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 OB2-2**

# Wsparcie dla operacji atomowych w SOA - 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-OB2-2-PR-09-3 |
| Koordynator | dr hab. inż. Paweł T. Wojciechowski |
| Autorzy | Paweł T. Wojciechowski (PP) |
| Poziom dostępności | Wewnętrzny konsorcjum |
| Słowa kluczowe | pamięć transakcyjna, atomowość, REST |

# Streszczenie

*Atomowość (niepodzielność)* zbioru operacji gwarantuje, że zbiór ten może być brany pod uwagę jako pojedyncze zdarzenie w systemie, niezależnie od jakichkolwiek innych operacji wykonywanych współbieżnie. W praktyce operacje z atomowego zbioru operacji mogą być przeplatane z innymi operacjami spoza tego zbioru, jednakże od strony logicznej nie można tego przeplotu zaobserwować. Atomowość pozwala na zachowanie spójności przetwarzania współbieżnego bez konieczności implementowania synchronizacji własnoręcznie oraz – w przypadku obliczeń rozproszonych – stanowi wygodne narzędzie do koordynacji rozproszonych operacji.

Raport zawiera podsumowanie prac wykonanych w I etapie projektu IT-SOA na Politechnice Poznańskiej nad zadaniem OB2-2, pt. "Wsparcie dla operacji atomowych w SOA". Wynikami tych prac są: (1) analiza i porównanie wybranych realizacji *Software Transactional Memory*, (2) analiza mechanizmów wsparcia dla operacji atomowych w systemach rozproszonych i opartych na paradygmacie SOA, oraz (3) koncepcja mechanizmu operacji atomowych 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

Przez ostatnie dziesięciolecia obowiązywało prawo Moore'a: wprowadzenie kolejnej nowej generacji procesorów sprawiało, że programy mogły działać szybciej. Te czasy minęły. Układy procesorowe nowych generacji będą miały więcej jednostek CPU, ale każda pojedyncza jednostka CPU nie będzie szybsza od modelu z poprzedniego roku. Aby programy działały szybciej, musimy pisać programy równoległe (współbieżne). Pisanie poprawnych programów współbieżnych jest zwykle znacząco trudniejsze niż pisanie analogicznych programów sekwencyjnych. Ponieważ programy współbieżne wykonywane są w sposób niedeterministyczny, ich testowanie jest trudne, a wykrycie i usunięcie wszystkich błędów programistycznych praktycznie niemożliwe. Dotyczy to oczywiście zarówno aplikacji, które są zgodne z paradygmatem SOA, jak i aplikacji, które nie mają nic wspólnego z SOA. Jednakże z uwagi na charakter aplikacji SOA, w szczególności wymóg integracji modułów opracowanych zupełnie niezależnie, nierzadko udostępnianych jedynie w postaci binariów lub w postaci interfejsu do usługi sieciowej, wspomniane wyżej problemy manifestują się w sposób szczególny. Przykładowo, analiza kodu jest utrudniona lub niemożliwa, kiedy nie można dokonać inspekcji kodu źródłowego.

W ramach zadania OB2-2, interesuje nas szeroka tematyka obejmująca metody, języki oraz narzędzia, które pozwalają na *bezpieczne programowanie*, tj. z gwarancją braku określonego rodzaju błędów programistycznych. Tematyka ta dotyczy zarówno nowych konstrukcji programistycznych, jak również metod i narzędzi do weryfikacji programów, takich jak statyczne systemy typów, dynamiczne analizatory kodu, *model checkers* i *theorem provers*. W tym zakresie, podjęliśmy prace w kierunku konstrukcji nowych mechanizmów osiągania atomowości w programowaniu współbieżnym i rozproszonym. *Atomowość (niepodzielność)* zbioru operacji gwarantuje, że zbiór ten może być brany pod uwagę jako pojedyncza jednostka obliczeniowa, niezależnie od jakichkolwiek innych operacji wykonywanych współbieżnie. W praktyce operacje z atomowego zbioru operacji mogą być przeplatane z innymi operacjami spoza tego zbioru, co wpływa korzystnie na efektywność, jednakże od strony logicznej nie można tego przeplotu zaobserwować. Atomowość pozwala na zachowanie spójności przetwarzania współbieżnego bez konieczności implementowania synchronizacji własnoręcznie oraz – w przypadku obliczeń rozproszonych – stanowi wygodne narzędzie do koordynacji rozproszonych operacji.

Istotne miejsce w kontekście zadania OB2-2 zajmuje *Programowa Pamięć Transakcyjna* (ang. *Software Transactional Memory – STM*) [1], która stanowi alternatywę dla niskopoziomowych konstrukcji synchronizacji. STM trochę przypomina atomowe transakcje bazodanowe. Implementacja STM także opiera się na optymistycznych algorytmach sterowania współbieżnością, ale operuje na dowolnych strukturach lub obiektach programu przechowywanych w pamięci operacyjnej. Zaletą STM oraz innych technik *lock-free* jest wyeliminowanie problemów, z którymi wiąże się programowanie synchronizacji przy użyciu *zamków* (ang. *locks*), np. możliwość *zakleszczenia* (ang. *deadlock*) lub różna ziarnistość synchronizacji (ang. *fine or coarse grain locking*) oraz związany z tym faktem *trade-off*, tj. niepotrzebne blokowanie fragmentów programu *vs.* mniej częste blokowanie kosztem komplikacji kodu, która może zagrozić jego poprawności. Stosowanie STM w pewnych sytuacjach jest jednak problematyczne, jak również semantyka różnych STM jest dość zróżnicowana, co łącznie utrudnia wybór odpowiedniego dla danej aplikacji rozwiązania. Problemy te wyjaśniamy szczegółowo w dwóch pierwszych raportach technicznych. Niewiele jest także rozwiązań STM dla obliczeń rozproszonych.

Naszym celem jest więc zbadanie abstrakcji programistycznych oraz mechanizmów osiągania atomowości (w szczególności STM) pod kątem ich wykorzystania do budowy usług sieciowych zgodnych ze stylem architektonicznym SOA. Powszechnie podnoszonymi postulatami (wymaganiami) wobec SOA jest wsparcie integracji oprogramowania oraz wysoka bezawaryjność usług sieciowych implementowanych zgodnie z tym stylem architektonicznym. Styl SOA nie specyfikuje jednakże, jakie środki powinny być zastosowane aby spełnić ten drugi postulat. Prace podjęte w ramach zadania OB2-2 pozwolą częściowo rozwiązać ten problem, gdyż wsparcie dla atomowości pozwala uniknąć wielu błędów programistycznych, na które narażeni są projektanci i programiści korzystający z konstrukcji

niskopoziomowych. Jednocześnie rozwiązania proponowane przez nas starają się uniknąć niektórych wad STM oraz wypełnić lukę, jeśli chodzi o wsparcie dla operacji atomowych w systemie rozproszonym. Aby zapewnić spełnienie pierwszego postulatu, tj. łatwej integracji oprogramowania tworzonego niezależnie, rozpoczęto prace w kierunku opracowania nowego interfejsu dla operacji atomowych, zgodnego ze stylem REST. *REST* (ang. *REpresentational State Transfer*) [2] 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 mechanizmu wsparcia dla operacji atomowych opartego o taki interfejs umożliwi koordynację dowolnych serwerów połączonych przez Internet.

Prace badawcze wykonane w ramach I etapu zadania OB2-2 i w fazie zerowej projektu IT-SOA oraz uzyskane wyniki, zostały opisane w następujących raportach technicznych i artykułach:

1. RAPORT: *Analiza i porównanie wybranych realizacji Software Transactional Memory* [3];
2. RAPORT: *Analiza mechanizmów wsparcia dla operacji atomowych w systemach rozproszonych i opartych na paradygmacie SOA* [4];
3. RAPORT: *Koncepcja usługi komunikacji grupowej oraz mechanizmu operacji atomowych dla systemów opartych na paradygmacie SOA* [5];
4. *Extending Atomic Tasks to Distributed Atomic Tasks.* Artykuł zreferowany na *Workshop on Exploiting Concurrency Efficiently and Correctly* $(EC)^2$, w ramach *CAV '08: the 20th International Conference on Computer Aided Verification* [6].

# 2. Opis uzyskanych wyników

## 2.1. Analiza i porównanie wybranych realizacji *Software Transactional Memory*

W raporcie [3] dokonano porównania wybranych realizacji pamięci transakcyjnej STM pod kątem składni i semantyki oferowanych operacji. Do porównania wybrane zostały naszym zdaniem najciekawsze i najbardziej wpływowe rozwiązania, które są dobrze udokumentowane w publikacjach naukowych lub innych. Były to: pierwsza implementacja STM, przeznaczona dla języka Java [7] (University of Cambridge), włączając późniejsze rozszerzenia o obsługę wyjątków i wsparcie dla operacji I/O [8] (Microsoft Research - Cambridge), monady STM dla języka Haskell [9] (Microsoft Research - Cambridge), transakcyjne rozszerzenie języka OCaml [10] (University of Washington), implementacja STM dla C/C++ [11] (Intel), SXM dla platformy .NET [12] (EPFL i Brown University) oraz JSTM (realizacja projektu XSTM w języku Java), będący rozszerzeniem rozproszonej pamięci współdzielonej o mechanizm pamięci transakcyjnej [13] (XSTM.net).

Porównanie powyższych rozwiązań STM objęło najważniejsze własności semantyczne oraz cechy implementacji, które wpływają bezpośrednio na sposób wykorzystania tych konstrukcji w programach oraz integrację kodu transakcyjnego z kodem nietransakcyjnym. Efektem dodatkowym tego porównania jest także dyskusja problemów dotyczących subtelnych szczegółów semantyki STM. Porównanie objęło następujące cechy, omówione w oddzielnych sekcjach: interfejs programisty (ang. *Application Programming Interface – API*), wsparcie dla silnej lub słabej atomowości w kontekście przyjętego modelu pamięci, wsparcie lub brak wsparcia dla transakcji zagnieżdżonych, moment zapisu modyfikacji do pamięci transakcyjnej, kontrola współbieżności, rozwiązywanie konfliktów, semantyka obsługi wyjątków, i wsparcie lub brak wsparcia dla operacji *wejścia/wyjścia* (ang. *input/output*). Wyniki porównania w każdej sekcji są zebrane w tabelkach, w których

przypisano poszczególnym realizacjom STM sposób rozwiązania problemu opisywanego w danej sekcji.

Kontynuacją pracy opisanej w raporcie [3] w etapie II projektu ma być porównanie wybranych realizacji STM od strony efektywnościowej. W tym celu podjęliśmy wstępne prace w kierunku opracowania odpowiednich benchmarków dla pamięci transakcyjnej oraz środowiska testowego.

## 2.2. Analiza mechanizmów wsparcia dla operacji atomowych w systemach rozproszonych i opartych na paradygmacie SOA

Standardy i technologie opracowywane dla paradygmatu SOA nie zajmują się bezpośrednio problemami konstrukcji narzędzi i metod bezpiecznego programowania oraz weryfikacji poprawności programów – dziedziny, która w ostatnich latach rozwija się prężnie nie tylko w ośrodkach akademickich, ale również w laboratoriach badawczo-rozwojowych liczących się firm IT. Metody te i narzędzia umożliwiają zwiększenie niezawodności usług informatycznych oraz skrócenie czasu tworzenia, testowanie i pielęgnowania oprogramowania, a więc obniżenie kosztów z tym związanych. Z drugiej strony wobec SOA formułuje się wysokie oczekiwania, iż związane z tym podejściem nowe technologie zapewnią nową jakość w produkcji oprogramowania. W związku z tym podjęliśmy prace w ramach zadania OB2-2, które zmierzają do zbliżenia się tych dwóch obszarów badawczo-rozwojowych. Jednocześnie należy zwrócić uwagę, że nasze cele i problemy formułowane w OB2-2 są zasadniczo różne od celów i problemów badawczych podejmowanych w zadaniu OB2-5, o podobnie brzmiącym tytule: *Mechanizmy transakcyjności w kompozycji usług w złożone procesy biznesowe*. Cytując luźno za Studium Wykonalności projektu IT-SOA, celem OB2-5 jest opracowanie mechanizmów transakcyjności a następnie wykorzystanie ich w obszarze OB4, tj. w protokołach do automatycznej kompozycji usług w złożone procesy biznesowe. W realizacji OB2-5 zostaną wykorzystane dotychczasowe standardy, np. Web Services Coordination (WS-C), Web Services Trust (WS-T), OASIS Business Transaction Protocol (BTP), OASIS Web Services Transaction (WSTX) oraz inne rozwiązania.

W raporcie [4] opisano krótko motywacje oraz sformułowano problemy związane z opracowaniem wsparcia operacji atomowych w implementacji usług sieciowych. Streszczono także wyniki badań autora niniejszego sprawozdania, zebrane w [14], które stanowiły punkt wyjścia dla prac podjętych w ramach projektu IT-SOA nad zadaniem OB2-2. W szczególności był to projekt nowego języka operacji atomowych (niepodzielnych) [15]. Język pozwala deklarować fragmenty kodu jako wolne-od-odtwarzania transakcje, które nazywane są zadaniami atomowymi. *Zadania atomowe* zapewniają atomowość (albo izolację, używając terminologii z baz danych) dla wybranych fragmentów kodu. Z punktu widzenia implementacji usług sieciowych, kluczową cechą jest w tym wypadku fakt, iż kod ten może zawierać dowolne operacje, w tym operacje mające efekty wejścia/wyjścia, np. wysłanie lub odbiór komunikatów. Warto też dodać, że zadanie atomowe w tym języku może być wewnętrznie wielowątkowe, co łącznie z poprzednią cechą pozwala na dużą elastyczność w deklarowaniu atomowości w stosach współbieżnych protokołów. Przetwarzanie zadań atomowych bazuje na *algorytmach wersjonowania*, tj. nowych algorytmach sterowania współbieżnością, zaprojektowanych specjalnie dla zadań atomowych [14] (wstępną wersję algorytmów opublikowano w [16]). Algorytmy te stanowią integralną część maszyny wirtualnej zaprojektowanego języka. Kluczową cechą tych algorytmów jest wersjonowanie dostępu do współdzielonych obiektów, tj. danych lub kanałów wejścia/wyjścia. Szeregują one operacje krytyczne na tych obiektach zgodnie z numerami wersji, w taki sposób, że własność atomowości (izolacji) jest zachowana.

Aby algorytmy wersjonowania mogły spełniać swoją funkcję, wymagają one podania określonych argumentów, które zależą od konkretnego programu. Jeśli argumenty podane w wywołaniu konstrukcji 'atomic' są błędne, atomowość będzie naruszona, co doprowadzi do niepoprawnego wykonania programu. Dlatego więc zaproponowano w [15], aby wymagane dane były specyfikowane (lub automatycznie dedukowane) w postaci adnotacji

typów. W artykule zaprojektowano odpowiedni statyczny system typów, którego implementacja (jako rozszerzenie kompilatora) będzie w stanie zweryfikować, czy podane adnotacje typów są prawidłowe dla danego algorytmu wersjonowania. W praktyce, pozytywna runda *weryfikatora zgodności typów* (ang. *type checker*) zapewni więc automatyczny dowód, że własność izolacji, stanowiąca kryterium poprawności wykonania zadań atomowych, rzeczywiście jest zachowana. Ponieważ dowód przeprowadzany jest w czasie kompilacji, zanim jeszcze program jest wykonywany, eliminuje to potrzebę sprawdzeń lub obsługi wyjątków z tytułu błędów fatalnych lub niepoprawnego działania algorytmów wersjonowania, np. w wyniku podania przez programistę nieprawidłowych danych. Formalny dowód poprawności zaprojektowanego systemu typów został przeprowadzony w [14, 15] dla *CBV rachunku lambda* (ang. *Call-By-Value Lambda Calculus*), rozszerzonego o konstrukcję zadań atomowych, oraz wyposażonego w semantykę operacyjną. W praktyce rezultat ten będzie obowiązywać także dla pełnowymiarowego języka, np. Java, choć przeprowadzenie formalnego dowodu tego rezultatu byłoby zbyt czasochłonne.

## 2.3. Projekt i implementacja algorytmów wersjonowania dla operacji atomowych

W fazie zerowej projektu IT-SOA rozpoczęto prace w kierunku zaprojektowania i implementacji zadań atomowych w środowisku rozproszonym, jako rozszerzenia popularnego mechanizmu *zdalnego wywołania metod* w języku Java (ang. *Java Remote Method Invocation – Java RMI*). W tym celu zaprojektowano konieczne rozszerzenia rachunku zadań atomowych i algorytmów wersjonowania. Wstępne wyniki opisane zostały w artykule *Extending Atomic Tasks to Distributed Atomic Tasks* [6]. Rozproszone zadania atomowe mogą wołać zdalne obiekty i tym samym obejmować swoim zasięgiem wiele miejsc (węzłów) sieciowych. Z kolei te inne obiekty mogą wołać także inne zdalne obiekty, jako część tego samego zadania atomowego. Przez zastosowanie algorytmów wersjonowania, współbieżne wykonanie wielu rozproszonych zadań atomowych gwarantuje zachowanie własności izolacji. Ponadto z uwagi na to, że wołania obiektów mogą być asynchroniczne, rozproszone zadania atomowe mogą wewnętrznie składać się z wielu współbieżnych wątków. Zaprojektowany w artykule rachunek obiektowy posłużył jako specyfikacja dla implementacji biblioteki rozproszonych zadań atomowych jako rozszerzenia zdalnego wywołania metod. Kombinacja dwóch cech: współbieżności oraz konieczności koordynacji wykonania rozproszonych zadań sprawiła, że implementacja nie była prosta. Efektem tych prac jest wstępna wersja biblioteki, nazwana *Atomic RMI*, będąca rozszerzeniem mechanizmu Java RMI o wsparcie dla atomowości. W implementacji biblioteki zastosowaliśmy algorytm sterowania współbieżnością, który jest kombinacją *Algorytmu Podstawowego Wersjonowania* (ang. *Basic Versioning Algorithm – BVA*) i *Algorytmu Najmniejszego Górnego Ograniczenia* (ang. *Supremum Versioning Algorithm – SVA*) [14].

Póki co w obecnej wersji eksperymentalnej implementacji biblioteki Atomic RMI, programista musi wyspecyfikować wszystkie argumenty wymagane przez algorytm wersjonowania. W najprostszym przypadku dla każdego zdalnego obiektu wystarczy wskazanie innych zdalnych obiektów, które mogą być wywoływane w ramach przetwarzania atomowego. Warto nadmienić, że implementacja biblioteki zachowuje w tym przypadku istotny postulat programowania obiektowego, tj. wymóg enkapsulacji danych. Dane odnośnie innych obiektów są zawsze podawane z punktu widzenia wiedzy lokalnej. Takie podejście wspiera kompozycję usług oferowanych przez różne zdalne obiekty w ramach jednego rozproszonego zadania atomowego. W celu potencjalnego zwiększenia równoległości przetwarzania współbieżnych zadań, w momencie rozpoczęcia zadania atomowego można dodatkowo przekazać także informacje o górnych ograniczeniach na liczbę odwołań do zdalnych obiektów. Pozwoli to algorytmowi sterowania współbieżnością na zastosowanie bardziej agresywnej polityki szeregowania zadań. Im więcej danych ilościowych na temat wywołania zdalnych obiektów jesteśmy w stanie przekazać przed rozpoczęciem zadania atomowego, tym bardziej równoległe będzie wykonanie współbieżnych zadań. Innymi słowy wykonanie zadań będzie rzadziej blokowane, a tym samym moc serwerów w różnych węzłach sieciowych będzie lepiej wykorzystana.

W ramach zadania OB2-2 podjęto też prace w kierunku implementacji analizatora kodu źródłowego w języku Java, przeznaczonego dla programów korzystających z biblioteki Atomic RMI. W naszym zamierzeniu analizator ten ma umożliwić automatyczną analizę kodu i dedukcję niektórych fragmentów kodu związanego z wywołaniami biblioteki. Ułatwi to dalej korzystanie z biblioteki, a tym samym przyczyni się do zmniejszenia liczby potencjalnych błędów programistycznych. Prace nad analizatorem są zaawansowane w około 50% i będą kontynuowane w drugiej fazie projektu. Równolegle z pracami nad analizatorem kodu, interesuje nas opracowanie mechanizmu pamięci transakcyjnej w oparciu o kombinację optymistycznych i pesymistycznych algorytmów sterowania współbieżnością. Pewnym krokiem w tym kierunku będzie *Atomic Locks* – implementacja rozszerzenia mechanizmu zamków w języku Java o algorytmy wersjonowania. Prace nad Atomic Locks są jednak wciąż w fazie początkowej.

## 2.4. Koncepcja mechanizmu operacji atomowych dla systemów opartych na paradygmacie SOA

Celem innego wątku zadania OB2-2, wspólnego z częścią zadania OB2-1, było opracowania koncepcji *Atomic REST*, tj. rozszerzenia stylu architektonicznego REST przeznaczonego do implementacji usług sieciowych o wsparcie dla rozproszonych operacji atomowych. Opis realizacji tych prac zamieszczono w oddzielnym raporcie technicznym [5]. Zaletą takiego wsparcia ma być ułatwienie koordynacji usług sieciowych, które komunikują się ze sobą korzystając z reguł oraz protokołów specyfikowanych przez podejście REST-owe. W praktyce, stosuje się w tym wypadku najczęściej protokół HTTP, który jest powszechnie używanym protokołem dostępu do danych hypermedialnych w ramach World-Wide Web (WWW). Tak więc naszym celem będzie rozszerzenie protokołu HTTP o mechanizm wsparcia dla atomowości, oparty na algorytmach wersjonowania. Z uwagi na to, że podejście REST-owe jest wymieniane obok standardów WS-* jako czołowe podejście do realizacji architektury SOA, mamy nadzieję, że nasza propozycja zyska zainteresowanie w środowisku twórców oprogramowania, w szczególności aplikacji biznesowych. Jej realizacja wymaga jednakże rozwiązania szeregu problemów technicznych, które wymieniono w raporcie. Niektóre z tych problemów wynikają ze specyfiki REST, inne z ograniczeń protokołu HTTP.

Pierwsza część raportu zawiera szereg 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 przedstawiono podstawowe założenia stylu REST, zwracając uwagę na takie cechy jak: adresowanie zasobów, bezstanowość interakcji, uniwersalny interfejs, buforowanie zapytań, oraz warstwowość architektury systemu. Termin REST używany jest w tym kontekście 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).

W raporcie 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 algorytmów wersjonowania 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 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 do algorytmów wersjonowania, 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. W ostatniej części raportu przedstawiono koncepcję REST-owego wsparcia operacji atomowych, pokazując na przykładzie przewidywany sposób korzystania z biblioteki Atomic REST.

# 3. Podsumowanie i kierunki dalszych prac

W ramach I etapu zadania OB2-2 powstały trzy raporty techniczne: (1) Analiza i porównanie wybranych realizacji Software Transactional Memory; (2) Analiza mechanizmów wsparcia dla operacji atomowych w systemach rozproszonych i opartych na paradygmacie SOA; i (3) Koncepcja usługi komunikacji grupowej oraz mechanizmu operacji atomowych dla systemów opartych na paradygmacie SOA. Jak pokazano w niniejszym podsumowaniu, implementacje wsparcia atomowości dla SOA przy użyciu pamięci transakcyjnej STM oraz algorytmów wersjonowania są interesujące zarówno z punktu widzenia praktycznych zalet tych rozwiązań, jak również z punktu widzenia nośnych celów naukowo-badawczych. Kolejnymi krokami w realizacji zadania OB2-2 będą: praktyczne porównanie wybranych realizacji STM, zaprojektowanie i implementacja biblioteki Atomic Locks, implementacja i walidacja mechanizmu analizatora kodu dla Atomic RMI i/lub Atomic Locks oraz rozszerzenia protokołu HTTP o operacje atomowe. W naszym zamierzeniu prace te pozwolą na opracowanie użytecznych metod i narzędzi wsparcia dla operacji atomowych dla usług sieciowych i aplikacji SOA.

# 4. Literatura

[1] Nir Shavit and Dan Touitou. Software transactional memory. In *Proceedings of PODCS '95: the 14th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, August 1995.

[2] 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.

[3] Piotr Kryger and Paweł T. Wojciechowski. Analiza i porównanie wybranych realizacji Software Transactional Memory. Technical Report TR-ITSOA-OB2-2-PR-09-2, Instytut Informatyki, Politechnika Poznańska, May 2009.

[4] Paweł T. Wojciechowski. Analiza mechanizmów wsparcia dla operacji atomowych w systemach rozproszonych i opartych na paradygmacie SOA. Technical Report TR-ITSOA-OB2-2-PR-09-1, Instytut Informatyki, Politechnika Poznańska, February 2009.

[5] 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.

[6] Paweł T. Wojciechowski. Extending atomic tasks to distributed atomic tasks. In *Proceedings of the Workshop on Exploiting Concurrency Efficiently and Correctly $(EC)^2$ (co-located with CAV '08: the 20th International Conference on Computer Aided Verification, Princeton, USA)*, July 2008.

[7] Timothy Harris and Keir Fraser. Language support for lightweight transactions. In *Proceedings of OOPSLA '03: the 18th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, October 2003.

[8] Tim Harris. Exceptions and side-effects in atomic blocks. *Science of Computer Programming*, 58(3):325–343, 2005.

[9] Tim Harris, Simon Marlow, Simon Peyton Jones, and Maurice Herlihy. Composable memory transactions. In *Proceedings of PPoPP '05: the 10th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, June 2005.

[10] Michael F. Ringenburg and Dan Grossman. AtomCaml: first-class atomicity via rollback. In *Proceedings of ICFP '05: the 10th ACM SIGPLAN International Conference on Functional Programming*, September 2005.

[11] Yang Ni, Adam Welc, Ali-Reza Adl-Tabatabai, Moshe Bach, Sion Berkowits, James Cownie, Robert Geva, Sergey Kozhukow, Ravi Narayanaswamy, Jeffrey Olivier, Serguei Preis, Bratin Saha, Ady Tal, and Xinmin Tian. Design and implementation of transactional constructs for C/C++. In *Proceedings of OOPSLA '08: the 23rd ACM SIGPLAN Conference on Object-oriented Programming, Systems Languages and Applications*, October 2008.

[12] Rachid Guerraoui, Maurice Herlihy, and Bastian Pochon. Polymorphic contention management. In *Proceedings of DISC '05: the 19th International Symposium on Distributed Computing*, volume 3724 of *LNCS*. Springer, 2005.

[13] XSTM. XSTM. `http://jstm4gwt.googlecode.com/files/Book.pdf`.

[14] Paweł T. Wojciechowski. *Language Design for Atomicity, Declarative Synchronization, and Dynamic Update in Communicating Systems*. Wydawnictwo Politechniki Poznańskiej, Pl. Marii Skłodowskiej-Curie 2, Poznań 60-965, Poland, 1st edition, 2007. 204pp.

[15] Paweł T. Wojciechowski. Isolation-only transactions by typing and versioning. In *Proceedings of PPDP '05: the 7th ACM-SIGPLAN International Symposium on Principles and Practice of Declarative Programming (Lisboa, Portugal)*, July 2005.

[16] 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.