INNOWACYJNA GOSPODARKA
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI FUNDUSZ
ROZWOJU REGIONALNEGO

Program Operacyjny Innowacyjna Gospodarka: Działanie 1.3.1

Projekt: Nowe technologie informacyjne dla elektronicznej gospodarki i społeczeństwa informacyjnego oparte na paradygmacie SOA

Raport z prac wykonanych w ramach zadania OB2-1

Komunikacja grupowa dla usług sieciowych (etap I)

Za okres:

styczeń – maj 2009

Koordynator zadania obszaru badawczego:

dr hab. inż. Paweł T. Wojciechowski

ITSOA Sieć Naukowa Technologii Informacyjnych SOA

Spis treści

Informacje o dokumencie

Numer raportu	TR-ITSOA-OB2-1-PR-09-4
Koordynator	dr hab. inż. Paweł T. Wojciechowski
Autorzy	Paweł T. Wojciechowski (PP)
Poziom dostępności	Wewnętrzny konsorcjum
Słowa kluczowe	komunikacja grupowa, rozproszony konsensus, REST

Streszczenie

Systemy *komunikacji grupowej* dostarczają abstrakcje programistyczne oraz protokoły sieciowe, które umożliwiają budowę usług sieciowych o zwiększonej odporności na awarie. Zwiększona odporność na awarie jest osiągana przez replikację usługi sieciowej na kilku niezależnych serwerach połączonych siecią oraz zastosowanie protokołów komunikacji grupowej do koordynacji interakcji klientów z tymi replikami. Takie podejście gwarantuje, że usługa sieciowa widziana przez klientów jest odporna na awarie pojedynczych serwerów, które udostępniają tę usługę. Ogólny model takiej replikacji nazywany jest *zreplikowaną maszyną stanów*.

Raport zawiera podsumowanie prac wykonanych w I etapie projektu IT-SOA na Politechnice Poznańskiej nad zadaniem OB2-1, pt. "Komunikacja grupowa dla usług sieciowych". Wynikami tych prac są: (1) analiza mechanizmów komunikacji grupowej w systemach rozproszonych i opartych na paradygmacie SOA, (2) projekt i implementacja w języku OCaml protokołów komunikacji grupowej dla grup statycznych w modelu *crash-stop*, a także (3) koncepcja usługi komunikacji grupowej dla systemów opartych na paradygmacie SOA. W podsumowaniu zawarto wstępne motywacje, sformułowano problemy i cele badawcze oraz krótko scharakteryzowano uzyskane do tej pory wyniki. Szerszy opis otrzymanych wyników zawierają cytowane raporty techniczne.

1. Wstęp

Współczesne systemy rozproszone mają najczęściej strukturę, która wyróżnia dwa typy procesów: klienci oraz usługi. Każda usługa sieciowa oferuje klientom szereg operacji, które klienci mogą zdalnie wywoływać. Dana usługa może być udostępniona na jednym lub wielu serwerach (maszynach) połączonych z siecią. Zastosowanie pojedynczego, scentralizowanego serwera jest najprostszym sposobem aby zaimplementować usługę sieciową. Jednakże taka usługa jest odporna na awarie jedynie w takim stopniu, w jakim odporna na awarie jest maszyna (procesor), która wykonuje dany serwer. Jeśli taki poziom zabezpieczenia przed awariami nie jest akceptowalny, należy zastosować wiele serwerów realizujących tę samą usługę, które ulegają awarii całkowicie niezależnie.

Systemy *komunikacji grupowej* (ang. *group communication*) dostarczają abstrakcje programistyczne oraz protokoły sieciowe, które umożliwiają realizację powyższego scenariusza zwiększonej odporności na awarie. Zwiększona odporność na awarie jest w tym wypadku osiągana przez replikację usługi sieciowej na kilku niezależnych serwerach połączonych siecią oraz zastosowanie protokołów komunikacji grupowej do koordynacji interakcji klientów z tymi replikami. Takie podejście gwarantuje, że usługa sieciowa widziana przez klientów jest odporna na awarie pojedynczych serwerów, które udostępniają tę usługę. Ogólny model takiej replikacji nazywany jest *zreplikowaną maszyną stanów* (ang. *state machine replication*) [1]. Wyróżnia się przy tym awarie typu *crash-stop*, *crash-recovery* oraz *Bizantyjskie*. Dwa pierwsze typy zakładają, że serwer który uległ awarii po prostu przestaje działać, przy czym w drugim przypadku dopuszcza się jego reaktywację po awarii (ang. *recovery*) wraz ze stanem sprzed awarii. Trzeci typ awarii zakłada natomiast, że awarie mogą manifestować się w dowolny sposób i mieć charakter przejściowy, np. serwer może czasami nie odbierać lub nie wysyłać wybranych komunikatów. W naszej pracy interesują nas przede wszystkim awarie typu *crash-stop*.

Kluczową abstrakcją programistyczną służącą do implementacji modelu zreplikowanej maszyny stanów są różnego rodzaju *rozgłaszania komunikatów* (ang. *message broadcasts*). Rozgłaszania te gwarantują niezawodne dostarczanie komunikatów do wszystkich zreplikowanych serwerów, zachowując przy tym określony porządek dostarczania komunikatów, także w przypadku gdy awarii ulegnie część serwerów. W przypadku awarii typu *crash-stop* lub *crash-recovery* zwykle co najmniej połowa serwerów w grupie nie może ulec awarii. W przypadku awarii typu Bizantyjskiego, zwykle ta liczba wynosi dwie trzecie. Zapewnienie powyższych silnych gwarancji w systemie rozproszonym wymaga rozwiązania problemu *rozproszonego konsensusu* (ang. *distributed consensus*). W pierwszej połowie lat 80-tych udowodniono, że problem rozproszonego konsensusu w ogólności nie jest rozwiązywalny w systemie asynchronicznym, nawet jeśli tylko jeden proces (serwer) może ulec awarii [2]. Wynika to z faktu, że w systemie rozproszonym każdy z procesów może wykonywać swoje operacje w dowolnym tempie, wobec tego nie można rozróżnić procesu, który uległ awarii od procesu który jest po prostu wolny. Szczęśliwie rzeczywiste systemy rozproszone są częściowo synchroniczne, co umożliwia rozwiązanie problemu rozproszonego konsensusu. Podejście stosowane w praktyce polega na wprowadzeniu tzw. *detektora awarii* (ang. *failure detector*) [3], pozwalającego zidentyfikować proces, który uległ awarii. Decyzje podejmowane przez detektor awarii mogą być błędne, ale nie ma to wpływu na resztę procesów, które są uznane za działające.

Pomimo, że w ostatnich latach zaproponowano wiele algorytmów rozwiązujących rozproszony konsensus dla wszystkich trzech typów awarii, oraz wiele eksperymentalnych systemów komunikacji grupowej zostało zbudowanych w ośrodkach akademickich [4, 5, 6, 7, 8, 9, 10, 11], niewiele jest przykładów tego typu produktów rozwijanych komercyjnie [12, 13]. Po części jest to spowodowane faktem, że istnieje sporo różnych modeli i specyfikacji komunikacji grupowej, uwzględniających bądź nie dodatkowe aspekty, takie jak np. partycje w sieci. Istotne znaczenie może mieć też fakt, że nawet w przypadku awarii typu *crash-stop*, protokoły rozwiązujące rozproszony konsensus należą do najbardziej skomplikowanych protokołów sieciowych. W przypadku awarii Bizantyjskich to skomplikowanie dalej rośnie. Natomiast alternatywne podejścia do replikacji, takie jak

replikacja transakcyjna (ang. *transactional replication*) są relatywnie prostsze w realizacji i w użyciu, choć wolniejsze. Z drugiej strony firmy IT stosują protokoły komunikacji grupowej w innych produktach. Przykładowo Google używa algorytmu Paxos (rozwiązującego rozproszony konsensus) w implementacji usługi *Chubby* rozproszonych zamków, do utrzymywania spójności replik w przypadku awarii. Google zastosował tę usługę do implementacji własnego systemu bazodanowego. Podobnie IBM i Microsoft korzystają z algorytmu Paxos w implementacji swoich produktów (odpowiednio w implementacji usługi wirtualizacji dostępu do kontenerów pamięci oraz usługi zarządzania klastrem). Wynika z tego, że abstrakcje i protokoły komunikacji grupowej, w szczególności bazujące na rozproszonym konsensusie, stanowią ważny mechanizm budowy usług sieciowych.

Celem prac w ramach zadania OB2-1 jest zbadanie abstrakcji programistycznych oraz mechanizmów komunikacji grupowej pod kątem wykorzystania ich do budowy usług sieciowych zgodnych ze stylem architektonicznym SOA. Powszechnie podnoszonymi postulatami (wymaganiami) wobec SOA jest wsparcie integracji oprogramowania oraz wysoka dostępność usług sieciowych implementowanych zgodnie z tym stylem architektonicznym. Styl SOA nie specyfikuje jednakże jakie technologie powinny być zastosowane aby spełnić ten drugi postulat. Prace podjęte w ramach zadania OB2-1 pozwolą częściowo rozwiązać ten problem, pokazując przykładowe zastosowanie komunikacji grupowej do replikacji usług sieciowych. Aby zapewnić spełnienie pierwszego postulatu, tj. łatwej integracji oprogramowania tworzonego niezależnie, rozpoczęto prace w kierunku opracowania nowego interfejsu do komunikacji grupowej, zgodnego ze stylem REST. Styl *REST* (ang. *REpresentational State Transfer*) [14] to styl architektury oprogramowania w pełni zgodny z SOA, przeznaczony dla rozproszonych systemów hypermedialnych, takich jak World Wide Web (WWW). Popularne realizacje stylu REST opierają się na powszechnie używanym protokole HTTP. Opracowanie systemu komunikacji grupowej opartego o taki interfejs umożliwi koordynację dowolnych serwerów połączonych przez Internet, a tym samym może się przyczynić do szerszego stosowania tej technologii do replikacji usług sieciowych.

Prace badawcze nad zadaniem OB2-1, wykonane w ramach I etapu projektu IT-SOA, i uzyskane wyniki, zostały opisane w następujących raportach technicznych i artykułach:

1. RAPORT: *Analiza mechanizmów komunikacji grupowej w systemach rozproszonych i opartych na paradygmacie SOA* [15];
2. RAPORT: *Biblioteka komunikacji grupowej: Architektura systemu dla grup dynamicznych w modelu crash-stop* [16];
3. RAPORT: *Koncepcja usługi komunikacji grupowej oraz mechanizmu operacji atomowych dla systemów opartych na paradygmacie SOA* [17];
4. *A Brief Survey of Message Queueing Systems for Group Communication Middleware*. Artykuł został zgłoszony do recenzji na *Workshop on the Challenges of SOA Implementation and Applications*, organizowany w ramach międzynarodowej konferencji *WISE 2009: the 10th International Conference on Web Information Systems Engineering* [18].

2. Opis uzyskanych wyników

2.1. Analiza mechanizmów komunikacji grupowej w systemach rozproszonych i opartych na paradygmacie SOA

W pierwszej części prac w ramach zadania OB2-1 opisano model komunikacji grupowej, obejmujący podstawowe pojęcia związane z komunikacją grupową takie jak: własności i rodzaje kanałów komunikacyjnych, własności detektora awarii, problem rozproszonego

konsensusu, własności rozgłaszania z uporządkowaniem komunikatów (np. uporządkowanie kolejkowe, uporządkowanie przyczynowe oraz uporządkowane atomowe lub globalne), usługę członkostwa oraz wirtualną synchronizację. W kontekście tego modelu przeanalizowano szereg standardów, technologii i przykładowych implementacji komercyjnych narzędzi do budowy systemów rozproszonych, pod kątem semantyki komunikacji grupowej. Poniżej scharakteryzujemy krótko wyniki tych analiz; kompletny opis zamieszczono w [15].

W zakresie standardów *Usług Sieciowych* (ang. *Web Services*) wyróżnione zostały *WS-ReliableMessaging* [19] oraz *WS-BaseNotification* [20]. Pierwszy standard dotyczy jedynie komunikacji między dwoma procesami, ale specyfikowane przez ten standard gwarancje dostarczania komunikatów są analogiczne do własności kanałów komunikacyjnych definiowanych w kontekście komunikacji grupowej. Drugi standard dotyczy protokołu asynchronicznej wymiany wiadomości między procesami w stylu *publikuj-subskrybuj* (ang. *publish-subscribe*). Zgodnie z tym stylem, użytkownicy mogą tworzyć *tematy* (ang. *topics*), do których zapisują się *nadawcy* (ang. *producers*) oraz *odbiorcy* (ang. *consumers*) wiadomości. Kiedy nadawca publikuje wiadomość z danego tematu, jest ona przesłana do wszystkich odbiorców zapisanych do tego tematu. Nadawca nie musi więc adresować odbiorców wiadomości bezpośrednio, a jedynie pośrednio, specyfikując temat wiadomości. Aby nie ograniczać konkretnych implementacji, standard WS-BaseNotication nie specyfikuje gwarancji dotyczących wiarygodności kanałów komunikacyjnych lub gwarancji przechowywania wiadomości, a więc własności istotnych w kontekście komunikacji grupowej.

W zakresie narzędzi budowy niezawodnych systemów rozproszonych na szczególną uwagę zasługą *systemy kolejek komunikatów* (ang. *message queueing systems*), które rozwijane są przez niemal wszystkie czołowe firmy IT. Systemy kolejkowe zapewniają asynchroniczny protokół komunikacji między procesami, udostępniając mechanizm kolejek komunikatów. Asynchronizm komunikacji oznacza w tym wypadku, że zarówno nadawca jak i odbiorca komunikatu nie muszą być dostępni w tym samym czasie, aby komunikat został dostarczony. Jeśli odbiorca komunikatu jest w danej chwili niedostępny, komunikat będzie czekać w kolejce aż do momentu kiedy zostanie odebrany. Jest to własność *subskrypcji trwałych* (ang. *durable subscriptions*). Ten typ subskrypcji wymaga przechowywania komunikatów w pamięci trwałej, aby nie były one tracone w przypadku awarii procesów. W przypadku *subskrypcji nietrwałych* (ang. *non-durable subscriptions*), dopuszcza się by komunikaty były trzymane jedynie w pamięci ulotnej. Wiele systemów kolejkowych wspiera komunikację typu jeden do wiele, zwykle w stylu *publish-subscribe*. W przypadku gdy odbiorca wiadomości nie jest osiągalny, wiadomość zostanie dla niego zachowana w systemie kolejkowym, jeśli tylko subskrypcja ma charakter trwały.

Istnieją prace w których zaproponowano realizację komunikacji grupowej w oparciu o systemy kolejkowe [21]. Można to osiągnąć stosując dość proste odwzorowania. W naturalny sposób temat subskrypcji reprezentuje grupę, a proces będący częścią grupy jest subskrybentem danego tematu. Opis tego odwzorowania streszczono w raporcie. W odwzorowaniu subskrypcje nietrwałe odpowiadają modelowi awarii typu *crash-stop* przyjmując, że nieosiągalność procesu oznacza jego nieodwracalną awarię. W przypadku subskrypcji trwałych, system kolejkowy przechowuje wiadomość aż do czasu gdy wcześniej nieosiągalny proces odbiorcy stanie się ponownie dostępny. Wiadomości przechowywane przez system kolejkowy mogą pomóc procesom odtworzyć swój stan. Taka semantyka odpowiada w komunikacji grupowej modelowi awarii typu *crash-recovery*. Naturalnie systemy kolejkowe nie były projektowane jako systemy komunikacji grupowej. Tym samym nie wszystkie elementy komunikacji grupowej, takie jak np. usługa członkostwa czy wirtualna synchronizacja, mogą być w łatwy sposób odwzorowane na operacje ogólnych systemów kolejkowych. Ponadto odwzorowanie dotyczy jedynie semantyki *API (Application Programming Interface)*, a więc abstrahuje od spraw architektonicznych (np. podejście scentralizowane *vs.* rozproszone) oraz efektywnościowych, które w przypadku zastosowań komunikacji grupowej mają niezwykle istotne znaczenie.

W pracy [15] scharakteryzowano kilka przykładowych systemów komunikacji grupowej: Spread [12], Appia [22], Fortika [10] i JGroups [13]. Następnie przeanalizowano kilka komercyjnych systemów kolejkowych pod kątem własności, które są istotne w realizacji

opisanego wyżej odwzorowania na podstawowe operacje komunikacji grupowej. Opisując powyższe systemy, wzięto pod uwagę następujące własności: gwarancje dostarczenia komunikatów, gwarancje uporządkowania komunikatów, usługę członkostwa, wirtualną synchronizację, metody dostępu do systemu oraz architekturę systemu. Analiza objęta następujące systemy kolejkowe lub środowiska zawierające systemy kolejkowe: Apache ActiveMQ, IBM WebSphere, Microsoft WCF, Open Message Queue, BEA MessageQ i Oracle Streams Advanced Queueing. Wszystkie te systemy implementują API usługi *Java Message Service (JMS)* [23], która stanowi *de-facto* standard systemów kolejek komunikatów, częstokroć rozszerzając JMS w specyficzny dla siebie sposób i oferując tym samym nową funkcjonalność. Wyniki tej analizy zebrano w osobnej tabeli. Najważniejsze wyniki tej analizy w odniesieniu do systemów kolejkowych przedstawiono w osobnym artykule [18] (w przygotowaniu).

2.2. Projekt i implementacja protokołów komunikacji grupowej w OCaml

Odwzorowanie semantyki komunikacji grupowej na systemy kolejkowe nie zawsze współgra z podstawowym zastosowaniem komunikacji grupowej, tj. implementacją zreplikowanej maszyny stanów. Jak wspominaliśmy we wstępie, implementacja takiej formy replikacji byłaby także możliwa przy zastosowaniu systemów transakcyjnych, jednak taka implementacja byłaby dość wolna (mało efektywna) i co gorsza niekoniecznie dawałaby te same silne gwarancje odporności na awarie jak to się dzieje w przypadku specjalistycznych systemów (i protokołów) komunikacji grupowej, które projektuje się dla konkretnego modelu awarii. Podobna sytuacja dotyczy także ewentualnych systemów komunikacji grupowej zbudowanych w oparciu o systemy kolejek komunikatów. Dodatkowo należy zauważyć, że nie wszystkie systemy kolejek komunikatów (a nawet nie wszystkie komercyjne systemy komunikacji grupowej!) mają architekturę w pełni rozproszoną. Przykładem była implementacja komunikacji grupowej JGroups. Systemy scentralizowane ze swej natury nie są odporne na awarie pojedynczych serwerów, co jest w sprzeczności z podstawowym zastosowaniem komunikacji grupowej.

Z drugiej strony istnieje wiele eksperymentalnych systemów komunikacji grupowej, które zostały zaprojektowane i zaimplementowane na uniwersytetach w ciągu ostatnich dwóch dekad. Nieliczne z tych systemów zostały skomercjalizowane, większość jest oferowana jako oprogramowanie typu *open source*. Niestety część z tych systemów nie jest już dalej rozwijana. Przykładem z ostatnich lat jest modułowy system komunikacji grupowej Fortika [10] (lub SAMOA [11]), zaprojektowane i zaimplementowane na EPFL w latach 2003-2006(2008). Fortika była jednym z pierwszych modularnych systemów komunikacji grupowej, wspierających model awarii *crash-recovery*. SAMOA [24] to efektywny framework protokołowy oferujący nowe API, przy pomocy którego zaimplementowano ponownie protokoły Fortiki, oraz wspierający zupełnie nową funkcjonalność – możliwość dokonania w sposób całkowicie transparentny dynamicznej zamiany protokołów (algorytmów) implementujących tę samą usługę komunikacji grupowej (np. rozproszony konsensus). Zarówna Fortika jak i SAMOA to systemy modularne, zbudowane w oparciu o nową architekturę systemów komunikacji grupowej, zaproponowaną w [25]. Implementacja obu systemów zależy od dodatkowych *third-party* bibliotek opracowanych dla języka Java. Niestety jak się okazało biblioteki te albo nie są już udostępniane albo zostały zmienione w ostatnim czasie, co nie pozwala na korzystanie z tych systemów bez znaczących modyfikacji kodu. Po analizie sytuacji doszliśmy do wniosku, że cechy które stanowią o wyjątkowości obu tych systemów, tj. wsparcie dla modelu awarii *crash-recovery* oraz możliwość dynamicznej zamiany protokołów, nie są krytyczne z punktu widzenia przewidywanych przez nas zastosowań komunikacji grupowej w projekcie IT-SOA, i wobec tego ostatecznie postanowiono sięgnąć po inne tego typu systemy. Przebieg prac z tym związanych, opisany zostanie w następnym rozdziale.

Niezależnie podjęto próbę reimplementacji uproszczonej architektury stosu protokołów Fortika w języku programowania funkcyjnego OCaml [26]. Ograniczono się przy tym wyłącznie do modelu awarii *crash-stop*. Celem tych prac jest nie tyle implementacja kolejnego

systemu komunikacji grupowej, ile wdrożenie studentów w tematykę komunikacji grupowej oraz poznanie algorytmów rozwiązujących podstawowe problemy z tej dziedziny, np. problem rozproszonego konsensusu. Wybór języka funkcyjnego OCaml jest w tym kontekście podyktowany dużą czytelnością kodu źródłowego oraz cechami języka takimi jak typy abstrakcyjne oraz moduły, które doskonale wspierają implementację modularnego stosu protokołów. Z punktu widzenia celów naukowo-badawczych nie jest uzasadnione implementowanie pełnego systemu komunikacji grupowej przy użyciu protokołów które są dobrze znane i opisane w literaturze, gdyż nie dałoby to rezultatów publikacyjnych. Ponadto implementacja takiego systemu nie byłaby możliwa uwzględniając posiadane zasoby czasowo-osobowe w ramach zadania OB2-1. Dlatego więc zdecydowano się na implementację uproszczonej architektury systemu, przeznaczonej dla grup dynamicznych w modelu *crash-stop*. Jest to jednak architektura w pełni funkcjonalna, która pozwala na implementację zreplikowanej maszyny stanów oraz może stanowić platformę testową do prototypowania nowych algorytmów w przyszłości.

W chwili obecnej architektura proponowanej biblioteki komunikacji grupowej, zwanej *CamlGroups*, składa się z następujących protokołów (implementujących odpowiednie usługi): protokół TCP tolerujący awarie sesji (ang. *Robust TCP*), protokół niezawodnego rozgłaszania (ang. *Reliable Broadcast*), protokół implementujący algorytm Chandra-Toueg'a [3] rozwiązywania rozproszonego konsensusu (ang. *Distributed Consensus*), protokół rozgłaszania zachowującego atomowe (lub globalne) uporządkowanie komunikatów (ang. *Atomic Broadcast*), a także detektor awarii oparty na prostym protokole *heart-beat*. Obecnie trwają prace implementacyjne nad powyższymi protokołami. Prace te będą kontynuowane w drugim etapie projektu. Przykładowo implementacja pierwszej warstwy (Robust TCP) to około 1200 linii kodu w OCaml, co odpowiada mniej więcej 3000 linii kodu w językach typu Java. Architektura ta implementuje odpowiednie usługi komunikacji grupowej zakładając model *grup statycznych*, tj. bez możliwości dodawania nowych procesów do grupy. W dalszej części prac planuje się rozszerzyć tę architekturę o wsparcie dla grup dynamicznych. Podstawowa modyfikacja będzie dotyczyć usługi atomowego rozgłaszania. Zamiast implementować rozgłaszanie oparte na synchronizacji wirtualnej (ang. *view-synchronous broadcast*), zamierza się zastosować podejście z systemu Fortika, które korzysta z *rozgłaszania generycznego* (ang. *generic broadcast*) [27]. Obie architektury (dla grup statycznych i dynamicznych) zostały krótko opisane w raporcie [16], łącznie z definicjami podstawowych pojęć i opisem własności usług implementowanych przez te architektury.

2.3. Koncepcja usługi komunikacji grupowej dla systemów opartych na paradygmacie SOA

Celem innego wątku zadania OB2-1, wspólnego z częścią zadania OB2-2, było opracowania koncepcji *REST Groups* – nowego interfejsu *API* (ang. *Application Programming Interface*) do systemu komunikacji grupowej, zgodnego ze stylem REST. Opis realizacji tych prac zamieszczono w oddzielnym raporcie technicznym [17]. Zaletą nowego interfejsu ma być uproszczenie implementacji replikacji usług sieciowych, które komunikują się ze sobą korzystając z reguł oraz protokołów specyfikowanych przez podejście REST-owe. W praktyce zwykle stosuje się w tym wypadku protokół HTTP, który jest powszechnie używanym protokołem dostępu do danych hypermedialnych w ramach World-Wide Web (WWW). Z uwagi na to, że podejście REST-owe jest wymieniane obok standardów WS-* jako czołowe podejście do realizacji architektury SOA, mamy tym samym nadzieję na upowszechnienie się systemów komunikacji grupowej, które udostępniają taki właśnie interfejs. Techniczne motywacje za realizacją takiego właśnie podejścia, wsparte krótkimi przykładami uzasadniającymi zalety wprowadzenia REST-owego API komunikacji grupowej, zawarto we wstępie raportu [17].

W pierwszej części prac zapoznano się z szeregiem podstawowych informacji na temat protokołu HTTP oraz modelu jego funkcjonowania. Omówiono między innymi sposoby komunikacji bezpośredniej i za pośrednictwem serwerów buforujących, format wiadomości, metody protokołu HTTP (tj. GET, POST, HEAD, PUT, DELETE, OPTIONS i

TRACE), a także zagadnienia bezpieczeństwa. Następnie opisano krótko zagadnienia komunikacji grupowej, ze szczególnym uwzględnieniem aspektów istotnych z punktu widzenia omawianego zakresu prac, np. różnice między tradycyjną architekturą systemów komunikacji grupowej, która jest oparta na abstrakcji *synchronicznych obrazów* (ang *view synchrony*), a nowszą architekturą komunikacji grupowej [25], która opiera się na abstrakcji *rozgłaszania ogólnego* (ang. *generic broadcast*) [27]. Jakkolwiek wybór architektury systemu komunikacji grupowej determinuje postać konkretnego API komunikacji grupowej opartego na stylu REST, to jednak nie wpływa on zasadniczo na samo podejście do projektowania i implementacji takiego API. W szczególności w opisywanym raporcie opisujemy ogólną koncepcję API, która nie determinuje wyboru architektury. Jednocześnie skłaniamy się obecnie ku architekturze opartej na abstrakcji obrazów synchronicznych, co jest spowodowane wyborem tego a nie innego systemu komunikacji grupowej, o czym za chwilę.

W dalszej części prac zapoznano się z podstawowymi założeniami stylu REST, zwracając uwagę na takie cechy jak: adresowanie zasobów, bezstanowość interakcji, uniwersalny interfejs, buforowanie zapytań, oraz warstwowość architektury systemu. Należy uściślić, że termin REST używany jest w kontekście zadania OB2-1 do opisu dowolnego prostego interfejsu API, który transmituje dane przez protokół HTTP bez pośrednictwa dodatkowej warstwy komunikacji wiadomości, takiej jak protokół *SOAP* (ang. *Simple Object Access Protocol*) lub warstwy śledzenia sesji przez *ciasteczka HTTP* (ang. *HTTP cookies*). Kluczową sprawą w realizacji REST-owego interfejsu API jest zdanie sobie sprawy ze specyfiki i ograniczeń tego stylu architektonicznego w porównaniu z, np. mechanizmem *zdalnego wywołania procedur* (ang. *Remote Procedure Calls – RPC*). W przypadku podejścia REST-owego, mamy do czynienia nie ze zdalnymi procedurami (lub metodami) zdefiniowanymi przez programistę, ale z zasobami określonego typu, adresowanymi przez unikalne *ujednolicone nazwy (identyfikatory) zasobów* (ang. *Uniform Resource Identifiers – URI*). Na tych zasobach klienci usług oraz serwery mogą wykonywać operacje, ujednolicone przez zastosowanie standardowego protokołu dostępu do tych zasobów (w naszym przypadku jest to HTTP). Z uwagi na to, że w podejściu REST-owym nie ma pojęcia sesji komunikacyjnej, wywołanie każdej operacji na zasobach musi przekazać wszelkie dane potrzebne do wykonania tej operacji. Ponieważ stan aplikacji i jej funkcjonalność są abstrahowane w postaci unikalnych i globalnie adresowalnych zasobów, styl REST umożliwia łatwą skalowalność i integrację aplikacji (co pokazał dynamiczny rozwój WWW).

Następnie opisano podstawowe różnice między stylem REST a RPC, pokazując na przykładzie jak zaimplementować prosty program w jednym i drugim stylu. Jakkolwiek w ogólności styl REST nie powinien naśladować RPC, to jednak w przypadku REST-owego interfejsu do systemu komunikacji grupowej nie da się uniknąć konieczności zaimplementowania mechanizmu przekazania do systemu (w jakiś sposób) argumentów i odbioru wyników. Porównanie obu stylów jest w tym wypadku pouczające. W szczególności zwrócono uwagę na pewne ograniczenia w stylu programowania opartego na REST oraz opisano metody obejścia (lub uniknięcia) tych ograniczeń. Przykładowo, jest dość problematyczne w REST zaimplementowanie mechanizmu asynchronicznego powiadamiania o zajściu jakiegoś zdarzenia. Z uwagi na to, że taki mechanizm jest niezbędny do implementacji interfejsu komunikacji grupowej, należało zaproponować sposób jego realizacji, który z jednej strony jest zgodny z REST, a z drugiej strony daje się efektywnie zaimplementować przy pomocy metod i technik przewidzianych dla protokołu HTTP. Kilka propozycji rozwiązania tego problemu przedstawiono w raporcie.

Obok prac nad koncepcją interfejsu, podjęte zostały także badania eksperymentalne. Przebadane zostały cztery systemy komunikacji grupowej: Spread (Spread Concepts LLC), Appia (University of Lisbon), Fortika (EPFL) oraz SAMOA (EPFL). Ostatecznie jako kandydat dla naszego REST-owego API wybrany został Spread. O wyborze zdecydowały zalety takie jak prosty interfejs oraz istniejące wsparcie dla tego projektu. Spread jest oferowany zarówno na licencji *Open Source* jak i komercyjnej. System nie jest pozbawiony wad, o których wspomniano w raporcie, jednakże nie mają one bezpośredniego znaczenia dla zastosowań przewidywanych w naszym projekcie. Ten wstępny wybór nie wyklucza innego systemu, jeśli zaszłaby taka potrzeba.

3. Podsumowanie i kierunki dalszych prac

W ramach I etapu zadania OB2-1 powstały trzy raporty techniczne: (1) Analiza mechanizmów komunikacji grupowej w systemach rozproszonych i opartych na paradygmacie SOA; (2) Biblioteka komunikacji grupowej: Architektura systemu dla grup dynamicznych w modelu crash-stop; i (3) Koncepcja usługi komunikacji grupowej oraz mechanizmu operacji atomowych dla systemów opartych na paradygmacie SOA. Jak wykazano w niniejszym podsumowaniu, obiecującym kierunkiem rozwoju dalszych prac w ramach zadania OB2-1 będzie implementacja nowego API komunikacji grupowej zgodnego ze stylem REST. Kolejnym krokiem w realizacji tego zadania będzie zastosowanie nowego API do implementacji zreplikowanej maszyny stanów dla przykładowej usługi sieciowej, celem uodpornienia jej na awarie pojedynczych serwerów. W naszym zamierzeniu eksperyment ten pozwoli nam na opracowanie ogólnych metod replikacji usług sieciowych i aplikacji SOA, budowanych zgodnie ze stylem REST.

4. Bibliografia

[1] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)*, 22(4):299–319, 1990.

[2] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, April 1985.

[3] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.

[4] Ken P. Birman and Robbert Van Renesse (eds.). *Reliable distributed computing with the Isis toolkit*. IEEE Computer Society Press, 1994.

[5] Robbert Van Renesse, Kenneth P. Birman, and Silvano Maffeis. Horus: A flexible group communication system. *Communications of the ACM*, 39(4):76–83, April 1996.

[6] Danny Dolev and Dalia Malki. The Transis approach to high availability cluster communication. *Communications of the ACM*, 39(4):64–70, April 1996.

[7] Yair Amir and Jonathan Stanton. The Spread wide area group communication system. Technical Report CNDS-98-4, Department of Computer Science, Johns Hopkins University, 1998.

[8] Mark Hayden. The Ensemble system. Technical Report TR98-1662, Department of Computer Science, Cornell University, January 1998.

[9] Hugo Miranda, Alexandre Pinto, and Luís Rodrigues. Appia, a flexible protocol kernel supporting multiple coordinated channels. In *Proceedings of ICDCS '01: the 21st IEEE International Conference on Distributed Computing Systems*, April 2001.

[10] EPFL. Fortika. http://lsrwww.epfl.ch/crystall/, 2006.

[11] EPFL. Samoa. http://lsrwww.epfl.ch/samoa/, 2008.

[12] Spread Concepts LLC. The Spread toolkit. http://www.spread.org/, 2006.

[13] Bela Ban / Red Hat. JGroups toolkit. http://www.jgroups.org/, 2009.

[14] Roy T. Fielding and Richard N. Taylor. Principled design of the modern Web architecture. *ACM Transactions on Internet Technology (TOIT)*, 2(2):115–150, 2002.

[15] Adam Gawarkiewicz, Jarosław Przybyłowicz, Michał Witkowski, Paweł T. Wojciechowski, and Jerzy Brzeziński. Analiza mechanizmów komunikacji grupowej w systemach rozproszonych i opartych na paradygmacie SOA. Technical Report TR-ITSOA-OB2-1-PR-09-1, Instytut Informatyki, Politechnika Poznańska, March 2009.

[16] Adam Gawarkiewicz, Jarosław Przybyłowicz, Michał Witkowski, Piotr Wysocki, and Paweł T. Wojciechowski. Biblioteka komunikacji grupowej: Architektura systemu dla grup dynamicznych w modelu crash-stop. Technical Report TR-ITSOA-OB2-1-PR-09-2, Instytut Informatyki, Politechnika Poznańska, May 2009.

[17] Tadeusz Kobus, Mateusz Mor, and Paweł T. Wojciechowski. Koncepcja usługi komunikacji grupowej oraz mechanizmu operacji atomowych dla systemów opartych na paradygmacie SOA. Technical Report TR-ITSOA-OB2-1-PR-09-3, Instytut Informatyki, Politechnika Poznańska, May 2009.

[18] Adam Gawarkiewicz, Jarosław Przybyłowicz, Michał Witkowski, and Paweł T. Wojciechowski. A brief survey of message queueing systems for group communication middleware. In *Proceedings of the Workshop on the Challenges of SOA Implementation and Applications (co-located with WISE 2009: the 10th International Conference on Web Information Systems Engineering)*, October 2009. Submitted for publication.

[19] OASIS. *Web Services Reliable Messaging (WS-ReliableMessaging) Ver. 1.1*, 2007.

[20] OASIS. *Web Services Base Notification 1.3 (WS-BaseNotification)*, 2006.

[21] Arnas Kupsys, Stefan Pleisch, André Schiper, and Matthias Wiesmann. Towards JMS compliant group communication - A semantic mapping. In *Proceedings of IEEE NCA '04: the third IEEE International Symposium on Network Computing and Applications*, August 2004.

[22] University of Lisbon. Appia. http://appia.di.fc.ul.pt/, 2007.

[23] Sun Microsystems. Java Message Service. http://java.sun.com/products/jms/docs.html, 2009.

[24] Paweł T. Wojciechowski, Olivier Rütti, and André Schiper. SAMOA: A framework for a synchronisation-augmented microprotocol approach. In *Proceedings of IPDPS '04: the 18th IEEE International Parallel and Distributed Processing Symposium (Santa Fe, USA)*, April 2004.

[25] Sergio Mena, André Schiper, and Paweł T. Wojciechowski. A step towards a new generation of group communication systems. In Markus Endler and Douglas Schmidt, editors, *Proceedings of Middleware '03: the 4th ACM/IFIP/USENIX International Middleware Conference (Rio de Janeiro, Brazil)*, volume 2672 of *LNCS*, pages 414–432. Springer, June 2003.

[26] INRIA. Objective Caml. http://caml.inria.fr, 2009.

[27] Fernando Pedone and Andreé Schiper. Generic broadcast. In *Proceedings of DISC '99: the 13th International Symposium on Distributed Computing*, volume 1693 of *LNCS*, pages 94–108. Springer, September 1999.