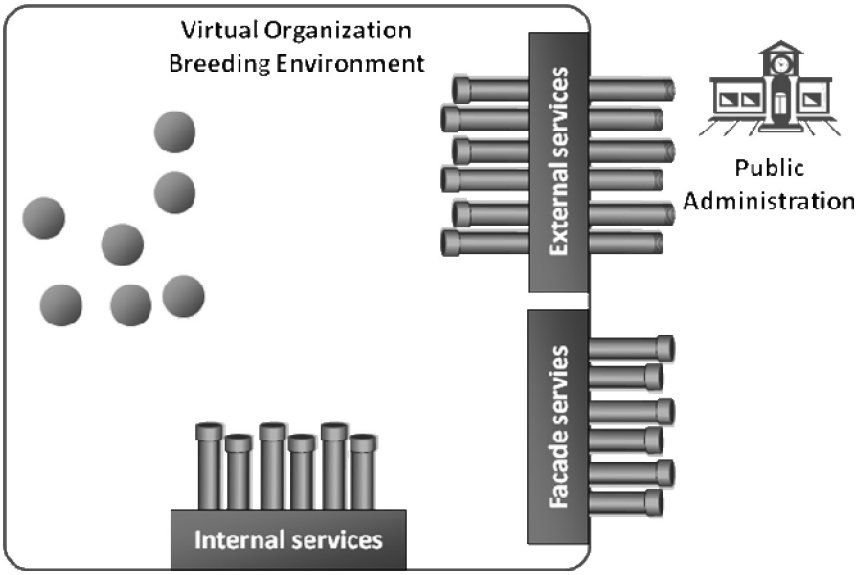

Fig. 3. Services provided by a SOVOBE for its members and the outside world

— *external services* — services provided by organizations operating outside the SOVOBE, but offered by the intermediation of SOVOBE to its members. External services facilitate interactions between external organizations (e.g., public administration units) with SOVOBE, its members and VOs;
— *façade services* — services provided by the SOVOBE to organizations outside the SOVOBE. Façade services provide external organizations with access to information about the SOVOBE and allow the submission of information to the SOVOBE. This set of services includes services for providing information about the SOVOBE and its members' profiles, and service for announcing business needs.

4 SOVOBE Core Internal Services

The number of internal services that a SOVOBE may offer to its members is theoretically unlimited. In this section, a set of internal services common to all SOVOBEs, i.e., *core internal services*, is proposed. SOVOBE core internal services focus on management of either SOVOBE members, or VOs. Other internal services, e.g. trust or security-related internal services, are not addressed in this paper,

neither is the impotant issue of SOVOBE management, including the management of the core internal services.

First, two core internal services focusing on the management of SOVOBE members are presented: the competence management service provides means for structured description of SOVOBE members, while the social network service addresses relations among SOVOBE members. Second, three core internal services focusing on the management of VOs: the VO creation service, the VO collaboration service, and the VO monitoring service are proposed.

4.1 Management Services of SOVOBE Members

Support for service selection is the main responsibility of a SOVOBE, which must provide methods and techniques facilitating this activity. Aspects taken into account during business service selection process are divided into:

— service description;
— service provider characteristics;
— business process context, in particular social context.

In SOA, standards supporting Web service description such as WSDL [6], OWL-S [7], and WSMO [8], have been developed to provide information necessary to find a service and interact with it. These standards permit [6]:

— *service discovery* — a process for location of services that can provide a particular functionality, while adhering to some client-specified constraints;
— *service invocation* — an invocation of a service by a computer program or a software agent, given only a declarative description of that service, as opposed to when the agent has been pre-programmed to be able to call that particular service;
— *service composition and interoperation* — a selection, composition, and interoperation of services to perform some complex tasks, given a high-level description of an objective.

In a business environment, where services are usually complex, the aim of business *service description remains* the same, but aspects going beyond service description have to be considered during service selection.

An important aspect of business service description is information about entities that may provide a required service. The characteristics of a candidate for a collaboration partner include its competences, certificates confirming competences,

capacities, former performance, realized projects, etc. In SOVOBEs, structured descriptions of SOVOBE members take the form of a competence model. Functionality of competence management and verification is provided by the *Competence Management Service*.

The second important aspect of business service description is the information about existing relations among service providers and service consumers, supported in SOVOBEs by the *Social Network Service*.

Competence Management Service. A *service provider* is an organization (in particular: company, public administration unit, business unit, team, person) that provides a given set of services.

Competency is defined as "the organization's capability to perform (business) processes, tasks, having the necessary resources (human, technological, physical) available, and applying certain standards (practices), with the aim to offer certain products and/or services" [9]. *Competency-based VO creation* is an approach to VO partner search and selection based on information available in a SOVOBE in a form of structured competence description (competence requirements).

To promote itself to acquire new VOs and to be included in VO partner search, each SOVOBE member provides detailed and up-to-date information about the activities it is able to perform and the services it can offer. The information typically includes "an accurate description of member capabilities, its free resources capabilities, the production costs for each of its product, as well as conspicuous proof of the validity of the provided information" [9]. In small SOVOBEs, the competency description can perhaps be transmitted in an unstructured and oral way to VO planners and the SOVOBE administrator. In medium and large SOVOBEs, this approach is not effective because the amount of information to be maintained is significant and complex. Moreover, it changes dynamically over time as a result of adaptation to market needs. In such SOVOBEs, computer support for management of competence descriptions is required. In [9] the following approaches and standards for competence modeling are presented: core competence notion (the first time proposed in 1990), core competence hierarchy (1998), HR-XML competences schema (2001), core competencies in the manufacturing clusters (1999), competence cells (2006), s-a-r-C model (2005). Most of these models have been elaborated to be used in traditional human resources management. Finally, the 4-C model has been proposed in [9] which relies on four key concepts: competence, capability, cost, and conspicuity.

Taking into account the characteristic of SOVOBEs drawbacks of the 4-C model are:

— unclear distinction between the competence and capability concepts;

— poor service-orientation — in the 4-C model a service is modeled as a *product*, output of the activity or process represented by capability — there is one-to-one relation between a capability and a service and, as already mentioned, it is unclear how capability relates to a competence;
— limitation of the competence model to a competence description model — the model misses a method for evaluation of quality and relevance of information in competence description;
— missing concept of competence description change and versioning — competence description can change over time;
— missing clear distinction between competence and service description model;
— very limited model of a service description;
— missing context in the description of competence — competence details may depend on circumstances, especially organization capabilities; including *context* in a service description allow for more detailed specification, i.e., production capabilities and use of resources, delivery time may depend on the season, a price may depend on target market, etc.

Therefore, there is a need for a competence model tailored to the needs of SOVOBE. Such a model should include:

— competence description model consisting of:
  - organization profile;
  - competence profile;
  - service business profile;
— competence verification method;
— competence management method.

Competence description model should encompass methods of verification of competence description relevance. This aspect is, unfortunately, usually ignored. A set of services for management of competences should be included in the model. Such set of services defines possible actions SOVOBE members playing different roles may take.

In SOVOBE, partner selection is strictly connected with service selection. Moreover, a structured competency description creates a possibility of spotting new potential collaboration — joint partners' competences permit offering new services for consumers.

Competence Description Model. In the SOVOBE member competence description, it is reasonable to distinguish organization profile, organization competency profile and a business-level service description. Services should be also described on the technical level allowing their consumption, but this kind of description is out of the scope of the competence description model.

Description of organization profile should include a non-service specific information such as: history of collaboration, past performance, formal certificates, recommendations, membership in associations, localization, financial capital, contact information, steering managerial board, etc.

Competence description consists of concepts referring to organization activities and processes an organization can perform, and their characteristic. Main concepts that should be modeled in SOVOBE as a competence profile, but which are not present or are understood differently in the 4-C model include:

— *service* — in the 4-C model, the term *service* refers to the output of a competence, while in SOVOBEs, the term *service* should be understood as in the SOA environment (cf. Section 1);
— *context* — is defined by a set of triplets: <*object, predicate, subject*>; context describes conditions under which a capability may deliver defined outcomes, with defined costs, use of resources, etc.;
— *capacity* — represents the amount of resources needed to perform a given number of processes and activities associated with a capability; a capacity depends on a context;
— *capability* — represents processes and activities of an organization whose many aspects depend on context including cost, capacity, and possible output;
— *competence* — every organization registered in SOVOBE must poses at least one competence; competence may be directly connected with services; a competence includes a non-empty set of capabilities connected with capacities depending on a context; in SOVOBE competences are in M-N relation with services (there is a number of competences needed to provide a given service, on the other hand a competence may be used in provision of many services);
— *product* — modeled as an output of a capability and may take a form of a product or service; in VOBEs this term Product is not used, and Service is a separate object being in M-N relation with Competence; also a concept of Output is introduced and represents the result of the activities represented by a competence;
— *version* — competence versioning allows for tracking of organization evolution and adaptation to market needs and particular collaboration opportunities;
— *visibility* — some competences may be a result of negotiation and adherence to a particular VO; this competence should not be visible for all SOVOBE members but data on such competences should be present in SOVOBE as it might be used for estimation of VO efficiency and other analysis; this issue strongly refers to data access management in SOVOBE.

The business service profile includes business characteristics of a service, free of any technical aspects, i.e., service reference to VO strategic goals, entities responsible for a service, strategy of service, formal requirements, etc. A number

of approaches to service description including business aspects has been proposed, including enterprise architecture modeling (TOGAF [10]), semantic-based approaches (WSMO [8]) and others [11].

Competence Description Verification. Information contained in competence description must be reliable. Information provided by an organization must be confirmed or verified against other sources of information about this organization. A competence description model allows for initial verification of data reliability with the use of conspicuities. Conspicuity is a formal or informal document justifying, confirming and explaining information provided in a competence description.

SOVOBE stores information about all organizations, history of collaboration, efficiency of collaboration, former and existing problems, etc. which creates possibilities for verification of information provided in competence descriptions. Sources of information existing in SOVOBE that can be used for verification of competence descriptions are presented in Table 1.

Table 1. Data sources for verification of competence descriptions

Category name	Description
1 Continuous monitoring of collaboration	Data provided by monitoring current provision of services and progress of running collaboration processes
2 History of collaboration	Data restored form the history of partners' performance and collaboration within SOVOBE
3 Description of competences	Conspicuities held in descriptions of SOVOBE members and competences
4 Social network	Data provided from social networks

Competence Verification and Management. SOVOBE must provide a set of services referring to competence management. This set should include: registration service, create, read, update and delete operations referring to profiles comprising competence description model, versioning services, search and compare services, competence verification and evaluation services, and storage services. Services referring strictly to competence include negotiation services. Access to appropriate services should depend on roles and defined restrictions.

Social Network Service. In SOA, interactions between service providers and service consumers are always performed within a *context*, in particular a *social context* defined by relations between them. A context is important as it has been noted by OASIS in the SOA Reference Architecture [13]: "The actions undertaken by participants, whether mediated by services or in some other way, are performed in a context that defines the meaning of the actions themselves. That context is

fundamentally a *social context* — a context that includes other participants. We can formalize that context as a social structure: the embodiment of a particular social context."

In this section the role of social context in functioning of virtual organization breeding environments and virtual organizations is described.

Support for VO Creation. Social aspects play an important role during VO creation. The VO creation process is based on competence requirements and social requirements [14]. While competence requirements determine the competences needed for future VO partners' cooperation (cf. Section 4.1), *social requirements* concern relations among future VO partners.

In a business environment, information referring only to a particular organization should be supplemented by information concerning relations among organizations. This information may come from sources different from the organization itself. This information concerns past performance, history, trust, recognition, etc. It may strongly influence the selection process. For instance, a VO planner may require candidates for VO partners to have a common successful cooperation history. Such requirement is strictly a social requirement, being in fact a requirement of a certain type of relation between organizations.

Support for VO Agility. The social context is significant for VO *agility* [15]. A VO is said to be *agile* if it can rapidly and cost effectively adapt the way it is functioning to changes [15, 18]:

— due to events within a virtual organization, e.g., when a VO partner bankrupts;
— due to events in the environment a VO is functioning in, e.g., when a natural disaster prevents some tasks to be realized.

The adaptation of a VO to new conditions can be simplified by using the social context of VO partners. Adaptation to changes often requires new resources. In a situation when performance of a VO partner suddenly decreases, a possible solution is to search for another organization, which may substitute or complement the current VO partner. The social context of VO partners may be a good place to search at first.

Social Context Services. The following social services are proposed in this chapter as SOVOBE core internal services:

— *social context structure management services* — services responsible for creating, reading, updating and deleting information about organizations within SOVOBE and relations among them;
— *social requirements definition services* — services used to define social requirements during VO creation and evolution;

— *social requirements validation services* — services used for validation of social requirements during VO creation and evolution;
— *social context properties calculation services* — services providing properties of social context; these properties can be used in monitoring of social context within SOVOBE or within VOs.

Social context can be modeled as a *social network*. A social network (SN) is a graph in which nodes represent social actors and edges reflect relations among those actors [16, 17].

In SOVOBEs, social contexts are represented by social networks in which actors are SOVOBE members and relations are social relations among SOVOBE members. Changes in a social context imply changes in the social network that represents the context, and in the graph modeling the social network. This way of social context modeling permits to apply well developed tools of graph management to provide computer support for social context services.

Social Network Metrics. Social network analysis (SNA) methods, based on the graph theory and matrix algebra, are mature and well described in the literature [16, 17]. SNA methods focus mainly on relations and on the overall structure of a social network. The most common social network metrics are [14, 16, 17]:

— *size* — the number of actors of the network;
— *average path length* — the average distance between all pairs of actors;
— *density* — the proportion of relations in a network relative to the total number of possible relations;
— *degree* — the number of relations of an actor;
— *closeness* — the inverse of the sum of the shortest distances between each actor and every other actor in the network;
— *eccentricity* — the maximum of the shortest paths to other actors in the network; eccentricity indicates how far a given actor is from the furthest one in the network;
— *neighborhood size* — the number of other actors to which a given actor is adjacent, i.e., has a direct path;
— *reciprocated ties density* — the ratio of the number of relations that are bidirectional to the neighborhood size.

Social requirements. Social requirements concern the nature, structure and characteristics of the relations among organizations in a social network. Social requirements are usually expressed as a set of social network metrics with associated expected values. An example of a simple social requirement concerning a future VO is “the VO must have five partners and one VO leader”, while the associated social network metrics are: the size of the social network and the neighboring size of

one actor, associated respectively with expected values six and five. Social requirements are usually at a high, business level of abstraction, whereas social network metrics are at much lower level of abstraction, focusing on the raw structure of a social network.

To provide computer supported social context services in SOVOBEs, social requirements have to be translated to social network metrics. The process of requirements translation is a complex issue because of difficult expression of usually soft social requirements. It may involve usage of complex techniques such as natural language processing. It needs more attention and investigation [14].

4.2 Management Services of Virtual Organizations

VO Collaboration Service. The concept of a *social protocol*, on which the VO collaboration service relies, is proposed in this chapter as an attempt to link process models to the social context within which the processes are instantiated.

Process models capture the structure of interactions among persons, organizations and software entities. The social context within which a given process model may be instantiated is not part of the process model.

Process model. An important characteristic of social protocols is the potential support for any type of process model. Therefore, social protocols may be based on various types of process model, e.g., Finite State Machines (FSM), Petri Nets, BPEL, and BPMN.

The main requirement imposed on process models is the possibility to extract a *service-oriented* summary of a given process model, which consists of:

— a set of *process roles* — each process role defines a set of rights to consume some services;
— a set of *service providers* — each service provider is associated with a set of services descriptions;
— a set of associated *service descriptions* — each service description consists of a set of attributes as pairs (name, value).

Social network schema. Additional requirements for elements of the service-oriented summary may concern:

— necessary characteristics of process roles, service providers, or service descriptions;
— necessary characteristics of relations among process roles, service providers, and service description.

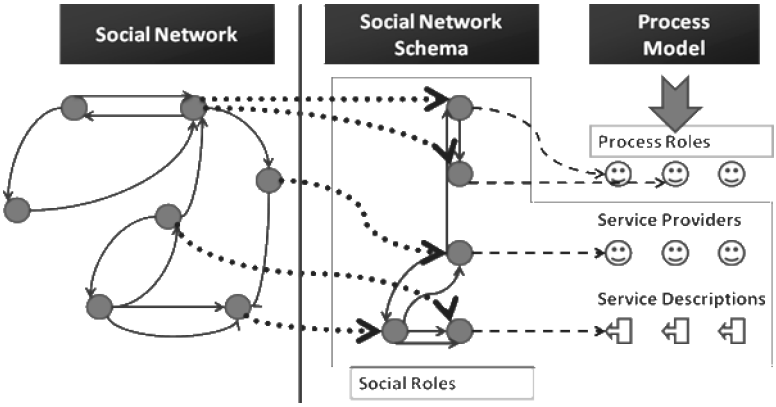

Fig. 4. Social Protocols

In social protocols, the requirements for elements of the service-oriented summary are modeled as a *social network schema*. A social network schema (SNS) consists of nodes and relations. Nodes capture the characteristics of elements of the service-oriented summary, while relations capture characteristics of the relations between elements of the service-oriented summary.

Mapper and Implementer. A *Mapper* is the link between the service-oriented summary and the SNS. A Mapper associates process roles, service providers and service description with nodes of the associated SNS. A Mapper is represented by dashed lines in Figure 4.

While process roles capture the rights to perform some activities, the Mapper captures *obligations*: elements of the service-oriented summary associated with nodes of the SNS have to satisfy requirements defined in the SNS node and its associated relations.

An *Implementer* associates nodes of the SNS with actors of the social network. An Implementer is represented by dotted lines in Figure 4.

An Implementer instantiates a social protocol by assigning actors of the SOVOBE social network, i.e., real persons, organization or software entities, to social roles.

VO Creation Service. Partner search and selection is a key part of VO creation. A SOVOBE provides its members with a service supporting searching for partners and services suitable for a particular cooperation process. The selection of partners and services on the business level is a complex task that can hardly be automated. It encompasses the following aspects:

— determination of requirements by VO stakeholders;
— determination of particular requirements for VO partners and their business services, processes, and services offered to the VO clients;
— identification of SOVOBE members able to play a particular role in a business process or fulfill a particular task;
— negotiation and settlement of cooperation rules and conditions;
— analysis of possible VO variants in terms of conformance to requirements and efficiency of cooperation.

The selection should include various viewpoints:

— subjective viewpoint of a VO planner;
— objective aspects contained in history of collaboration stored in SOVOBE;
— opinions of SOVOBE members represented in a social network;
— anticipated efficiency of collaboration estimated basing on both objective and subjective sources of information;
— requirements imposed on a potential VO by SOVOBE.

Therefore, various services, providing such functionalities as SOVOBE members comparison, VO variants comparison, and support for analysis and filtering of organizations, should be a part of SOVOBE core services supporting partner search and selection.

Requirements for VO Creation Service. The VO Creation (VOC) Service is proposed in this chapter as a SOVOBE core internal service. The following are requirements for the VOC service:

— conformance with virtual organization breeding methodology defined in [12];
— human control over the process — in context of complex business processes such search and selection cannot be fully automated, so human control over the process is vital;
— multi aspect and multi criteria process of search and selection:
  - inclusion of social aspects;
  - competence-based approach (cf. Section 4.1);
  - requirement-based approach — social and non-social requirements coming from many sources (SOVOBE, VO planner, potential VO customers, etc.) defined for various elements of virtual organization (process, partner, etc.);
  - definition of preferences referring to requirements and taken into account in VO variant evaluation;
  - multi-variant analysis — evaluation of various partner and service compositions to maximize requirements satisfaction;

Table 2. Outline of the partner search and selection procedure implemented in the VOC service

Category name	Description
1 Definition of requirements and preferences	Social requirement extraction from social roles defined in social protocols Requirement definition for VO partners, services, processes, subgroups of partners and services Preference definition Policy formulation
2 Partner selection for social roles	Search for partners satisfying requirements defined for each social role
3 VO variant analysis	Generation of possible VO variants and their validation against requirements and preferences
4 Partner selection for process roles	Alignment of partners to process roles Verification of requirements
5 VO inception	Registration of the created VO Update of SOVOBE repositories: competence model repository and the social network

— collaborative partner search and selection;
— persistency — processes of VO creation must be monitored and recorded for further reuse and analysis.

In Table 2, general steps of the procedure implemented in the VOC service are presented. First, a set of functional and performance requirements identified. These requirements may refer to various VO components: partners, services, processes, subgroups of partners and services. Additionally social requirements are extracted from *social roles* defined in *social protocols* (cf. Section 4.2). Final definition of requirements includes specification of a method step in which the particular requirements will be used in (steps 2–4). Full definition of requirements constitute *VO specification*. Next, potential partners fulfilling requirements of social roles are chosen from SOVOBE members. As a consequence, a non-empty set of organizations is associated to each social role. In step 3, various VO variants are created and evaluated on the basis of above sets. Finally, the best VO variant is chosen and registered in SOVOBE. The whole process is accompanied with negotiations among the VO planner and potential VO members.

In every step, human action is possible and may take a form of requirements redefinition, preference modification, reconfiguration of tools, reordering and repetition of a step.

4.3 VO Monitoring Service

Definition of Performance Measurement. VO Performance Measurement (PM) is defined as a "systematic approach to plan and conduct the collection and monitoring of data for performance indicators. The Performance Measurement is focused on the process of data collection. The input are defined Performance Indicators (PI) including targets. For these PIs appropriate methods and tools of measuring have to be developed and planed. The collected data are then prepared into indicators. As a result, performance measurement produces reports that could be used for further analyses and interpretation to assess the performance and to derive measures for improvement" [19].

At a higher level of abstraction, PIs are usually combined to define Key Performance Indicators (KPIs). *Key Performance Indicator* (KPI) has been defined as "a performance indicator which represents essential or critical components of the overall performance. Not every performance indicator is vital for the overall success of a company. To focus attention and resources the indicators with the highest impact on performance are chosen as key performance indicators. [...] An *indicator* is a variable which is feasible to assess the state of an object in scope. Indicators could be as well quantitative as qualitative measures. They can not only consist of a single measure but also be aggregated or calculated out of several measures" [19].

VO PM is a difficult task, as it usually implies many KPIs based on data from multiple sources. In a SOVOBE, data for performance indicators may be supplied by most internal services, by business services, and by external services. Supporting the heterogeneity of data sources for PM is a challenge that requires a systematic classification of indicators, as proposed in [19], and providing means for the description and selection of suitable indicators and sources of data essential for their calculation.

Reference Model for PM. For SOVOBEs, a *Reference Model for Performance Management* (*RM4PM*) has been proposed in [19]. The goal of RM4PM is twofold: on the one hand, the reference model defines a set of common terms that can be used to describe key performance indicators (KPIs). On the other hand, the reference model defines four taxonomies of KPIs, each taxonomy focusing on a given aspect of PM:

— availability and type of data sources;
— subject of measurement (e.g., a process or a VO);
— scope (e.g., a VO or the SOVOBE);
— collaboration characteristics (e.g., responsiveness or flexibility).

KPIs Calculation. In SOVOBE, KPIs calculation is based on a function that may be defined in any programming language supported by the Java Scripting technology [20], e.g., AWK, Java, Javascript, PHP, Python, Scheme, SmallTalk. As a consequence, the learning curve is reduced as the probability that the user willing to define a KPI already knows one of the supported programming languages is rather high.

Various data sources may be associated with the parameters of the KPI function, e.g., a KPI may aggregate data coming from the competence management service and the social network service.

Application of KPIs. Performance anticipation and monitoring are the two applications of KPIs. The performance of VO variants may be anticipated due to calculated KPIs. Then, VO variants may be ranked according to their anticipated performance.

In the case of KPIs monitoring application, changes in data sources lead to recalculation of the appropriate KPIs. Then the results of the recalculated KPIs are compared with a threshold value associated with a given KPI. If the threshold is reached, a special event is fired to associated services responsible for proper reaction.

5 Conclusions

The concept of Service-Oriented Virtual Organization Breeding Environment (SOVOBE) presented in this chapter proves that the Service-Oriented Architecture (SOA), which has its roots in the research area of information system integration, may be applied at the coarser level of organizations. In particular, concepts such as service reuse, service loose coupling, service abstraction, etc. may be referred to virtual organizations.

The main advantage of building SOVOBE systematically around the concept of a service is separation of SOVOBE core services functionality from their implementation, which facilitates changes of SOVOBE service implementations. Changes of service implementations usually follow from SOVOBE member needs to adjust to evolving business environment. Concluding, application of SOA to VOBE significantly contributes to VOBE agility.

As a target of future works, detailed comparison studies are planned concerning SOA concept application at the Web services and virtual organizations levels. It has been noticed that while concepts used in the communities working independently at both levels are often similar, the terms used to refer to these concepts are usually different. Detailed analysis of similarities and dissimilarities of SOA

implementations at both levels may help to reuse solutions developed by one community in the other community that focused on other aspects of the problem, and vice versa.

References

1. Camarinha-Matos L.M., Afsarmanesh H., Ollus M. ECOLEAD and CNO base concepts. Methods and tools for collaborative networked organizations. Springer, New York, 2008.
2. Rabelo R., Gusmeroli S. The ECOLEAD collaborative business infrastructure for networked organizations. Pervasive collaborative networks PRO-VE 2008. Springer, New York, 2008.
3. OASIS, reference model for Service Oriented Architecture 1.0. OASIS Standard, 2006.
4. Camarinha-Matos L.M., Afsarmanesh H., Galeano N. Characterization of key components, features, and operating principles of the Virtual Breeding Environment. ECOLEAD deliverable 21.1, 2004.
5. Camarinha-Matos L.M., Afsarmanesh H. Elements of a base VE infrastructure. J. Comp. Ind. 51, 2, 139–163, 2003.
6. Martin D., Burstein M., Lassila O., Paolucci M., Payne T., McIlraith S. Describing Web Services using OWL-S and WSDL. 2005, `http://www.daml.org/services/owl-s/1.1/owl-s-wsdl.html`.
7. Martin D., Burstein M., Hobbs J., McDermott D., Narayanan S., Paolucci M., Parsia B., Payne T., Sirin E., Srinivasan N., Sycara K. OWL-S: semantic markup for Web Services. 2004, `http://www.daml.org/services/owl-s/1.1/overview/`.
8. WSMO. 2005, `http://www.w3.org/Submission/2005/06/`.
9. Ermilova E., Afsarmanesh H. Competency modeling targeted on promotion of organization towards VO involvement. Pervasive collaborative networks PRO-VE 2008. Springer, New York, 2008.
10. TOGAF version 9. Enterprise Edition. The Open Group, 2009, `http://www.opengroup.org/togaf/`.
11. Baukadi K., Vincent L., Burlat P. Modeling adaptable business service for Enterprise Collaboration. Leveraging knowledge for innovation in collaborative networks PRO-VE 2009. Springer, New York, 2009.
12. Picard W., Paszkiewicz Z. Modeling virtual organization architecture with the virtual organization breeding methodology. Leveraging knowledge for innovation in collaborative networks PRO-VE 2009. Springer, New York, 2009.
13. OASIS Reference Architecture Foundation for Service Oriented Architecture 1.0. Committee Draft 02, OASIS, 2009, `http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-cd-02.pdf`.

14. Świerzowicz J., Picard W. Social requirements for virtual organization breeding environments. Leveraging knowledge for innovation in collaborative networks PRO-VE 2009. Springer, New York, 2009.
15. Picard W. Social protocols for agile virtual teams. Leveraging knowledge for innovation in collaborative networks PRO-VE 2009. Springer, New York, 2009.
16. Hanneman R.A. Introduction to social network methods. University of California, Riverside, 2001, `http://faculty.ucr.edu/ha006Eneman`.
17. Wasserman S., Faust K. Social networks analysis: methods and applications. Cambridge University Press, 1994.
18. Picard W. Computer support for agile human-to-human interactions with social protocols. Business Information Systems BIS 2009. Springer, New York, 2009.
19. Paszkiewicz Z., Picard W. Reference model for performance management in service-oriented virtual organization breeding environments. Leveraging knowledge for innovation in collaborative networks PRO-VE 2009. Springer, New York, 2009.
20. The Java Scripting Project home page. 2006, `https://scripting.dev.java.net/`.

Service-Oriented Architecture for Public Administration Organization

Sergiusz Strykowski, Rafał Wojciechowski, Jan Świerzowicz, and Paweł Baszuro

Department of Information Technology
Poznań University of Economics,
Mansfelda 4, 60-854 Poznań, Poland
{strykow,rawojc,jswierz,baszuro}@kti.ue.poznan.pl

Abstract. Electronic economy requires public administration to apply information technology for servicing its clients — businesses and citizens. To do that public administration must base its organization on an architecture that enables collaboration and integration of heterogonous IT systems, which are the norm when considering public agencies as a whole. The approach to public administration organization based on Service-Oriented Architecture is presented in this chapter.

Keywords: public administration, administrative process, e-Government, Service-Oriented Architecture

1 Introduction

Nowadays, a rapid process of expanding the scope of electronic economy can be observed worldwide. Electronic economy is based on making the use of information technology to conduct business processes. At the operational level, electronic economy involves two fundamental changes. First, replacing paper documents with electronic ones. Second, replacing face-to-face meetings and interactions with electronic communications conducted through the net, especially the Internet. These two changes are the foundation of four main characteristics of electronic economy: shortening transaction times, reducing costs, independence of geographical distances separating business partners, and an opportunity of automatic reaction to a signal initiating a business process which opens a possibility of mass-personalization. Due to these characteristics, electronic economy gains the great advantage over traditional economy based on paper documents and face-to-face meetings and interactions. As a consequence, capital is invested more willingly in

cities, regions and countries which support electronic economy and create conditions for its operation and development.

The key role in these supporting activities is played by the public administration, both at the national and the local level. This role follows from the fact that public administration is a necessary participant of many citizen's and industry's business processes [1]. Effectiveness of public administration has a direct impact on citizens' quality of life and businesses' effectiveness. It is quite important for public administration to keep up with citizens and businesses on information technology usage. Public administration should not force citizens and businesses that use information technology on a daily basis to use paper and face-to-face meetings when contacting public administration offices and agencies. Due to social role of public administration, the situation ought to be even opposite: public administration should provide the broad range of electronic services to encourage, stimulate, educate, and break down sociological barriers of extensive usage of information technology. If people learn how to use information technology by interacting with public administration and see advantages of this usage, they begin to use this technology also in their ever-day life and business and that will contribute to the development of a given city, region and even the whole country.

Electronic services delivered by public administration must be able to collaborate with the broad range of heterogonous public registries and back office systems, both at the national and local levels. Therefore, for successful implementation of these services, an architecture that enables integration of heterogeneous systems must be applied. This requirement is fulfilled by Service-Oriented Architecture.

In this chapter, an approach to public administration organization based on Service-Oriented Architecture is presented and discussed. Section 2 presents the fundamentals of electronic public administration. Section 3 explains how Service-Oriented Architecture can be used for public administration organization. The reference public administration organization architecture is proposed and explained; also guidelines for public administration systems and administrative processes implementations are presented. Section 4 concludes the chapter.

2 Electronic Public Administration

Traditional organization of public administration follows the following approach: public administration is organized the way to be able to store, send and process documents in a paper form. These document include applications, decisions, rulings, certifications, forms, etc. Paper documents have several significant shortcomings: storage of paper documents is space consuming; delivery requires physical realization which is slow and costly; but the most important drawback is that paper documents must be processed by humans — processing of paper documents cannot be shifted to computers and automated.

The situation turns quite the opposite in the case of electronic documents, which can be stored, transferred and processed by IT systems. One of the main characteristics of electronic documents is geographical independence. Each electronic document stored in a computer database can be shared with any authorized person having access to this database via network. Transfer of electronic documents is significantly faster and less costly than paper ones, regardless of the number of recipients.

Opposite to paper documents, it is not necessary to "have" electronic documents physically to get access to information included in them. Computer networks make it irrelevant if a server storing electronic documents is placed under public servant's desk or in a data center located hundreds kilometers away. The fundamental consequence of document dematerialization is the separation of two main functions performed by public servants in relation to documents: the archiving function and the decision-making function. Public servants released from archiving function; i.e., released from cabinets, binders and folders can focus on decision-making function, which is directly related to citizens and businesses. There are here such issues like recognition of citizens and businesses problems, establishing and sustaining contact with them to help them to solve their problems, creating politics, and making decisions.

The very important characteristic of electronic documents is a possibility of automated processing, i.e., classification, matching, comparing, sorting, searching and analyzing. Automated processing enables to release public servants from the next unproductive task: the routine, manual processing of paper documents within administrative processes and hand over this task to IT systems. Implementing such a change requires creating models of those processes for steering IT systems on document's content operations.

A process model is a sequence of activities which must be preformed to do a given work and therefore achieve a certain goal. There are two main levels of process modeling: a conceptual level and an execution level. The conceptual level aims at representing processes from the managerial point of view. Models created at this level should present activities which must be undertaken to reach a goal specified in a managerial plan. In many cases, especially in a case of processes composed of activities conducted by humans, modeling at the conceptual level is sufficient. Models at this level are meant to be a kind of instruction how to do a certain work. However, if the process or a part of it are intended to be automated and performed by an IT system, the model of such process must be transformed from the conceptual level to the execution level. Models at the execution level present a sequence of technical operations which must be performed by IT systems to conduct a process.

Currently, when building models of public administrative processes, it is taken the same approach as in a case of modeling business processes performed in commercial companies. This approach is based on creating models according to information gained from operational practices. Meanwhile, the form of administrative

processes results from legal rules and regulations which are the primary sources defining the way public administration operates.

In a typical situation the course of a specific administrative process is influenced by several legal acts, which can be divided into two main groups. The first group consists of general acts; for example, administrative process code which regulates general rules, common for all administrative processes regardless of the nature and essence of a case the given process applies to. These general acts are superior ones to acts of the second group. The acts of the second group have much more detailed character and regulate processing rules of a specific class of cases. For example, building code regulates details of granting building permissions. Additionally, if the administrative process was modeled based on these two groups of legal acts only, it would not include options which are correct from the legal point of view but which are regulated by legal acts not directly related either to administrative processes as a whole or the specific class of cases. For example, the civil code regulating the issue of granting a power of attorney. Creating models of administrative processes requires thus taking approach based on legal acts and not on operational practices.

To summarized, advantages of public administration organized around electronic documents results directly from characteristics of digital information. Therefore successful implementation of this approach requires a proper technical architecture. Most of public agencies are already running IT systems. In the case of electronic documents and electronically conducted administrative processes there must be a common, efficient collaboration among different IT systems of public administration. Unfortunately, the significant obstacle hindering the introduction of the collaboration is the low level of interoperability between public administration IT systems. The problem results mostly from heterogeneity of the systems, which were developed by various software houses at different time over the period of last forty years, so they are built in different, mutually incompatible technologies [2]. The interoperability of heterogeneous IT systems running in public administration can be achieved by application of Service-Oriented Architecture and its principles at both technical and managerial levels.

3 SOA-Based Approach to Public Administration Organization

3.1 Service-Oriented Architecture Overview

Service-Oriented Architecture (SOA) is a paradigm of design and development of solutions composed of services. These solutions embrace a whole organization and support it to achieve its business goals [3].

A *service* is a separate and autonomous business logic unit, which has a specific functional context and offers a set of functional features associated with that context. The functional features offered by a service are called *service capabilities*. A service can therefore be considered as a container for capabilities within a specific range. The capabilities of a service are independent of the technical implementation of the service. Principles and methods of access to a service and offered capabilities are specified in the *service contract*. Services are independently designed and implemented, and therefore can be repeatedly used in different *service compositions* comprising different solutions.

A *service composition* is an ordered combination of invocations of specific services with the aim of automating the execution of a task or a business process. The composition must consist of at least two constituent services and an initiator of the composition, which arranges and coordinates the invocations of the constituent services. Otherwise, a point-point interaction between services merely occurs.

A *service repository* is a collection of services working within an organization. The services are arranged and governed according to a *policy* established by the organization. Typically, an organization operates one service repository. When an organization operates more than one repository, the individual repositories are called domain *service repositories*. The primary objective of the service repository is to ensure that competences offered by services available in the organization are complementary and disjunctive to each other, and it is possible to establish seamless interoperability between the services. *Interoperability*, in turn, translates into the possibility of building a variety of service compositions in response to changing demands and new managerial requirements.

Service-Oriented Architecture Design Principles. The service-oriented architecture is founded on eight main design principles [4]. Each of these principles refers to different aspects of the design process and ensures that certain characteristics of service orientation are equally present in each service. The common application of all these principles causes a solution to conform to service-oriented architecture paradigm. Such a solution can be classified as "service-oriented".

Service-oriented architecture principles are as follows:

— *Standardized Service Contract* — contract of each service in a given service repository must conform to established contract design standards. Systems of public administration should provide interfaces compatible with open standards. All newly developed systems should have such interfaces, while legacy systems should be equipped with special adapters, which makes it possible to offer their functionality in a standardized form.

— *Service Loose Coupling* — service contracts enforce the requirement of clients weak coupling, and the contracts themselves are not dependent on any external entities. Loose coupling increases independence of requestors from services, so changes to business logic of the services have no impact on the requestors.

Public administration systems may require changes due to changes in law and objectives of citizens and businesses. The requestors can be other public administration systems as well as the systems of businesses and citizens.

— *Service Abstraction* — service contracts include only significant information about a given service. The implementation details of business logic are hidden within a service and are unavailable to requestors. The requestors have only access to the service contract that allow them to easily understand the service functionality.

— *Service Reusability* — services include independent application logic and therefore can be considered as reusable resources. In public administration, a number of services that perform basic tasks can be reused within various administrative processes. Public administration operates under common law, and therefore the same processes can be carried out by a number of public administration agencies. Thus, different agencies may require similar functionality from electronic systems, and as a result services may be reused by a number of public administration agencies.

— *Service Autonomy* — services are autonomous and able to control runtime environment. Due to service autonomy, each public administration agency can control the business logic directly related to the competence of this agency. While electronic systems may come under the administration of a specific agency, the services offered by these systems can be available to other agencies and can be used to create integrated services.

— *Service Statelessness* — services reduce their resource usage and restrict themselves to storing and processing state information only when truly necessary. Services offered by public administration systems should not be involved in processing the state information because the services can be intensively used and reused in a large number of different compositions.

— *Service Discoverability* — services are described with metadata and can be discovered and interpreted efficiently. The descriptions of the services offered by public authorities must be stored in the registry of public services, which allows discovering the services by other public administration agencies, as well as citizens and businesses.

— *Service Composability* — services can be arranged into various compositions regardless of the dimensions and the density of the given composition. Composite services allow the automation of entire administrative processes, as well as business processes and living processes of citizens.

Service orientation introduces a new way of thinking. In a traditional approach, applications were created as monolithic systems supporting a specific business need; in a service-oriented approach applications are created by connecting (orchestrating) services supporting basic (low-level) functionalities. Services can be reused in a number of applications because supporting different business needs often requires the same basic functionalities.

Service orientation should not be considered as software architecture only. The concept of a service can also be applied at the management level: it can be used when designing an organizational structure. Service orientation should be taken up as the fundamental principle for all other integration activities.

Service-Oriented Approach for Public Administration. The good suitability of the service-oriented architecture for implementing public administration IT systems follows from its immanent characteristics [5]:

— Service orientation and its special focus on the definition of interfaces make integration a primary issue. During the design phase, service interface definitions draw the same level of attention as databases and system logic. With integration built-in by design, data and functionality can be seamlessly shared between internal processes and external partners without the necessity to deploy costly and complex integration solutions.
— SOA paradigm assumes heterogeneous environment. This makes it particularly suited for public administration and for the collaboration of public administration entities with private ones. The primary infrastructure component here is an Enterprise Service Bus (ESB), which enables system integration in a diversified environment [6]. ESB enables the integration of a variety of legacy systems and components appearing quite often in public administration IT environments.
— Designing IT solutions in a form of shared and collaborating services which map directly to atomic units of administrative processes. The result is an IT architecture that directly reflects the organizational and management architecture. This remains in strong contrast to the traditional approach to designing IT systems where administrative rules and logic are implemented in a form of programming code, which makes them inaccessible to external software developers without knowledge on these systems.
— Collaborative and iterative project approach. Development of service-oriented solutions requires close collaboration between IT and other departments because of close relationships between services and units of administrative processes. Also the development process is an iterative approach (through services) rather than the monolithic one based on collecting requirements and then involving into long-running implementation projects.

Public administration agencies are autonomous, but they are required to collaborate. Under such circumstances service orientation seems to be perfectly suited for them. In a service-oriented architecture a particular agency can retain a high level of autonomy in terms of its internal functionality but at the same time it can deliver to the outside world a set of services, which can be used by other public administration agencies and businesses to create integrated services spanning across many entities, both public and private.

A service-oriented architecture enables information sharing both inside and outside the public administration; i.e., on the one hand among various public administration agencies, offices and departments, and on the other hand with external entities, i.e., businesses and citizens. The service-oriented architecture facilitates dynamic composition and orchestration of services delivered by different providers without spending public funds on huge, time-consuming, and risky projects of development new centrally integrated software systems. In this way, service-oriented architecture stimulates citizen centricity, improves time-to-market factor, and encourages development of electronic economy.

Enterprise Service Bus. An enterprise service bus (ESB) is defined as a middleware based on the exchange of messages, called Message Oriented Middleware — MOM, using open standards of web services, integrating applications, which are service containers, and providing services to enable this integration [7]. The most important architectural feature of ESB is a logical bus consisting of middleware services that implement the logic of integration, which consists of features such as message routing, data transformation, protocol conversion, security, management and monitoring services [8]. Applications are connected to the bus using *intelligent adapters*.

The intelligent adapters linking the applications with the ESB are responsible for the transformation of messages from the format specific to a given application to a common format used by the bus, and vice versa. The intelligent adapters used in the ESB architecture, as opposed to the lightweight adapters used in the hub-and-spoke architecture, offer an additional opportunity to implement part of the integration logic, which allows dynamic coupling of the applications.

The main difference between the ESB architecture and the architecture with a central hub is that the integration features are offered by the ESB services, which are physically distributed across the bus. The ESB services can be independently deployed in appropriate service containers. The service container model allows independent scalability of the various integration components and incorporating into the bus as required. The solutions built in accordance with the ESB architecture offer a much greater scalability and performance than the solutions with a central hub. However, the systems based on an ESB bus are more difficult to configure and to manage in comparison to the other solutions.

The standards-based integration is one of the foundations of the ESB architecture. The ESB bus is an open architecture, which may be extended by new elements based on open standards and components that use proprietary standards through the use of standardized interfaces.

There are a number of solutions on the market that offer the ESB functionality. These solutions are available either as open-source software (e.g., Apache ServiceMix, Mule ESB, GlassFish ESB, JBoss ESB) or as commercial distributions (e.g., Oracle Service Bus, IBM WebSphere ESB, Microsoft BizTalk ESB,

TIBCO ActiveMatrix Service Bus). The choice of a specific ESB product for a given public administration agency depends on many aspects, such as: number of systems to integrate, implementation technologies of legacy systems, size of the agency, number and complexity of tasks and administrative processes realized within the agency, and financial capabilities.

3.2 Service-Oriented Architecture for Public Administration

The characteristics and advantages of the SOA paradigm predestine it to be a basis for the construction of an electronic public administration system. Building such a system, however, entails the integration of multiple heterogeneous public administration systems, which have been built using different technologies and architectures. To build an integrated SOA-based public administration system, the existing local and national public administration systems must be transformed into services according to the SOA paradigm. Such a SOA-based electronic public administration system could be seen as a coherent system that would provide citizens and businesses with a single point of entry to carry out a whole variety of administrative processes.

Service-oriented architecture for public administration is presented in Figure 1.

The national public registries are used to store and manage data related to citizens and businesses at the national level. In industrialized countries almost all public registries are maintained as IT systems. Examples of national public registries are *Citizen Repository* or *Vehicles and Drivers Registry*. National public registry systems are integrated using the National ESB. Therefore, these systems can cooperate with each other and with the local public administration agency systems.

A local public administration agency system consists of the following components:

— Administrative Process Modeling Platform,
— Administrative Process Management Platform,
— Administrative Process Execution Engine,
— E-Government Portal,
— Public registries and Back office systems, and
— Local ESB.

Administrative Process Modeling Platform. The platform enables process analysts to model administrative processes in accordance with the guidelines described in Section 2. The administrative process models are stored in a process repository being an integral part of the platform. The platform offers an intuitive graphical user interface that makes it easy to use and understand for a process

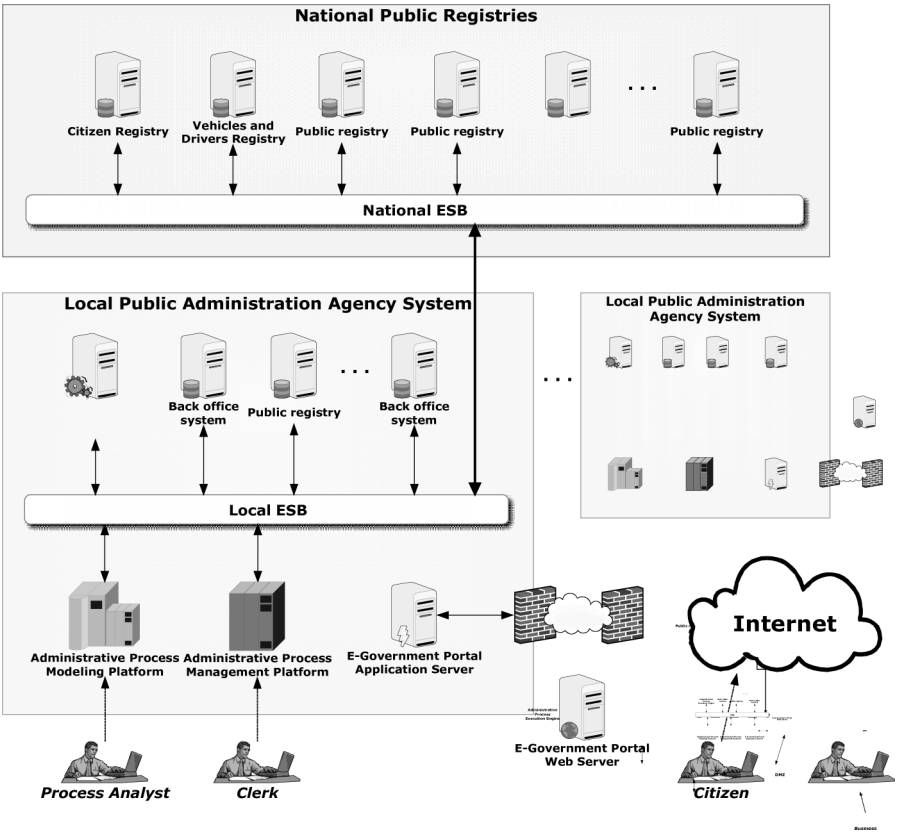

Fig. 1. Service-oriented architecture for public administration. The service-oriented architecture for public administration is composed of *National Public Registries* and a number of *Local Public Administration Agency Systems*

analyst who may not be an IT expert. Administrative processes are modeled at the conceptual level, for example, using BPMN or other suitable notation [9]. Such process models can be then converted into a format executable by an orchestration engine. The two most popular executable languages for describing business processes are BPEL and XPDL [10, 11]. The process models in an executable format can be executed by the Administrative Process Execution Engine.

Administrative Process Management Platform. The platform is used by public servants for managing the administrative processes, as well as to perform tasks associated with their operational activity. Process management involves monitoring the execution of administrative processes, initializing new processes, carry-

ing out human activities within processes, and terminating processes. An example of a human activity is making a decision on a case. In turn, the tasks related to an operational activity include viewing and modifying the data stored in public registries, if this data is not directly related to ongoing administrative processes.

Administrative Process Execution Engine. The process execution engine is a service responsible for performing instances of administrative process models, which are represented in an executable format. The executable models are generated based on the models at the conceptual level created by the Administrative Process Modeling Platform. While performing an administrative process instance, the service is involved in the orchestration of other services connected to the Local ESB, according to the process definition. The state of the process instances is stored in the execution engine.

E-Government Portal. The E-Government Portal allows citizens and businesses to submit an application for initiating an administrative process, monitor the process progress status and read the available data stored in public administration systems. From the perspective of citizens and businesses, the E-Government Portal is a website on the Internet where they can comprehensively and effectively carry out whole administrative processes.

The E-Government Portal consists of the *E-Government Portal Application Server* and the *E-Government Portal Web Server*. The E-Government Portal adopts a three-layer architecture, which consists of a data layer, a business logic layer and a presentation layer. The data layer consists of the data repositories of both local and central public administration systems. The E-Government Portal Application Server is included in the business logic layer, whereas the E-Government Portal Web Server is contained in the presentation layer.

The E-Government Portal Application Server is responsible for processing the business logic offered by electronic public administration to citizens and businesses. The application server is connected to the local ESB bus through which it contacts the internal systems of public administration. These systems include, in particular, the Administrative Process Execution Engine, but also public registries and other back office systems of the public administration. The E-Government Portal Web Server is responsible for building a Web-based user interface for the citizens and businesses which use the E-Government Portal.

One of the key issues that must be taken into consideration during the design of an electronic public administration is security of the data stored in its internal systems. Therefore, a local public administration agency system is separated from the Internet by a demilitarized zone (DMZ) [12]. A demilitarized zone is a network located at the border between the local public administration agency system and the Internet. In the proposed architecture, the E-Government Portal Web Server is located in such a demilitarized zone, because it is accessed by users through

the Internet, and thus, is especially vulnerable to attacks. The E-Government Portal Web Server does not contain any business logic essential to processing the requests of citizens and businesses. The whole business logic is contained in the E-Government Portal Application Server, which is an internal component of the local public administration agency system and is separated from the Internet by at least two firewalls, as presented in Fig. 1.

Public Registries and Back Office Systems. A public registry is a collection of data on citizens and businesses aimed for accomplishing public tasks, gathered by public administration agencies according to specific legal regulations. The scope of data stored in the public registry, the type of available operations on it and the access procedures are strictly regulated by appropriate legal acts. The example of a public registry is the Civil Registry used to record information on citizens' vital events. Back office systems are responsible for the processing of data required to perform the steps of administrative processes.

Local ESB. The features of the ESB predestine it to be used as the backbone of local public administration agency systems. The internal systems comprising the local public administration agency system are integrated by the Local ESB, which enables the interoperability between the local systems, and between these systems and the national public registries. In particular, the Administrative Process Execution Engine through the Local ESB can have access to the data stored in local public registries, local back office systems, and the national public registries.

System Usage Scenario. The exemplary system usage scenario is as follows. A citizen wants to do some business with a local administration agency; e.g., he wants to request a renewal of his driver license. Using a browser of his choice, the citizen logs in at the E-Government Portal Web Server. The citizen authenticates himself with his certified digital signature. After successful login, the citizen chooses "Driver License Renewal" from a list of on-line services offered by the local administration agency. The citizen is presented with the detail description of the driver license renewal process, especially with the cost and time frame. Citizen clicks on the Start button to confirm his choice and start the renewal process. The process is performed according the model created earlier by a process analyst on the Administrative Process Modeling Platform. After creating, the model was uploaded on the Administrative Process Execution Engine. For each case, the Process Execution Engine starts a new process instance. In the case being discussed, this is a new instance of the driver license renewal process. The first activities in the process model consist of collecting all data needed by a citizen to form an application. That includes citizen's personal data and data on his driver license which he wants to renew. Including this data in an application is required by the law. In

the traditional approach, it is the citizen duty to collect all necessary data and include it in a paper application. In electronic public administration, collecting and including activities can be performed automatically by a software system. Citizen personal data is stored in the Citizen Registry, and data on citizen driver licenses is stored in the Vehicles and Drivers Registry; both registries are nationwide. Both registries expose its functionality as services; the similar approach can be applied for exposing functionality of other national registries. The services exposed by registries are registered in the National Enterprise Service Bus. Therefore, the Process Execution Engine does not have to know any details about the location of those services; it just sends a request to the Local ESB, which routes it to the National ESB, which routes it to the appropriate service. The data returned by the services of the Citizen Registry and Vehicles and Drivers Registry is forwarded by the Process Execution Engine to the E-Government Portal Application Server to fulfill the application for the driver license renewal. The citizen is presented with the application completed with all necessary data. Everything what he must do is to digitally sign the application. After signing, the application is stored in a repository of a Back office system. Next, the Process Execution Engine presents the application to a public clerk who must make a decision; from a legal point of view a driver license renewal is granted as an administrative decision. The application is presented to the clerk through the Administrative Process Management Platform. If there are no obstacles, the decision is positive and the clerk clicks the Accept button. The Process Execution Engine sends a request to the E-Government Portal Application Server to generate the decision document. The Process Execution Engine takes the document and sends it to the Process Management Platform, which presents it to the clerk. The clerk signs the document with his certified digital signature. The Process Execution Engine gets the signed document and sends one copy to a Back office system for storage, and the other copy to the E-Government Portal Application Server where it can be reached by the citizen. The Process Execution Engine sends a request to the appropriate service of the Certification Documents Manufacturing House to manufacture a new driver license. The request routes through the Local ESB to the National ESB and further through the extranet to the service of Manufacturing House. The renewal process waits until a new driver license is manufactured and sent back to the local administration agency. It should take no longer than two weeks but unfortunately delays happen quite often. When the agency receives the driver license, its employee marks this fact in the Process Management Platform. The Process Execution Engine notifies the E-Government Portal Application Server to send email to the citizen for which the driver license is waiting to be collected. When the citizen picks up his new driver license, the agency employee marks this fact in the Process Management Platform. The Process Execution Engine stores information about collecting the driver license in a Back office system and concludes the process.

3.3 Guidelines for Public Administration System Implementation

To incorporate a public administration system into the service-oriented architecture, the system must expose its functionality as a service interface through which other systems may interact. However, most of legacy systems functioning in public administration are devoid of such an interface. Those systems must be extended by adapters enabling interoperability with other public administration systems.

In the SOA-based public administration the component systems must be designed and implemented in accordance with specific functional and quality requirements. Furthermore, to ensure compliance with the SOA paradigm, the systems should be compatible with a reference architecture proposed in this section.

Functional Requirements. Functional requirements include the specification of the following issues: input, output, data format, communication interfaces, external software and hardware, user tasks, and data used in each task. In the case of public registries, all the service interfaces are determined by legal acts which implies that input, output and data format are explicitly imposed. In the SOA-based approach, communication interfaces accompany with SOA standards, i.e., WSDL, SOAP and XML Schema. Software is built as a service provider, so an application server and a database management server are used during its development.

Services contain functions which are human-oriented tasks. These tasks fulfill the common functionality needed by citizens and businesses. The same applies to the data used in communication with external systems.

Quality Requirements. Quality requirements for SOA-based systems are similar to those for distributed systems [13]: high availability, reliability, stability of interface, and security.

High availability and reliability requirements have both very similar and tightly coupled features and system principles. First of all, communication issues are one of the biggest concerns of SOA. The reference system must be prepared for communication delays and breakdown. Delays might drive execution processes to timeout, breakdown, or to the indeterminate state. Execution should retry to communicate with an external service, and not revert the entire process. It is important to have the right coordination of the execution to restrict the concurrency to limited pool of resources. A good SOA system should be error resistant, and fault tolerance. Everyday use by a high numbers of users should be made easier by incorporating management and monitoring tools. First, to give the possibility to reconfigure the system at runtime, and second, to report what happens with the system at any moment.

A different problem is a runtime data synchronization which might drive system resources requirement to excessive levels. In many cases bypassing a message, with

a current value to a different service using a pull or push strategy might be easier to implement using less demanding systems.

More legal-depend than technical requirements might be features like load and information distribution. Systems should be designed with information distributed across services and sites from an early beginning. Thus new rules incorporating SOA systems should avoid information redundancy, which is hard to maintain across the sites. Different aspect of designing system is to prepare for a load distribution. System at some point might be assigned to a different business process and new groups of users might try to use the system resources. Loose coupling, separating the data and services from interfaces allows the system to be reused under different conditions.

A service interface should be kept as stable as possible, because each interface change drives the service consumers to an incompatibility which ends with a communication breakdown. Therefore, at first systems should be designed very carefully, concentrating on a both process and data stage.

Security should imply four different stages [14]: authentication — where system checks for authenticity of a request; authorization — system performs the mapping of services and operations to the requesting user; auditing — detection of the possible actions on the SOA items; and assurance — system wide features to support general security considerations (also bugs and mistakes).

Reference Architecture. The reference architecture of public administration systems consists of a service provider and a service consumer. A provider offers Web services, whereas a consumer uses them [15]. In a common scenario, an application server publishes Web services using WSDL; Web services offer Create, Read, Update, and Delete (CRUD) operations on backend database storage; Web services are called by the client using SOAP and WSDL protocols.

A system designer should follow the same principles as in the object-oriented programming:

— Abstraction — to avoid technology specific data and use more business specific data structures,
— Encapsulation — to hide case specific data into meaningful complex data types (described in an external XML Schema),
— Inheritance — to reuse data definitions from XML Schema,
— Polymorphism — to reuse interfaces described in WSDL documents for different implementation purposes.

A service provider site is divided into three layers: a data storage layer, a data access layer, and a business logic layer. Relational Database Management System (RDBMS) should be chosen as the data storage layer. The database schema contains common elements: data indicating a person in the system, data indicating its mailing and electronic address, data specific to system purposes. The data

access layer specifies all components used to access the RDBMS and serves all the CRUD operations on data tables. The components should support an object-relational mapping. The business logic layer includes access requests to the other layers, conversion to business objects and implements the interface of Web services.

Equivalent to the service provider, the service consumer is also composed of three layers: a user interface, a Web services management layer, and a Web services communication layer. The user interface layer is responsible for showing data and manipulating controls to the user, as well as error handling. The user interface layer communicates with the Web services management layer, which is responsible for managing the calls to the concrete Web services. The Web services communication layer contains proxy classes for Web service calls.

3.4 Guidelines for Administrative Processes Implementation

Administrative processes associated with citizens and businesses are implemented as invocations of services offered by various public administration systems. To successfully execute those processes, the service invocations must be accordingly coordinated. The most widely used method of coordination is orchestration, in which there is a process acting as a coordinator [16]. A coordinator process controls the services offered by other systems and coordinates the invocations of their operations. The orchestration is centralized, since only the coordinator process has the knowledge of the component services and the order of their invocations.

The lifecycle of an administrative process is as follows. First, a process analysts use dedicated tools to graphically design a *conceptual model* of the process. The conceptual model is a diagram that can be appropriately annotated to allow simulation and analysis but which lacks the technical details required for execution. Afterwards, in a design environment, process designers import the conceptual model and convert it into an executable one by linking it with the components built by developers. That executable model can be deployed from the design environment to an execution engine to automate and manage the administrative process. The language for describing executable processes depends on the execution engine.

Executable Process Languages. In process execution engines based on the well-established reference model of the Workflow Management Coalition (WfMC), process models are encoded in XPDL. XPDL provides a file format that support every aspect of the Business Process Modeling Notation (BPMN) notation including graphical descriptions of the diagram, as well as executable properties used at run time. With XPDL, a product can export a process definition with full fidelity, and another product can import it and reproduce the same diagram that was sent.

Processes modeled along with the SOA-based approach are executed using Business Process Execution Language from OASIS. BPEL is a de facto standard

for implementing business processes on top of Web services technology. Numerous platforms (such as Oracle BPEL Process Manager, IBM WebSphere Application Server Enterprise, IBM WebSphere Studio Application Developer Integration Edition, and Microsoft BizTalk Server) support the execution of BPEL processes [17].

The key objective of BPEL is to standardize the format of process flow definitions so public administration agencies can work together seamlessly using Web service technology. BPEL extends the Web services interaction model and enables it to support transactions. The BPEL process model is layered on the top of the service model defined by WSDL. It is the notion of peer-to-peer interaction between services described in WSDL.

To define how processes should be executed, BPEL has XML definitions or commands that specify the order of operations, the looping of operations, and synchronous/asynchronous requirements for operations. BPEL also has commands to take care of fault conditions and commands to undo or reverse operations [18]. Basic BPEL XML elements are as follows: Partners — actors in a transaction, Containers — transmitted messages, Operations — required Web services, Port types — kinds of Web services connections required for operations.

BPEL is used for creating XML-based descriptions of processes based on the interactions between the process and its partners. It is based on Web services in the sense that each of the process involved is assumed to be implemented as a Web service. Processes written in BPEL can orchestrate interactions between Web services using XML documents in a standardized manner. These processes can be executed on any platform or product that complies with the BPEL specification.

Despite having "execution language" in its name, BPEL has features of a design language. In BPEL-based design environments, some diagram elements that correspond with elementary BPEL activities. In the real world, BPEL is often used as a design language by process designers technical enough to use it.

Executable Process Modeling Principles. There are ten basic principles that should be followed during modeling executable processes [19]:

1. Web services as the base: model a process that communicates with external entities through Web services (defined using WSDL) and that manifest itself as a Web service (defined using WSDL). BPEL processes are inseparably connected with Web Services, thus during creation of information system that would use Web services to interface itself with other partners, especially if one wants to build BPEL processes that use the system.
2. XML as the form: although there are many graphical editors (e.g., BPEL Designer, ActiveVOS, JDeveloper) that support BPEL process creation; it is worth remembering that process is defined using an XML-based language. Graphical representation of process enables designers to detect bottlenecks and loops by themselves by looking at the model. It is recommended to use

graphical editors for they make designing processes easier and models "readable". However, how the process is represented graphically has no meaning in terms of execution, thus no particular design methodology for processes is needed.

3. Common set of core concepts: a set of Web service orchestration concepts for external (abstract) and internal (executable) views of a process must be defined. WS-BPEL specification introduces a notion of compatibility between an under-specified abstract process and a fully specified executable one. Basically, this compatibility notion defines a set of syntactical rules that can be augmented or restricted by profiles(the Abstract Process Profile for Observable Behavior and the Abstract Process Profile for Templates). Besides common concepts, it is recognized that both abstract and executable view require a few specialized extensions such as conformance checking that link the two usage patterns.
4. Control behavior: hierarchical and graph-like control regimes need to be introduced. This concept stems from two BPEL predecessors: XLANG and WSFL. Both of these languages were XML based languages of Web services' composition. XLANG followed hierarchical structure while WSFL represented graph structure. Each of the languages had its advantages and flaws over one another. WSFL was strong on model presentation while XLANG did well with long-running interaction of Web Services. While WSFL complements WSDL and is transition-based, XLANG is an extension of WSDL and block structured-based. In BPEL both hierarchical and graph-like control regimes are merged together. As a result of this approach the fragmentation of the process modeling space is reduced.
5. Data handling: since data manipulation functions in BPEL are just powerful enough to perform only the simple tasks necessary in defining process models including very simple message construction and data expressions associated with control. Complex operations should be performed by the Web services invoked by the process. In order to optimize BPEL process all the complicated computations should be done "outside"; i.e., there should be a set of Web services providing outcome that needs no further or very little processing by the BPEL engine.
6. Properties and correlation: BPEL supports an identification mechanism for process instances that allows the definition of instance identifiers at the application message level. Instance identifiers should be partner defined and may change over time. BPEL creators thought that creating artificial process instance identifier is not a good idea. They have proposed a mechanism that allow users to create self-defined identifiers which are kept in WSDL abstract message properties. the best way to keep identifiers would be to embed them into WSDL abstract messages.
7. Lifecycle: in BPEL no explicit distinction between stateless and stateful services or processes is needed. Furthermore, BPEL mechanisms ensure implicit

lifecycle management of a process, i.e. process instance is automatically created when message is sent to appropriately annotated receiving operation of service, and deleted when control reaches terminal activity.

8. Long-running transaction model: it is highly recommended to use mechanism to support failure recovery should anything go wrong in BPEL process. Mechanisms such as scope and compensation help in doing so. Scopes are blocks of code in BPEL that may have a compensation handler associated with them. These scopes, and their associated compensation handlers, can be nested to an arbitrary depth. When scope is successfully completed the compensation handler and the current state of the process are saved for possible later invocation. Should Invocation of a compensation handler can only be done from within one of BPEL's fault (exception) handlers, and when actually invoked, the compensation handler is run on its associated saved state.
9. Modularization: discover subprocesses in the process and run them as Web services. In that way processes may be easily decomposed and assembled. One of the biggest advantages of BPEL processes is their reusability. It is a very good practice to divide process into rational subprocesses. Having in mind abovementioned rule 1 (Web service as a base) running BPEL processes by another process comes down just to invoking one Web service. With this mechanism one can easily build more complex processes just by combining existing more basic subprocesses.
10. Composition with other Web services functionality, One must also remember that once process is modeled correctly it can be further reused by anyone. That is why it is important that processes should be designed on Web services standards (approved and proposed) as much as possible in a composable and modular manner (with respect to the previous rule: Modularization).

4 Conclusions

Public administration organization based on Service-Oriented Architecture, presented in this chapter, ensures the unified access to heterogonous public registries and back office systems. Due to SOA, public administration can provide electronic services drawing information from any internal IT systems, both at the national and local levels. Furthermore, these services can be used as building blocks to create and offer integrated services handling comprehensive needs of citizens and businesses. Integrated services may include partial services delivered not only by the public administration, but also by entities originating from the private sector.

SOA-based public administration organization ensures civil servants to have direct access to information stored in public registries and back office systems. As the result, they can conduct administrative processes more efficiently. They do not

have to ask either clients or other public agencies to provide them with necessary information on paper; they can simply reach all necessary information themselves, of course according to all legal regulations they are bound upon.

In the approach presented in this chapter it is also assumed that administrative processes models are derived from legal rules and regulations. The models originating from law as the primary source of public administration operations are more accurate than models pursuant to operational practices.

References

1. Cellary W., Strykowski S. e-Government based on cloud computing and Service-Oriented Architecture. T. Janowski, J. Davies (eds.). Proc. 3rd Int. Conf. Theory and Practice of Electronic Governance ICEGOV 2009, Bogota (Colombia), ACM Int. Conf. Proc. Ser. ACM Press, 5–10, 2009.
2. Binildas C.A. Service oriented Java business integration: enterprise service bus integration solutions for Java developers. Packt Publishing, Birmingham, 2008.
3. Erl T. Service-Oriented Architecture: concepts, technology, and design. Prentice Hall PTR, Boston, 2005.
4. Erl T. et al. SOA Design Patterns. Prentice Hall, Boston, 2009.
5. Leganza G. Why is SOA hot in government? Forrester Research, Cambridge, 2006.
6. Davies J., Schorow D., Ray S., Rieber D. The definitive guide to SOA: oracle service bus. Apress, Berkeley, 2008.
7. Chappell D.A. Enterprise Service Bus. O'Reilly Media, Sebastopol, 2004.
8. Rademakers T., Dirksen J. Open-Source ESBs in Action. Manning Publications, Greenwich, 2008.
9. Business Process Modeling Notation. `http://www.bpmn.org/`.
10. Web Services Business Process Execution Language Version 2.0. OASIS Standard, `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`.
11. XML Process Definition Language (XPDL). `http://www.wfmc.org/xpdl.html`.
12. Kanneganti R., Chodavarapu P. SOA Security. Manning Publications, Greenwich, 2008.
13. Brown P.C. Implementing SOA, Total Architecture in Practise. Addison-Wesley, New Jersey, 2008.
14. Peterson G. Security in SOA — It's the Car, Not the Garage. SOA Mag. 15, 2008.
15. Connolly T.M., Begg C.E. Database Systems: A Practical Approach to Design, Implementation and Management. Addison-Wesley, Harlow, 2002.
16. Juric M.B., Mathew B., Sarang P. Business process execution language for Web services. Packt Publishing, Birmingham, 2008.

17. Vasiliev Y. SOA and WS-BPEL: Composing Service-Oriented Architecture Solutions with PHP and Open-Source ActiveBPEL. Packt Publishing, Birmingham, 2007.
18. Matis, J. Explainer: BPEL. (Computerworld, 30/05/2005) `http://www.computerworld.com.au/article/134827/explainer_bpel/?relcomp=1`.
19. Leymann, F., Roller, D., Thatte, S. Goals of the BPEL4WS Specification. `http://xml.coverpages.org/BPEL4WS-DesignGoals.pdf`.

A Revision of the SOA Paradigm from the e-Business Process Perspective

Stanisław Ambroszkiewicz, Waldemar Bartyna, Mirosław Barański, Krzysztof Cetnarowicz, Marek Faderewski, Dariusz Mikułowski, Marek Pilski, Marcin Stepniak, and Grzegorz Terlikowski

Institute of Computer Science Polish Academy of Sciences,
Al. Ordona 21, 01-237 Warsaw, Poland
Institute of Computer Science
University of Podlasie,
08-110 Siedlce, Poland
sambrosz@ipipan.waw.pl
http://www.ipipan.waw.pl

Abstract. The classic version of the SOA paradigm has its origins in software engineering. From the point of view of e-business processes (that are supposed to be also software applications), the concept of service need not be related to RPC (as it is in SOA) where a service is passive and is waiting for a client to be invoked. In other words, a service may be active and looking by itself for clients' tasks that can be accomplished. This corresponds to the reverse auctions in business practice. On the other hand, a business service has well founded structure where its operations (corresponding to request-quote, order-contract, invoice-payment) are related to each other. These relations cannot be expressed in WSDL. Hence, the concept of service in SOA should be discussed. A discussion on a possible revision of the SOA paradigm related to the above issues is presented in this chapter.

Keywords: business process, service arrangement, service composition, ontology, task execution

1 Introduction

The classical version of the SOA paradigm, see [1] (Fig. 1), may be summarized as follows. "SOA provides a standard programming model that allows self-contained, modular software components residing on any network to be published, discovered, and invoked by each other as services. There are essentially three components of SOA: Service Provider, Service Requester (or Client), and Service Registry. The

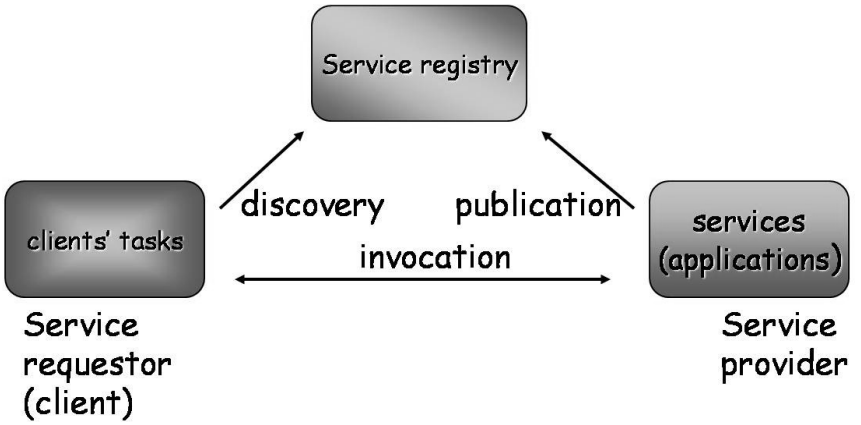

Fig. 1. The classical version of the SOA paradigm

provider hosts the service and controls access to it, and is responsible for publishing a description of its service to a service registry. The requester (client) is a software component in search of a component to invoke in order to realize a request. The service registry is a central repository that facilitates service discovery by the requesters".

The key point of the SOA paradigm is service integration, that is, "...other applications (and other services) can discover and invoke the deployed service..." In order to realize this vision, simple ubiquitous and automatic process modeling techniques are needed. The service integration is interpreted in several ways, e.g., as service composition, service orchestration, service choreography, process modeling, etc. These terms have been widely used to describe business interaction protocols. Generally, business processes comprise collaborative services that can provide much more complex functionalities, than the single services [2].

One of the classical definitions of a business process is following [3] (1993):

"It is a structured, measured set of activities designed to produce a specific output for a particular customer or market. It implies a strong emphasis on how work is done within an organization, in contrast to a product focus emphasis on what. A process is thus a specific ordering of work activities across time and space, with a beginning and an end, and clearly defined inputs and outputs: a structure for action. ... Taking a process approach implies adopting the customers' point of view. Processes are the structure by which an organization does what is necessary to produce value for its customers".

More detailed definition of electronic business process may be the one that can be found in IST CONTRACT Project [4]: "A business process specifies the potential execution order of operations from a collection of Web Services, the data shared between these Web Services, which partners are involved and how they

are involved in the business process, joint exception handling for collections of Web Services, and other issues involving how multiple services and organizations participate. In other words it defines the composition of Web Services in order to provide higher functionalities".

The terms *higher functionalities* and *customer's point of view* from the above definitions may be interpreted as a client (customer) task the business process is supposed to accomplish.

Although languages and technologies such as WSDL, BPEL4WS and OWL-S seem to be adequate for modeling and execution of the electronic business processes, automated process composition is still a challenge in IT.

The notion of *e-business process* (as the main point of the discussion) is strongly related to the several crucial issues, such that the SOA paradigm, and the concept of e-business service in SOA. The classic version of the SOA paradigm has its origins in software engineering. Service is viewed as server application (in the client-server model)waiting to be invoked by clients. From the point of view of e-business process, a service may be also active and looking by itself for clients' tasks that can be accomplished. This corresponds to the reverse auctions in business practice.

A business service has well founded structure where its operations (corresponding to request-quote, order-contract, invoice-payment) are related to each other. These relations cannot be expressed explicitly in WSDL. Hence, the concept of service in SOA should be discussed. Generally, from the e-business perspective the following interrelated problems seem to be important: *(1) Service architecture. (2) Communication protocols in e-business processes.*

According to the SOA paradigm, services and clients are engaged in several activities, i.e., publication, discovery and service invocation. Usually, the invocation is preceded by a negotiation for a service level agreement (SLA) that is (by its nature) "output" based, i.e. the result of the service, as required by the client, is the subject of the agreement. The SLA negotiations require semantic interoperability, that is, understanding between client and service provider. Actually all activities related to how a service is used by a client (also publication and discovery) require understanding between the client and the service provider. These activities must be realized in the form of communication protocols as it is usually done in distributed systems. Since the system is supposed to be open and heterogeneous, the semantic interoperability cannot be hardcoded in the protocols. It must be incorporated in the contents of the protocol messages. The contents must be expressed in a generic declarative language. Client task as well as service description, and the output (final result) description of service performance must be expressed in this language. The important question is what this language is about? In other words, what is the grounding, i.e., the real(nor abstract model-theoretic) semantics of the language. In our approach the grounding is a class of XML documents that services process (input documents and output documents). Once the documents are processed, they have precisely defined impact in the real world. Since this "impact" is hard (perhaps not possible) to describe formally, the

proposed grounding is simple and sufficient to provide semantic interoperability. OWL was chosen as the language for describing XSD schemata of the documents processed by services. In our previous research projects the language was Entish, our own invention. The rest of the paper (except the following Section devoted to existing approaches to service composition) describes our approach in details.

To summarize the introduction, there are two interrelated problems addressed by the chapter. The first one is a possible revision of the SOA paradigm whereas the second one is how to realize SLA negotiations in business services. The proposed solution assumes that once the XML-documents processed by the services are described in OWL, the SLA negotiations as well as rest activities between service and client can be automated.

The structure of the chapter is as follows. Section 2 is devoted to existing approaches to service composition. In Section 3 our approach to e-businesses is presented with OWL as the description language, as well as experimental software tools for realizing these e-businesses. Section 4 presents our discussion on active services and Task Oriented Architecture. Since the chapter presents the work in progress, the conclusion is short and indicates only some future work.

2 Approaches to Service Composition

There are several classical frameworks and approaches to formal investigation of the distributed systems, as well as service composition and business processes (e.g. π-calculus and Petri Nets).

Formal approaches to service composition (see for example, [5, 6, 7, 8]) presuppose that a service is described in a formal language (modeled) by a precondition and the effect of its execution. This allows the application of techniques from AI planning and program synthesis to orchestrate the services into a composite service. However, it puts the burden on the service provider to formally specify its services and to annotate the WSDL specifications with this additional information.

There are also technical (as opposed to formal ones) approaches to service integration and composition. Each of them focuses on different aspects of the service arrangement and composition and proposes a different solution. An extensive overview of these approaches is given in literature (see i.e. [9, 10]). The approaches that are closely related to our proposal are briefly described below.

OWL-S (Web Ontology Language for Services) [11] and other Semantic Web projects, for example, WSMO (Web Services Modeling Ontology) [12] aim to make Web services computer-interpretable, i.e., described with sufficient information to enable automated service discovery, invocation, composition and execution. They also provide a process-level description of services, which is defined as a special ontology. It allows describing more general business processes that consist of any

services that can be discovered, bind and invoked. The OWL-S Ontology comprises ServiceProfile, ServiceModel, and ServiceGrounding. ServiceProfile is like the yellow page entry for a service. It relates and builds upon the type of content in UDDI, describing properties of a service necessary for automatic discovery, such as what the services offers, and its inputs, outputs, and its side-effects (preconditions and effects). ServiceModel describes the service's process model, i.e., the control flow and data-flow involved in using the service. It relates to BPEL and is designed to enable automated composition and execution of services. ServiceGrounding connects the process model description to communication-level protocols and message descriptions.

The SWORD [13] is a set of tools for the composition of a class of web services including information-providing services. In this project, services are defined in terms of their inputs and outputs in a rule (as in rule-based expert systems). These rules define which outputs can be obtained by the service given which inputs. When a developer wishes to create and deploy a new composite service, she specifies the inputs and outputs of the composite service in the world model and submits it to SWORD. After this submission SWORD determines using a rule engine if the composite service can be realized using the existing services. If so, SWORD generates a composition plan for the composite service. After receiving the plan, the developer can then view it and if appropriate, request that a persistent representation of the plan be generated. This representation contains the sequence of services that need be invoked to obtain the composite service outputs from its inputs. When an actual request for the composite service is received, the service(s) specified in the plan are executed, starting with the known inputs, in order to compute the desired outputs. SWORD is independent of other Web Service Standards.

The SHOP2 planning system [14] is a Hierarchical Task Network (HTN) planner. Rather than having a set of goals to achieve (as in classical planning), it instead has a partially ordered set of tasks to perform. To solve planning problems in some domain, SHOP2 needs a domain-specific knowledge base of methods. Each of these methods gives a way to decompose tasks recursively into subtasks until it reaches primitive tasks that can be performed directly by planning operators. The preconditions of SHOP2's methods and operators can include logical inferences, complex numeric computations, and calls to external programs. SHOP2 always builds plans forward from the initial state: given several tasks to decompose, SHOP2 plans for them in the same order that they will be executed.

There are also several academic projects, e.g., SELF-SERV [15], EFlow [16], and our own enTish project (see [17, 18]). In the enTish project we have proposed the declarative way to describe the client's request, i.e., to specify formally *What*

is to be realized?, and then to realize the request by discovering, arranging and composing services according to one defined protocol.

3 Our Approach to Service Composition with OWL as the Description Language

The key assumption of our approach is that a service may be described externally, that is, by the local change in the environment, caused by a performance of service. The very *local change* is usually described by precondition and postcondition (effect) of the performance. The environment should be represented in a formal way, and then described in a formal language. In the case of business processes, the environment is represented by documents that are exchanged and processed by the partners involved in the process. The language describes the documents. So that the crucial notions are: *environment of interaction between services and clients*, its *representation* denoted by (Service Environment Representation, SER for short), and *a language describing the representation*. The language is grounded (has semantics) in the environment representation (SER). The OWL syntax may be used as the syntax of the description language. Although OWL has well defined model-theoretic semantics (see `http://www.w3.org/TR/owl-semantics/`), the proposed semantics for the description language (based on the OWL syntax) has another semantics, this time concrete one, i.e., SER. For this very reason the description language used in our approach is denoted by OWL+SER.

OWL+SER is complementary to WSDL where the input and output documents, to be processed by a service operation, are defined. Actually, these documents are also in SER. OWL+SER allows to specify service precondition and effect as OWL formulas in the same way as it is done in OWL-S, however, in the case of OWL+SER the precondition formula describes the input documents of the WSDL operations of the service, whereas the effect formula describes the corresponding output documents of the operations.

In fact, we follow the IOPE (Input Output Preconditions and Effects) approach from OWL-S. However, contrary to OWL-S and its ServiceModel, the internal service structure cannot be specified; it is treated as a black box.

OWL+SER is also used to define tasks to be accomplished by (composite) services. Tasks are expressed in a declarative way as a pair of OWL formulas (initial situation formula and intention formula) describing the client initial situation, and a final situation intended by the client. This is similar to *(precondition, effect)* of the service description in OWL-S. OWL+SER serves also to arrange necessary conditions (to be fulfilled by the client) for service invocation in order to accomplish the client's task.

3.1 Service Architecture

WSDL is regarded as the standard for describing web services. General structure of service in WSDL 2.0 consists of the following elements:

— The data types used by the web service.
— The abstract interface as a set of abstract *operations*, each operation representing a simple interaction between the client and the service. Each operation also specifies a message exchange pattern that indicates the sequence in which the associated messages are to be transmitted between the parties. Usually, there are input and output messages.
— The communication protocols used by the web service.

The usual non automatic use of a business service is decomposed onto the four following phases:

1. The service requester sends a query to a service specifying roughly what she/he wants from the service, and then gets back a quotation (a pro forma invoice) from the service provider. The quotation specifies details of what (and how) the service can be performed for the requester.
2. Having the quotation, the requester creates an order and sends it to the provider. Usually, the provider replies with an appropriate contract.
3. If the service is performed according to the contract, the provider sends expertise or carriage letter to the requester, whereas the requester sends back a delivery note (or acknowledgement of receipt) or a complaint.
4. Finally, the service provider sends invoice and the requester realizes the payment for the service performance.

Sometimes, after sending the order, the service provider sends back an invoice, so that the payment must be done before service delivery.

In order to automate the use of a service, all those four phases above must be automated. For any specific type of service this may be done by a dedicated software application implemented, for example, in BPEL. Note that in each of the phases above, there is a document flow, that is, query and quotation in the first phase, order and contract in the second phase, carriage letter, delivery note, and complaint in the third phase, and finally invoice and payment. These documents can be created in XML. For each of the phases, the document flow can be described in WSDL. Finally, the complete process of using a single service can be implemented in BPEL. This can be done (hardcoded) separately for any service type in a dedicated way. However, our goal is to do so generically, that is, to construct tools that allow automation of service use for any service type.

Note that all the above phases are interrelated with each other, that is, a request (in the second phase) is created on the basis of the quote from the first phase,

whereas the carriage letter, delivery note, complaint and payment are strictly related to the contract. Each of the phases must be implemented as a separate WSDL operation. There are no means to explicitly express these interrelations in WSDL. An implicit way to do so is to use the same types of elements in documents from different phases. However, to automate the service use these interrelations must be explicitly expressed in a formal language; in our approach the language is OWL+SER. These interrelations, as OWL formulas, can be processed automatically.

The first phase is of special interest for the automation in question. Actually it concerns SLA (Service Level Agreement) negotiations. The agreement concerns the service output, i.e. the result of the service required by the client. Hence, the negotiations are about the contract, delivery, complaint, and payment. It is natural to have a generic language for such negotiations independent of the type of service to be used. The language should describe documents to be processed in the next phases; it is clear that it must be OWL+SER. Then, the negotiation may proceed as follows. Given its task, the client sends its *intention formula* (OWL formula) to a service, describing roughly what is to be accomplished by the service. This intention formula corresponds to the query in the first phase of business service. Service sends back a commitment consisting of two OWL formulas: *precondition formula*, and *effect formula*. This corresponds to the quote from the first phase. The precondition allows creating, by the client, XML-document required as input by the service. Actually, the precondition formula proposes several options (one to be chosen by the client) of the task accomplishment, and describes the input document (corresponding to the client choice) in details. The effect formula specifies the service output in details with relation to the input. Actually, the effect formula implies (logically) the client intention formula. It means that the intention formula describes (roughly) the output document, whereas the effect formula describes the output completely, usually, in several options. If one of the options is chosen by the client, SLA is set and its terms are incorporated in the order and in the contract (or invoice) in the next phases.

The conclusion from the above discussion is the following. The first phase of service use (corresponding to SLA negotiation) can be done, in a universal automatic way, using OWL-SER language. Based on this, the next phases of service use can also be automated. So that the crucial points of the proposed approach are as follows.

— Service interaction environment is represented by XSD schemata of the XML documents processed in the order-contract phase of service use.
— The language (OWL+SER) describing the representation (i.e., XSD schemas) is based on the syntax of OWL. The grounding (semantics) of the language is constituted by the above XSD schemata.
— Query and quote (of the first phase of the service use) are expressed in this language.

So that automation of the use of a single service is relatively simple. If, however, several single services must be composed to perform a complex task, then the problem is how this composition can be done automatically and in a universal way.

Once the first phase is realized according to our approach, the automatic service composition is possible to some extent. The composition is done in the following three steps.

— **Step 1:** Given an initial situation and a goal (that constitute together a request) expressed in OWL+SER, construct generic plans that may realize this goal starting with the initial situation.
— **Step 2:** Choose one generic plan and discover appropriate services. Then, arrange them into a workflow. The arrangement is done as query-quote phase in OWL-SER, and corresponds to SLA negotiations.
— **Step 3:** Present the workflow to the client specifying the initial situation more precisely. Once the initial situation is satisfied by the client, execute the workflow, control the execution, and realize the distributed transaction, if the execution is successful.

Note, that if a business process is more complex, the client interaction may be necessary in the Step 3. In this case, depending of a client decision, a new task must be realized, so that the Steps 1, 2, and 3 must be repeated.

To realize the Step 1, an automatic reasoning for plan construction is needed. There is a considerable work done in this area, see for example PDDL (planning domain definition language) [19] for Estimated-regression planning, and SHOP2 domain-independent planning system [14].

The Step 2 is realized by sending queries (called also intentions in our approach) and getting back quotations (called also commitments in our approach). Step 3 may be realized by using a process modeling language, i.e., by BPEL.

In our approach we propose a universal protocol for realizing simple client's requests (that do not require client interactions) according to the Step 2 and Step 3. Generally, the protocol specifies message exchange between services, agents (realizing the requests on behalf of the clients) and service registry.

The protocol consists of two general phases: The first one is called query phase (corresponding to SLA negotiations) whereas the second one is called execution phase. The query phase consists of the following two steps:

1. The agent sends a query (intention) to the service specifying the desired output (effect).
2. Then, the service answers with the quotation (commitment), i.e., specification of the input (precondition) required producing the desired output.

The execution phase consists of the following three steps:

1. The agent (dedicated for the task accomplishment) creates input data (an order) according to the quotation and sends it to the service.
2. The service receives the order and produces output data (a contract) that is sent back to the agent.

The previous versions of our approach (where Entish was used as the description language instead of OWL+SER) were verified by several prototype implementations. The first implementation (called enTish [20]) focused merely on composition of services that process data, i.e., services were ordinary software applications. The last completed implementation (called ELA-enT [21]) is dedicated to realization of Electronic Market for simple services. The details of the description language Entish can be found in [17] and on the project web site [20].

3.2 Architecture of the Proposed System

The ongoing implementation of our approach (based on OWL+SER) is designed for automatic planning, arrangement, and execution of complex business processes. The system consists of the four main elements: Dictionary, Services, Kernel, and Task Manager.

The Dictionary stores definitions of types of services and documents that are processed by the services. Each service type may have several documents type (as its input) and several document types as its output. Each document type consists of simple attributes and nested complex types. The nested types may be added through aggregation or inheritance. The domains of attributes may be defined as generic data types, enumerations or sets of ranges. The Dictionary provides a GUI for browsing, defining and modifying types of services and documents. It can also generate XSD schema for document types and WSDL descriptions of defined service types, as well as their descriptions in OWL.

Services communicate with the system trough the Service Manager. It is a system component responsible for receiving and sending messages in OWL language (in the query phase), and SOAP envelopes (containing documents) in the execution phase. Kernel (Fig. 2) consists of two autonomous components: Service Registry and Query Agent. Communication between these elements as well as with other system components is based on universal language (OWL) describing the service environment representation. The communication protocols define the format of messages, the order in which the messages are exchanged, and the actions that should be taken after receiving each of the messages. Protocols handled by the Kernel include the following functionalities: service registration, commitment requests, and service discovery. In order to be visible (discovered) in the system, a service must publish its interface to the Service Registry. The service interface describes functions or actions that the service can provide by specifying appropriate precondition and postcondition (effect) in the OWL language. In the response to

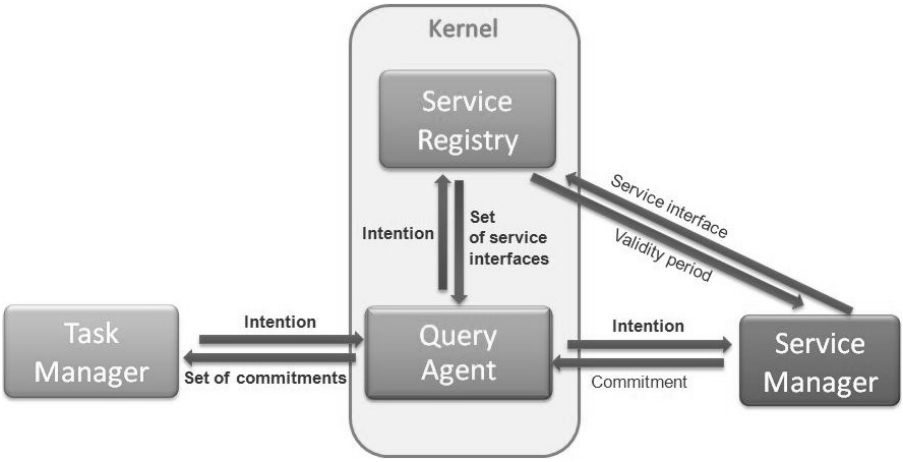

Fig. 2. Kernel

a registration request, the service receives the validity period of the entry made in the Service Registry. If the service does not repeat its registration in the specified time period, the corresponding entry is removed from the Registry.

Task Manager, in order to find and arrange a service that can realize the client's intention, queries appropriate services through the Kernel. It sends the intention to a Query Agent, and receives a set of service commitments (quotes) in response. Based on these commitments, the most suitable service is chosen, directly by the client, or automatically, by the appropriately configured Task Manager.

After receiving a commitment request, the Query Agent discovers appropriate services through the Service Registry. A service is considered as appropriate, if it realizes the same function as described in the intention, and postcondition (effect) of the service interface implies the intention. After discovering the services (with their network addresses), the Quote Agent sends commitment request with the client's intention to each of them, and waits for their responses. A service replies with its commitment if it is able to realize the intention. The commitment specifies the precondition that must be satisfied for the service to produce desired output. If there is no precondition that could allow realization of the intention, the service replies to the Query Agent with the commitment refusal. After receiving responses from all the discovered services or when the timeout passes, the Query Agent sends the gathered commitments to the Task Manager.

The functionality of the Kernel (used in the query phase), i.e. service registration, service discovery, and handling commitment requests, in realized in the same manner independently from the types of services and intentions.

The Task Manager itself is composed from several modules that interact with each other through precisely defined interfaces (Fig. 3). The goal of such design is to increase automation of client's intention realization, beginning from automatic generation of an abstract plan, through the plan arrangement, to the execution of the arranged plan.

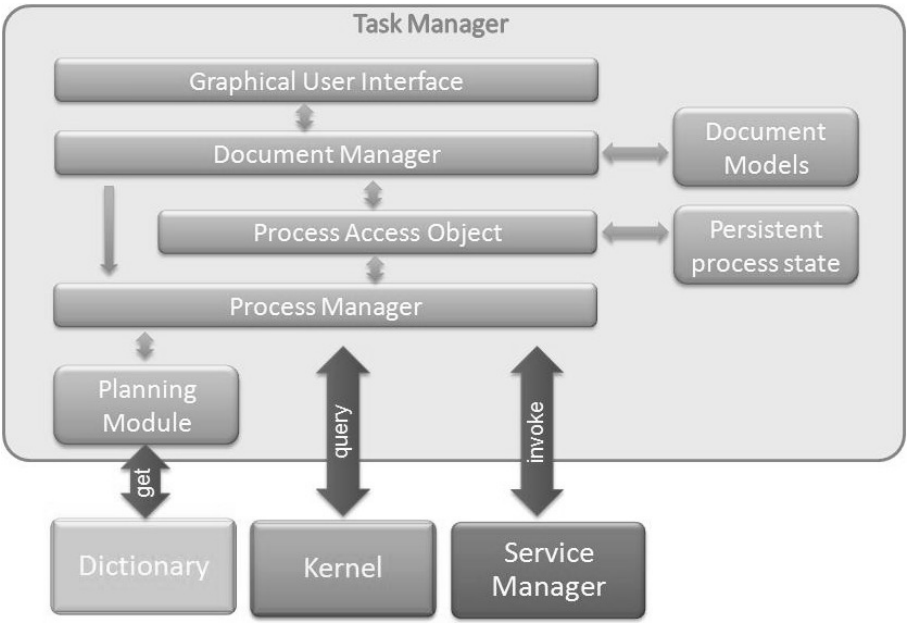

Fig. 3. Architecture of Task Manager

The Graphical User Interface is responsible for interaction with the user, i.e.:

— specifying an intention;
— presenting the process steps, their state and possible actions;
— choosing and carrying out actions during each of the process steps, including:
  - browsing through existing documents;
  - choosing most suitable service;
  - creation and modification of documents.

The process steps are handled in the same way during the entire process of the intention realization, which also includes the method of presenting them to the client.

During each of the steps, the client operates on documents. There are input and output documents. A process step is arranged by choosing an appropriate service that can execute given task. Input documents are to be sent to the service, and output documents are the results of the service execution. Processing of the documents as well as generating views for the documents (which are presented to the client) are handled by the Document Manager. The Document Models, used in the mentioned activities can be prepared by a programmer for each of the document types, or can be automatically generated based on the information from the Dictionary.

The state of each of the processes being currently realized is stored in a persistent way (Persistent process state) in a database. The state consists of process identifier, its status, and set of steps. Each step consists of step identifier, its status, intention, chosen commitment, set of all commitments from the query phase, input documents, and output documents. Universal access (independent from the method of persistence) to the process state is achieved by defining the Process Access Object interface that provides methods for accessing the information about current state of a process and its steps as well as methods for updating the state.

The life cycle of the process of an intention realization includes:

1. Automatic generation of the abstract plan that leads to the realization of the intention.
2. Initialization of the process steps. Each abstract service in the plan is represented by a single process step. The order of the steps corresponds to the order of abstract services in the generated plan.
3. Query phase. Beginning from the last step, the client partially defines the output documents (i.e. defines the intention for given step). The intention is sent to a Query Agent, and the most suitable service is chosen for that step. The commitment of chosen service may become the intention for the precedent step.
4. Execution phase. After successful arrangement of all steps, the client completes the input documents for the first step. Then, the documents are sent to appropriate service. The output documents received from the service may automatically become input documents of the next step (if the types of documents are appropriate).

The Process Manger is responsible for changing the process state. The change can be initiated by the client (by defining the intention, choosing a service, accepting an input document), or by receiving relevant messages from other system components involved in the process of the realization of the client's intention. The Process Manager communicates with Planning Module in order to obtain an abstract plan, the Kernel in query phase, and appropriate Service Managers in execution phase.

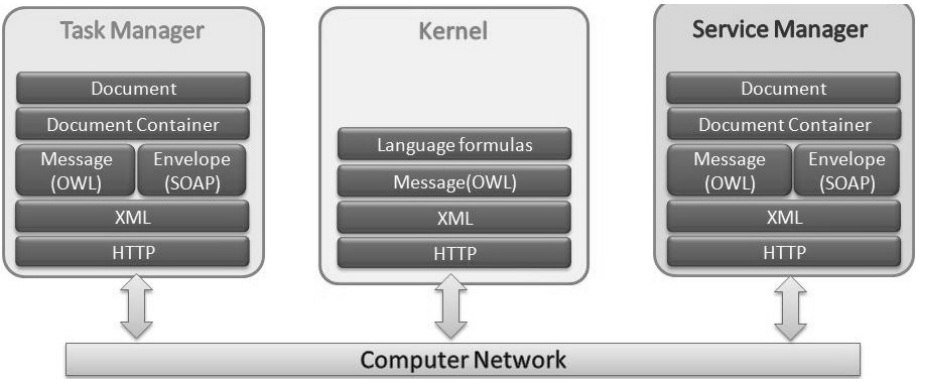

Fig. 4. Schema of communication stacks

The Planning Module can be an integral part of the Task Manager (as shown in Fig. 3) or can be a separate system component. Its task is to automatically generate an abstract plan leading to the creation of the documents described in the client's intention. The plan can be generated based on the information about defined types of services and documents stored in the Dictionary.

The implementation of the prototype of the proposed system also includes (beside Kernel, Task Manager and Service Managers) tools allowing for easy operating on documents and universal handling of massages exchanged between the system components (Fig. 4). OWL messages are used in query phase as they describe partially specified output documents (intentions) and the service conditions (quotes, commitments) to realize the intention. Because of this, there is no need for defining additional WSDL operations and document schema for each of the services types, and more importantly, the query phase may be carried out in a fully automatic manner. In the execution phase, the documents are exchanged using the SOAP protocol. The message formats (OWL message, and SOAP envelope) are translated to a common representation allowing for unification of the document management, that is, reading, defining, and modifying.

4 Active Services and Task Oriented Architecture

Classical version of the SOA paradigm has its origins in software engineering. From the point of view of e-business processes (that are supposed to be also software applications), the concept of service need not be related to RPC (as it is in SOA) where a service is passive and is waiting for a client to be invoked. In other words,

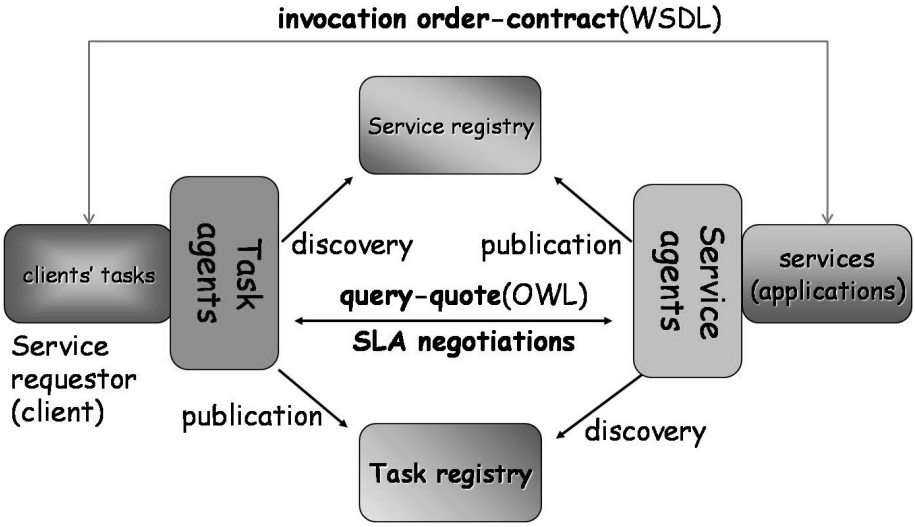

Fig. 5. SOA and TOA together

a service may be active and looking by itself for clients' tasks that can be accomplished. This corresponds to the reverse auctions in business practice. In the SOA paradigm the services must be described (as it is done in WSDL and OWL) in order to be published in a service registry, and discovered there by clients. Task oriented architecture, as opposed to Service Oriented Architecture, requires tasks to be expressed declaratively, so that the tasks can be published by the clients in a special task registry, and discovered there by services. Unfortunately the use of the term *Task Oriented Architecture* is a bit restricted by the US patent [22].

In the Figure 5, the classic version of the SOA paradigm (presented in the Fig. 1) is augmented with its TOA counterpart. That is, several components are added: Task repository for task publication and discovery, as well as task agents and service agents, representing clients and service providers (respectively) in business processes. These agents together with their interactions, constitute a multiagent system that is interesting to investigate by itself. One of the fundamental points of our approach is the separation of the query-quote phase (corresponding to SLA negotiations) and its generic realization in declarative language (as shown in the Fig. 5) from the execution phase (invocation order-contract in WSDL). The execution phase is realized by direct interaction between client and service on the basis of the service level agreement done in the query-quote phase. All the interactions corresponding to task publication and discovery, service publication and discovery, as well as SLA negotiation are delegated to task agents and service agents.

It seems that this very separation makes the revised SOA paradigm more generic and flexible.

4.1 A Revision of the SOA Paradigm

The main and starting point of the proposed SOA revision is that service, in the SOA paradigm, should be considered as business service, so that it has well defined structure corresponding to the following consecutive phases of service use: query-quote, order-invoice, payment, and so on. These phases may correspond to operations in WSDL. However the relations between the phases cannot be expressed in WSDL explicitly. Since a service may be active (as opposed to passive service as it is in classic SOA), a declarative language is necessary to express client tasks. Once we have such declarative language; it may also be used to describe service interfaces in a declarative way. Although, the WSDL (by its name) is supposed to describe Web services, actually it serves to define XSD schema of the input and output documents processed by service operation.

Separation between WSDL and the description language is crucial in the proposed revision. Although this was done in Semantic Web services in OWL-S, the semantics of OWL is not directly related to the WSDL, that is, to the XSD schema of the documents processed by services. In our approach, introduction of the notion SER (Service Environment Representation) as the direct grounding (concrete semantics) of the description language, allows to explicitly express the relations between the phases of service use.

The Figure 6 summarizes the proposed approach and a revision of the SOA paradigm. The basis is the representation of service interaction environment consisting of XSD schemata of the documents processed by services and clients presented respectively in the right and the left side of the figure. In the next layer there is OWL+SER (grounded in the layer below) as a declarative language for expressing client tasks and service interfaces. In the third layer the separation between the execution phase (XML document flow by BPEL, SOAP, WSDL, and UDDI), and SLA negotiation phase (as a multi agent system) is shown.

4.2 Prototype Implementation

Based on the proposed approach to building SOA systems a prototype version was implemented. The prototype consists of the following components: Task Manager, Service Registry, Quote Agent, Dictionary, and Service Servers corresponding to the service types defined in the Dictionary.

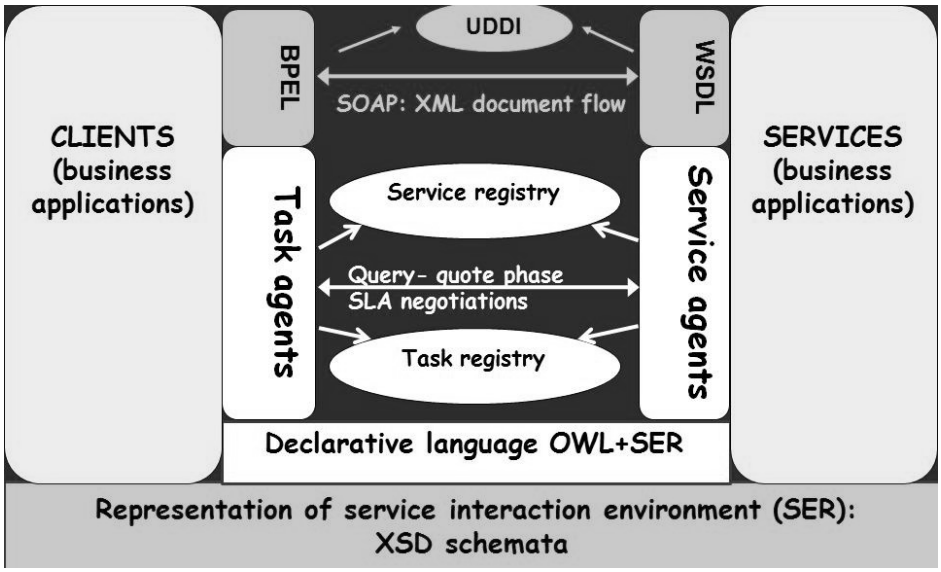

Fig. 6. The general architecture of our approach

To verify the prototype system a testing environment was created. This environment is described by concepts (from Dictionary) and concerns a realization of construction investment by an investor who wants to build a house.

The prototype system supports the realization of the complete business process for the construction investment, as well as any of its parts, for example, the subprocess of obtaining a license to build a house. If the execution of any service (being a part of the business process) fails, it is immediately reported to the Task Manager (and then to the investor). The investor (via the Task Manager) can decide whether to continue the process or not. In the case of continuation, the Task Manager replaces service that failed by another one.

Furthermore, the system allows to reuse any document obtained as a result of a processes execution (not necessarily completed successfully) in a new processes. This approach allows reducing services calls.

The functioning of the components of the prototype system (mainly the Task Manager and Service Servers) has been verified by performing several business scenarios and corresponding business processes related to construction investment. The prototype system is available at website: `http://www.itsoa.ipipan.eu/`

5 Preliminary Conclusions

We have described our ongoing project that aims at creating tools for automate (as much as possible) modeling, planning, constructing, execution and control of sophisticated business processes. The work is in progress, and is based on the approach presented above. To verify our approach as well as the tools, two testing environments are being built. The first one relates to crisis management, whereas the second one for business processes is related to real estate development.

The idea presented in the chapter seems to be a novel approach to SLA negotiations as well as to e-business processes. A potential application of the proposed solution may improve automation of modeling, creation and executions business processes for accomplishing clients' tasks.

Acknowledgments. The work was done within the framework of the ITSOA POIG01.03-01-00-008/08-00 project supported by the European Commission and the Polish Government.

Appendix A List of acronyms

AI	Artificial Intelligence
BPEL	Business Process Execution Language
BPEL4WS	Business Process Execution Language for Web Services
GUI	Graphical user interface
HTN	Hierarchical Task Network
IOPE	input, output, pre-condition and effect
IT	Information Technology
OWL-S	Semantic Markup for Web Services
PDDL	Planning Domain Definition Language
RPC	Remote procedure call
SER	Service Environment Representation
SLA	Service Level Agreement
SOA	Service-oriented architecture
SOAP	Simple Object Access Protocol
TOA	Task Oriented Architecture
WSDL	Web Services Description Language
WSMO	Web Service Modeling Ontology

XML	Extensible Markup Language
XSD	XML Schema Definition
UDDI	Universal Description Discovery and Integration

References

1. IBM Services Architecture Team, Web Services architecture overview: the next stage of evolution for e-business. 2000, `http://www.ibm.com/developerworks/webservices/library/w-ovr/`.
2. Peltz C. Web Services orchestration and choreography. Computer 36, 10, 46–52, 2003.
3. Davenport T. Process innovation: reengineering work through information technology. Harvard Business School, Boston, 1993.
4. Napagao S.A., Biba J., Confalonieri R., Dehn M., Kollingbaum M., Jakob M., Oren N., Panagiotidi N., Solanky M., Salceda J.V., Willmott S. Contract based electronic business systems state of the art. Ist Contract Project, 2007.
5. Berardi D., Calvanese D., Giacomo G.D., Lenzerini M., Mecella M. Automatic service composition based on behavioral descriptions. Int. J. Coop. Inf. Syst. 14, 4, 2005.
6. Duan Z., Bernstein A., Lewis P., Lu S. A model for abstract process specification, verification and composition. Proc. 2nd Int. Conf. on Service Oriented Computing ICSOC04, New York, USA, 232–241, ACM Press, 2004.
7. Matskin M., Rao J. Value-added web services composition using automatic program synthesis. Workshop on Web Services, E-Business, and the Semantic Web CAiSE02, Rev. Pap. C. Bussler, R. Hull, S. McIlraith, M. Orlowska, B. Pernici, J. Yang (eds.), 213–224, LNCS 2512, Springer Verlag, Berlin, 2002.
8. Traverso P., Pistore M. Automated composition of semantic web services into executable processes. Proc. 3rd Int. Semantic Web Conference ISWC, S. McIlraith, D. Plexousakis, F. van Harmelen (eds.), 380–394, LNCS 3298, Springer Verlag, Berlin, 2004.
9. Rao J., Su X. A survey of automated Web Service composition methods. Proc. 1st Int. Workshop on Semantic Web Services and Web Process Composition SWSWPC 2004, 43–54, Springer Verlag, Berlin, 2004.
10. Milanovic N., Malek M. Current solutions for Web Service composition. IEEE Internet Comp. 8, 6, 51–59, 2004.
11. The OWL Services Coalition, oWL-S 1.1. 2004, `http://www.daml.org/services/owls/1.1/`.
12. Web Service Modeling Ontology (WSMAO). ESSI cluster WSMO working group, `http://www.wsmo.org/`.
13. Ponnekanti S.R., Fox A. SWORD: a developer toolkit for web service composition. Proc. 11th Int. WWW Conf. WWW2002, Honolulu, Hawaii, 2002.

14. Wu D., Sirin E., Hendler J., Nau D., Parsia B. Automatic Web Services composition using SHOP2. Workshop on Planning for Web Services, Trento, Italy, 2003.
15. Benatallah B., Dumas M., Sheng Q.Z. The Self-Serv environment for Web Services composition. IEEE Internet Comp. 40–48, 2003.
16. Casati F., Ilnicki S., Jin L. Adaptive and dynamic service composition in EFlow. Proc. 12th Int. Conf. on Advanced Information Systems Engineering CAiSE, Springer Verlag, Berlin, 2000.
17. Ambroszkiewicz S. Entish: a language for describing data processing in open distributed systems. Fund. Inf. 60, 1–4, 41–66, 2004.
18. Ambroszkiewicz S., Mikulowski D., Rozwadowski L. Entish: e-lingua for Web Service integration. Proc. 5th Int. Conf. on Business Information Systems BIS-2002, Grinbelt, USA, 2002.
19. McDermott D. PDDL — the planning domain definition language. AIPS98 Planning Competition Committee, 1998, `http://www.cs.yale.edu/homes/dvm`.
20. Project enTish. `http://www.ipipan.waw.pl/mas/`.
21. Project ELA-enT. `http://ent.ipipan.waw.pl/`.
22. Apparatus, method and architecture for task oriented applications (Fujitsu Limited). US Patent 6442749 issued Aug. 27 2002, exp. Oct. 30 2018, 2002.

Adaptable User Interfaces for SOA Applications in the Construction Sector

Jacek Chmielewski, Krzysztof Walczak, Wojciech Wiza, and Adam Wójtowicz

Department of Information Technology
Poznań University of Economics,
Mansfelda 4, 60-854 Poznań, Poland
{chmielewski,walczak,wiza,awojtow}@kti.ue.poznan.pl

Abstract. The problem of integrating heterogeneous *back-end platforms* using the SOA paradigm has been widely addressed in a number of research and development projects. However, in some application domains — such as the construction industry for example — there is an additional requirement: integration of heterogeneous front-end platforms — from desktop PCs to mobile PDAs. In this chapter, a new method of building adaptable user interfaces for SOA applications is described. In the adaptation process, the displayed content is adjusted, the best way of presenting the content is selected, and interaction methods are adapted to the capabilities of a particular front-end device, as well as the preferences and privileges of an end-user. Therefore, adaptable user interfaces proposed in this chapter make any SOA service easily accessible on any device.

Keywords: SOA, user interface, adaptation, mobile devices

1 Introduction

One of the main advantages of using the Service Oriented Architecture (SOA) paradigm [1] to develop software is the ability to seamlessly integrate diverse systems running on heterogeneous hardware and software platforms. This feature is of special importance for the efficient deployment of systems associated with complex business processes. An application domain, which is characterized by the intrinsic complexity of the underlying business processes is the *construction industry*. Construction industry requires close cooperation of multiple participants representing different domains, playing different roles, and having different areas of interest. Moreover, these participants may come from both the *private sector* and the *public sector* including the *public administration*. Therefore, in general,

they have different ways of operation and are subject to various regulations and constraints. In such a complex and heterogeneous environment, the application of the SOA paradigm can be particularly beneficial. In many cases, it may enable the creation of new types of applications that could not be built based on traditional approaches or — at least — may significantly reduce the software development costs.

The problem of integrating heterogeneous *back-end platforms* using the SOA paradigm has been widely addressed in a number of research and development projects [2]. However, in some application domains — such as the construction industry for example — there is an additional requirement: integration of heterogeneous *front-end platforms*. In many cases, work with an application must be performed in several different environments — on a desktop computer in an administration office, on a notebook computer in the construction office, or on a handheld computer in the construction field. These devices significantly differ in their communication capabilities and processing power, as well as the offered presentation and interaction methods [3].

Often, efficiency of work with an application can be significantly improved by allowing users to adjust the application interface and to choose the content and the functions, which they need most in given circumstances. In many cases, the method of content presentation is important as well. Also, in general, users differ in their privileges to access different types of content. All these elements should be taken into account in the development of SOA applications.

In this chapter, we describe a new method of building adaptable user interfaces for SOA applications, called *ASIS — Adaptable SOA Interface System*, and provide examples on how this method can be used to build SOA applications for the construction sector. A system, called *PSS — PINB Support System*, is described. PSS is a set of tools and interfaces built to provide access to location/building/case documentation for the local construction supervisory body [4].

By the use of the ASIS method it is possible to provide end-users with a convenient and flexible way of accessing SOA services. The way of accessing the services can be adjusted to the users' requirements, the device capabilities, the characteristics of the communication channel and the current context (place, time, previous interactions). The adjustment is performed automatically, without explicit actions of the users. At the same time, the method is generic and is not bound to any specific set of services or underlying business logic.

This chapter is organized as follows. In Section 2, an overview of the overall architecture of the ASIS interface adaptation framework is provided. In Section 3, the architecture of the ASIS Service Interface Generator is described. Section 4 contains the description of the SOIL language. In Section 5, examples of service visualization templates are presented. Section 6 provides an overview of the management application used to manage the data processed by the ASIS system and to control the ASIS Service Interface Generator components. Section 7 concludes the chapter.

2 ASIS Architecture

The overall architecture of the ASIS interface adaptation framework is presented in Figure 1.

The ASIS framework consist of four main elements:

— the Data Aggregation Service,
— the Service Interface Generator,
— the Service Interface Templates, and
— the Multimedia Data Adaptation Subsystem.

The *ASIS Data Aggregation Service* (DAS) is used to collect all kinds of persistent data processed within the framework. Examples include information about locations and buildings, maps, photographs, audio or audiovisual recordings, information about legal proceedings and documents, metadata describing these objects, information about users and groups as well as information about front-end devices and access channels. The data may originate from other available SOA services or may be entered manually or imported by the use of the *ASIS Content Management Application* (ACMA) attached to this module.

The *ASIS Service Interface Generator* (SIG) is used to dynamically create the content of a user interface in response to a request coming from the front-end device. This module is implemented as a servlet running on a web application

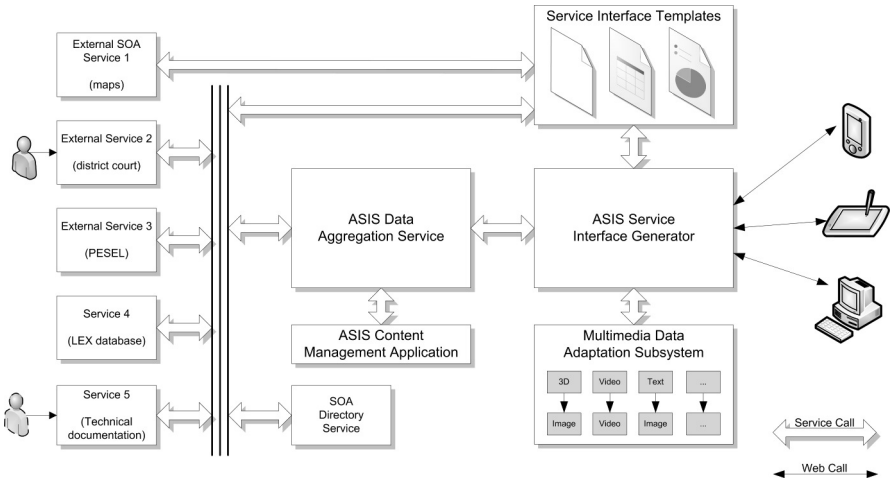

Fig. 1. The overall architecture of the ASIS framework

container and uses all the other elements of the ASIS framework. The SIG automatically adjusts the way of interface presentation based on the request, the available presentation templates, as well as the information about devices, access channels and user accounts stored in the database. The SIG uses services provided by all the other elements of the ASIS framework.

The *ASIS Service Interface Templates* enable SOA services to be presented to the end-users in a user-friendly way. The templates are encoded in a newly developed language called *SOIL* (*Service Oriented Interface Language*). SOIL is based on XML [5]. It provides commands that can execute calls to SOA services and control the process of interface generation. SOIL is independent of the content description language used (e.g., HTML [6], XML, or PDF [7]).

The *ASIS Multimedia Data Adaptation Subsystem* (MDAS) provides services adjusting the formats and properties of multimedia data (e.g., type, format, resolution, sampling, precision, compression) to make them suitable for presentation on a particular front-end device. For example, to display a three-dimensional model of an object on a device, which is not 3D capable, a particular view of the object can be rendered to a 2D raster image. Formatted 2D text can be converted into image to make the presentation independent of the text presentation capabilities of the particular front-end device.

3 The Interface Generator

The overall architecture of the ASIS Service Interface Generator is presented in Figure 2.

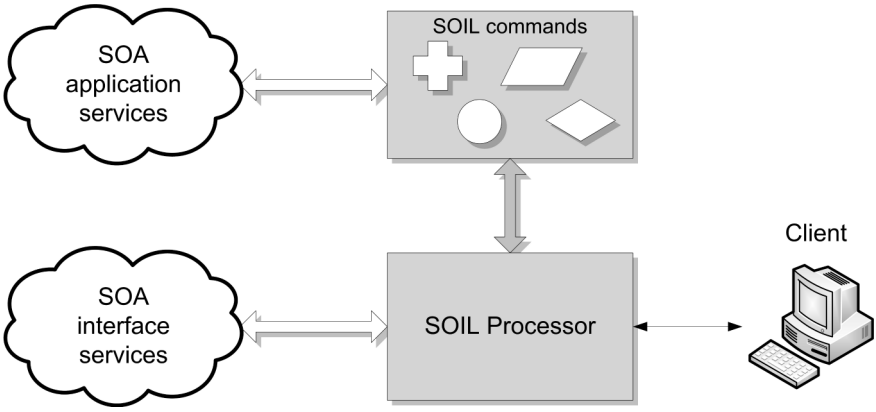

Fig. 2. The overall architecture of the ASIS Service Interface Generator

The ASIS Service Interface Generator consists of two components: the *SOIL Processor* and a collection of *SOIL command implementations*. The SOIL processor is the main unit responsible for the generation of the user interface. The unit contains the SOIL engine, which can interpret SOIL commands contained in the interface templates (cf. Section 5 *Service Visualization Templates*). In response to a request received from a client, the unit generates the final form of the interface description based on the collection of available interface templates and calls to the available *SOA interface services*. The interface generated is then being sent back to the client.

The SOA interface services provide the SOIL processor with all information needed in the process of generating the user interface. It includes information about the templates to be used, their parameters, users and their preferences and privileges, etc.

The interface templates consist of fragments of the interface description interwoven with SOIL commands. Implementations of the SOIL commands are provided as a collection of Java [8] classes independent of the main SOIL processor. An XML file provides mapping between the language elements and their implementations in the Java classes.

The SOIL commands can execute *SOA application services*. These are the services, which provide application's business logic. Examples of application services available for the PSS system include retrieval of locations at a given address, retrieval of administrative cases related to a location, and storing the list of selected cases in a briefcase for later access from a different device.

4 The SOIL Language

This section contains a description of the SOIL language, which has been developed to enable building user interfaces for SOA services and applications. First, an overview of the language is provided, followed by the description of the language syntax and examples of SOIL commands.

4.1 SOIL Overview

SOIL (*Service-Oriented Interface Language*) is a new interface programming language, which has been designed to enable efficient creation of computer interfaces for SOA services and applications. The SOIL language enables creation of interface templates combining static elements and dynamically generated elements including results of calls to SOA services.

The SOIL language is based on XML. A SOIL template is an XML representation of an algorithm that generates the interface description. The template is a program built of SOIL commands. The program can execute calls to SOA services, retrieve or update data from/in a database, use values of template parameters, etc. Examples of SOIL commands are: `set` to assign a value to a variable, for implementing a numerical loop, `if/then/else` implementing conditional statements, `db_query` to retrieve data from a database, and `service_call` to execute external SOA services. The SOIL commands are encoded in XML and placed inside the text of the interface description written in XHTML [9], XML or any other interface description language. The target language can be either XML-based or not. All XML elements that are not known to the SOIL processing unit (e.g., use a different namespace) are ignored and included in the outcome as elements of the interface description.

In SOIL, empty XML elements represent single-line commands such as `<set/>` or `<insert/>`. Non-empty elements represent block-statements like loops, conditions, service calls, database queries, iterations, etc. Examples of non-empty elements are `<for>` ... `</for>` and `<service_call>` ... `</service_call>`. The non-empty elements can contain fragments of interface description and SOIL code. The code located inside a repeating element is interpreted multiple times. Parameters for the command execution are provided in the form of values of element attributes. In SOIL, all parameters are expressions and are evaluated prior to command interpretation.

4.2 Modularization of SOIL

One of the fundamental concepts in the design of SOIL is extensibility. New language elements can be easily added to the language — either implementing some generic operations or operations specific to a particular application domain. The new elements can take the form of new XML SOIL commands or new functions for use in expressions provided as command parameters.

To ease the management of the SOIL implementation and documentation, the language has been divided into modules. Specialized SOIL modules with functionality specific to a particular domain or application can be added to the implementation.

Currently, four SOIL modules are available:

— the Core Module (SOIL CM), which contains basic language elements independent of a particular application,
— the Database Module (SOIL DB), which contains commands that enable retrieving and updating data in databases,
— the Service Module (SOIL SRV), which contains elements enabling executing requests to SOA services implemented, for example, as web services,

— the PSS Module (SOIL PSS), which contains elements specific to the PSS application (cf. Section 5 *Service Visualization Templates*).

The SOIL Core Module provides the basic language functionality that can be used in most applications. The basic functionality allows such general operations as assigning values to variables, inserting calculated values of expressions to the output interface description, conditional interpretation of fragments of the SOIL code, loops, nesting SOIL templates, and accessing databases.

The SOIL Database Module contains generic commands enabling operations on databases, including connecting to a database, retrieving data, and updating data in databases.

The SOIL Service Module contains commands enabling the SOIL templates to execute external calls to SOA services, thus retrieving data or updating data in external modules.

The SOIL PSS Module contains elements, which are specific to the PSS application. SOIL PSS commands perform more advanced tasks, such as retrieving and analyzing complex data structures, accessing documents in the repository, and processing metadata descriptions.

4.3 SOIL Core Module

Variables and Expressions. The SOIL language enables the use of variables. Variables can be set inside SOIL programs by the use of SOIL commands (e.g., `set`, `for`, `iteration`, or `db_query`), can be read from a configuration file, or can be provided in a URL. Values of variables provided in the URL can be statically specified by a hypertext reference or dynamically set on the client side (e.g., in an HTML form filled by a user).

In SOIL, all values of element attributes are treated as expressions and are evaluated prior to element interpretation. Expressions can contain:

— constant numerical, textual and Boolean values,
— variable references,
— operators, and
— functions.

Basic Operations. Basic operations of the Core Module include assigning values to variables, inserting calculated values of expressions in the outcome interface descriptions, conditional processing of fragments of SOIL code, loops, nesting templates, and connecting to databases. Examples of basic SOIL commands are provided below:

```
<set name="..." value="..."/>
```

```
<insert value="..." type="..."/>

<if condition="...">
  <then>
  ...
  </then>
  <else>
  ...
  </else>
</if>

<for name="..." from="..." to="..." step="...">
...
</for>

<iteration name="..." list="...,...,...">
...
</iteration>

<while condition="...">
...
</while>

<inline url="..."/>

<import url="..." in="..." out="..."/>
```

4.4 SOIL Database Module

Connecting to Databases. SOIL enables reading and updating data from/in databases. Since connections to multiple databases are allowed from a single SOIL template, the connections must be established explicitly. Before the interpreter connects to a database, it must be provided with connection parameters. These parameters include the network address of the server where the database is running, the port number, the database name, and the user name/password. In order to allow connections to different types of data sources, the interpreter must also be provided with the database driver name.

In SOIL, a connection to a database is established by the `db_connect` element. The required connection details are provided in two parameters: `conn_string` and `conn_driver`. If the parameters are omitted, the necessary connection information

is taken from two SOIL variables: `conn_string` and `conn_driver`. The syntax of the `db_connect` element is as follows:

```
<db_connect conn_string="..." conn_driver="...">
...
</db_connect>
```

All database access elements must be included within a `db_connect` element. More than one `db_connect` element may appear in one SOIL template to enable retrieving and updating data from/in more than one database.

Retrieving Data from Databases. Retrieving data from databases may be accomplished using a special `db_query` element. To simplify the process of retrieving data and constructing the output interface description, the `db_query` element has been designed to behave as a loop. The loop is being repeated for each row of data retrieved from a database as a result of the query execution. The values retrieved from the database are assigned to a set of loop control variables.

In the command parameters, a list of names of variables and an SQL [10] query are specified. The number of names of variables should be equal to the number of attributes retrieved from the database by the query. For each loop traversal, the retrieved values are assigned to variables identified by the provided list of names. The syntax of the `db_query` element is the following:

```
<db_query names="...,...,..." sql="...">
...
</db_query>
```

The `db_query` element has two mandatory attributes: `names` and `sql`. The names attribute contains a coma-separated list of the names of the loop control variables. The `sql` attribute represents the text of the SQL query to be executed. Since the `sql` attribute is a SOIL expression, it may use SOIL variables, operators and functions. Each `db_query` element must be located inside a `db_connect` element.

Updating Databases. A SOIL template can be used to update information in a database. In the SOIL language, updating databases is accomplished by the use of the `db_update` element. The syntax of this element is the following:

```
<db_update sql="..."/>
```

The `db_update` element has one mandatory attribute: `sql` that contains the SQL command to be executed. The `sql` attribute is a SOIL expression that is evaluated before the element is interpreted allowing the use of SOIL variables, operators and functions in SQL commands. Each `db_update` element must be located inside a `db_connect` element.

4.5 SOIL Service Module

Executing a Service Call. SOIL enables direct execution of external services from within a template. For this purpose, a special command `<service_call>` is used. The command contains the specification of the service URI and the name of the result variable provided as attributes, and the service request body provided as a child element of the command. The result of the service call is stored in a SOIL variable in the XML format.

```
<service_call uri="..." result="...">
  <soap:Body xmlns:x="...">
    <x:...>
      ...
    </x:...>
  </soap:Body>
</service_call>
```

The `service_call` command enables the inclusion of data dynamically obtained from the execution of SOA services in an application interface. It also enables updating data in other connected modules directly from the level of the application interface.

4.6 SOIL PSS Module

The SOIL PSS Module contains language commands specific to the PSS application. These elements are used to enable efficient retrieval and processing of various kinds of data from the PSS repository through the available application services. Data which can be retrieved includes folders, locations, cases, documents, media objects and metadata. Some of the data elements can be also updated. Examples of SOIL PSS commands are the following.

Retrieving Properties of Publication Folders. A publication folder is a hierarchical collection of content objects (locations or cases) published in a number of presentation domains (corresponding to different target devices) through a number of available interface templates. In SOIL PSS, properties of publication folders can be retrieved by the use of the `pfProps` command. The `pfProps` element has the following syntax:

```
<pfProps pfId="..." propName="..." varName="..."/>
```

where:

— `pfId` is the identifier of the publication folder,

— `propName` is the name of the property of the folder to be retrieved (see Table 1),
— `varName` is the name of a SOIL variable where the result is stored.

Table 1. Properties of publication folders

PropName value	Result stored in the variable	Type of result
PF_NAME	The name of the publication folder.	String
PF_DESCRIPTION	The description of the publication folder.	String
PARENT_PF_ID	The identifier of the parent folder.	Integer
CHILD_PF_IDS	The list of identifiers of child folders (subfolders).	List
PARENT_PF_IDS	The list of identifiers of all parent folders, forming a complete path to the folder identified by `pfId`. The list is ordered from the parent folder id to the root folder id.	List
PF_PATH	The full path to the publication folder.	String
CHILD_CO_IDS	The list of identifiers of content objects assigned to the folder identified by `pfId`.	List

In Table 1, examples of `propName` attribute values are presented with the description of the result produced by the `pfProps` command.

Retrieving Properties of Content Objects. Content objects are the basic units of information storage in the PSS repository. A content object can correspond to a location or an administrative case associated with a location. Content objects can form a hierarchy: cases are associated with locations and inherit their properties; cases can also inherit other cases. Content objects contain media objects — documents, notes, photographs, recordings, etc.

To retrieve properties of a content object, the `coProps` command is used. The `coProps` element has the following syntax:

```
<coProps coId="..." propName="..." varName="..."/>
```

where:

— `coId` is the identifier of the content object,
— `propName` is the name of the object property to be retrieved (see Table 2),
— `varName` is the name of a SOIL variable where the result is stored.

In Table 2, examples of `propName` attribute values are presented with the description of the result produced by the `coProps` command.

Table 2. Properties of content objects

PropName value	Result stored in the variable	Type of result
CHILD_MO_IDS	The list of identifiers of media objects associated with the content object identified by coId.	List
CO_NAME	The name of the content object.	String
CO_DESCRIPTION	The description of the content object.	String
CO_TYPE	The type of the content object. May have two values: 'location' and 'case'.	String
EXTENDS_CO_ID	The identifier of the parent content object in the inheritance hierarchy — if the object identified by coId is of type 'case', or NULL otherwise.	Integer
PARENT_PF_IDS	The list of identifiers of the publication folders to which the content object identified by coId is assigned.	List

Retrieving Properties of Media Objects. To retrieve properties of media objects the moProps command can be used. The moProps element has the following syntax:

```
s<moProps moId="..." propName="..." varName="..."/>
```

where:

— moId is the identifier of the media object,
— propName is the name of the object property to be retrieved (see Table 3),
— varName is the name of a SOIL variable where the result is stored.

Table 3. Properties of media objects

PropName value	Result stored in variable	Type of result
MO_NAME	The name of the media object.	String
MO_MIMETYPE	The MIME-type of the media object (if any) or NULL value.	String
MO_TYPE	The type of the media object type of the media object, e.g. 'Word document', 'Text', 'Image'.	String
MO_DATA	The content of the media object, e.g., a text value (if any) or NULL.	String
MO_CREATION_DATE	The date of creation of the media object.	String
PARENT_CO_IDS	The list of identifiers of content objects to which the media object identified by moId is assigned.	List

In Table 3, the list of valid `propName` attribute values is presented with the description of the result produced by the `moProps` command.

Manipulating Object Metadata. Content objects are associated with metadata descriptions. These descriptions can be retrieved and manipulated by SOIL commands. These commands can be either generic or specific to the application. An example of a command specific to the PSS application is `pssFlag`. This command enables flagging objects selected by a user, e.g., to be displayed on a mobile device. The syntax of the command is the following:

```
<pssFlag coId="..." userId="..." flagName="..." method="..."
varName="..."/>
```

where:

— `coId` is the identifier of the content object,
— `userId` is the identifier of a user,
— `flagName` is the name of the flag to be retrieved or changed,
— `method` is the name of the method to be executed on the flag (see Table 4),
— `varName` is the name of a SOIL variable where the result is stored.

Table 4. Names of methods for the pssFlag command

Value of the method attribute	Result
`CHECK`	Retrieve the value of the flag: TRUE or FALSE.
`SET`	Set the flag to TRUE.
`RESET`	Set the flag to FALSE.
`GETCO`	Retrieve the list of content objects with the flag set to TRUE.

In Table 4, the list of valid `method` attribute values is presented with the description of the result produced by the `pssFlag` command.

Searching Object Metadata. Metadata associated with content objects can be searched by the use of the `pssSearch` command. This command has the following syntax:

```
<pssSearch pfId="..." scope="..." text="..." varName="..."/>
```

where:

— `pfId` is the identifier of the publication folder,
— `scope` describes the object attributes on which the search should be performed (optional),

— `text` is the text fragment to be searched,
— `varName` is the name of a SOIL variable where the result (list of content objects) is stored.

If the `scope` attribute is not provided, the search is performed on all metadata elements associated with the object.

5 Service Visualization Templates

A number of SOIL templates have been implemented for the PINB Support System (PSS). Different templates have been designed to be used with different devices supporting the adaptation of the interface. In the test PSS environment sample templates have been prepared for a wide range of end-user devices: desktop PCs, netbooks, Tablet PCs, various PDAs and smartphones like Toshiba TG01 [11], Apple iPod Touch [12], HTC Touch Pro2 [13], and others.

5.1 Template for Desktop Interfaces

The main PSS interface is implemented for desktop computers. This interface is used mainly as an application to browse and search locations and cases. The desktop environment is characterized by rich graphics capabilities, high screen resolution, easy manipulation of screen content with mouse or other pointing devices, easy scrolling, and multiple window capabilities.

A simple search interface is presented in Figure 3. The interface provides a text-based search functionality for a PSS system user. After pressing the "Search" button, a search is performed with given phrase in all locations/objects the user has right to browse. The search is performed by the dedicated SOIL element `<pssSearch>`. The template code containing the form and the tag is the following:

Fig. 3. Simple search interface

```
[...]
<form action="{@pfPath}pfid={@pfid}&amp;domain=PINB"
  method="post" name="form_1">
  <if condition="{(@searchstr==NULL)}"><then>
    <set name="searchstr" value=""/>
  </then></if>
  <br/>
  Wyszukaj obiekty/sprawy:
  <input name="searchstr" size="30" type="text"
    value="{@searchstr}"/>
  <input name="search" type="submit" value="Szukaj"/>
</form>
</div>
<div id="search_res">
<if condition="{(@searchstr!=‘‘)}"><then>
  <pssSearch pfId="{@pfid}" text="{(@searchstr)}"
    varName="results"/>
[...]
```

The search result contains the list of locations and cases that contain the specified phrase in the metadata. The SOIL template code used to present the list is the following:

```
[...]
<set name="nos" value="{sizeOf(@results)}"/>

<if condition="{$nos gt 0}"><then>
  Znalezionych obiektów/spraw:
  <b><insert value="{#nos}"/></b>
  <br/><br/>
</then><else>
  Nie znaleziono obiektów/spraw z wyszukiwanym tekstem:
  <i><insert value="{@searchstr}"/></i>
</else></if>
<for from="0" name="i" to="{$nos-1}"><evaluate>
  <set name="currentObjId" value="{@results[$i]}"/>
  <coProps coId="{#currentObjId}" propName="CO_TYPE"
    varName="roar"/>
  <set name="objPath"
    value="{@pfPath}pfid={#pfid}&amp;co={#currentObjId}"/>
```

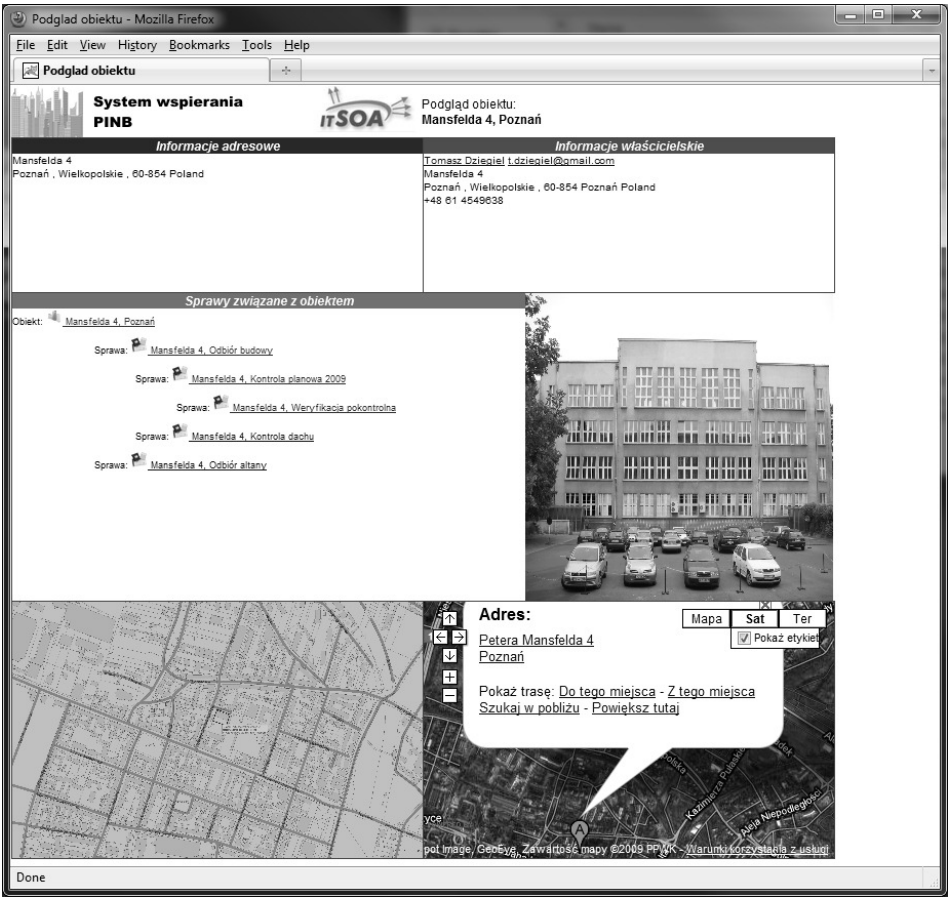

Fig. 4. An interface presenting a location

```
<if condition="{@roar=='location'}"><then>
  <a href="{@objPath}&amp;domain=PINB.OBIEKT">
    <img Border="0" src="{@adamToPath}18207"/>
  </a>&amp;nbsp;
  <coProps coId="{$currentObjId}" propName="CO_NAME"
    varName="objName"/>
  <a href="{@objPath}&amp;domain=PINB.OBIEKT">
    <insert value="{@objName}"/>
  </a>
```

```
  </then><else>
[...]
```

A user may click on a link pointing to a case or a location. In case of a location, the user is directed to the page presented in Figure 4. The page presents information retrieved for a particular location (from top left to bottom right):

— Information about the address (police data);
— Owner data (retrieved from court registers);
— Hierarchy of cases;
— Photo of the location;
— Map retrieved from a geodesy office [14];
— Map retrieved from Google Maps [15].

The hierarchy of cases for a particular location is created by a dedicated SOIL PSS tag `getCOHierarchy`. The code presenting all cases for a given location is presented below:

```
[...]
<span class="styl2">Sprawy związane z obiektem</span>
<getCOHierarchy coId="{#co}" varNameObj= "objl"
  varNamePos="positionlist"/>
<set name="listsize" value="{sizeOf(@positionlist)}"/>
<coProps coId="{@objl[0]}" propName="CO_NAME"
  varName="objName"/>

<evaluate>
  Obiekt:
  <a href="{@pfPath}pfid={#pfid}&amp;domain=PINB.O&amp;
    co={@objl[0]}">
    <img Border="0" src="{@adamToPath}18221"/>
    <insert value="{@objName}"/>
  </a><br/>
</evaluate>
<for from="1" name="i" to="{$listsize-1}">
  <coProps coId="{@objl[$i]}" propName="CO_NAME"
    varName="objName"/>
[...]
```

A user may enlarge an image displaying a location by clicking on it. In this case, a separate window is opened showing the image. A user may navigate from the

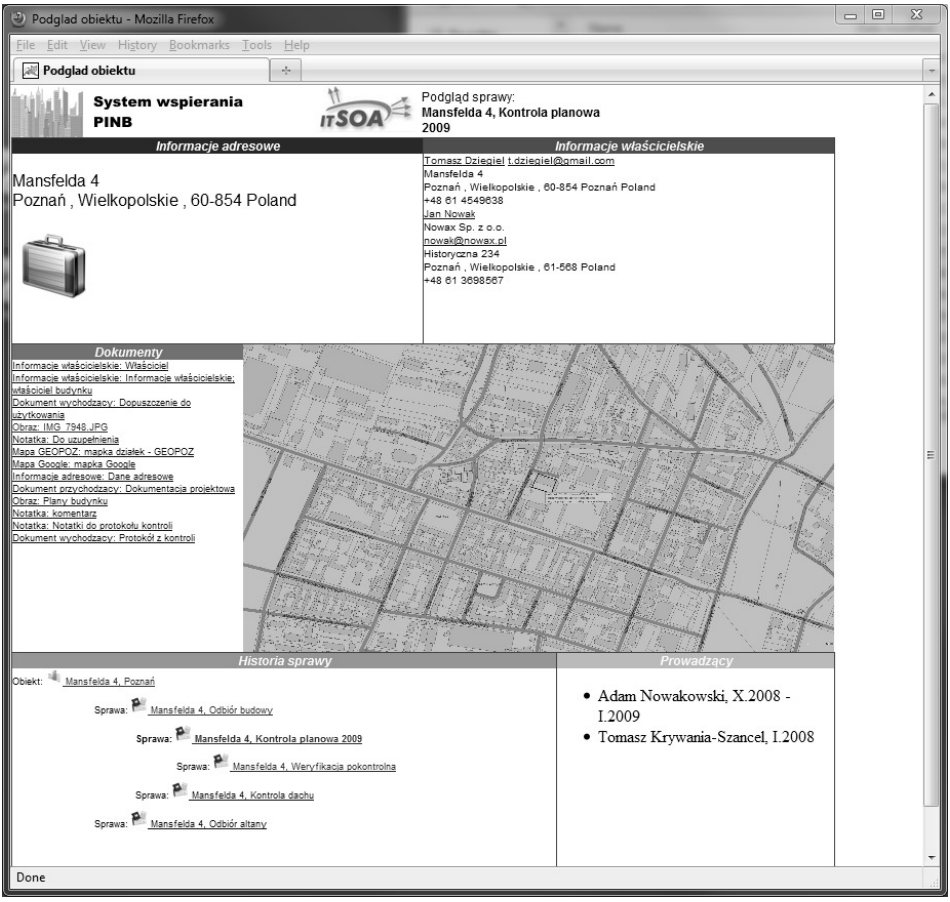

Fig. 5. A window presenting a case

location window (by clicking on hierarchy links) or from the search result window to a page presenting a case (Fig. 5). The page consists of the following sections:

— Information about the address (police data) with briefcase;
— Owner data (retrieved from court registers);
— List of associated documents;
— Map retrieved from a geodesy office;
— Hierarchy of cases;
— Responsible people.

A briefcase is an element permitting a user to select a particular case to be

displayed on the mobile interface. If the briefcase is red, it denotes that the case will not be presented on mobile interfaces; blue briefcase denotes the opposite. Clicking on the briefcase a user may switch between these two states. By enabling the briefcase on selected cases, a user creates a list of cases that will be displayed on his/her mobile device, i.e. that will be accessible outside of the office.

A SOIL tag responsible for managing the state of the briefcase is `<pssFlag>`. It is used in the case template in the following way:

```
[...]
<pssFlag coId="0" method="GETCO" flagName="BRIEFCASE"
  userId="{#uid}" varName="v"/>
[...]
```

The briefcase switching is performed in a separate template and the result is included in the case window by the following code:

```
[...]
<evaluate>
  <iframe frameborder="no" height="100px"
    src="{@pfPath}pfid={#pfid}&amp;domain=PINB.TECH&amp;
      co={#co}&amp;uid={#uid}" width="100px">
  </iframe>
</evaluate>
[...]
```

The part of the SOIL template responsible for maintaining the state of the briefcase is the following:

```
[...]
<if condition="{$set==1}"><then>
  <pssFlag coId="{#co}" method="SET" flagName="BRIEFCASE"
    userId="{#uid}" varName="v"/>
</then></if>
<if condition="{$set==0}"><then>
  <pssFlag coId="{#co}" method="UNSET" flagName="BRIEFCASE"
    userId="{#uid}" varName="v"/>
</then></if>
<pssFlag coId="{#co}" method="CHECK" flagName="BRIEFCASE"
  userId="{#uid}" varName="v"/>
[...]
```

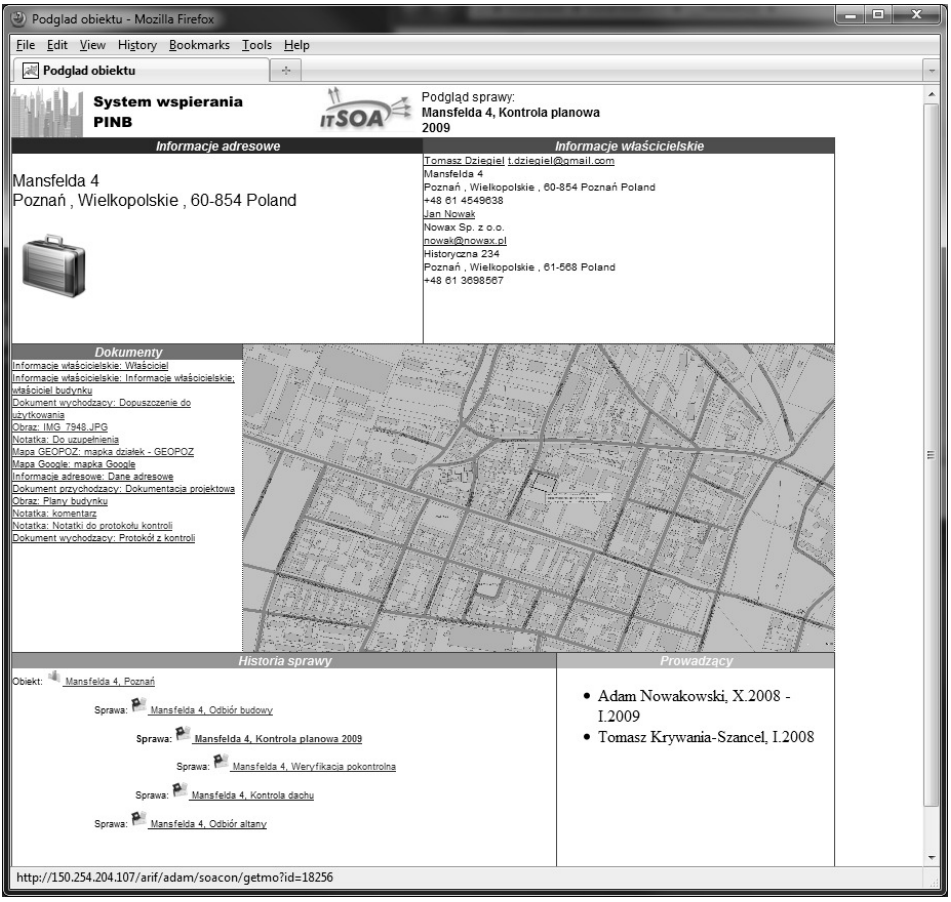

Fig. 6. A window presenting a case with a briefcase on

In Figure 6 a case is presented with a briefcase switched on. Thus, this particular case will be displayed on a user's mobile device.

A user may interact with the content by selecting a document from the list of documents associated with the current page, navigating to other cases or clicking on a map in order to enlarge it.

All interfaces may be customized by a user. The level of customization depends on the designer of the template. A customization affects either interfaces for all locations/objects in a particular presentation domain or a single interface only. An example of a customization is the change of the background color in all interfaces

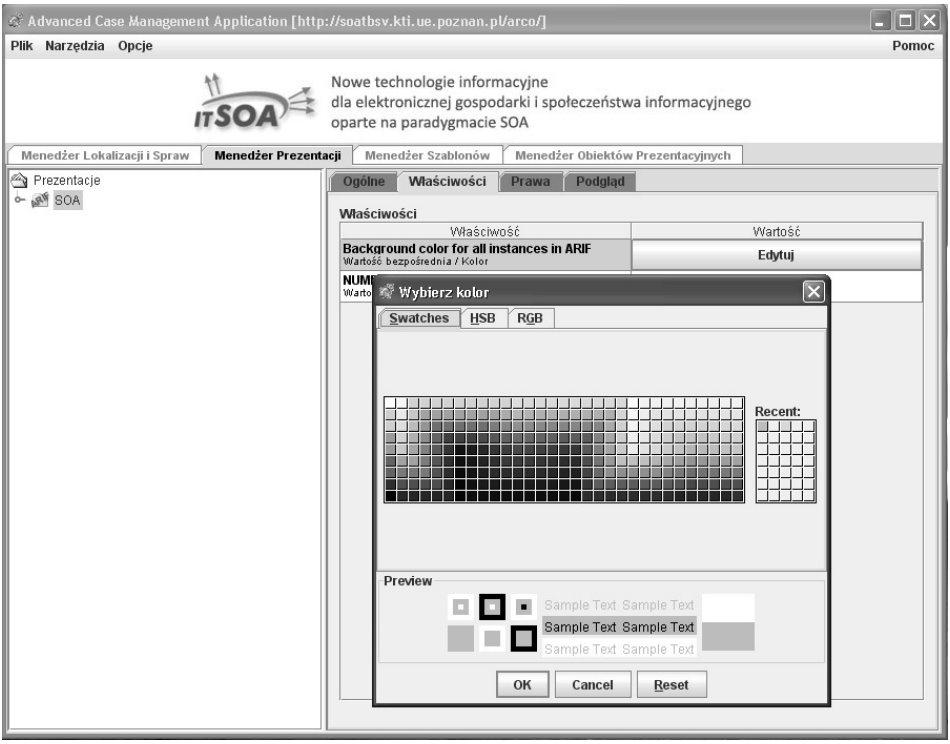

Fig. 7. Change of a parameter in ASIS Content Management Application

within a given presentation domain (e.g. desktop). A dedicated property is defined on the level of publication folders. The property denotes RGB color of the background. A user changes it using PSS instance of the ASIS Content Management Application (see Fig. 7). The SOIL template tag used for retrieval of parameters of publication folders is `pfProps`. The code below presents usage of the tag in the template:

```
<pfProps pfId="{#pfid}" propName="BGCOLOR" varName="pfbg"/>
[...]
<style>
  body { background-color: <insert value="{@pfbg}"/>; }
</style>
```

An example of color selection is presented in Figure 7, while in Figure 8 the result of the customization is shown.

Fig. 8. Case page after user customization

5.2 Templates for Mobile Interfaces

Mobile devices are characterized by a small display area, limited connectivity, and low interaction possibilities. Therefore, special templates that allow a user to interact with the content had to be prepared. In most cases, for each class of devices a special template has to be prepared.

When accessing the PSS system from a mobile device, a user may see a list of cases that have been previously selected by the use of the briefcase icon. The list

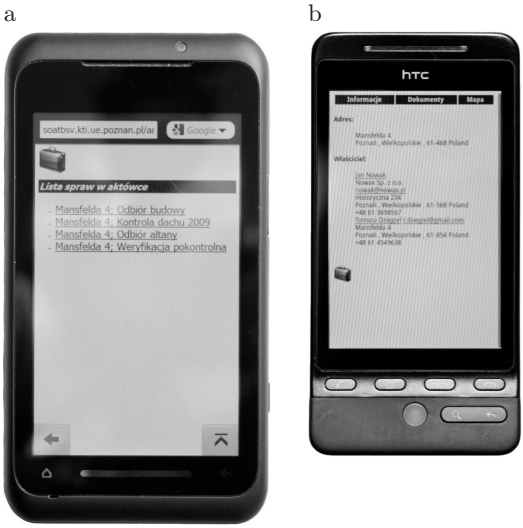

Fig. 9. Visualization interfaces on different mobile devices displaying: a — list of cases (on left), b — information about a case (right)

is displayed on a device in a simple way adapted to the current size of the display (see Fig. 9a).

A fragment of SOIL template responsible for generating such a list is presented below.

```
[...]
<pssFlag coId="" method="GETCO" flagName="BRIEFCASE"
  userId="{#uid}" varName="colist"/>

<set name="listsize" value="{sizeOf(@colist)}"/>
<span class="hd1">Sprawy w aktówce:</span>
<ul>
  <for from="0" name="i" to="{$listsize-1}">
    <coProps coId="{@colist[$i]}" propName="CO_NAME"
      varName="objName"/>
    <evaluate>
      <li>
        <a href="{@pfPath}pfid={#pfid}&amp;
          domain=PINB.INFO&amp;co={@colist[$i]}">
          <Insert value="{@objName}"/>
      </a></li>
```

```
      </evaluate>
    </for>
  </ul>
  [...]
```

If a user wishes to see a case he/she can click on a link. Then the user is presented with a case window, adapted to the current device. For instance, on a device with a small display and limited interaction possibilities (no scrolling), the entire content is divided into three tabs (see Fig. 9b) eliminating the need to scroll long pages.

6 PSS ASIS Content Management Application

The PSS implementation of the ASIS Content Management Application is an application specially designed and implemented to allow managing data processed and stored within the PINB Support System. The same application is used also to manage ASIS Service Interface Generator (SIG) objects and configuration.

The PSS ACMA application consists of four main independent but inter-related managers:

— the Location and Case Manager,
— the Presentation Manager,
— the Template Manager, and
— the Template Object Manager.

The Location and Case Manager is targeted at PSS user and is used to manage all kinds of data related to locations and cases processed and stored by the DAS. The other three managers are targeted at ASIS system administrator. The Presentation Manager and Template Manager are used to prepare, store and manage ASIS presentations and presentation templates (SOIL). The Template Object Manager is used to manage all kinds of multimedia data used in ASIS presentations but not related to locations or cases. These managers are described in details in Sections 6.1, 6.2, 6.3 and 6.4.

There is also one technical tool: the Configuration Manager, which is an advanced tool used for creating and editing technical SIG configuration sets. It is described in details in Section 6.5.

6.1 Location and Case Manager

The Location and Case Manager enables administering the content objects and media objects processed and stored by the DAS. There are two kinds of content objects (CO): location and case. Each location or case can contain multiple media objects (MO). COs are stored in a hierarchical structure of folders.

Content objects — locations represent real locations of buildings identified by the PINB, while content objects — cases represent all cases run by the PINB for given buildings (locations). Media objects are digital versions of information elements available for each location or case: documents and multimedia files. Examples of information elements processed by the PSS ACMA application are: inbound document, outbound document, note, address info, owner info, maps, and photos.

A view of the Location and Case Manager is presented in Figure 10. The Location and Case Manager window is divided into two main panels. The left panel contains a tree presenting the structure of folders; folders can contain locations and/or cases. The tree is organized as follows. The tree root can contain folders only. Each folder can contain other folders, locations and cases. Each location or

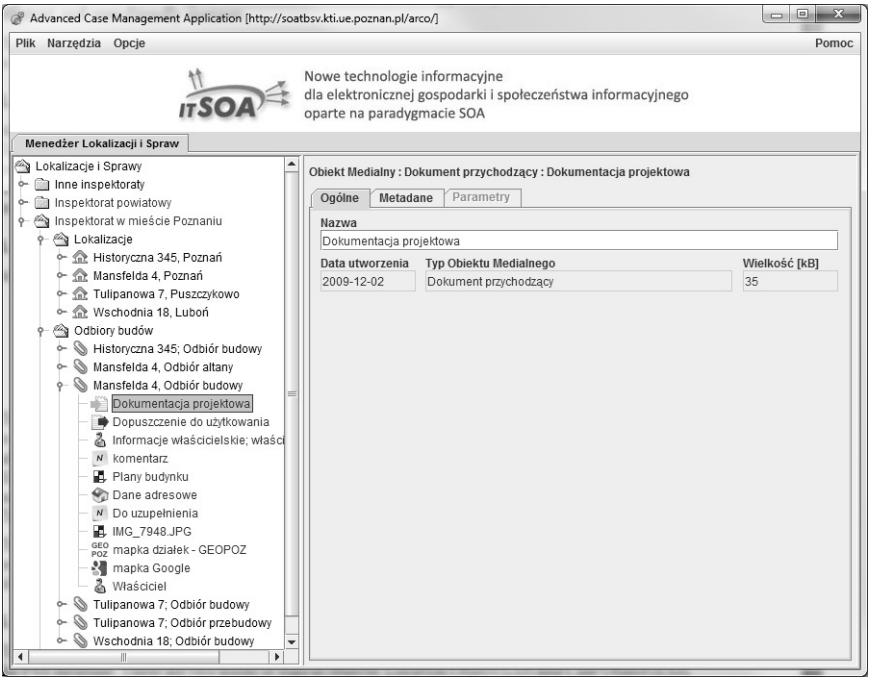

Fig. 10. Location and Case Manager

case can be associated with a number of media objects of different types. The name of any tree element may be highlighted in red, which means that it is used in some ASIS presentation. There is an icon next to the name of the object. It represents the type of the object (— location, — case) or the type of the media object. Icons for media object types can be defined in a configuration file.

Every case is related to a location or another case. Such relations represent a hierarchy between location and cases. Every case can contain its own media objects and media objects inherited from its parent in the location/case hierarchy. An inherited MO icon is grey.

Detailed information about each selected item in the tree is displayed in the right panel. For folders these are the name and the description.

For locations and cases there are three tabs containing the following data. In the 'General' tab there are: the name of the location or case, the creation date, and the list of folders containing this location or case. In the 'Metadata' tab, there is the Metadata Editor: a tree representing object metadata (it can be edited via popup menus for adding and deleting tree elements and via a double click editor for every tree node). In the 'Case Hierarchy', a visual presentation of the hierarchy of cases for a given location is contained. A sample Case Hierarchy is presented in Figure 11.

In the case of a media object, there are three tab panels: 'General', 'Metadata', and 'Parameters'. The first ('General') tab panel contains general information: the name (editable), the creation date, the media object type and the media object size in kilobytes. The 'Metadata' tab panel holds the metadata tree (similarly as for locations and cases). The 'Parameters' tab panel provides access to technical object parameters.

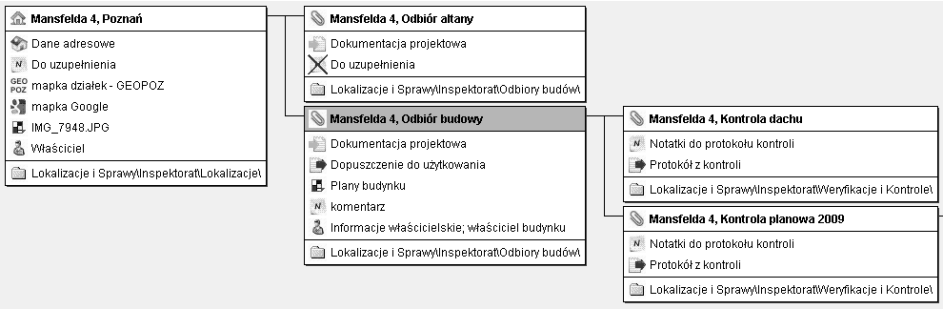

Fig. 11. Full view of a Case Hierarchy

6.2 Presentation Manager

The Presentation Manager allows administering the presentation structures and presentation template instances used by the SIG. The Presentation Manager is shown in Figure 12.

Presentation structures are defined by a hierarchy of publication folders containing content objects and template instances. Multiple template instances corresponding to different domains can be assigned to a single publication folder.

Each template can have parameters, which can be set in its instances. If values for all template parameters are set, the result is a rigid presentation. Otherwise, i.e. if not all parameters are provided, the end user will be able (in case of non-required parameters) or will have to (in case of required parameters) to provide the parameter values when the presentation is about to be shown. Optionally, some parameter values are gathered from the invocation context, device capabilities or user preferences.

The Presentation Manager is divided into two main panels. The left panel contains a tree presenting the structure of folders, template instances and content

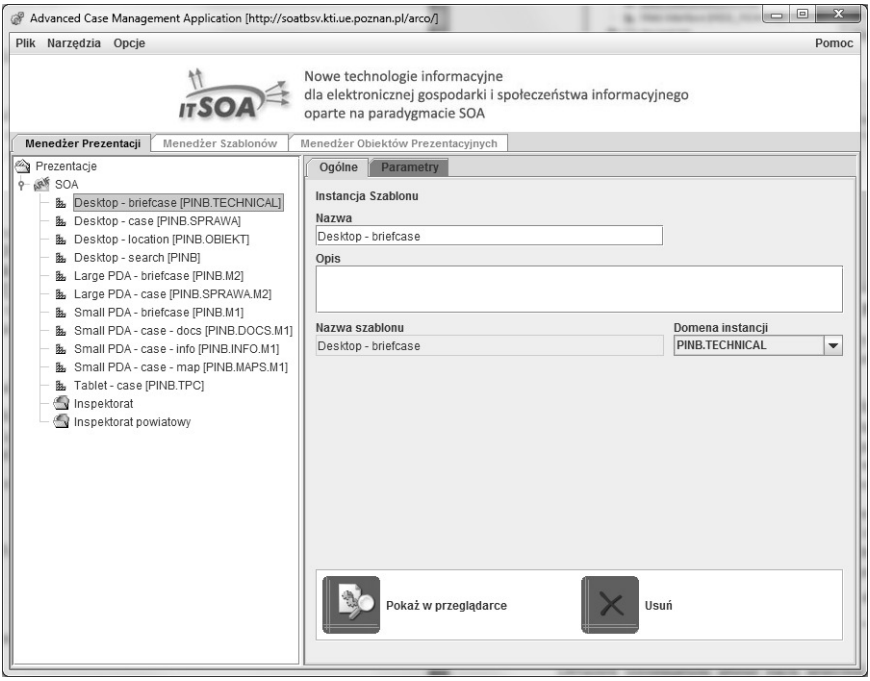

Fig. 12. Presentation Manager

objects. The tree is organized as follows. Each publication folder can contain other publication folders, template instances, content objects and content object folders. Each template instance can have a number of embedded template instances. There is an icon next to the name of each object. The icon represents the type of the object (— presentation, — publication folder, — instance of a template, — inherited instance of a template).

Detailed information about each selected item in the tree is displayed in the right panel. For publication folders, these are the name and the description. For template instances — the name, the description, the template name and the instance domain. In the case of a content object, the same information as in the Location and Case Manager is presented.

For template instances, a tab panel named 'Parameters' is displayed containing information about parameters, their names, types and values in a tabular form. This is the place where values of template parameters can be set. The value can be either a number or a string or a certain object (e.g. media object, template object).

For publication folders, content objects and media objects, a tab named 'Properties' appears. It enables managing the visualization properties of publication folders, content objects and media objects. Value of a property can be a number, a string or a certain object.

There is a popup menu available for each item in the tree. It provides actions for managing all components of a presentation: publication folders, template instances, content objects, and media objects.

6.3 Template Manager

The Template Manager allows administering the SOIL presentation templates used by the SIG. The Template Manager is divided into two main panels. The left panel contains a tree that presents the structure of folders, templates and their parameters. The tree is organized as follows. The root of the tree contains folders and templates not assigned to any folder. Each folder can contain other folders and templates. Each template can have a number of template parameters.

Detailed information about each selected item in the tree (such as the name, the description, etc.) is displayed in the right panel. For templates, these are the name, the description and a table with the presentation domains where the template can be used. In the case of a template parameter — the name, label, data type, data type description, and the default value can be accessed. Moreover, for templates there is another tab (named 'Parameters') which contains information about parameters, their names, types and values in tabular form. This is the place where default values of parameters can be set. The default value can be either a number, a string, or a certain object (e.g. media object, template object).

There is a popup menu available for each item in the tree. It provides actions for managing folders and templates.

6.4 Template Object Manager

The Template Object Manager enables administering template objects used by the SIG. Template objects are all kinds of multimedia data that can be used in presentations, but which are not semantically related to content objects. Examples of template objects are images for page background, audio and textual descriptions. All template objects are organized in a hierarchical structure of folders. Each template object can be assigned to more than one folder.

The main window of the Template Object Manager consists of two panels. The left panel contains a tree representing folders and template objects. The tree is organized as follows. The root of the tree contains folders. Each folder can contain other folders and template objects. There is an icon close to the name of each template object. The icon represents the MIME type [16] of the object. Each template object belongs to one of the template object types. Parameters associated with the type are represented as child nodes in the tree.

There is a popup menu available for each item in the tree. It provides actions for managing the folder hierarchy and template objects.

The right panel shows detailed information about the currently selected node in the tree. For nodes representing folders, the name and the description are presented. In the case of template objects, the name, the description, the type and the MIME type are presented together with the preview of the template object. In the case of a parameter of the template object type, additionally the value of the parameter is displayed.

6.5 Configuration Manager

The Configuration Manager is a tool for editing the configuration data used by the ASIS Service Interface Generator. It allows creating, deleting and copying different configurations (configuration categories). Within a configuration it is possible to add a new configuration item and edit or delete existing items.

6.6 Technology Used

The PSS ACMA application is implemented in Java. It runs as a standalone application. Data are stored in an Oracle10g [17] database. To access the data stored in the database the PSS application uses web services provided by an application

server. The application server is implemented in Java and runs on Tomcat [18] with Axis [19] libraries. A user interacts with the PSS application by means of an intuitive graphical user interface (GUI).

7 Conclusions

The ASIS solution for building adaptable user interfaces for SOA applications, presented in this paper, provides a flexible way of accessing SOA services. Due to the fact that the ASIS Service Interface Generator generates the final service interface on-demand and takes into account the capabilities of a particular front-end device, the preferences and privileges of end-users, and additional context information like location, it is possible to provide a user with a final interface in the most appropriate form for a given request, without explicit actions of the user. The method of adjusting service interfaces is not bound to any specific set of services or underlying business logic, permitting the ASIS system to be used as a front-end interface generation middleware with virtually any system based on the SOA paradigm. Examples provided in this paper are related to the construction industry, however, the method is general and can also be successfully used in other application domains.

References

1. Bih J. Service oriented architecture (SOA) a new paradigm to implement dynamic e-business solutions. Ubiquity, 1-1, 2006.
2. Martins A., Carrilho P., Mira da Silva M., Alves C. Using a SOA paradigm to integrate with ERP systems. Advances in Information Systems Development, Springer US, 2007.
3. Cellary W., Rykowski J., Chmielewski J., Walczak K., Wiza W., Wójtowicz A. Personalizacja dostępu do usług. ITSOA project internal report: ITSOA — Raport z prac wykonanych w ramach zadania OB3-5, 2009.
4. PINB — Powiatowy Inspektorat Nadzoru Budowlanego dla Miasta Poznania. `http://www.pinb.poznan.pl/`.
5. XML — Extensible Markup Language. `http://www.w3.org/XML/`.
6. HTML — HyperText Markup Language. `http://www.w3.org/TR/html4/`.
7. PDF — Portable document format. Document management — Portable document format — Part 1: PDF 1.7. ISO 32000-1:2008.
8. Java programming language. `http://java.sun.com/`.

9. XHTML — The Extensible HyperText Markup Language. http://www.w3.org/TR/xhtml1/.
10. SQL — Structured Query Language. Information technology — Database languages — SQL. ISO/IEC 9075.
11. Toshiba TG01. https://www.toshiba-europe.com/mobilerevolution/default.aspx.
12. Apple iPod Touch. http://www.apple.com/ipodtouch/.
13. HTC Touch Pro2. http://www.htc.com/www/product/touchpro2/overview.html.
14. Zarząd Geodezji i Katastru Miejskiego GEOPOZ. http://www.geopoz.pl/.
15. Google Maps. http://maps.google.com/support/?hl=en.
16. MIME — Multipurpose Internet Mail Extensions type. http://www.iana.org/assignments/media-types/.
17. Oracle database. http://www.oracle.com/us/products/database/index.htm.
18. Apache Tomcat — open source servlet container. http://tomcat.apache.org/.
19. Apache Axis — implementation of the W3C SOAP specification. http://ws.apache.org/axis/.

SOA-Based Multi-Server Agent System — Application for Integrated Rescue Action

Krzysztof Cetnarowicz[1], Tadeusz Dyduch[1], Jarosław Koźlak[1], Małgorzata Żabińska[1], Piotr Błaszczyk[3], Michał Niedźwiecki[1], Krzysztof Rzecki[2], Lev Belava[1], Grzegorz Wąchocki[1], Przemysław Bech[1], Jan Dziedzic[1], and Michal Ptaszek[1]

[1]Department of Computer Science AGH University of Science and Technology, Al. Mickiewicza 30, 30-059 Kraków, Poland
[2]Department of Teleinformatics Cracow University of Technology, Warszawska 24, 31-155 Kraków, Poland
[3]Faculty of Electrical and Computer Engineering Cracow University of Technology, Warszawska 24, 31-155 Kraków, Poland

Abstract. In this paper, we have presented an idea of agent-based SOA systems that use different solutions and mechanisms analysed in the multi-agent domain for providing a flexible construction and an execution of the business processes. We are especially focusing on the problem of composition and execution of the complex business processes containing web services as their elements. These may be performed thanks to negotiations inspired by multi-agent domains or thanks to a rule-based agent, which selects the most appropriate business processes described in BPEL for execution. We have chosen the management and coordination of activities performed in the framework of the integrated rescue action to undertake a verification of the algorithms proposed and pilot tools implemented.

1 Introduction

The concept of the computation model based on interreacting services (Service Oriented Computation — SOC) and the architectonic models, dedicated to such computations are the research and realiZation domains widely explored nowadays. The grand expectations are linked with the SOA concept, especially in the domain of integrating the existing approaches as well as elaborating models, which make it possible to build complex applications, and having control of their complexity. This control of complexity is thanks to basing the works on the construction scheme, which contains service providers, service registers and service requestors (clients) and using numerous programming patterns.

It seems to us that the multi-agent approach might be profitable to support the development of SOA systems, and in particular, to provide the choice-of-best services function of the cooperation and leading the execution and monitoring of complex business processes. The multi-agent approach concerns the development of systems consisting of cooperating autonomous elements — agents. The application of artificial intelligence methods, while developing multi-agent systems, makes it possible to use the techniques of automatic planning, machine learning, working on knowledge expressed using advanced methods (such as ontology languages) or complex interaction protocols.

The aim of our work was to propose a concept of an SOA agent system and create prototypes for testing fragments of functionalities and chosen tools. SOA (Service Oriented Architecture) systems being the main subject of our interest comprise of a great number of services deployed into many servers distributed between many different locations. Within the framework of functioning of the mentioned SOA systems, execution of complex services takes place. A complex service process consists of ordering and realizing many inter-related (logically and functionally, etc.) services.

To manage such sets of services, and provide the possibility to realize complex scenarios of activities when using them, it is necessary to work out flexible solutions, which enable a proper choice of services to collaborate. It is necessary both to match services of proper functionalities (described by the sort of data supplied as an input and obtained as an output) as well as to guarantee they perform actions required from the point of view of the scenario logic. For example, if we search for a place in a hospital for a victim of an accident, e.g. an injured person with a given class of damage, then s/he will be directed to the hospital specializing in these sorts of injuries, which is able to supply necessary resources and data (thus, the needed resources are available at an institution delivering a given service; whereas if they are not available, it is necessary to look for a more suitable institution, which offers appropriate resources of a particular type).

To create a system of such properties we intend to apply the following solutions:

— multi-agent approach, which should give a choice of services or business scenarios, that offer the most satisfactory solution of a given problem;
— perform conversations with services to select such a service, that best fulfils our needs, negotiates conditions of its execution (i.e. carrying out the arrangement process), as well as monitors the construction and running of business processes, to enable reaction to changing situations;
— create a complex description of notions, useful for services operations, as well as relations between them; we plan to use ontology language OWL [26, 27] and ontology OWL-S [21] for this purpose.

2 Comparison of SOA Systems and Multi-agent Systems

In ([23], p. 298) the basic limitations of SOA are indicated. They are related to the lack of a grouping function — grouping services having functions in consent with given needs, into appropriate layers. This is a result of introducing a concept xSOA (extended SOA) [25, 24], based on distinguishing the three layers: basic (offered by classic SOA, comprising of the architecture and description of services and simple operations), composing (responsible for matching, coordination and the description of transactions), management and monitoring. The currently created solutions for a basic layer are based on such standards as SOAP, WSDL, UDDI, WS-Notification and WS-MetaDataExchange. We plan to concentrate our work on solutions for these two higher layers. The existing standards, which may be applied here (WS-Transaction, WS-Coordination, WS-DistributedManagement, WS-Policy, WS-PolicyFramework) solve only some tasks, which the layers have to face. It seems that research in the domain of multi-agent systems, and the worked out solutions may be of a great help. Current solutions of SOA on the composition level are limited to orchestration with the use of BPEL [30] and preliminary proposals of choreography with the use of WS-CDL [7].

There are numerous definitions of agents and multi-agent systems, the wide overview of them is given in [12]. Among others, our group also participated in these works during a 15 years period and formulated some definitions and models [22, 11, 31, 8]. One of the agent definitions is the definition given in the work [28], which concerns the application of the multi-agent approach to Service Oriented Computing: "An agent is an active computational entity that has a persistent identity; can perceive, reason about and initiate activities in its environment; can communicate with other agents, including humans".

There are many similarities between agent architectures and SOA but agents extend services considering several important elements [28]. In contrast to services, agents are often self-aware, learn and build models concerning other agents and their capabilities. As a consequence, agents may construct cooperating groups or coalitions. It is a feature useful in SOA systems, where an important problem to solve is finding services matching well with one another. Another feature of agents is that they participate actively, initiating communication while the services are passive and reacts only from being called.

In [20] three possible kinds of relations between agents and services were identified and analysed. In the first approach, the agents use existing, simple services. The role of the agent may be, for example, the automatic composition of services. In the second approach, the services are the agents with limited capabilities, however still having agent features such as autonomy, proactivity and persistence. The third approach assumes that agents are built from web services, which are their composing elements.

In [14] the authors mention fundamental properties and characteristic features of agent approach, which are closely related to Service Oriented Computation (SOC) and Service Oriented Architecture (SOA):

— ontology (representation of a knowledge domain, supporting interoperability);
— models of processes (representation of actions and their realization);
— choreography (simplified business protocols, services may cooperate within their frames);
— catalogues and facilities (intermediary agents);
— contracts of service levels and criteria describing the quality of service.

The authors postulate that the problems present in the SOC approach cannot be solved without the application of concepts and methods worked out as a result of research on the multi-agent approach. Such a crucial meaning may have: an approach guaranteeing persistence of software solutions, application structures (Framework) supporting society management (such as TEAMS — Task Analysis and Environment Modeling System) as well as ensuring realization of goals owing to bonds of different strength.

3 Multi-server Complex SOA Systems

In the previous sections the SOA systems, multi–agent systems and the relations between them were presented. This section focuses on a given type of the SOA systems and gives a preliminary description of a problem domain, to which we are going to apply SOA systems with presented features.

The kind of SOA systems which we would like to focus our attention on, are large-scale systems with a substantial degree of distribution, where services are available on many servers deployed in the Internet. The essential problem, which may be faced when realizing such a multi-server system, is related to its execution as well as the management and control of complex business processes. We propose an execution of a decentralized multi-server system with the use of an appropriately constructed platform. The platform has additional servers to monitor a decentralized multi-server system. The servers will be named execution-monitoring servers. The above-mentioned execution-monitoring servers enable successive switching on the servers and verifying their correct actions according to their roles in the decentralised multi-server system. After fulfiling its role, the execution-monitoring servers are successively withdrawn from the realized system. Finally, we consider support of our system by one of ESB (Enterprise Service Bus) [6] realizations, which might supply some of the communication-management functions. There are still complex functions left, connected with the domain problems of the realized application.

As the domain of application for such a multi-server system we selected a management of integrated rescue action. Let's consider the witness of the accident, which informs the Crisis Management Centre (number 112) about it. The witness plays the role of the client and the Crisis Management Centre is the service provider in this interaction. The Crisis Management Centre informs further service providers about the accident: Dispatcher of Fire Brigade, Dispatcher of Medical Emergency Services, Duty Officer at the Police. As the rescue action develops, it deploys numerous subjects cooperating with one another in the client-server interaction scheme, however the clients initiating a given action or a subsequent state of the rescue action process (for example, witness of the accident, and after the Crises Management Centre) does not afterwards participate in future activities. As a consequence, the current concept of the client-server application needs to be extended into the form of a service client/service provider. The solution of this problem may be obtained by applying Peer2Peer technology, however, such a solution is not fully satisfying. The characteristic feature of the P2P solution is that it consists of elements, which play roles of both the client and the server. The idea of introducing some differences between client and service (in the played role and its structure) is justified and clarifies the designing, realization and functioning of systems composed from numerous clients and server. However, quite often one can notice situations where clients send requests to the server to perform given services and for this, the servers have to send on the next requests to other servers, that are necessary for fulfiling the client request. So, in these cases, the servers also play the roles of clients and work as a P2P application.

We have a similar situation in the case where a given client addresses a server to perform a service whose results should be delivered to another client, so the service is realized in the interest of another client. For the client, which initiates the process, it is only important whether, the service was successfully performed and not the result returned by it. Afterwards one can examine a situation where the result of a request sent to the server by the client is that another client has addressed another server to perform a service. In conclusion, the problem is to determine what should be the result of a service for which the client addresses a server.

One can consider the following results of realization of the services whose execution was requested from server S1 by client C1:

— the returned message, created only by the server — service provider S1 and sent to the client C1;
— the message created by the server — service provider S1 and other servers participating in the performing services for the client C1;
— the message created only by the server - service provider S1 and returned to the other client C2, indicated by the client C1;
— executed action client-server between client C2 and server S2 (client C1 obtains only the confirmation that the service was performer by server S1).

The extension of the activities of the client-server interaction protocol presented above may be realized using the functionalities provided by agents. For example, one can consider a server, which provides the following service: after the request of the client it creates an adequate agent and executes it in a suitable environment. The choice of the kinds of agent and environment depends on the task to be performed in the given system.

4 Approach to Create Multi-server Agent SOA Systems

In comparison to classic SOA systems, an agent SOA system, differs in:

— clear differentiation of two phases of the service choice, which are as follows:
  - a choice of services of compatible interfaces either by classic matching or by the use of a semantic matcher and an automatic planner, maybe in collaboration with patterns of business processes;
  - a querying of matching services so as to choose these, which assure the expected quality of results from the point of view of an agent that initialises and monitors a business process;

— presence of agents, whose role is querying services about abilities to satisfy needs, or making decisions (on the basis of such techniques as a rule-oriented system or machine-learning) on the choice of the most suitable business process to be executed.

According to the applied approach, a process of service realization may be divided into the following steps:

— *Identification of a service.* During this phase, a potential client finds services of the anticipated profile. The result of this phase is information whether services of a searched profile are located in the environment surrounding the mentioned client, as well as obtaining information on how to find the needed service (e.g. the address of the appropriate repository of services, etc.). Thus, the effect is a selected group of services, which may be considered as potentially useful from the point of view of the clients needs.
— *Negotiation with the service.* During this phase, services belonging to the selected group of services are checked from the point of view of a definition of how the mentioned services are suitable for needs of the considered client. The result of the phase is one chosen service, the most appropriate for client's needs. The process of negotiation may be performing using multi-agent interaction protocol like Contract Net [29] or one of different auction protocols.
— *Service arrangement.* During this phase, the ability to execute the certain service, with the parameters (conditions) given by the client is negotiated with

a service-provider. Usually the result of this stage is to inform the provider about real conditions of the service execution, collecting the appropriate resources by the provider and preparation of appropriate algorithms to execute the certain variant of the service, and informing the client about it.

— *Service execution contract.* As a result of the phase, a client accepts (or not) conditions of service execution by a service provider and concludes a contract to execute a certain service. There are conditions of service execution fixed in the contract, consequences of failure of service execution, as well as guarantee conditions.

— *Service realization.* A service is realized by a system, or in the real world (then, a computer system supervises service execution). A service provider obtains in this case, the information about the beginning, the ending, a success or a failure of service execution.

— *Service billing.* In this phase, both a client and a service provider negotiate and fix conditions of the service, i.e. whether it is performed satisfactorily, or whether there are shorcomings, potential correction of the obtained result, a due payment or a negotiated reimbursement.

Key problems, which are a part of the subject of research, are the following:

— The automatic creation of a proper plan leading to aims defined by a client. It may be carried out by applying methods of automatic planning [13], while assuming that the set of services are actions that can be possibly executed and that the requests received from the client represent the goals to be achieved.

— Selection of appropriate services, which may realize the plan. Choice of services should take place on two levels: a compositional consent of services interfaces and the usability for the performed domain scenario. For the former, we plan to take into account the classic approach based on repositories (such as UDDI) and the use of language for a service semantic description (such as OWL and OWL-S). On the second level, it is necessary to converse with a given service and get to know which resources it is able to make available and which resources to reserve.

— Monitoring of an execution plan and remembering a status of an action. Business processes of our interest have a complex nature; it is possible to distinguish some autonomic fragments, which may be executed concurrently and independently to some extent. Thus a task of automation of the process of composition and decomposition of single business processes appears.

5 Application of an Exploratory Start-platform

The developed platform has the following special features:

— Comprises of special agent servers, which provide an environment for agents.
— Constitute an infrastructure for the implementation of tools for arranging services, as well as direct negotiation of service details on servers (all or chosen).
— Contains special monitoring servers, which enable running a network of cooperating servers.

Services. Services are installed on distributed servers owned by different organizations or providers with a some decisional autonomy. After the arrival of the request, it becomes necessary to prepare and execute a business process, which will communicate with distributed services necessary for successful execution of the request.

Agent Environment. Cooperation of servers in complex actions (e.g. rescue actions) is aimed at the realization of given actions on the basis of mutual communication. The action mentioned is performed by executing numerous appropriate tasks in a client-server system. Roles of a client and a server are often changed and a server playing a role of a service provider becomes a client and communicates with another server — provider to realize a necessary service. A status of the action having been realized is represented by states of appropriate collaborating clients and servers. In particular, there is one or there are some selected servers participating in the mentioned action, which remember the action state (more or less in detail). We are going to analyze scenarios of executing complex actions, when it is not possible (or it does not make sense) maintain availability of servers, which take part in an action, during the whole time. It is not possible to distinguish a server or a group of servers, which remember the distributed state of the business process during its complete execution. It seems that the application of the concept of the autonomous agent to solve this problem might be justified. The performed action (whole or in some fragments) is connected with agents acting in an environment composed of a network of collaborating servers. Since a server is not always adjusted to make an environment of agents, it is proposed to introduce a group of special servers into the system, which are able to play the role of environments for the agents.

Tools for Arrangement. Negotiation of services that enable a choice of an appropriate one makes up the basic element of the SOA systems functioning. Usually, a client searches through appropriate service repositories and analyses information about available services. After making a choice of a proper service, it turns to an appropriate service provider – a server, to realize a chosen service. Such a manner

of arranging services is called indirect. However, a certain service may not always be realized at that very moment. Moreover, additional arrangements to precise conditions of service execution may be necessary. In such a case, a client may perform an exchange of information with a service provider, that defines whether a given service may be executed at the very moment and what current conditions of the service execution are. The mentioned exchange of information is called an arrangement and requires the use of appropriate means. For this purpose, Entish language [5, 4] was proposed which enables the above-mentioned arrangement of services. In the multi-server system performing a complex action, tools for realization of service arrangement process have been implemented.

Monitoring Servers. A complex action realized by a network of servers – a multi-server system, requires cooperation of many servers at different phases of performing the action. Triggering the multi-server system, which requires causing simultaneous cooperation of appropriate servers is especially complex. To make it easier, as well as to perform a proper maintenance of the system, the introduction of special monitoring servers has been proposed. A task of monitoring servers, performed or supported by agents, is supervision and monitoring (tracing) communication in the framework of service realization. After having been used, some of the monitoring servers (not useful any more) are withdrawn from the system. Some of them are left and they enable maintenance and development of the system.

6 Description of the Integrated Rescue Action Scenario

The application domain of the developed agent SOA system [10] is a management and a coordination of the integrated rescue action [9]. We assume that the action may be performed in a consequence of large-scale accidents which cause different losses: numerous wounded people with different injuries, damaged vehicles, persons jammed in cars, blocked roads or contamination of the environment. In the rescue actions performed in such conditions, different services have to participate: an ambulance service, the police, a hospital and the activities of these services is the task for the developed agent SOA system.

Below, one of the scenarios of carrying out the integrated rescue action is presented. The case with several injuries was selected.

In the first stage of the operation (Fig. 1), the person reporting the accident informs about the event, when calling 112 and asking for help. The dispatcher, who receives the call, introduces data concerning the accident into the system with the help of an interface. Here, the integrated rescue system starts functioning. After the receipt of the call/request, the system creates the *Agent responsible for the*

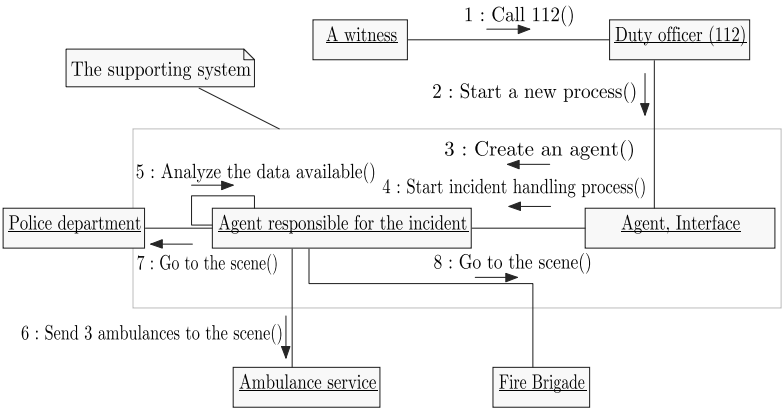

Fig. 1. A scenario of a rescue action for an accident with multiple injuries. The first stage

incident, which is responsible for solving the problem and carrying out the rescue action.

The agent responsible for the incident, after the analysis of data decides whether to send a police team, an ambulance and a fire brigade, which handles the problem of the neutralization of the chemical substances, to the location of the event.

The doctor from the ambulance, after arrival to the place of the event, performs the examination of the injuries and for each injury, an *Agent Responsible for the Patient* is created, which is responsible for future aid. For the severely injured, agents will organize necessary places in the hospitals and will provide adequately speedy transport. Light injuries will be delivered to hospitals for future examination using ambulances present in the place of the event or additional ambulances, which will be sent if necessary. It is important that each patient is served by an independent agent, whose goal it is to organize help/aid for them. In the event of a high number of injuries, agents may be in competition. Each of them is led by criteria associated with the state of injury, the price and others. For example, the agent responsible for light injuries selects slower but less expensive transport whereas the agent responsible for severe injuries does not take the price into consideration and selects the fastest transport. The ambulance service itself may offer different kinds of services for different prices. It gives a guarantee that the help will be provided for the injuries with a correct order, the most serious injuries with the highest priorities.

Another aspect of this scenario is the synchronization of actions (Fig. 2). A reorganization of the traffic onto side roads by the police should last as long as the last services leave the place of the accident.

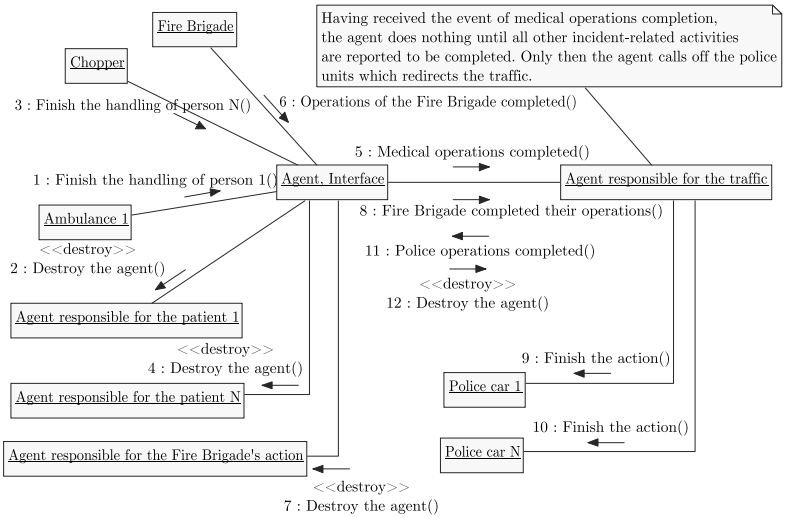

Fig. 2. A scenario of a rescue action for an accident with multiple injuries — synchronization of activities

In the example of the organization of the detour, a synchronization of agent activities is presented. Namely, the *Agent responsible for the traffic* does not finish the traffic change until all necessary actions at the place of the accident are ended. Despite the end of medical actions — all injuries are delivered to hospitals and all agents responsible for serving the injuries are removed from the system, the agent does not restore the normal organization of the traffic. It is aware that operations of police and fire brigade are still in progress. Only when all these services have ended their activities, the agent will restore the previous traffic organization and inform the police cars about the end of operation. At the end, the agent notifies/reports the end of rescue action and is removed from the system.

7 Developed Tools

Prototype systems and tools have been created in the frames of carried out works. They offer, on the one hand, fragments of functionality necessary to create a final system, and on the other, enable testing the most important, from our point of view, elements of the system as well as checking existing, available complex tools and approaches. In particular, our work comprises of the following:

— a dynamic choice of business processes described by BPEL; for this, an agent equipped with a rule-oriented inference machine was used;
— a demonstration of a protocol of arranging services applying Entish environment;
— an analysis of abilities of basic tools needed for manipulation on semantic Web services, using an ontology/ontological description, and their composition, thus creating a concept of a semantic repository, as well as an analysis and a choice of external complex tools which might be used in a semantic layer being realized, thus a matcher (to match services to queries/questions) and a planner (for a dynamic composition of Web services and/or business processes);
— creation of a prototype version of the system for management of an integrated rescue action. Its aim was, on the one hand, to offer an evaluation environment for tools having been created, and on the other hand collecting the necessary experience, required by the process of creating mature system versions.

There are descriptions of the above-mentioned solutions and tools presented below in brief:

Library of Arranging Services. The library of services arrangement based on a solution from IPI PAN-Siedlce has been worked out. This arrangement consists of questioning services: whether they are able to deliver the required results, and making a choice of such a service, which offers the best solution. Both agents and services may take part in a conversation. Entish language is used for this purpose.

Log Server. The created tool is a server of logs: LogServer. Its task is to trace messages, their sequence and contents and their visualisation in the form of dynamically drawn sequence diagrams. LogServer collects logs from all elements of the system and stores them into a database. Then the logs are made available by LogView which analyses them and builds sequence diagrams based on them.

Monitoring Server: MapServer. It is a tool, which enables storing the information about the status of system elements. Visualization of a map together with a network of roads, units with available and occupied resources, events and moving cars is done. The tool has been created for the purpose of examination of a problem of an integrated rescue action. However, it has a wider application; it may be useful for creating a wide class of systems related to optimizing the realization of transport requests, modeling of city traffic, or the optimization of solutions of other logistic problems (e.g. supply chains management).

Agent Server. The agent server is a tool being an agent environment, which enables the creation of subsequent instances of different agents, realization of their algorithms, check/ control, communication, etc. A pilot version of the environment

has been implemented. The server of agents is a relatively simple and universal tool. It consists of two servlets externally available:

— AgentServer — a client who orders an agent to perform a task communicates with the servlet;
— ClientService — ServicesServer communicates with it when it sends back an answer in a phase of an arrangement or an execution.

A Pilot Version of a System for management of an integrated rescue action has been realized. The system uses and evaluates the above-mentioned tools: Library of services arrangement, Agent server, Map Server, Server of logs. An environment based on tools proposed in this phase of a project is a universal motor, which enables both: modeling to test a concept, as well as realization of real systems.

The environment comprises of the five following elements: a client, an agent server, services servers, a repository of services and monitoring servers.

A client orders agents server to perform a service, transferring a name of a function (type of a service) and a document with parameters. A new instance of an Agent is created in an agent server; it takes care of the execution of the order. We can distinguish two phases in an action of an agent's instance: an arrangement phase and an execution phase. The aim of an arrangement phase is to find servers offering the ordered service. The aim of an execution phase is to select one server, which may execute the service, order its execution of the service and collect results. In the phase of execution an agent analysis received answers. It compares prices of execution and chooses a server, which offers the cheapest one. The choice of a service in the subsequent phases may be done with the use of many criteria. Next an agent sends an order to the selected server (confirmation of willingness), and to others, which were not selected, but have not returned a rejection at the same time, it sends a message about resignation from the service. The service server, when receiving an order to perform a service, executes it using the preliminarily reserved resources, and next it returns an answer to an agent. A service server, which received information about resignation from the service, frees the earlier reserved resources.

Rule-Based Agent-Manager of Business Processes. A tool offers the ability to compose Web services with the help of a reactive agent, which manages a choice of execution of web processes using a rule-oriented system based on the Drools platform [1]. An environment for an agent is a web interface, web services and BPEL processes seen also as web services.

Functions of the system comprise of:

— Collection of a query from a user, by a web interface. We assume different possibilities of communication with the system, for example a user friendly web interface made available to a dispatcher.

— Web interface provides an agent with the preliminary data, which is the base of facts, fundamental for a definition of the sequence of processes and web services execution.
— Exchange of data between web services is performed on a level of a knowledge base rules.
— Results of web services execution are added to the fact base and may later become a basis to call other web services or may be transferred as an argument for operation call.
— A Web service of an agent returns a log of executed operations consisting of rules, which were active in a certain case as well as web services called by certain rule.

The Semantic Infrastructure for Search and Composition of Web Services

The designed semantic infrastructure is based on the following:

— Implementation of its own modules, which makes, as a whole, a semantic repository of services.
— Use of existing, examined solutions [19, 18], or application of solutions created by other subgroups of an IPIPAN team [15]. In particular, it concerns a planner and a matcher, maybe also other tools guaranteeing conversion between different formats. Such a choice has already been done and they are used in other works.
— Creation of a set of a domain ontology in OWL, for an integrated rescue action and a description of web services concerning a rescue action, in OWL-S. A pilot version of such ontology, which makes a basis for further works has been created.

The designed repository comprises the following key elements:

— A transport layer — a repository has to have ability to communicate remotely in any indicated technology.
— A communication interface with an authorization system — it is a key element of the whole repository, because it integrates all the elements available in the repository, it triggers a transport layer, serves the user authorization, communicates with a matchmaker's adapter and with an OWL/RDF repository.
— A matchmaker's adapter — it was decided, that the repository for service matching uses existing implementation of matchmakers, so it is necessary to create an adapter which will separate the repository from a certain implementation of matching algorithms and owing to this it gives the opportunity to change a matchmaker into another one in the future.
— A repository OWL/RDF — this function will be performed by the Sesame [2] system, which integrates a reasoner in itself, and enables execution of requests on the stored data files. It stores the data files related to services (OWL-S),

possibly added ontology and its descendants from processing by the matchmaker adapter.

So far, we examined and tested two existing tools, which might be used as a part of our system: the matcher and a planer. OWLS-MX [18, 17] is a hybrid semantic matchmaker, which matches services for a given query (question) written in OWL-S. Matchmaker uses Pellet [3] as a reasoner of a description logic (OWL-DL). The aim of the application of the planer is to use it as a tool for the service composition. We intend to apply different techniques of dynamic service composition, use of automatic planners and treating a service as an action with a given input and output is one of these techniques. We have concentrated on the OWLS-XPlan [19, 16]. It is a program (distributed on an Open Source license) for generating plans, thus a chain of calls of web services on the basis of the given information. The program receives a description of a current state described in OWL notation, and a description of an expected state, also defined in OWL. Additionally, it is necessary to give a list with services descriptions, which may be used (services are also described in OWL).

Ontologies. The created domain ontology currently concerns a sub-domain of a rescue-emergency domain. We have based the creation of the ontology on our own knowledge in the area of the domain including the general knowledge, and real documents as a card of patient transfer and an order for emergency transport. The created preliminary version of the ontology has 96 classes, 23 properties of objects (possible relations), 9 properties of data types. Preliminary descriptions of parts of services in WSDL and OWL-S have also been prepared.

8 Conclusions

The carried out work has concentrated on:

— working out elements of architecture of an agent SOA system;
— testing basic solutions which enable composing and arrangement of services;
— preparing a scenario of an integrated rescue action;
— realization of prototype systems reflecting the chosen scenarios of functioning of the system to manage an integrated rescue action.

The hitherto performed works to check different techniques and tools, have been carried out in various directions, only partially having their common parts. We plan to integrate the examined solutions in the future. To make the process of integration of different elements of the system easier, we plan to use one of the ESB platforms, extended by the examined solutions.

Further works will be carried out in the following way:

— extension and development of an agent architecture based on services (SOA-A);
— enrichment of scenario descriptions for an integrated rescue action, taking into account existing documents filling in currently by certain units, regulations and rules currently being in force;
— development of pilot versions with wider functionalities and tending to their integration;
— further examination of existing solutions and tools, as well as implementation of complex tools needed in the next steps;
— work on semantic Web services, more detailed description of larger sets of such services and the necessary domain ontology, as well as on the use and integration with the rest of the system;
— analysis of the problem of cooperation between BPEL and OWL-S. There is research work on the subject, however obtained results are preliminary ones and have a limited character;
— carrying out experiment to use abilities offered by Web Services Choreography Description Language (WS-CDL), which is the subject of works, and describes a pattern of messages exchange between Web services taking part in a business cooperation.

References

1. Drools — JBoss Community. 2009, `http://www.jboss.org/drools/`.
2. openRDF.org: home of sesame. 2009, `http://www.openrdf.org`.
3. Pellet: the open source OWL 2 reasoner. 2009, `http://clarkparsia.com/pellet`.
4. Ambroszkiewicz S. Entish: a language for describing data processing in open distributed systems. Fund. Inform. 60, 1–4, 41–66, ISO Press, 2004.
5. Ambroszkiewicz S. enTish: an approach to service description and composition. Instytut Podstaw Informatyki Polskiej Akademii Nauk, Warszawa, 2003.
6. Binildas C.A. Service Oriented Java Business Integration. Enterprise Service Bus integration solutions for Java developers. Packt Publishing, Birmingham, 2008.
7. Web Services Choreography Description Language version 1.0. W3C Candidate Recommendation, Nov. 2005, `http://www.w3.org/TR/ws-cdl-10/`.
8. Cetnarowicz K. M-agent architecture based method of development of multiagent systems. Proc. 8th Joint EPS-APS Int. Conf. on Physics Computing, 591–596, ACE Cyfronet, Kraków, Mar. 1996.

9. Cetnarowicz K., Cetnarowicz E. Multi-agent decentralised system of medical help. Management and Control of Production and Logistics, IFIP, IFAC, IEEE Conf. Grenoble, France, 2000, ENSIEG, LAG Grenoble, 2000.
10. Cetnarowicz K., Dyduch T., Koźlak J., Żabińska M. Koncepcja zastosowania podejścia agentowego w systemach soa. Automatyka 13, 2, 231–238, AGH, 2009.
11. Cetnarowicz K., Nawarecki E. Systeme d'exploitation décentralisé réalisé a l'aide de systemes multi-agents. 3eme Journées Francophone sur l'Intelligence Artificielle Distribuée et les Systemes Multi-agents, St Baldoph, France, 1995.
12. Franklin S., Graesser A. Is it agent, or just a program? A taxonomy for autonomous agents. Proc. 3rd Int. Workshop on Agent Theories, Architectures and Languages, Budapest, Hungary, 1997.
13. Ghallab M., Nau D., Traverso P. Automated planning. Theory and practice. Morgan Kaufmann Publishers, San Francisco, 2004.
14. Huhns M.N., Singh M.P., Burstein M., Decker K., Durfee E., Finin T., Gasser L., Goradia H., Jennings N., Lakkaraju K., Nakashima H., Van Dyke Parunak H., Rosenschein J.S., Ruvinsky A., Sukthankar G., Swarup S., Sycara K., Tambe M., Wagner T., Zavala L. Research directions for service-oriented multiagent systems. IEEE Internet Computing, IEEE Computer Society, 2005.
15. Jarocki M., Niewiadomski A., Penczek W., Półrola A., Szreter M. Towards automatic composition of web services: abstract planning phase. tech. rep., ICS PAS, Warsaw, 2010.
16. Klusch M. OWLS-XPlan. tech. rep., Feb. 2010, `http://www-ags.dfki.uni-sb.de/klusch/owls-xplan/`.
17. Klusch M. The OWLS-MX project web page. tech. rep., Feb. 2010, `http://www.w3.org/TR/owl2-overview/`.
18. Klusch M., Fries B., Sycara K. Automated semantic web service discovery with OWLS-MX. ACM Press, AAMAS, Hakodate, Japan, 2006.
19. Klusch M., Gerber A., Schmidt M. Semantic web service composition planning with OWLS-Xplan. Proc. 1st Int. AAAI Fall Symp. on Agents and the Semantic Web, Arlington, Virginia, AAAI Press, 2005.
20. Martin D., Burstein M., McIlraith S., Paolucci M., Sycara K. Owl-s and agent-based systems. Extending Web Services Technologies. The Use of Multi-Agent Approaches, 53–78, Springer, Berlin, 2004.
21. W3C Member Submission. Semantic markup for web services. tech. rep., Nov. 2004, `http://www.w3.org/Submission/OWL-S`.
22. Nawarecki E., Cetnarowicz K. Decentralized intelligent systems with multi-agent model. Proc. 11th IASTED Int. Conf. Applied Informatics Annecy, France, 1993.
23. Papazoglou M.P. Web services: principles and technology. Prentice Hall, New York, 2008.
24. Papazoglou M.P. Extending the service oriented architecture. Busin. Integr. J. 18–24, 2005.
25. Papazoglou M.P., Georgakapolus G. Introduction to the special issue about service-oriented computing. Commun. ACM 46, 10, 24–28, 2003.

26. OWL Web Ontology Language. W3C Recommendation, Feb. 2004, http://www.w3.org/TR/owl-features/.
27. OWL 2 Web Ontology Language. W3C Recommendation, Oct. 2009, http://www.w3.org/TR/owl2-overview/.
28. Singh M.P., Huhns M.N. Service-oriented computing. Wiley, New York, 2005.
29. Smith R.G. The contract net protocol: high-level communication and control in a distributed problem solver. IEEE Trans. Comput. C-29, 12: 1104–1113, 1981.
30. OASIS Standard. Web services Business Process Execution Language version 2.0. tech. rep., Apr. 2007, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf.
31. Żabinska M., Cetnarowicz K. A study of agents' orientation in decentralized information system. Proc. 1st Int. Workshop: Decentralized Multi-Agent Systems DIMAS'95, LIFIA-IMAG, II AGH, Kraków, 1995.