OPHELIA
Software Development Tools Integration Technology*

Maciej Hapke, Andrzej Jaszkiewicz, Krzysztof Kowalczykiewicz,
Dawid Weiss, Piotr Zielniewicz
Instytut Informatyki, Wydział Informatyki i Zarządzania, Politechnika Poznańska,
{hapke, jaszkiewicz,krzysztof.kowalczykiewicz, dawid.weiss,
piotr.zielniewicz}@cs.put.poznan.pl

Long and painful is the road to success in software development – use shortcuts!

Abstract

This paper is a summary of the OPHELIA project, a tool integration platform for distributed software development. We present basic concepts of the project and some of the novel approaches that proved to be interesting from software engineering point of view. We describe project's outcomes and outline the key future perspectives.

1. Introduction

Contemporary software development methodologies define numerous artifact types like error reports, source code, schedules, test cases or documentation. Development tools aid in creating and managing these software engineering artifacts. Unfortunately very often these tools are offered by different software vendors and they vary significantly with the scope and functionality. They are also specialized, offering aid in a specialized area of software engineering like requirements management, software model design, project management, documentation management or risk assessment and control. Finally, it is a rare case for a software vendor to offer a broad range of tools for all of these different

* OPHELIA is a project co-funded by the 5th Framework Program of European Union - IST-2000-28402/ IST-2000-28402D - and jointly developed by a consortium of academic and industry partners. http://www.opheliadev.org

domains that could cooperate and exchange information across between different project areas.

Meanwhile one of the most frequent problems with software development is the lack of consistency between its artifacts. For example, keeping source code and UML model in sync is a nightmare of every programmer unless the case tool has reverse engineering capabilities and integration with an IDE environment. The same problem applies to updating documentation, test cases, schedules and other project elements. It is in general the problem of reacting to changes introduced to the project and propagating those changes on related elements. The additional effort this operation requires, unless supported by integrated tools, is usually higher than the payoff; that is the main reason most software projects abandon documentation and UML models at certain point of complexity or their lifecycle. Tight development tools' integration can help in keeping the software project consistent and prevent proliferation of redundant (and obsolete) project artifacts by automating some of tedious tasks related to change propagation and notification. Unfortunately, what we have already pointed out, most software vendors provide specialized, black-box products that lack wide integration facilities.

All these problems need resolution to improve the way software projects are run. Two possible approaches could be distinguished. One of them is to develop a set of tools that build complete unified and distributed environment for managing all software artifacts. Some vendors already provide solutions of such flavor [WWW2003a,c,d]. Commercial software packages are usually of limited availability to small or mid-size companies because of their price (on the other hand, one could argue whether such small companies really need complex distributed development aid). Open source packages and solutions, such as [WWW2003c] require shifting the development process to specific tools, which is sometimes an obstacle.

An alternative approach is to develop a framework that could bring together *existing* software development tools from different domains and create an abstraction layer consolidating them and adopting them for use in an enterprise environment. The abstraction layer here would not only create a bundle of different tools, but also provide a 'global' view of project artifacts, regardless of the tool used to create them, and an additional functionality of relationship tracking and event notification. The OPHELIA project is an example of this second approach.

The OPHELIA project has been created and developed by a number of partners from software and academic fields. Our work and contribution to the project has been focused on creating traceability subsystem and integrating it with change notifications. The challenging task was to define the level of inter-artifact links on the abstract level that OPHELIA provides, without the knowledge of what the underlying tools might be.

In the following part of this paper we describe the basics of the OPHELIA project and our experiences with the abstract tools layer and tools integration. Chapter 2 compares several existing approaches to tools integration. Chapter 3 describes the OPHELIA project approach in more details, presenting basic architecture, components and integration strategies. The following chapter outlines possibilities the OPHELIA technology could enable – a new class of tools making use of the abstract access layer to project artifacts. Chapter 5 presents an implementation of OPHELIA technology – Orpheus solution that

has been developed to prove the concept. Chapter 6 presents project curriculum and concepts evolution throughout the project period followed with the chapter on future plans and perspectives for continuing project effort.

2. Other development tools integration efforts

There are already some solutions on the market that provide different software engineering tools integration in more-or-less limited form. Feature comparison matrix has been shown in the Table 1.

2.1. Rational Suite

Rational Suite [WWW2003a] is a toolset including acclaimed modeling and requirements management applications. The suite includes most of the software engineering tools developed by Rational. As a toolset provided from a single vendor the integration between different applications is very tight. They share their data, allow relationship tracking and provide some traceability services. With software configuration management mechanisms provided the toolset supports distributed development.

Rational Suite is one of the most mature solutions on the software engineering tools market. Years of development made this environment industry standard. Unfortunately price of this solution is quite high and may not be acceptable for many, especially mid- and small software companies.

2.2. Eclipse

Eclipse [WWW2003b] is a development environment created initially in IBM laboratories and contributed as free, open-source product to the developers community. This tool represents new approach to the IDE (Integrated Development Environment) concept. Eclipse is rather a platform for embedding different tools, not only purely development-related, but also for modeling, configuration management, documentation generation etc.

The Eclipse project has been widely acclaimed and has been in active development ever since its public release by IBM. Many new plug-ins appear, both free ones and commercial. It seems to be a very good platform for building toolsets and integration services. At the time of writing, distributed development is supported in this platform only using common versioning repository for development files.

2.3. Sourceforge Enterprise

Sourceforge website [WWW2003c] became recently one of the centers of open-source development. Its services host over 60000 projects (May 2003).

A version of Sourceforge called Sourceforge Enterprise [WWW2003d] is available for commercial institutions. It consists of a web-based collaboration environment extended with project management, requirements tracking and several other software engineering modules.

2.4. Feature comparison

Table 1. Comparison of tools integration solutions

Feature	OPHELIA (Orpheus)	Rational Suite	Eclipse	Sourceforge Enterprise
Open integration platform	Yes	No	Yes	No
Open source	Yes (except of project management module)	No	Yes (with many commercial plug-ins)	Yes/No
Distributed development	Yes	Yes	No	Yes
Requirements management	Yes	Yes	No	Yes *
Project management	Yes	No	Yes *	Yes
Bug/issue tracking	Yes	Yes	Yes *	Yes
Repository access	Yes	Yes	Yes	Yes
Relationship tracking	Yes	Yes *	No	Yes *
Modelling support	Yes	Yes	Yes *	No
Metrics support	Yes	Yes	No	Yes *
Development environment	Yes *	Yes	Yes	No
Testing support	No	Yes	Yes *	No
Process definition	Yes *	Yes	No	Yes

* - limited support

3. OPHELIA - tools integration, unification and abstraction environment

OPHELIA is an international European Union – co-founded project running within the 5th Framework Programme – Information Society Technologies domain. It started in

September 2001 and is due to finish in September 2003. The aim of the project is to build a unified software engineering tools integration technology. The concept behind the project involves defining standardized set of interfaces abstracting functionalities of different kinds of software development tools. These unified interfaces are access points to the specific tools behind them. This abstraction layer would make OPHELIA environment adaptable to any set of tools. To maintain implementation language independence CORBA technology has been chosen for defining these interfaces.

A number of most popular tools categories have been chosen and for each of them a CORBA interface has been defined representing its common internal object model. All these interfaces define the OPHELIA technology. They are described in a set of documents called Module Interface Specification. Base OPHELIA technology includes the following MIS documents:

- Kernel Services Module Interface Specification
- Requirements Module Interface Specification
- Modeling Module Interface Specification
- Project Management Module Interface Specification
- Metrics Module Interface Specification
- Bug Tracking Module Interface Specification
- Repository Module Interface Specification
- Knowledge Management Module Interface Specification
- Traceability Module Interface Specification
- Notifications Module Interface Specification

Depending on the software configuration in a given environment, suitable OPHELIA instances could be created. As different tools may expose different set of functionalities the effort needed to OPHELIA-enable them may differ. The approach is to develop a plug-in that exposes application's functionalities through CORBA interfaces. Because each OPHELIA module becomes in the end a server of a certain artifact type, the underlying tools must have the ability to work in client-server mode. For desktop tools missing this functionality, an additional effort is required to develop the server part of such software. These approaches are summarized in the Table 2.

Table 2. Integration of development tools with OPHELIA platform

Type of tool	**Steps needed to integrate into OPHELIA**
Desktop tool	Server component needs to be written (or reused) implementing appropriate MIS interface. Plug-in for the software needs to be written to communicate with the server.
Client-server tool	Plug-in for the server part of the software needs to be written implementing MIS interface.
Web tool	Adapter needs to be written to access the data in the tool's database through module's MIS interface.

To enable multi-project and multi-user environment additional interfaces have been defined for managing projects and user accounts as well as security mechanisms. These compose OPHELIA kernel services interfaces. The architecture of an OPHELIA solution is presented in Figure 1.

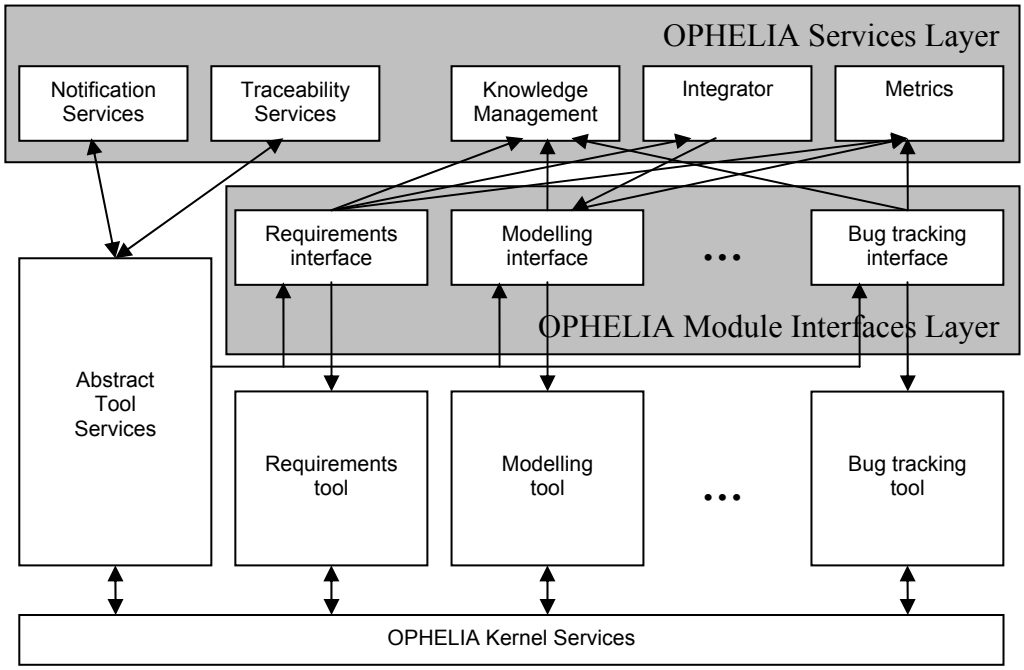

Figure 1. Architecture of OPHELIA platform

Summarizing OPHELIA itself as a technology is only composed of several CORBA interfaces that should be implemented to build OPHELIA instance. In other words, OPHELIA technology could be compared to a bus in computer architecture forming a base for co-operation of various modules.

4. OPHELIA services – utilizing tools abstraction

With different tools' interfaces defined, new services can be built on top of the existing abstraction layer. These services do not need to rely on any particular tools that lay behind these interfaces. This abstraction creates uniform view on the software artifacts and possibilities emerge to create new kinds of services [KOW2002]. A few of such possible services are listed in this chapter.

4.1. Integrators

An example of the new service type that can be built using OPHELIA technology is an integrator-type application. Such application can extract data from one tool (using its OPHELIA interface), process it automatically or semi-automatically (with user assistance) and create new project artifacts utilized by different tool type. This type of operation facilitates transferring data from one type of tool to another while minimizing the human effort required for such task.

An example of such integrator software could be a wizard that migrates the requirements defined in the project directly to use cases within the modeling tool. Depending on the software process reverse action could be taken as well. Another example of such integration process is an integrator for creating schedule tasks using information about classes and use cases. Relationships defined within modeling tool between these objects affect precedence constraints of the tasks created [HAP1999], [HAP2000]. Integrators that only migrate the data are of course of limited usefulness since they amplify data duplication and synchronization problems mentioned at the beginning. We think that once we have the abstraction layer, the integrators can be pushed further toward intelligent, automated synchronization agents. These, working behind-the-scenes, agents would be responsible for keeping artifacts in sync among all components of an OPHELIA instance.

4.2. Abstract Tool Services

The MIS interfaces define a unified and public way of accessing project artifacts. With this transparent layer it is possible to provide services for tracing relationships and receiving events from any project element, regardless of the tool used to create or maintain it. Within OPHELIA this layer is called an ATS (Abstract Tool Services). Using the ATS services it is possible to retrieve basic artifact properties and subscribe to events fired from it. The ATS forms the base for other OPHELIA services.

4.3. Traceability

Traceability is one of the key aspects of the OPHELIA project. Traceability involves defining and managing relationships between any kinds of objects in the OPHELIA environment. This constitutes a brand new value in a distributed multi-tool environment.

Table 3. Types of relationships in OPHELIA

Type of relationship	**Description**
Internal explicit	Some tools may manage relationships between their objects internally. To maintain their usefulness they should be transferred into OPHELIA traceability services in order to make them propagable.
External implicit	This kind of relations is created automatically by integrator-type applications. Any integrator used to transfer data from one tool to another should create traceability links between the items transferred.
External explicit	Traceability services allow defining explicit relationships. It is possible to select any two objects and create a traceability relationship propagating or not propagating the events.

With traceability relationships defined it is possible to determine the impact potential changes may have on the project by analysis of the graph of affected objects. Moreover it is possible to use traceability relationships to propagate events and supply consistency warnings to items (and users responsible for these items) that may be potentially involved in the change propagation chain.

All the objects in the OPHELIA environment and the relationships defined within traceability service make a global traceability graph of the project. This graph includes several types of relationships, which may be created in different ways (see figure 3).

4.4. Notifications

Using the notification service one can subscribe and receive information about any changes made to a particular project artifact. Artifact owners and managers can track the changes on objects *and* the set of depending objects easier than it was the case when performed manually. An example of such process could be a requirement change that is propagated through previously defined traceability links to the related use cases and then to their implementation files. This way a developer in charge of a given implementation file gets a notification when requirement is changed.

We have found out that the fact that no change to an artifact or its dependency graph gets unnoticed is both an advantage and a potential problem. When pushed to the extremes, the number of notification can exhaust any user's patience limits… A careful study of usability of notifications and traceability links is scheduled as part of the OPHELIA project, but the results are currently not yet available.

4.5. Knowledge management

Knowledge management module allows the user to search project artifacts of different types according to various search criteria. It monitors also changes made in project's artifacts and update project's documentation after each change. If automatic documentation generating facilities are available, this results in always up-to-date documentation available to project stakeholders.

What is particularly beneficial and distinguishes the OPHELIA project is that the documentation is generated using the ATS services, so the documentation-engine has no knowledge of what exact tool type it is accessing or how the artifact should be documented (the module that manages that artifact knows these facts).

4.6. Metrics

Unified events management in OPHELIA environment allows building software which can monitor these events and perform certain operations when predefined actions happen. An example of such a service can be an event-driven metrics services. Such services may monitor artifacts that are used for calculating project metrics. With any depending artifact changed or added, the metrics daemon receives an event notification and may recalculate metric value to its current state. It could be even possible to define alert thresholds that would notify project managers if the threshold is exceeded (for example, when module-cross-references exceed a given level).

5. Orpheus – OPHELIA solution example

One of the key tasks for the OPHELIA project was to create an instance of the OPHELIA technology. Example implementation was assumed to use free, open-source tools where possible. If such tools had not been available, they were to be developed. Commercial tools could be used as a last resort. To present the diversity of approaches for integrating different tools into OPHELIA solution the Orpheus system has been proposed. The components of this solution are presented in the Table 4.

Table 4. Orpheus solution components

Tool type	Orpheus implementation
Kernel services	custom implementation (free, open-source)
Requirements management	DRES (custom implementation, free, open-source)
Modelling	ArgoUML (free, open-source)
Project management	Microsoft Project (commercial)
Bug tracking	Bugzilla (free, open-source)
Metrics	The Metrix (custom implementation, free, open-source)
Repository	CVS (free, open-source)
Knowledge management	(custom implementation, free, open-source)
Traceability	Traceplough (custom implementation, free, open-source)
Notifications	(custom implementation, free, open-source)
Integrators	Modeling → Project Management Integrator (custom implementation, free, open-source)
Portal	custom implementation based on Apache Jetspeed (free, open-source)

Additionally, several utility software components have been developed to help future OPHELIA solution developers with their work. For instance, common storage repository has been developed to supply versioning XML storage facility for any tools not offering server counterpart.

To provide unified front-end for the users utilizing Orpheus solution a web-based portal application has been proposed. It provides users with a common startup point for their daily operations with Orpheus.

6. Project curriculum summary

The OPHELIA project is a joint research, development and evaluation effort. The intent was to provide OPHELIA technology specification, develop example solution that relies on this technology and then evaluate the product created in the software development companies there were part of the project consortium.

The project started in September 2001 with a debate on the OPHELIA architecture. Few approaches have been identified and discussed upon. These finally transformed into two alternative architectures to choose from. The first one was based on a centralized repository of all software artifacts. This repository would take care of events management, change detection and would provide common storage facility for all the modules. The second approach was to create a more distributed infrastructure with the tools utilizing their own storage facilities where applicable. OPHELIA interfaces would be implemented not by the repository, but by the server counterparts of the tools. As can be concluded from the architecture description in the previous chapters, the second approach has been finally chosen in December 2001.

Soon after the architecture debate new concepts arose to supply functionalities not described in the initial project description. It turned out that the modules abstraction OPHELIA will provide can be utilized to create new services for managing inter-object relationships and events management. This is how traceability and notification services concepts started in early 2002. Later, a portal application has been proposed to provide a front-end for the Orpheus user.

In the spring of 2002 module interface specification documents were prepared including initial interface definitions Orpheus solution should implement. It soon turned out that one of the major issues with these specifications is a level of objects granularity they should provide. At this stage events management facilities have been decided to include in the interface specifications.

Major development effort was conducted during summer 2002 resulting with early alpha version of the Orpheus solution in autumn 2002. Further development had led to the beta version released during spring of 2003. This version was passed to two companies – Azertia (Spain) and ICCC (Czech Republic) – for conducting evaluation stage. The two companies run test projects using Orpheus solution during summer-autumn 2003. Unfortunately no results were known at the time this paper has been submitted for publication.

7. Future plans

After OPHELIA project finishes the consortium plans to continue efforts to popularize the OPHELIA technology. Consulting services for companies desiring to adopt the technology are planned as well as supporting broader range of tools. The software created with the project effort will be free and will be contributed to the open-source community.

Starting from the beginning of the OPHELIA project it was obvious that this technology opens new possibilities for managing software projects and for developing new services utilizing software artifacts abstraction. Soon the scope became too broad and some of the aspects have been excluded from the OPHELIA project.

Members of the consortium have further research and development plans on extensions to OPHELIA platform. The potential research topics are:

- New tool types specifications (risk management, workflow, testing)
- Intelligent project monitoring software, detecting process disturbances, management reporting
- Workflow integration and process definition facilities
- Integrating with IDEs, e.g. Eclipse [WWW2003b] development platform
- Migrating to alternative distribution providers (web services)

8. Summary

The assumptions and achievement of OPHELIA project have been presented. OPHELIA platform constitutes a new approach to software development tools integration in distributed environment. Its main advantages are openness, low cost, support for distributed work in decentralized environment and the additional functionality like event-driven metrics or traceability, provided on top of the existing software development tools.

Within OPHELIA project a specific instance of OPHELIA technology has been built. At the time of writing the paper it is intensively tested in several projects run by partners of project consortium. It is expected, that, in the future, further solutions, both open-source and commercial, based on OPHELIA technology should be developed.

To some extend OPHELIA platform is a competitor to other development tools integration systems (see chapter 2). At the same time, however, due to distinct and often complementary approaches used in OPHELIA platform and other systems, new promising paths for further integration appear. For example, OPHELIA based solutions may be integrated with open IDEs, e.g. Eclipse. Existing, closed collaborative development environments, e.g. Sourceforge, may re-use OPHELIA technology in order to assure openness of their systems and take advantage of some OPHELIA-specific tools like traceability.

Bibliografia

[BOL2003] Boldyreff C., Dewar R., et al., *Environments to Support Collaborative Software Engineering*. 2nd Workshop on Cooperative Supports for Distributed Software Engineering Processes, Benevento, Italy, March 25, 2003.

[DEW2003] Dewar R., Smith M., et al., *The OPHELIA Traceability Layer*. 2nd Workshop on Cooperative Supports for Distributed Software Engineering Processes, Benevento, Italy, March 25, 2003.

[HAP1999] Hapke M., Jaszkiewicz A., Kominek P., Integrated tools for project scheduling under uncertainty (in Polish). Proceedings of 1st National Conference on Software Engineering, Kazimierz Dolny 11□13.10.1999, Informatyka Stosowana S4/99, Politechnika Lubelska, 65□76.

[HAP2000] Hapke M., Kominek P., Jaszkiewicz A., Slowinski R., Integrated tools for software project scheduling under uncertainty. In: P. Brucker, S. Heitmann, J. Hurink, S. Knust (Eds.) *Proc. 7th Int. Workhop on Project Management and Scheduling PMS'2000*, Osnabrueck, Germany, April 17-19, 2000, pp.149-151.

[HAP2001] Hapke M., Jaszkiewicz A., Perani S.: *OPHELIA - Open platform and methodologies for development tools integration in a distributed environment*. Proceedings of 3rd National Conference on Software Engineering, Otwock/Warsaw, p. 189-198.

[KOW2002] Kowalczykiewicz K., Weiss D., Traceability: Taming uncontrolled change in software development. Proceedings of 4th National Conference on Software Engineering, Tarnowo Podgórne, Poland, 2002.

[WWW2003a] http://www.rational.com/products/entstudio/index.jsp

[WWW2003b] http://www.eclipse.org

[WWW2003c] http://www.sourceforge.net

[WWW2003d] http://www.vasoftware.com/products