AKADEMIA GÓRNICZO-HUTNICZA

Wydział Elektrotechniki, Automatyki,
Informatyki i Elektroniki

QoS driven Semantics Based SOA Applications Composition and Execution

---

Zarządzanie jakością kompozycji i wykonania w oparciu o informacje semantyczne aplikacji zorientowanych na usługi

Autoreferat rozprawy doktorskiej

Autor:
mgr inż. Tomasz Szydło

Promotor:
prof. dr hab. inż. Krzysztof Zieliński

Kraków, 10 października 2010

1 Wprowadzenie

Współczesne systemy komputerowe stają się coraz większe i bardziej złożone niż poprzednio. Muszą one zapewniać odpowiednią jakość usług przy równoczesnym skróceniu czasu na wprowadzenie aplikacji na rynek. Analizując projekty programistyczne z lat 2003 - 2006, naukowcy z firmy IBM zauważyli, że wiele z tych projektów jest zbudowana z reużytkowalnych komponentów które mogą być wykorzystywane w różnych produktach.

Architektura zorientowana na usługi *(ang. SOA)* pozwala na budowanie systemów komputerowych z luźno powiązanych reużytkowalnych serwisów. Systemy komputerowe są projektowane w taki sposób, że mogą w się adaptować do zmian w środowisku uruchomieniowym. Od procesu adaptacji oczekuje się aby był on dokonywany z możliwie minimalnym udziałem człowieka. W miarę rozbudowywania systemów komputerowych, proces adaptacji staje się problemem złożonym i stanowiącym wyzwanie.

1.1 Wykonanie i kompozycja aplikacji SOA

Obecnie aplikacje są tworzone w oparciu o usługi dostarczane przez różnych dostawców. Proces kompozycji i wykonania składa się ze zdefiniowania procesu biznesowego, który realizuje tworzoną funkcjonalność, a następnie jest on rozmieszczany i wykonywany na infrastrukturze wykonawczej. Architektura zorientowana na usługi *(ang. SOA)* jest szeroko akceptowaną architekturą dla integracji systemów komputerowych.

Aplikacja SOA składa się ze zbioru serwisów połączonych odpowiednio w platformie integracyjnej. Taka aplikacja opisana jest w postaci abstrakcyjnej, a następnie wykonywana w kontenerze uruchomieniowym. Kontener jest odpowiedzialny za odwzorowanie abstrakcyjnych serwisów na konkretne ich instancje w trakcie wykonania. Proces odwzorowywania może być dokonany przez użytkownika systemu albo przez odpowiedni moduł przed wykonaniem serwisu albo może być dokonany w trakcie wykonania serwisu złożonego. Podczas odwzorowania powinien być uwzględniony aspekt jakości dostarczanej usługi *(ang. QoS)*. Ze względu na luźne powiązanie serwisów w architekturze SOA, ta klasa aplikacji jest szczególnie odpowiednia do wprowadzenia mechanizmów rozszerzających funkcjonalność o elementy adaptowalności ponieważ instancje serwisów mogą być zamieniane w trakcie wykonania.

W literaturze wiele miejsca poświęcono rozwiązaniom dotyczącym procesu kompozycji i wykonania serwisów *(ang. Web Service Composition and Execution - WSCE)*[18][8][5]. Niemniej jednak większość prac proponuje rozwiązania polegające na automatycznym komponowaniu serwisów abstrakcyjnych na podstawie zadanego celu. Następnie tak wygenerowane rozwiązania są uruchamiane w odpowiednio przygotowanych kontenerach. Takie podejście zakłada bardzo precyzyjne zdefiniowanie celu kompozycji, tak aby stworzony serwis spełniał wszystkie wymagania użytkownika. W praktyce takie rozwiązanie napotyka na problemy wynikające z technologicznej niezgodności otrzymanego rozwiązania i jego integracji z już istniejącym oprogramowaniem. W szczególności dotyczy to obsługi błędów, transformacji protokołów komunikacyjnych i aspektów technologicznych będących wynikiem integracji elementów systemu od różnych dostaw-

ców. W nieniejszej pracy autor wprowadza nowatorską koncepcję kompozycji i wykonania aplikacji SOA z elementami adaptowalności. Programista będzie mógł stworzyć aplikację dbając o różnego rodzaju szczegóły techniczne, a następnie dostarczyć politykę adaptacji oraz model wykonania serwisu złożonego tak aby wzbogacić stworzoną aplikację o elementy adaptowalności.

Adaptowalność w trakcie wykonania wymaga szczegółowej wiedzy dotyczącej zachowania systemu i tego jak zmiana właściwości niefunkcjonalnych poszczególnych serwisów wpływa na wykonanie serwisu złożonego. W rozwiązaniach przedstawionych w literaturze, informacja o zachowaniu się systemu jest zaszyta w postaci automatycznie wygenerowanego serwisu złożonego, a podczas wykonania jest on odpowiednio modyfikowany. W rozwiązaniu przedstawionym w rozprawie, autor zakłada że aplikacje jest nieświadoma zastosowanej polityki adaptacji i nie jest w żaden sposób modyfikowana. Cała wiedza o zachowaniu aplikacja potrzebna do procesu adaptacji dostarczana jest w postaci statystycznego modelu wykonania serwisu złożonego. Taki model reprezentuje wiedzę o serwisie złożonym i jest uaktualniany w trakcie jego działania. Modele wykonania nie tylko filtrują zbyt szczegółową informację pochodzącą z systemu monitorowania, ale również wzbogacają opis aplikacji o aspekty niefunkcjonalne lub informacje semantyczne o jakości dostarczanej usługi z perspektywy klienta[3]. Obecnie, szyna integracji usług *(ang. ESB)* uważana jest za obiecujące rozwiązanie do integracji aplikacji SOA i jest wykorzystywana przez wiodące firmy branży informatycznej. ESB jest technologią integracyjną, która pozwala architektom na komponowanie aplikacji z serwisów wykorzystując różne protokoły komunikacyjne. Duże zainteresowanie szyną integracji usług przenosi rozważania dotyczące kompozycji i wykonania usług SOA do środowisk integracyjnych bazujących na przesyłaniu komunikatów. Dodatkowo, autor wprowadza koncepcje wirtualnej szyny ESB *(ang. Virtual ESB)*, która pozwala współdzielić jedną fizyczną instancję szyny ESB przez wiele logicznych instancji jednocześnie zwiększając skalowalność rozwiązania oraz upraszczając jego zarządzalność.

1.2 Semantyczny opis serwisów

Aktualnie tworzone serwisy przeznaczone są głównie do zarządzania i wykorzystania przez człowieka. Semantyczny opis serwisów bazuje na wiedzy człowieka o tworzonym systemie. Serwisy opisane semantycznie za pomocą ontologii pozwalają na automatyczne wyszukiwanie, selekcje oraz wykonywanie serwisów. Ontologia jest formalnym opisem koncepcji w konkretnej dziedzinie oraz opisem powiązań pomiędzy nimi. Począwszy od roku 1990 wiele rozpraw naukowych poświęcono zagdnieniom związanym z analizą jak koncepcja opisów ontologicznych może wpłynąć na rozwój WWW. Opisy semantyczne usług pozwalają na opisanie co serwis dostarcza, jak go używać oraz gdzie serwis jest dostępny. Te informacje stwarzają możliwośc budowania środowisk oferujących możliwość automatycznego wykrywania serwisów, komponowania oraz ich wykonywania.

1.3 Teza i cele badawcze rozprawy

Teza rozprawy została sformułowana następująco:

> *Service Oriented Architecture could be extended by mechanisms that enable adaptability of applications to the changes in the execution environment in accordance to an adaptation policy and semantic service description.*

W tłumaczeniu na język polski:

> *Architektura zorientowana na usługi może być wzbogacona o mechanizmy adaptowalności do zmian w środowisku uruchomieniowanym w oparciu o zdefiniowaną politykę adaptacji i semantyczne opisy serwisów.*

1.4 Osiągnięcia rozprawy

W ocenie autora, główne osiągnięcia prezentowanej rozprawy obejmują:

- zaproponowanie statystycznego modelu wykonania serwisu złożonego, który jest abstrakcyjną reprezentacją serwisu złożonego zawierającą informacje statystyczne na temat zachowania modelu w przeszłości;
- sformułowanie koncepcji kompozycji i wykonania aplikacji SOA wzbogaconej o mechanizmy adaptowalności w taki sposób, że aplikacja nie jest świadoma procesu adaptacji;
- sformułowanie koncepcji adaptacji aplikacji SOA wykorzystującej statystyczny model wykoania serwisu złożonego jak również politykę adaptacji;
- zaproponowanie mechanizmów rozszerzających warstwę intgracji usług dla systemów SOA o mechanizmy adaptowalności;
- prototypową implementację zaproponowanych koncepcji kompozyji i wykonania aplikacji SOA dla ogólnie przyjętej platformy integracyjnej;
- praktyczną weryfikację i ewaluację proponowanych koncepcji i rozwiązań.

2 Prace związane z obszarem badań

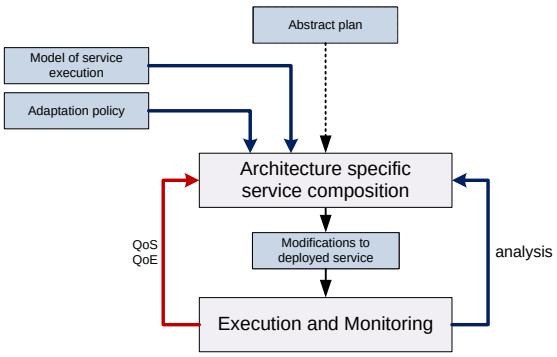
Istnieje szereg prac, które rozszerzaja ESB o elementy adaptowalności, ale zwykle te rozwiązania skupiają się na wyborze odpowiedniej instancji serwisu w trakcie wykonania bez analizy jak taka zmiana może wpłynąć na całość wykonania serwisu złożonego. Chang proponuje Dynamic Composition Handler(DCH)[6]. Procesy biznesowe są zdefiniowane z wykorzystaniem języka BPEL[2] i wykorzystują szereg serwisów. Część z nich może posiadać kilka instancji które realizują tą samą funkcjonalność, a DCH wybiera najbardziej odpowiednią instancję do aktualnych warunków wykonania.

Chen[7] prezentuje adaptowalną szynę usług (ang. ASB) *Abstract Routing Table* która do pewnego stopnia umożliwia dynamiczna kompozycję serwisów. Koncepcja polega na przeniesieniu informacji konfiguracyjnych dla serwisu do zewnętrznego repozytorium. Aplikacja może być modyfikowana w czasie wykonania poprzez zmianę konfiguracji w zewnętrznym repozytorium.

Interesujące wydaje się wykorzystanie modeli w procesie adaptacji i wykonania oprogramowania. Modele są odwzorowaniem systemu pokazując jego strukturę, zachowanie lub cele z perspektywy domeny problemu[3]. Modele mogą przedstawiać szereg aspektów działającego systemu oraz mogą zawierać dodatkowe informacje. Modele strukturalne pokazują jak system jest skonstruowany z wykorzystaniem obiektów oraz ich wykonaniem. Modele dynamiczne kładą nacisk na przedstawienie procesu wykonania systemu z uwzględnieniem przepływów lub śladów wykonania.

W prezentowanych rozwiązaniach założono, że abstrakcyjny model serwisu jest znany i może byc wykorzystany w procesie kompozycji i wykonania. Abstrakcyjny model jest automatycznie komponowany jako wynik semantycznej kompozycji serwisów lub jest opisany za pomocą języków takich jak BPEL gdzie zamiast konkretnych instancji serwisów używane są ich abstrakcyjne opisy. Kolejnym problem jest brak nawiązania do rozwiązań stosowanych w przemyśle głównie dlatego, że są one implementowane jako prototypowe rozwiązania w oderwaniu od istniejących i stosowanych środowisk wykonawczych.

Autor proponuje rozwiązanie, które wykorzystuje platformę integracyjną dla aplikacji SOA stosowaną w rozwiązaniach komercyjnych, a mechanizmy adaptowalności zaprojektowane są w taki sposób, że nie wymagają zmian w istniejącej aplikacji gdyż są dla.



Rysunek 1: Adaptacja z wykorzystaniem statystycznego modelu wykonania

3 Koncepcja kompozycji i wykonania aplikacji zbudowanej w oparciu o architekturę SOA

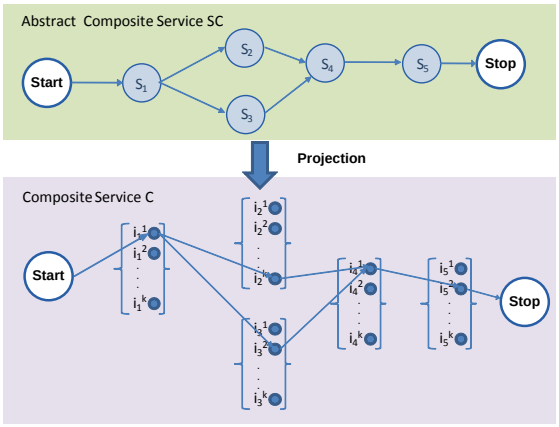
Analiza własności istniejących rozwiązań systemów adaptacyjnych prowadzi do konkluzji, że kompozycja i wykonanie serwisów złożonych w uwzględnieniem gwarancji jakości usług QoS może być osiągnięta poprzez rozszerzenie koncepcji SOA o adaptację z wykorzystaniem modeli statystycznych. Według autora wydaje się to być interesujące podejście do adaptacji w czasie wykonania. Modele statystyczne reprezentują wiedzę na temat działającego systemu i ukrywają nadmiarową informacje, dostarczając tylko tą która jest istotna dla domeny problemu. Uaktualnianie statystycznego modelu wykonania w czasie działania systemu wprowadza możliwość reagowania na zmiany w środowisku wykonawczym. Ten rozdział wprowadzana koncepcje statystycznych modeli wykonania serwisów złożonych i ich zastosowania w procesie adaptacji.

3.1 Serwis złożony

Koncepcja kompozycji i wykonania aplikacji SOA przedstawiona w rozprawie może być określona jako adaptacja z wykorzystaniem statystycznego modelu wykonania. System analizuje serwis złożony uruchomiony w środowisku wykonawczym i adaptuje go do zmian w nim występujących, a które mogą wpłynąć na zaburzenia dostarczanej jakości usług QoS i QoE.

Dostawca serwisu komponuje aplikacje w wybranej przez siebie technologii i dostarcza politykę adaptacji wraz ze statystycznym modelem wykonania tworzonego serwisu. Koncepcja przedstawiona jest na Rysunku 1 i rozróżnia cztery rodzaje serwisów:

- **Abstract Service** (s_k) Serwis, który jest opisany za pomocą swojej funkcjonalności oraz interfejsu, ale jego implementacja nie jest podana;
- **Service instance** (i_k^j) Instancja serwisu dostarczona przez jego dostawcę;
- **Composite service instance(C)** Serwis złożony, który składa się z instancji serwisów i może zostać wykonany;



Rysunek 2: Rzutowanie abstrakcyjnego serwisu złożonego na serwis złożony

- **Abstract composite service (SC)** Serwis złożony, który zawiera przynajmniej jeden serwis abstrakcyjny.

Abstrakcyjny serwis złożony SC składają się ze zbioru serwisów S oraz ze zbioru przejść T_s reprezentującego interakcję pomiędzy serwisami:

$$SC = (S, T_s) \tag{1}$$

gdzie:

$$S = \{s_1, \ldots, s_p\} \tag{2}$$

$$T_s \subset S \times S \tag{3}$$

Wykonanie zaczyna się w wierzchołku *Start* a kończy w wierzchołku *Stop*. Aby możliwe było wykonanie takiego serwisu, to dla każdego serwisu abstrakcyjnego należy wybrać odpowiednią instancję. Wprowadźmy pomocniczą funkcję *abstract*, która dla instancji serwisu zwróci odpowiadający mu serwis abstrakcyjny, oraz funkcję *instances*, która dla serwisu abstrakcyjnego zwróci zbiór instancji:

$$instances : S \to 2^I, instances(s_k) = \left\{ i_k^l \in I : abstract(i_k^l) = s_k \right\} \tag{4}$$

$$abstract : I \to S, \ \textit{zależy od technologii implementacji} \tag{5}$$

Kiedy dla każdego abstrakcyjnego serwisu złożonego s_k należącego do abstrakcyjnego serwisu złożonego SC konkretna instancja serwisu. Rysunek 2 przedstawia rzutowanie abstrakcyjnego serwisu złożonego SC na serwis złożony C:

$$C = (I_S, T_i) \tag{6}$$

gdzie:

$$I_S = \left\{ Start, Stop, i_1^{f(1)}, \ldots, i_p^{f(p)} \right\}, \bigwedge_{k=1,\ldots,p} abstract(i_k^{f(k)}) = s_k \tag{7}$$

$$T_i \subset I \times I, \bigwedge_{(i_a^k, i_b^l) \in T_i} \Big(abstract(i_a^k), abstract(i_b^l)) \Big) \in T_s \tag{8}$$

$$f : N \to N, f(k) \text{ wskazuje wybrana instancję dla k-tego serwisu abstrakcyjnego} \tag{9}$$

$$N = \{1, 2, \ldots\} \tag{10}$$

Jeden abstrakcyjny serwis SC generuje całą rodzinę $F(SC)$ serwisów złożonych, które następnie mogą być wykonane. Jeżeli dla każdego serwisu s_k można wybrać dowolną z jego instancji to dla całego serwisu SC, liczność rodziny $F(SC)$ wynosi:

$$|F(SC)| = \prod_{k=1}^{p} |instances(s_k)| \tag{11}$$

Proces adaptacji jest w tym wypadku ciągłą tranzycją pomiędzy serwisem abstrakcyjnym a serwisem złożonym zgodnie z zadana polityką adaptacji. Zwykle wybór odpowiedniego zestawu instancji dla serwisów opiera się na analizie parametrów niefunkcjonalnych. Przejście pomiędzy abstrakcyjnym serwisem złożonym a serwisem złożonym polega na wybraniu jednego rozwiązania C z całej rodziny $F(SC)$.

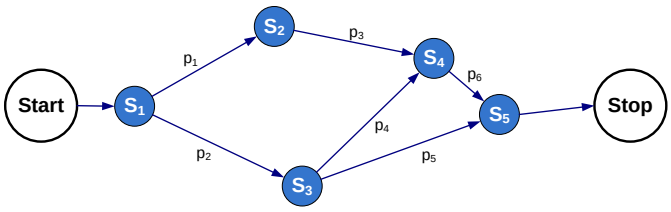
3.2 Model wykonania serwisu złożonego

Wykonanie serwisu złożonego stanowi zbiór ścieżek przetwarzania obserwowany w trakcie pojedynczego wykonania. Stanowi ono zatem konkretne przejście przez graf możliwych aktywności reprezentujący model M wykonania serwisu złożonego wraz z informacjami statystycznymi opisującymi ilościowo zachowanie się procesu złożonego podczas wykonania.

Model M wykonania serwisu złożonego jest opisany w postaci acyklicznego skierowanego grafu *(ang. DAG)* $M = (V, E)$ oraz posiada dwa wyróżnione wierzchołki *Start* i *Stop*. Wierzchołkami w grafie są serwisy abstrakcyjne, a krawędzie oznaczają, że dwa serwisy komunikują się ze sobą. Krawędź $(s_1, s_2) \in E$ oznacza zatem, że serwis s_1 przesyła wiadomości do serwisu s_2 z pewnym prawdopodobieństwem $p(s_1, s_2)$. Dodatkowo, poprzez $PRED(s_2) = \{s \in V : (s, s_2) \in E\}$ będziemy rozumieć listę poprzedników wierzchołka s_2, a $SUCC(s_1) = \{s \in V : (s_1, s) \in E\}$ oznacza listę następników wierzchołka s_1.

Dla każdego wierzchołka s_j, suma prawdopodobieństw na krawędziach wychodzących z niego sumuje się do 1, chyba że serwisy należące do $SUCC(s_j)$ wykonują się równolegle:

$$\sum_{s_j \in SUCC(s_k)} p(s_j, s_k) = 1 \tag{12}$$



Rysunek 3: Przykładowy statystyczny model wykonania serwisu złożonego

$$
\begin{aligned}
Time_C \quad = \quad & p_1 p_3 p_6 (t_{i_1} + t^{RTT}_{i_1,i_2} + t_{i_2} + t^{RTT}_{i_2,i_4} + t_{i_4} + t^{RTT}_{i_4,i_5} + t_{i_5}) \\
& + p_2 p_4 p_6 (t_{i_1} + t^{RTT}_{i_1,i_3} + t_{i_3} + t^{RTT}_{i_3,i_4} + t_{i_4} + t^{RTT}_{i_4,i_5} + t_{i_5}) \\
& + p_2 p_5 (t_{i_1} + t^{RTT}_{i_1,i_3} + t_{i_3} + t^{RTT}_{i_3,i_5} + t_{i_5})
\end{aligned} \tag{13}
$$

3.3 Podobieństwo serwisów

Zbiór $instances(s_i)$ dla każdego serwisu S_i może być podany przez administratora systemu lub może być uaktualniany i wyliczany w trakcie jego działania. Podobieństwo serwisów może być rozpatrywane w dwóch aspektach:

- **Zgodność interfejsów** – serwisy powinny udostępniać wspólny zbiór metod które można wykonać na serwisach;
- **Zgodność funkcjonalności** – funkcjonalność dostarczana przez serwisy powinna być identyczna.

Przykładowo, serwis i_a może dostarczać informacje o aktualnej temperaturze otoczenia, a serwis i_b o najniższej temperaturze z ostatniego tygodnia. Mimo, że oba serwisy mogą mieć identyczny interfejs, to ich funkcjonalność jest zdecydowanie różna. W związku z tym, te serwisy nie mogą być wykorzystywane zamiennie.

Niech funkcja $L(i_a, i_b) \in [0, 1]$ będzie fukcją określającą podobieństwo serwisów i_a oraz i_b. Wartość funkcji $L(i_a, i_b) = 1$ oznacza, że serwisy i_a oraz i_b mają jednakową funkcjonalność i interfejs, więc mogą być wykorzystywane zamiennie. Dla każdego serwisu abstrakcyjnego s_i prawdziwy jest zapis:

$$
\bigwedge_{i_1, i_2 \in instances(s_i)} L(i_1, i_2) = 1 \tag{14}
$$

Funkcja $L(i_a, i_b)$ jest zależna od technologii implementacji serwisów oraz również od technologii użytej do opisu funkcjonalności serwisów. W przypadku opisów semantycznych, rozpatruje się tutaj podobieństwo na poziomie ontologi dziedzinowej.

3.4 Proces adaptacji

Adaptacja w dużym uproszczeniu polega na ciągłym wybieraniu odpowiedniego serwisu złożonego C z całej rodziny serwisów $F(SC)$. W zależności od miejsca gdzie podejmowana jest decyzja o wyborze C, możemy podzielić proces adaptacji na dwie kategorie:

- **Statyczna adaptacja** przed wykonaniem serwisu złożonego;
- **Dynamiczna adaptacja** w trakcie wykonania serwisu złożonego.

3.4.1 Statyczna adaptacja

Przed wykonaniem złożonego serwisu abstrakcyjnego podejmowana jest decyzja który proces C z rodziny $F(SC)$ wywołać. Decyzja może być podejmowana na podstawie współczynnika dopasowania serwisu C do wymagań czy też na podstawie polityki adaptacji. Podczas wykonania serwisu złożonego ta decyzja jest niezmienna. Ilość rozwiązań może być bardzo duża, co w znacznym stopniu komplikuje przedstawiony algorytm, ale można wprowadzić szereg usprawnień heurystycznych, które w znacznym stopniu ograniczą przestrzeń rozwiązań i czas przetwarzania. Statyczna adaptacja potrzebuje jedynie do swojego działania ilościowe informacje o przebiegu historycznych wywołań serwisu złożonego. Narzuty na monitorowanie zachowania systemu są niewielkie, więc doskonale nadaje się do adaptacji lekkich serwisów o krótkim czasie działania.

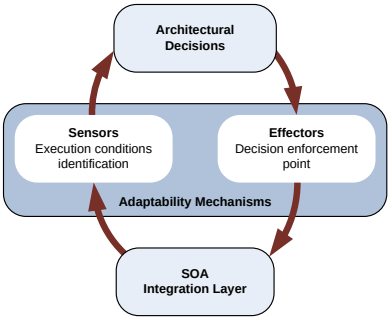
3.4.2 Dynamiczna adaptacja

W trakcie wykonania serwisu złożonego, przed każdym wykonaniem serwisu abstrakcyjnego podejmowana jest decyzja który proces C z rodziny $F(SC)$ wywołać. Przy czym proces decyzyjny bierze pod uwagę serwisy już wywołane w obrębie aktualnego wykonania serwisu złożonego, jak również przewidywaną jakość dostarczonej usługi analogicznie jak w przypadku statycznego wiązania. Podczas wykonania serwisu złożonego, ilość możliwych rozwiązań do wyboru maleje wraz z kolejnymi podejmowanymi wyborami.

3.5 Podsumowanie

W rozdziale przedstawiono koncepcję adaptacji aplikacji SOA z wykorzystaniem modelu wykonania serwisu złożonego. W omawianym rozwiązaniu aplikacja jest nieświadoma działającej strategii adaptacji. Z punktu widzenia SOA, statyczna adaptacja wydaje się być bardziej odpowiednia ze względu na mało znaczący narzut na proces adaptacji. Statystyczny model wykonania serwisu złożonego jest abstrakcyjnym opisem działającej aplikacji dla procesu adaptacji. Model może posłużyć do wygenerowania rodziny serwisów złożonych które moga być wykonane, a różniących się własnościami niefunkcjonalnymi. Cel procesu adaptacji definiuje kryteria wyboru konkretnego rozwiązania z tej rodziny serwisów złożonych.

Statystyczny model wykonania serwisu złożonego może być aktualizowany w trakcie działania aplikacji. Prezentowana koncepcja jest poparta wstępnymi badaniami autora w tym zakresie[17]. Pętla adaptacji obejmuje transparentny wybór serwisów w trakcie działania aplikacji nie wpływając na funkcjonalność samej aplikacji. Oznacza to, że przedstawiona koncepcja może być rozważana w kontekście środowiska integracyjnego, które może być rozszerzone o mechanizmy adaptowalności.



Rysunek 4: Rozszerzenia warstwy integracji o elementy adaptowalności

4 Koncepcja adaptowalnej szyny ESB

Zaproponowana koncepcja adaptowalnego wykonania serwisu złożonego koncentruje się na warstwie integracji serwisów i wymaga zdefiniowania elementów o które istniejące środowiska integracyjne powinny zostać rozszerzone.

Powszechnie stosowaną platformą integracyjną dla aplikacji zorientowaych na usługi jest szyna ESB, przez co autor proponuje rozbudowę tej platformy o mechanizmy adaptowalności. Jako, że jedną z głównych funkcjonalności ESB jest przesyłanie wiadomości pomiędzy serwisami to zmiana instancji serwisów użytych podczas wykonania aplikacji może się odbywać poprzez zmianę routingu wiadomości pomiędzy instancjami serwisów.

4.1 Mechanizmy adaptowalności

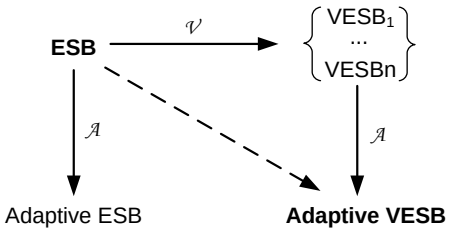
Autor proponuje w pracy transformacje, które rozszerzają warstwę integracji SOA o mechanizmy adaptowalności. Rysunek 4 przedstawia zaproponowany cykl przetwarzania informacji, który koresponduje do klasycznego podejścia sterowania w pętli zamkniętej.

Moduł podejmujący decyzje wpływające na architekturę aplikacji monitoruje system rozpoznając sytuacje wymagające reakcji. Monitorowanie odbywa sie za pomocą szeregu zainstalowanych sensorów. Pozyskane dane są następnie analizowane i jeżeli sytuacja tego wymaga to element *Architectural Decision Block* podejmuje odpowiednie decyzje które są wykonywane za pomocą zainstalowanych efektorów.

Szyna integracji usług ESB w swojej bazowej formie nie dostarcza mechanizmów adaptowalności. Autor proponuje dwie transformacje wzbogacające szynę integracji usług o dodatkowe mechanizmy:

- Transformacja $\mathcal{A}$ dostarcza mechanizów adaptowalności dla ESB. System zintegrowany z z wykorzystaniem adaptowalnej szyny ESB potrafi adaptować się do zmian w środowisku wykonawczym zgodnie z wysokopoziomowymi celami opisanymi za pomocą polityki adaptacji.

- Transformacja $\mathcal{V}$ rozszerza ESB o koncepcję wirtualnych ESB *(ang. Virtual ESB)*. Jest to mechanizm pozwalający na współdzielenie jednej fizycznej instancji ESB



Rysunek 5: Transformacje ESB

przez wiele wirtualnych szyn integracyjnych. Takie rozwiązanie zwiększa skalowalność szyny integracji usług oraz upraszcza zarządzenie aplikacją.

Rysunek 5 przedstawia omawiane transformacje. Obie transformacje mogą być dokonane niezależnie tworząc *Adaptive VESB*. Jednym z założeń kompozycji i wykonania aplikacji SOA przedtawionej w rozprawie jest transparentność propnowanego rozwiązania. Biorąc to pod uwagę zdefiniowano szereg wymagań niefunckjonalnych, które powinny zostać spełnione:

- płynna instalacja *sensorów* i *efektorów* w istniejącym środowisku wykonawczym;
- transparentność *sensorów* i *efektorów*;
- możliwośc zmiany strategii adaptacji podczas pracy systemu;
- niski narzut na zużycie zasobów obliczeniowych.

4.2 Transformacja $\mathcal{A}$

Przedstawiona transformacja dostarcza elementy potrzebne do zamknięcia pętli sterowania dla ESB. Jest ona wykorzystywana do wpływania na kompozycje aplikacji bez zmiany jej funkcjonalności. ESB jest szczególnie odpowiednia do implementacji proponowanej adaptacji, ponieważ może ona wpływać na routing wiadomoscu pomiędzy serwisami zgodnie z wysokopoziomową polityką.

Transformacja $\mathcal{A}$ jest procesem instrumentacji warstwy integracji, która instaluje *sensory* i *efektory* i jest ważnym elementem architektury *Adaptive VESB*. *Sensory* są odpowiedzialne za zbieranie informacji o działaniu szyny ESB i służą do aktualizacji modelu wykonania serwisu złożonego. *Efektory* są odpowiedzialne za wpływanie na środowisko wykonawcze, a w tym wypadku na routing wiadomości w szynie ESB.

Analiza określonych wymagań niefunkcjonalnych prowadzi do konkluzji, że mogą one byc spełnione przez wzorzec interceptora *(ang. Interceptor Design Pattern)*, który posłuży do implementacji *sensorów* i *efektorów*.

4.2.1 Sensory

System adaptowalny działa zgodnie z zadaną polityką, która wymaga szeregu informacji dotyczących działania szyny ESB. Zwykle takie informacje wymagają specjali-

zowanych sensorów dedykowanych do konkretnego zastosowania. Wspomniany wzorzec interceprtora pozwala na instalowanie i odinstalowywanie interceptorów w trakcie działania systemu.

Autor proponuje zastosowanie łańcucha interceptorów do pobierania informacji z sensorów. Każdy interceptor ma przypisany odpowiedni priorytet określający porządek przetwarzania wiadomości. Wiadomość jest przetwarzana w interceptorach w porządku zgodnie z malejącą wartością określonego priorytetu.

4.2.2 Efektory

Kiedy wiadomość jest przetwarzana przez NMR, algorytm routingu dopasowuje ją do reguł zapisanych w tablicy routingu. Kiedy reguła zostaje dopasowana do wiadomości, to intencjonalny serwis docelowy zostaje zamieniony na ten z reguły zapisanej w tablicy routingu. Kiedy wiadomość nie jest dopasowana do żadnej z reguł, to domyślna polityka przepuszcza taką wiadomość dalej, czyli NMR przesyła ją do intencjonalnej instancji serwisu.

Tablica 1: Tablica routingu NMR

Priority	VESB	CID	Intentional	EID	Destination
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

Reguły są pogrupowane zgodnie z elementami jakie są porównywane podczas ich przetwarzania. Każda grupa ma odpowiedni priorytet i wiadomości są dopasowywane do reguł zgodnie z rosnącą wartością priorytetu. Elementy które są porównywane przedstawia Tabela 1.

4.3 Transformacja $\mathcal{V}$

Koncepcja wirtualnej szyny ESB jest w swych założeniach podobna do sieci wirtualnych VLAN[10] stosowanych w sieciach komputerowych, ale na znacznie wyższym poziomie abstrakcji. Koncepcja przedstawiona w rozprawie zakłada, że zarówno serwisy jak i wiadomości moga byc przypisywane do konkretnych VESB.

4.3.1 Przypisanie serwisów

Serwisy dostępne w szynie ESB są albo przypisane do konkretnego *VESB* albo mogą być wykorzystywane w dowolnym *VESB*. Reguły przetwarzania wiadomości przez serwisy w zależności od przywiązania do konkretnych VESB przedstawiono w Tabeli 2.

Przykładowo serwisy przypisane do *VESB1* nie przetworzą wiadomości przypisanych do *VESB2*.

Tablica 2: Reguły przypisywania wiadomości do VESB

Service VESB	Message Input VESB	Message Processing	Message Output VESB
none	none	OK	none
none	*VESB1*	OK	*VESB1*
VESB1	none	OK	*VESB1*
VESB1	*VESB1*	OK	*VESB1*
VESB1	*VESB2*	—	—

Serwisy nie będące przypisane do konkretnych VESB mogą być postrzegane jako serwisy agnostyczne, co oznacza że mogą byc współdzielone pomiędzy różnymi VESB i niezależnie od tego do jakiego VESB wiadomość należy będzie ona przetwarzana przez serwis.

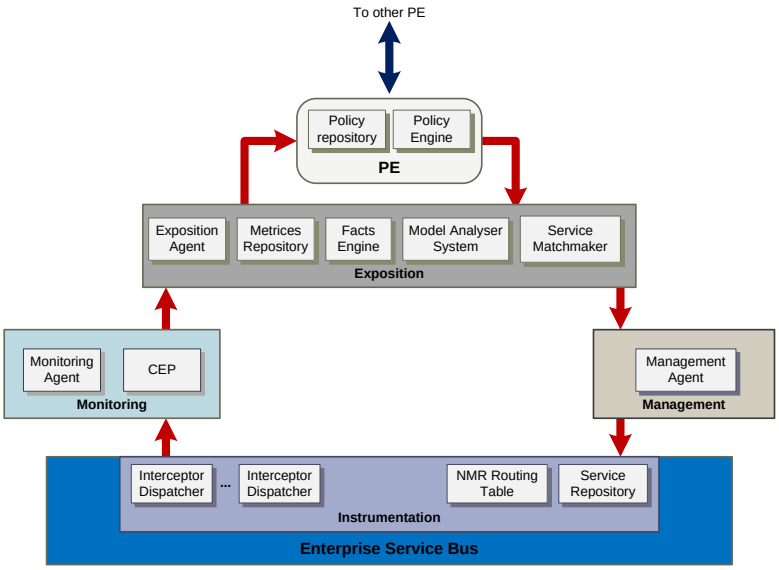
4.3.2 Przypisanie wiadomości do VESB

Wiadomości przesyłane poprzez szynę ESB są przypisywane do konkretnych VESB. Przypisanie może być dokonywane przez specjalizowany serwis, który analizuje zawartość wiadomości wyszukując przykładowo identyfikator klienta i odpowiednio przypisując do VESB na podstawie podpisanego uprzednio kontraktu na dostarczanie usług. Inna możliwość zakłada dynamiczne przypisanie wiadomości do VESB. W takiej sytuacji wiadomość jest przypisana do *VESB* serwisu który wystąpił jako pierwszy podczas przepływu wiadomości dla aktualnego wykonania serwisu złożonego. Takie rozwiązanie pozwala na stworzenie dedykowanych punktów dostępowych *(ang. Binding Components)* dla poszczególnych *VESB* i na tej podstawie rozróżnianie grup klientów.

4.3.3 Wykorzystanie VESB

Koncepcja VESB ma szeroki obszar zastosowań, w tym do:

- **Rozróżnianie klientów** – załóżmy, że dostawca usługi na podstawie podpisanych kontraktów SLA dostarcza klientom usługi o zróżnicowanej jakości. W typowym rozwiązaniu zarządzanie takim wykonaniem wymaga śledzenia każdego wywołania i każdej wiadomości przesyłanej w szynie ESB, a następnie ich grupowania na podstawie identyfikatora klienta i podejmowania odpowiednich decyzji. W przypadku zastosowania koncepcji VESB, dostawca może przypisać konkretne wiadomości na podstawie identyfikatora klienta czy też serwisy do różnych VESB i na podstawie analizy wirtualnych szyn integracji usług odpowiednio zarządzać wykonaniem.
- **Grupowanie klientów** – koncepcja jest również przydatna w sytuacji, gdy dostawca rozróżnia kilka grup klientów o zróżnicowanej jakości dostarczanych usług



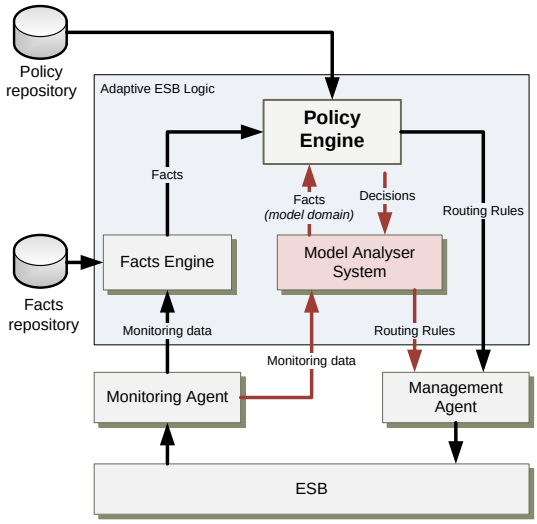
Rysunek 6: Warstwowa architektura *Adaptive VESB*

tj. złotych, srebrnych i brązowych. W celu uproszczenia zarządzania takim systemem istnieje możliwość przypisania każdej z grup klientów do osobnych VESB i odpowiedniego przypisania polityki adaptacji.

4.4 Warstwowa architektura systemu

Rysunek 6 przedstawia warstwową architekturę *Adaptive VESB* z zaznaczonymi warstwami funkcjonalnymi oraz ich powiązaniem. Architektura systemu zawiera pięć warstw:

- **Enterprise Service Bus(warstwa instrumentacji).** Szyna ESB wzbogacona o elementy umożliwiające adaptację wykonania serwisów na niej uruchomionych.
- **Monitoring.** Zbieranie informacji o stanie systemu, jak również monitorowanie wykonania serwisów celem wykorzystania tych informacji do adaptacji.
- **Management.** Komponenty służące do zarządzania wykonaniem serwisów na szynie ESB poprzez zmianę tablicy routingu.
- **Exposition(warstwa ekspozycji).** Komponenty, których zadaniem jest wyeksponowanie stanu ESB, oraz informacji z monitoringu serwisów w postaci faktów dla silników regułowych. Komponenty w tej warstwie odpowiedzialne są również za wykonywanie wysokopoziomowych operacji na szynie ESB oraz analizę modelu statystycznego.
- **Policy Engine.** Silniki regułowe odpowiedzialne za wykonanie polityki adaptacji oraz podejmowanie decyzji.



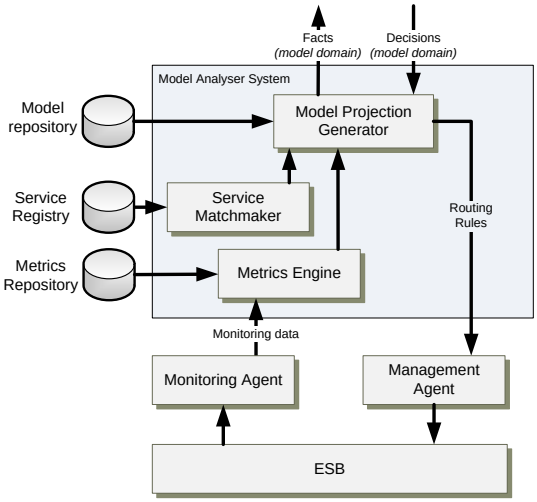
Rysunek 7: Wewnętrzna architektura *Adaptive ESB*

Przepływ wiadomości pomiędzy elementami wchodzącycmi w skład tej architektury przedstawiony jest na Rysunku 7. Ze względu na dużą ilość informacji pochodzących z monitorowania szyny ESB, zaobserwowane zdarzenia są korelowane przez agenta monitorującego *(Monitoring Agent)* w zdarzenia złożone *(ang. Complex Event Processing)*[11] na podstawie zdefiniowanych wyrażeń i tylko takie informacje są przekazywane do warstwy ekspozycji. Agent zarządzający *(Management Agent)* pozwala na modyfikowanie tablicy routingu w szynie ESB.

Silnik faktów *(Facts Engine)* zamienia informacje o zdarzeniach złożonych w fakty, na podstawie których odbywa się wnioskowanie przez silnik polityk *(Policy Engine)*. Warstwa ekspozycji zawiera również moduł analizy modelu *(Model Analyzer System)*, który na podstawie dostarczonego modelu wykonania serwisu złożonego przedstawia ESB i dostępne instancje serwisów w postaci wysokopoziomowej abstrakcji w domenie w której dokonuje się adaptacji.

4.5 Moduł analizy modelu

Model wykonania serwisu złożonego jest abstrakcją działającej aplikacji w środowisku wykonawczym, która pomija szczegóły dotyczące wykonania aplikacji, a dostarcza silnikowi przetwarzania polityk *(Policy Engine)* informacje z domeny problemu. Decyzje podjęte w domenie problemu są następnie zamieniane na operacje, które mogą byc wykonane w środowisku wykonawczym. Moduł analizy modelu połączony jest z szyną integracji usług za pomocą agenta monitorującego i zarządzającego. Wewnętrzna architektura tego elementu przedstawiona jest na Rysunku 8. Do głównych zadań modułu należy uaktualnianie modelu, znajdowanie przestrzeni rozwiązań poprzez rzutowanie serwisów abstrakcyjnych z modelu na konkretne instancje serwisów oraz wyliczanie oczekiwanych wartości metryk dla znalezionych rozwiązań.



Rysunek 8: Wewnętrzna architektura modułu analizy modelu

Repozytorium metryk *(Metrics Repository)* zarządza definicjami metryk, które są wykorzystywane do obliczania oczekiwanych wartości dla zdefiniowanych modeli wykonania. *Service Matchmaker* jest modułem odpowiedzialnym za analizę i porównanie serwisów pod względem semantycznym. Jako rezultat dostarcza grupy serwisów o odpowiadającej sobie funkcjonalności, a więc takich które mogą być wykorzystywane zamiennie.

Na podstawie statystycznego modelu wykonania serwisu złożonego oraz zbiorów równoważnych serwisów, moduł projekcji *(Model Projection Generator)* wylicza przestrzeń rozwiąń oraz oczekiwane wartości metryk dla każdego rozwiązania, które są następnie wykorzystywane przez moduł przetwarzania polityk adaptacji.

Zdefiniowana polityka adaptacji może podejmować akcje bazując na dostarczonych faktach w domenie modelu. Przykładowo, podjęcie decyzji, że klient nie powinien mieć dostępu do dostarczanych usług wymaga zwykle informacji o stanie konta użytkownika. Jeżeli natomiast ten sam klient oczekuje odpowiendiej jakości usług, to polityka adaptacji będzie uwzględniać fakty dostarczone przez moduł analizy modelu.

4.6 Podsumowanie

Rozdział definiuje rozszerzenia szyny integracji usług. Rozszerzenie ESB o elementy adaptowalności pozwala na dynamiczną i transparentną podmianę instancji serwisów wykorzystywanych w trakcie wykonania serwisu złożonego w myśl zadanej polityki adaptacji. Kolejne rozszerzenie ESB o elementy wirtualizacji jest technicznym aspektem poprawy zarządzalności i skalowalności poprzez możliwość współdzielenia jednej fizycznej instancji szyny integracji usług prze wiele logicznych. Przedstawione rozszerzenia mogą być wdrożone jednocześnie tworząc *Adaptive VESB*. Przedstawiono również proponowaną warstwową architekturę rozwiązania, a w kolejnym rozdziale opisano prototypową implementację systemu.

5 Implementacja systemu

W celu weryfikacji przestawionych koncepcji kompozycji i wykonania aplikacji SOA, rozszerzono funkcjonalność szyny integracaji usług o elementy adaptowalności.

Dostępne implementacje szyn ESB porównano pod kątem zgodności ze specyfikacją JBI, rozpoznawalności produktu, społeczności rozwijającej produkt oraz ich możliwości w zakresie szeroko pojętej adaptacji. W wyniku analizy wybrano implementację ServiceMix 3.3 do wzbogacenia jej o przedstawione koncepcje.

5.1 Rozszerzenia szyny ESB

Ze względu na to, że elementy rozszerzające warstwę integracji są mocno związane z technologią jej implementacji, autor zdecydował o zaprojektowaniu tych elementów w sposób lekki przenosząc większość przetwarzania na warstwę ekspozycji.

Sensory i efektory zostały zaimplementowane przy użyciu programowania aspektowego, co pozwala na ich uruchomienie bez modyfikacji kodu źródłowego szyny ESB. Agenta monitorującego oraz agenta zarządzającego zaimplementowano jako serwisy uruchomiony w szynie ESB. Agent monitorujący korzysta z bliblioteki Esper[16] do zaawansowanego przetwarzania zdarzeń i ich korelacji. Agent zarządzający udostępnia funkcjonalność modyfikacji tablicy routingu szyny ESB. Komunikacja agentów z warstwą ekspozycji odbywa się za pomocą technologii JMX. Szczegółowy opis implementacji rozszerzeń ESB jest przedstawiony w rozprawie.

5.2 Komponenty warstwy ekspozycji

Do implementacji elementów warstwy ekspozycji wykorzystano środowisko OSGi[1], który pozwala na dynamiczne ładowanie i zarządzanie cyklem życia modułów wchodzących w skład aplikacji. Zestaw mechanizmów wbudowanych w OSGi pozwala na automatyczne wykrywanie oraz odszukiwanie dostępnych usług i ich wykorzystanie.

5.2.1 Silnik faktów

Silnik faktów jest odpowiedzialny za eksponowanie w postaci faktów zdarzeń zaistniałych w warstwie instrumentacji. Fakty są definiowane jako adnotowane klasy POJO i instancjowane wtedy kiedy dane zdarzenie zaistnieje. Powiązanie fakty z rzeczywistym zdarzeniem jest opisywane za pomocą odpowiednio zdefiniowanych adnotacji.

Fakt określający czas wykonania serwisu może być określony za pomocą wyrażenia EQL*(Esper Query Language)* opisującego zdarzenie złożone. Jeżeli dla każdego serwisu ma istnieć osobna instancja faktu reprezentująca czas wykonania pojedynczego serwisu, to należy za pomocą adnotacji `@CEPUnique` określić pole które będzie rozróżniać instancje faktów. Może to być np. nazwa serwisu.

Listing 1 przedstawia przykładowy fakt, który zostanie zainstancjonowany dla każdej instancji serwisu w ESB i będzie reprezentowa średnie wykorzystanie danej instancji w ciągu każdej sekundy.

Listing 1: Przykładowa definicja faktu

```
package pl.edu.soa.agh.as3.vesb.examples.simple;
import pl.edu.soa.agh.as3.vesb.components.facts.*;

//CEP statement definition
@CEPFact(statement="select sname, count(*) as cnt
            from MessageEvent.win:time_batch(1 second)")
public class InvocationsRate {

    //methods definition and mapping
    @CEPUnique
    @CEPProperty(property = "sname")
    public String serviceName;

    @CEPProperty(property = "cnt")
    public String invocationsRate;

    //setters and getters
    public String getServiceName() {
      return serviceName;
    }

    public void setServiceName(String serviceName) {
      this.serviceName = serviceName;
    }
}
```

5.2.2 Moduł analizy modelu

W prototypowym rozwiązaniu do opisu semantycznego serwisów wykorzystano podzbiór ontologii OWL-S[4]. W celu znalezienia zbiorów serwisów, które mogą byc wykorzystywane zamiennie w module semantycznego dopasowania *(Service Matchmaker)* zastosowano algorytm oparty na rozwiązaniu zaproponowanym przez M.Paolucci[12].

Na podstawie modelu wykonania serwisu wyliczane są możliwe zbiory instancji serwisów jakie mogą być wykorzystane podczas kolejnych wywołań serwisu złożonego, a następnie wyliczane są spodziewane wartości metryk dla każdego z rozwiązań. Metryki są implementowane w postaci serwisów OSGi i są dostępne z poziomu silnika przetwarzania metryk*(Metrics Engine)*.

Wyliczanie wartości metryk złożonych na modelu wymaga zdefiniowania algorytmu lub funkcji jaką należy wykonać na grafie oraz wierzchołkach grafu, tak aby uzyskać wartość zadanej metryki. Analiza metod wyliczania typowych metryk dla serwisów złożonych doprowadziła do opracowania algorytmu *MapGraphReduce*, który w znacznym stopniu upraszcza ich implementację. Główna idea polega na podzieleniu obliczeń na dwa etapy: (i)wyliczanie wartości tymczasowych niezależnie w każdym wierzchołku grafu, (ii)

agregowanie poprzednio wyliczonych wartości tymczasowych w wierzchołkach zgodnie z porządkiem sortowania topologicznego grafu.

Uaktualnianie modelu wykonania polega na wyliczaniu aktualnych wartości prawdopodobieństwa tranzycji na krawędziach modelu. W sytuacji gdy statystyczny model wykonania serwisu złożonego zawiera wszystkie serwisy pośredniczące w przesyłaniu wiadomości, to wartość prawdopodobieństwa na krawędziach modelu jest proporcjonalna do ilości wiadomości przesyłanych pomiędzy serwisami, a odwrotnie proporcjonalna do ilości wiadomości wychodzących z węzła źródłowego krawędzi. Ze względu na to, że tak zdefiniowany model wykonania serwisu złożonego opisuje usługę na abstrakcyjnym poziomie to może nie zawierać wszystkich serwisów pośredniczących w przekazywaniu wiadomości. W szczególności może nie zawierać wzorców integracji biznesowej EIP *(Enterprise Integration Patterns)*[9]. W rozwiązaniu prototypowym przyjęto, że ilość wiadomości przesłanych pomiędzy serwisami s_x i s_y jest wartością minimalną ilości wiadomości przesyłanych pomiędzy serwisami na najkrótszej ścieżce pomiędzy s_x i s_y. Rozwiązanie wydaje się nie umniejszać skuteczności przedstawionej koncepcji wykorzystania statystycznych modeli wykonania serwisów złożonych. W celu efektywnego wyliczania najkrótszych ścieżek w grafie, w narzędziu zastosowano zmodyfikowaną wersję algorytmu Bellmana-Forda.

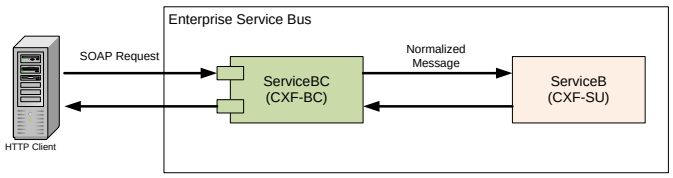
5.3 Silnik przetwarzania polityk

Silnik przetwarzania polityk adaptacji jest zarejestrowany w silniku przetwarzania faktów na notyfikacje o zdarzeniach dostarczane przez w w chwili zaobserwowania określonych zdarzeń złożonych. Dodatkowo do silnika przetwarzania polityk dostarczane są przez model analizy systemu możliwe odwzorowania modelu wykonania serwisu złożonego na zbiór instancji serwisów i oczekiwane wartości metryk. Polityka adaptacji może wpływać na środowisko wykonawcze poprzez zmianę reguł w tablicy routingu. W implementacji wykorzystano silnik regułowy Drools[13] w wersji 5.0.

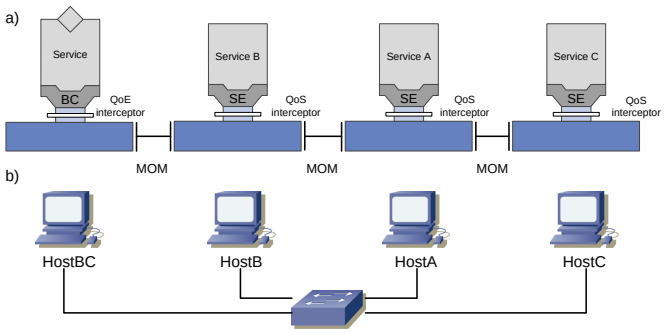
5.4 VESB Studio

W celu uproszczenia zarządzania Adaptowalnym VESB została stworzona graficzna konsola *VESB Studio*. Pozwala ona na zarządzanie aspektami adaptowalności ESB i jest przewidziana do używania przez administratorów systemu. Jej funkcjonalność pozwala na monitorowanie warstwy instrumentacji, edytowanie modeli wykonania serwisów złożonych oraz zarządzanie politykami apdatacji.

Jako środowisko graficznego interfejsu użytkownika wykorzystano Eclipse RCP. Jest to otware środowisko stworzone w języku Java, którego funkcjonalnośc może być rozszerzena poprzez instalowanie wtyczek z dodatkową funkcjonalnością.



Rysunek 9: Testowa aplikacja zbudowana z wykorzystaniem wzorców EIP



Rysunek 10: Testowa aplikacja a)architektura logiczna b)architektura fizyczna

6 Weryfikacja

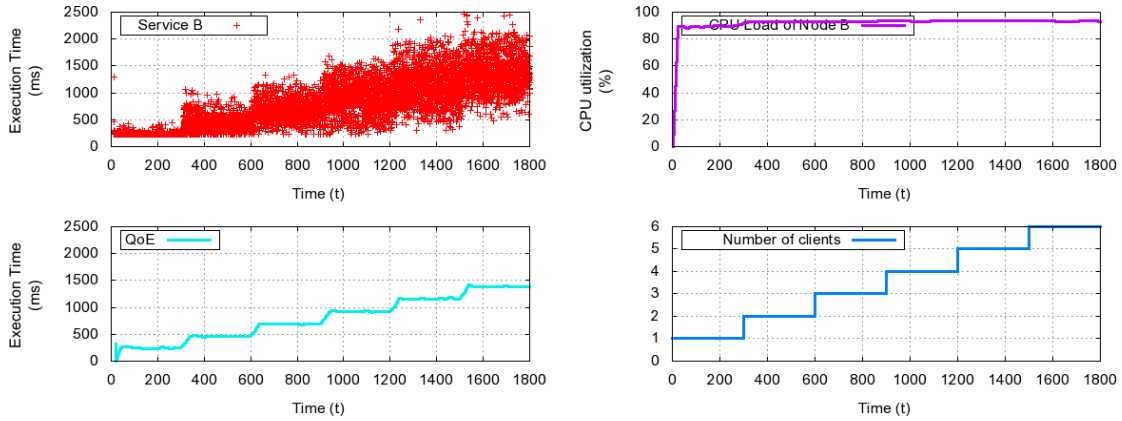
Głównym celem wykonanych testów była weryfikacja i ewaluacja zaproponowanych koncepcji i stworzonego prototypowego rozwiązania. W eksperymentach wykorzystano testowe aplikacje w których nacisk położono głównie na wymagania niefunkcjonalne. Eksperymenty zakładały porównanie działania aplikacji oryginalnej i adaptowanej zgodnie z zadaną polityką. Pierwszy eksperyment wykorzystuje wszystkie elementy stworzonego systemu za wyjątkiem modułu analizy modelu. Drugi eksperyment wykorzystuje statystyczny model wykonania, w znacznym stopniu redukując złożoność polityki adaptacji.

6.1 Eksperyment 1

Przedstawione badanie ma na celu weryfikację jak opisywane rozwiązanie zachowuje się przy zastosowaniu prostej strategii adaptacji i adresuje głównie opisywane mechanizmy adaptowalności wprowadzone jako rozszerzenie szyny ESB. Aplikacja, która jest przedmiotem tego testu jest udostępniona, jako web serwis i zbudowana w oparciu o wzorce integracji biznesowej EIP.

Test został przeprowadzony na sfederowanej szynie ESB zbudowanej z trzech instancji połączonych siecią komputerową. Wszystkie instancje ServiceMix ESB działają na różnych serwerach tak jak to przedstawia Rysunek 10.

Maszyny będące elementami infrastruktury należą do Sun Blade System 6000[15], który pozwala na łączenie urządzeń o różnych architekturach procesorów oraz o różnych systemach operacyjnych. W omawianym teście użyto cztery maszyny:



Rysunek 11: Wykonanie aplikacji niezmodyfikowanej

- 2 x Sun Blade X6220 – dwa procesory AMD Opteron 3GHz, 1MB pamięci podręcznej poziomu drugiego oraz 4 GB pamięci RAM (uruchomione serwisy `ServiceC` i `ServiceB`)
- 2x Sun Blade X6250 – dwa procesory Intel Xeon 3.5GHz, 12 MB pamięci podręcznej poziomu drugiego oraz 4GB pamięci RAM (uruchomione serwisy `ServiceA` oraz `ServiceBC`)

Serwery używane w tym teście połączono siecią LAN o przepustowości 1Gbit, a zainstalowane oprogramowie podsumowano w Tabeli 3.

Tablica 3: Konfiguracja platformy testowej

Software	Version
Operating system	Solaris 10 update 4
JVM	Java(TM) SE Runtime Environment (build 1.6.0_16-b01)
JBI implementation	Apache ServiceMix 3.3

6.1.1 Test 1

W pierwszym teście aplikacja została uruchomiona w niezmodyfikowanym środowisku i tylko `ServiceB` przetwarzał wywołania klientów. Wyniki testu przedstawia Rysunek 11. Średnio przetwarzanie zajmuje około 250ms wliczając w to komunikację pomiędzy węzłami ESB. Czas odpowiedzi wzrasta prawie liniowo do liczby użytkowników korzystających jednocześnie z serwisu. Przykładowo, kiedy sześciu klientów jednocześnie korzysta z serwisu, czas odpowiedzi sięga 1400ms czyniąc ten serwis praktycznie bezużytecznym.

6.1.2 Test 2

Test rozpatruje przypadek, w którym dostawca usługi ma dostępnych kilka maszyn, które mogą służyć do świadczenia usługi, ale zakłada się, że jednocześnie nie powinno

pracować więcej maszyn niż to jest konieczne. Dostawca przyjął, że dopuszczalnym średnim czasem odpowiedzi serwisu jest 500ms.

Ponieważ ten test nie wykorzystuje modelu wykonania serwisu złożonego, wszystkie informacje potrzebne do adaptacji muszą być zawarte w polityce adaptacji. Polityka zawiera więc reguły adaptacji jak i nazwy serwisów, które mogą być wykorzystane zamiennie z serwisem `ServiceB`. Tablica 4 zawiera politykę adaptacji złożoną z siedmu reguł.

Tablica 4: Prosta polityka apdatacji

Rule	Description
`Self-learning initialization`	Initializes routing from `ServiceB` to `ServiceB`
`Adaptation - stage up`	Rule triggered by QoE fact. When QoE > 520ms then system should go one stage up.
`Adaptation - stage down`	Rule triggered by QoE fact. When QoE < 480ms then system should go one stage down.
`Adding routing for Service A`	Rule triggered by stage up modification if there is not routing for Service A already.
`Removing routing for Service A`	Rule triggered by stage down modification if there is not routing for Service C already.
`Adding routing for Service C`	Rule triggered by stage up modification if there is routing for Service A already.
`Removing routing for Service C`	Rule triggered by stage down modification if there is routing for Service C already.

Graficzny interfejs użytkownika *VESB Studio* został wykorzystany do stworzenia polityki adaptacji, a następnie po dołączeniu do zdalnej instancji adaptowalnego ESB, gdzie polityka została zainstalowana i uruchomiona.

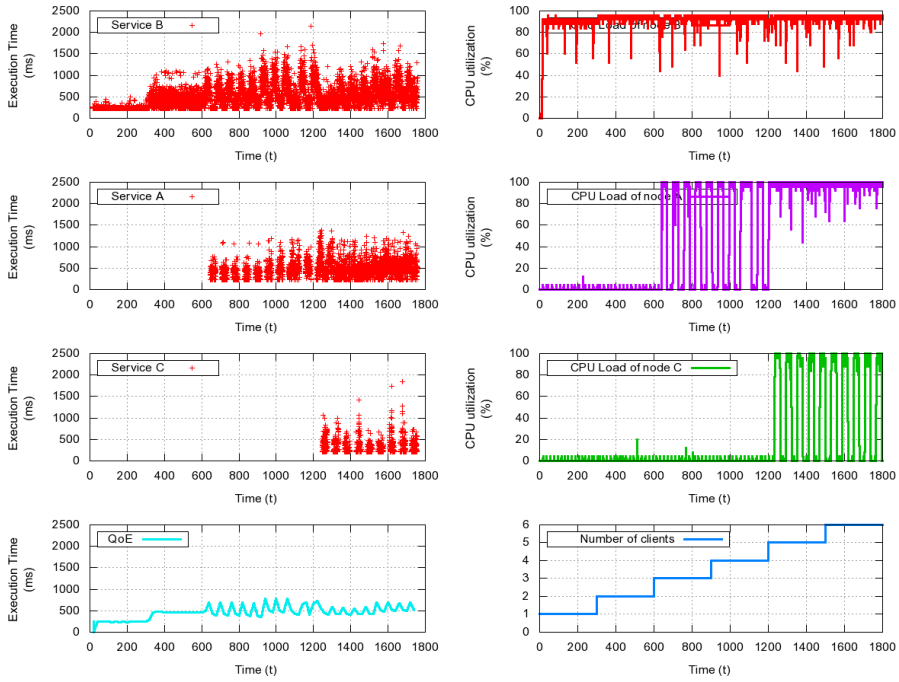
Informacja dotycząca aktualnej wartości QoE jest przesyłana, jako fakt do polityki adaptacji, która następnie wpływa na zawartość tablicy routingu wzbogaconej szyny ESB.

Rysunek 12 przedtawia wynik działania przedstawionej polityki adpatacji. W przypadku dużej ilości zapytań QoS obserwowany przez klienta oscyluje wokół zadanej wartości. Przy małej ilości zapytań, tak że są one przetwarzane przez jedną instancję serwisu, to czas odpowiedzi jest mniejszy niż zadany, gdyż system nie ma możliwości modyfikacji konkretnych instancji, a jedynie na politykę ich wyorzystania.

6.1.3 Test 3

W tym scenariuszu, tablica routingu zawiera trzy reguły przedstawione w Tabeli 5 i nie jest wykorzystywana żadna polityka adaptacji.

Kiedy jest kilka reguł routingu z tym samym priorytetem pasujących do przesyłanej wiadomości, to są one wykorzystywane zgodnie z regułą *Round-Robin*. W takim wypadku wiadomości są sprawiedliwie przesyłane do dostępnych instancji. Rysunek 13 przedstawia wykorzystanie dostępnych instancji serwisów. Czas odpowiedzi w tym wypadku



Rysunek 12: Wykonanie aplikacji z polityką adaptacji

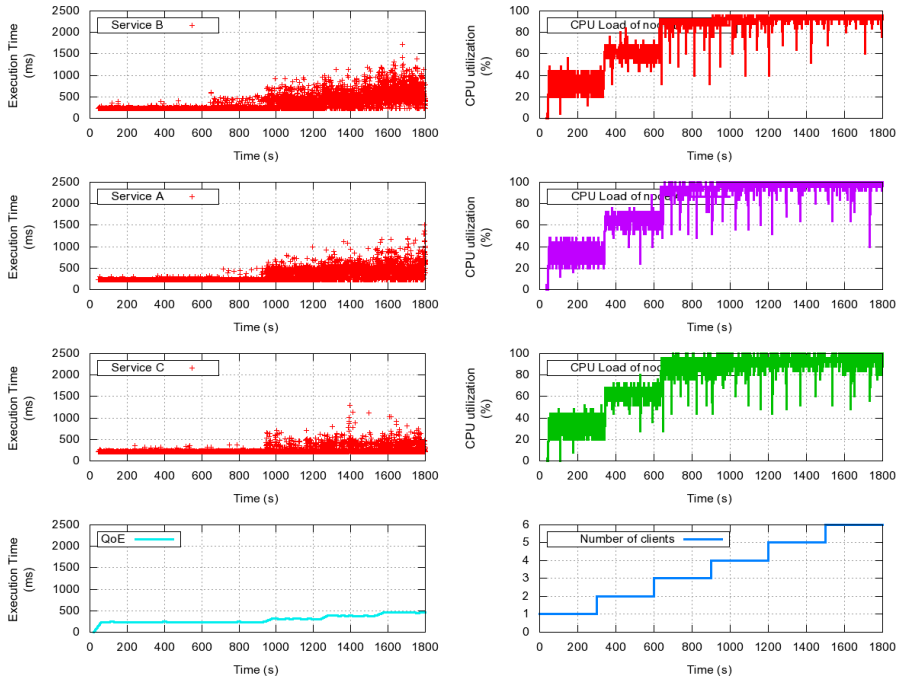
Tablica 5: Wpisy w tablicy routingu dla testu III

Priority	VESB	CID	Intentional	EID	Destination
4	n/a	n/a	ServiceB	n/a	ServiceA
4	n/a	n/a	ServiceB	n/a	ServiceC
4	n/a	n/a	ServiceB	n/a	ServiceB

dla zwiększającej się liczby klientów był znacznie poniżej 500ms, ale wszystkie serwery ze względu na to, że przetwarzały żądania musiały być aktywne.

6.1.4 Podsumowanie

Omawiane rozwiązanie zostało z powodzeniem zastosowane do adaptacji aplikacji z wykorzystaniem prostej strategii adaptacji. Główne elementy składające się na zamkniętą pętlę sterowania zostały przetestowane. Dodatkowe serwery na których były uruchomione instancje serwisów były włączane tylko wtedy, kiedy już działające instancje serwisów nie byłyby zdolne do przetwarzania zapytań bez naruszenia kontraktów SLA. Ilość wykorzystywanych instancji serwisów była zależna od ilości klientów zainteresowanych usługą. W ostatnim scenariuszu badawczym żądania klientów były przetwarzane zgodnie z regułą *Round-Robin* toteż wszystkie dostępne maszyny były uruchomione przez cały czas korzystania z serwisu. W obydwu przypadkach kontrakt SLA nie był złamany, a zastosowane polityki adaptacji mogą być użyte do zarządzania i oszczędzania energii konsumowanej przez serwery i ograniczenia ich czasu pracy.



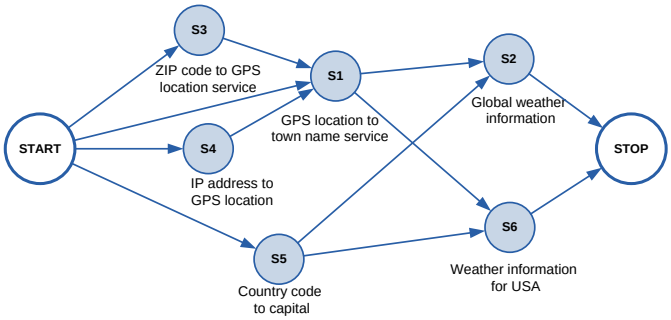
Rysunek 13: Wykonanie aplikacji z wykorzystaniem wielu instancji serwisów

6.2 Eksperyment 2

Celem tego eksperymentu jest weryfikacja jak stworzony system może być wykorzystywany ze złożoną polityka adaptacji. W przeciwieństwie do poprzedniego eksperymentu, w procesie adaptacji wykorzystany będzie statystyczny model wykonania serwisu złożonego.

6.2.1 Analiza modelu wykonania serwisu złożonego

Celem tego testu jest weryfikacja jak model wykonania serwisu złożonego jest uaktualniany w trakcie wykonania aplikacji i jak informacje w nim zawarte mogą być wykorzystane do poprawy działania aplikacji. W tym celu zdefiniowano profil użytkowników



Rysunek 14: Złożony serwis abstrakcyjny

korzystających z serwisu. Zakładając, że serwis pogodowy jest dostępny dla użytkowników z całego globu, profil użytkowników powinien uwzględniać przesunięcia stref czasowych oraz różne wykorzystanie serwisu dla klientów korzystających z urządzeń mobilnych. W celu obserwacji zmian zachodzących w modelu zaadptowano profile aktywności użytkowników w sieci internet bazując na danych statystycznych jednego z dostawców Internetu[14] w Polsce.

Rysunek 15 przedstawia profile wykorzystywania serwisu przez użykowników służące do analizy zachowania statystycznego modelu wykonania serwisu złożonego. Jak można zauważyć największa aktywność użytkowników przpada na godziny pomiędzy 10 a 12 oraz w godzinach popołudniowych około godziny 18. Jednocześnie można zaobserwować przesunięcia czasowe tych aktywności odpowiednio dla USA, Europy, Azji, Polski oraz Anglii. Profil dla użytkowników mobilnych jest zdecydowanie różny – typowa aplikacja na urządzenie mobilne działa w tle i w odpowiednich interwałach czasowych aktualizuje swoje dane niezależnie od używania telefonu przez użytkownika. Dlatego też dla klientów mobilnych przyjęto jednostajne wykorzystanie serwisu pogodowego w ciągu dnia.

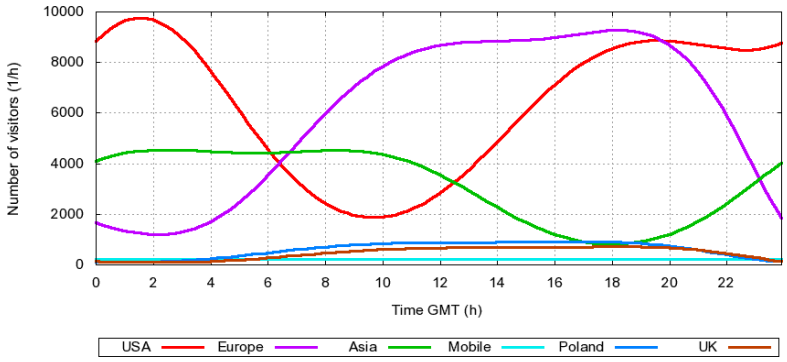
Rodzaj danych lokalizacyjnych zawartych w zapytaniu o aktualna pogodę może się znacznie różnić w zależności od kraju z którego ono pochodzi. Przykładowo w Anglii kody pocztowe są rozmieszczone tak gęsto, że służą jako określenie konkretnego adresu z dokładnością niejednokrotnie do konkretnych budynków, domów czy nawet mieszkań. W zapytaniach od użytkowników mobilnych dominować będzie przedstawienie lokalizacji w postaci pozycji GPS, a w pozostałych przypadkach informacja o lokalizacji będzie wnioskowana na podstawie adresu IP komputera z którego przyszło zapytanie.

Serwis *GeoWeather* był wykonywany zgodnie z przedstawionymi profilami aktywności użytkowników. Podczas badania, prawdopodobieństwa na krawędziach statystycznego modelu wykonania serwisu złożonego były rejestrowane. Analizując dane zamieszczone na Rysunku 16 możemy stwierdzić, że w celu poprawy jakości dostarczania usług dostawca powinien skoncentrować się na poprawie wydajności usługi odpowiedzialej za zamianę adresu IP na pozycję geograficzną, jako że jest ona wykorzystwana najczęściej. Dodatkowo, analizując dane przedstawione na Rysunku 17 można stwierdzić, że zasoby dostawcy powinny być skierowane na poprawę jakości serwisów w zależności od pory dnia. Pomiędzy godziną 8 a 13 dostawca powinien przenieść część zasobów na poprawę jakości serwisu `GlobalWeatherService` kosztem serwisu `USAWeatherService` tak aby sprostać zapotrzebowaniu użytkowników.

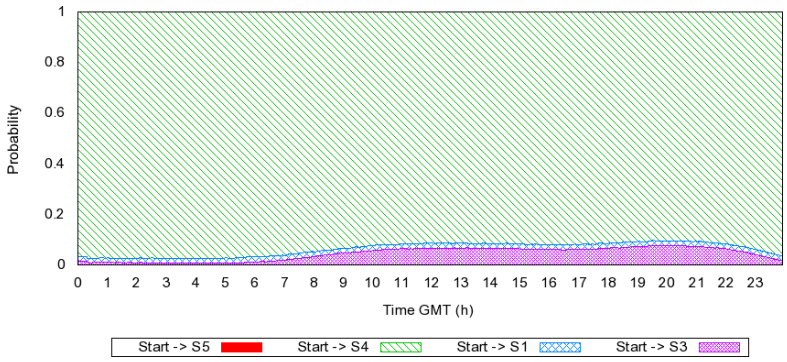
6.2.2 Zaburzenia czasu wykonania serwisów

Celem tego testu jest zbadanie jak system reaguje na zmiany zaobserwowane w środowisku wykonawczym. W celu analizy i przedstawienia reakcji systemu, autor analizuje pewien fragment całego serwisu. W tym przypadku przeanalizowane zostaną tylko wywołania serwisu zawierające położenie geograficzne zapisane w postaci pary składającej się z długości i szerokości geograficznej.

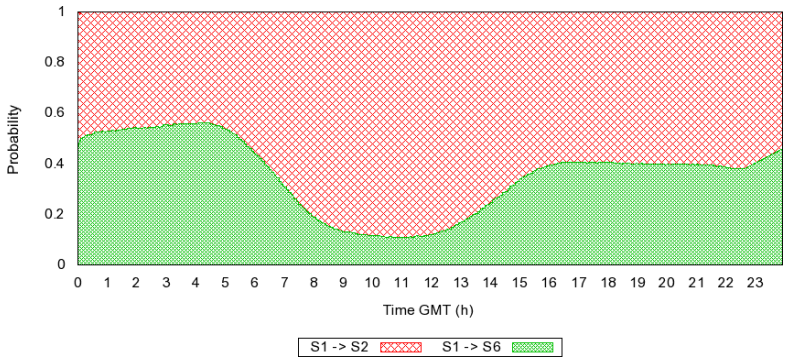
W trakcie wykonania serwisu złożonego bez uaktywnionej polityki adpatacji żądania



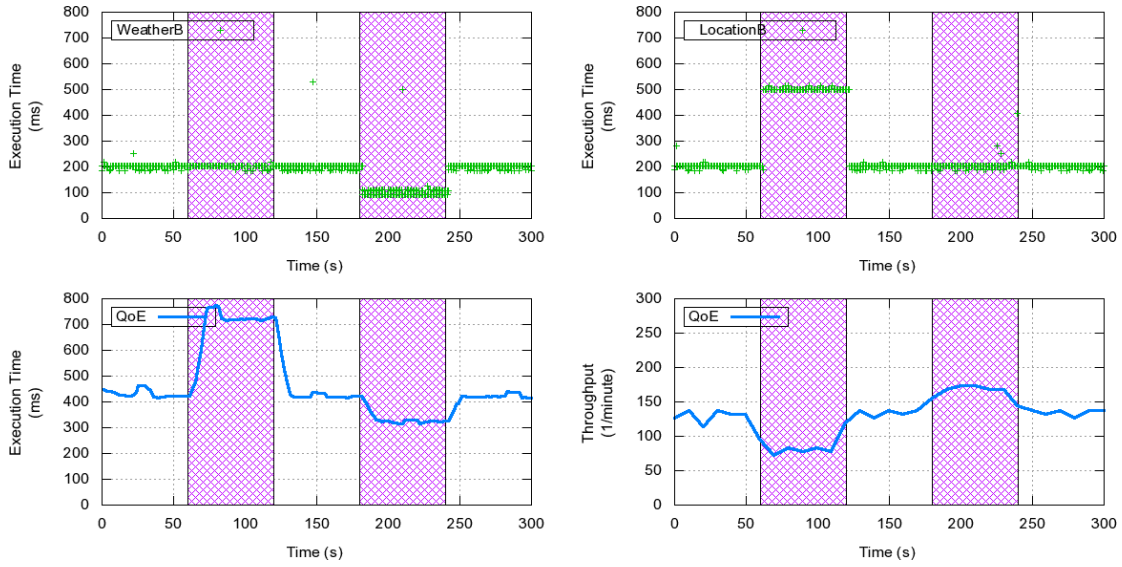
Rysunek 15: Rozkład ilości odwiedzin strony internetowej



Rysunek 16: Prawdopodobieństwa krawędzi modelu



Rysunek 17: Prawdopodobieństwa krawędzi modelu



Rysunek 18: Wykonanie aplikacji z zaznaczonymi przedziałami obserwacji zaburzeń wykonania

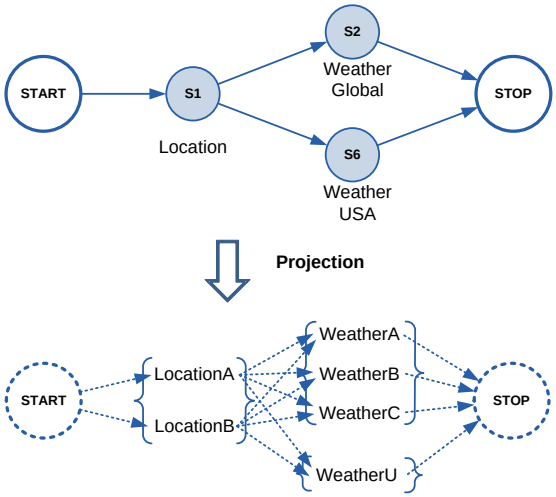
Tablica 6: Podobieństwo instancji serwisów

Service Similarity	*LocationA*	*LocationB*	*WeatherA*	*WeatherB*	*WeatherC*	*WeatherU*
LocationA	100%	100%	42%	42%	42%	42%
LocationB	100%	100%	42%	42%	42%	42%
WeatherA	42%	42%	100%	100%	100%	80%
WeatherB	42%	42%	100%	100%	100%	80%
WeatherC	42%	42%	100%	100%	100%	80%
WeatherU	42%	42%	80%	80%	80%	100%

klientów są przetwarzane przez serwisy `LocationB` i `WeatherB`, a średni czas wykonania serwisu złożonego waha się w granicach 400-450ms. Niestety, w pewnych przedziałach czasu zaobserwowano zaburzenia w środowisku wykonawnczym skutkujące wahaniami czasu wykonania. Zaobserwowane zmiany są przedstawione na Rysunku 18. Podczas pierwszego okresu, w którym zaobserwowano zaburzenia, klient nie otrzymał odpowiedzi w założonym czasie, a podczas drugiego okresu serwis zwrócił odpowiedź szybciej aniżeli wynikałoby to z kontraktu SLA co może skutkować zbędnym wykorzystaniem zasobów dostawcy usługi.

W szynie ESB dostępne są również inne instancje serwisów. Wynik procesu porównania semantycznego podobieństwa pomiędzy serwisami przedstawia Tabela 6. Wynika z zniej, że są dwie grupy serwisów które mają podobieństwo 100% co oznacza, że mogą być wykorzystwane zamiennie.

Aby móc reagować na zmiany w środowisku uruchomieniowym, model wykona-



Rysunek 19: Rzutowanie modelu wykonania serwisu złożonego na konkretne instancje serwisów

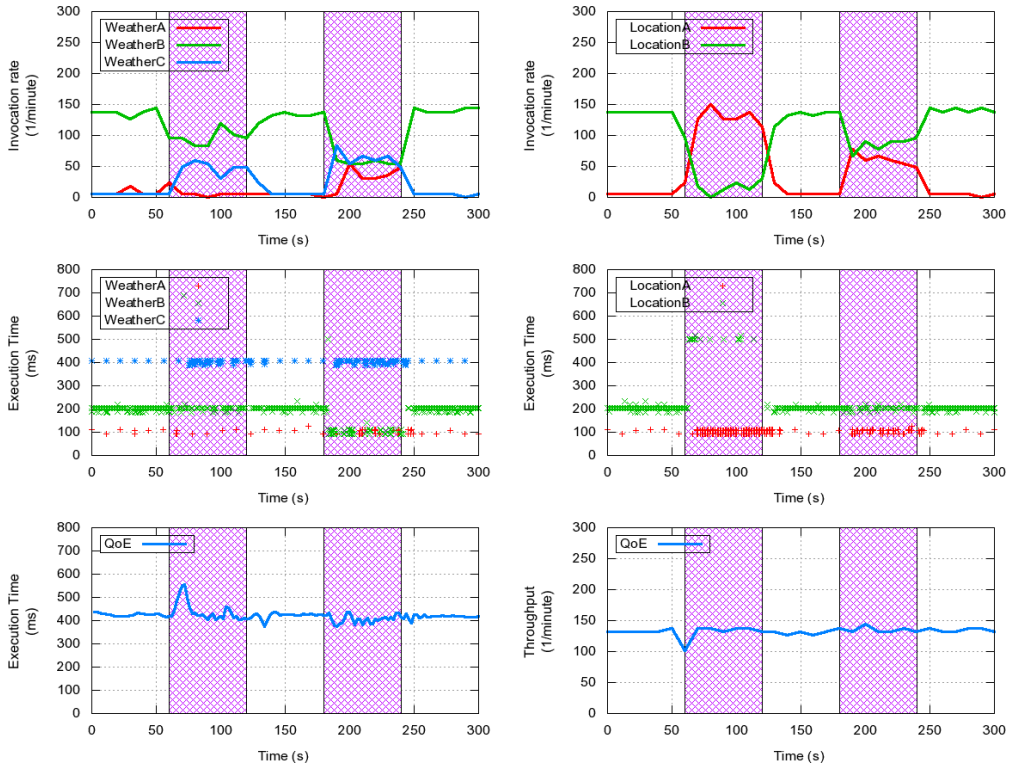
nia serwisu złożonego oraz polityka adaptacji powinna byc zdefiniowana. Model analizowanego serwisu złożonego oraz jego rzutowanie na istniejące instancje serwisów dostepnych w szynie ESB jest przedstawiony na Rysunku 19. Jest sześć możliwych kombinacji serwisów, które moga byc wykorzystanie w trakcie wykonania serwisu złożonego. Model oraz oczekiwane wartości metryk dla każdej z możliwych kombinacji są uaktualniane w trakcie działania systemu i wykorzystywane przez politykę adaptacji.

Polityka adaptacji dla przedstawionego eksperymentu składa się z siedmiu reguł adaptacji przsedstawionych w Tabeli 7. Reguły `Boostrap` są wyzwalane przez fakt QoE i są zdefiniowane w taki sposób, że w losowych odstępach czasowych każdy z możliwych serwisów złożonych C jest losowo wybierany do wykonania na krótki okres czasu. Jest to potrzebne w sytuacji gdy system zaobserwuje degradację w jakości dostarczanej usługi przez konkretna instancję serwisu i zadecyduje o jej nie uzywaniu w trakcie kolejnych wywołań serwisu. Bez sprawdzenia, czy poprzednio używana instancja ponownie dostarcza usługę z typową dla siebie jakościa, system nie użyłby jej w dalszym procesie adaptacji.

Rysunek 20 przedstawia wynik działania polityki adaptacji. W tym scenariuszu, system reaguje na zmiany w środowisku wykonawczym i podmienia w trakcie działania systemu instancje serwisów biorące udział w przetwarzaniu zapytań klientów. W przedziale czasowym 60s–120s system używał instancję `LocationA` zamiast instancji `LocationB` ponieważ w tym czasie zaobserwowano wzrost czasu wykonania tej instancji o około 200ms. Analogicznie w przedziale czasowym 180s–240s serwis `WeatherB` odpowiadał w czasie krótszym niż poprzednio, system zdecydował o wykrozystywaniu naprzemiennie wszystkich z trzech dostępnych instancji serwisów pogodowych, tak aby zapewnić czas odpowiedzi serwisu złożonego zgodnie z kontraktem.

Tablica 7: Złożona polityka adaptacji

Rule	Description
`Bootstrap - solutions`	Rule triggered by `QoE` fact. This rule asserts all of the `MPG` solutions as facts into working memory.
`Bootstrap - QoSparm`	Rule triggered by solution fact. This rule asserts `QoSparm` facts for all of the `QoS` parameters for particular solution.
`QoS control - execution time`	Rule triggered by `QoE` fact and `QoSParm` facts. This rule retracts solutions from working memory that does not fulfil desired execution time.
`Cleaning QoE facts`	Rule triggered by `QoE` fact. This rule retracts all of the `QoE` facts from working memory.
`Reducing solution space - fastest selection`	Rule triggered by two different solutions. Solution with the `QoS` value too far from desired value is retracted. This rule repeats until only one solution is left.
`Invocation`	Rule triggered by solution fact. This rule invokes chosen solution.
`Cleaning QoSparm`	Rule triggered by `QoS` fact. This rule retracts all of the `QoSparm` facts from working memory.



Rysunek 20: Wykonanie aplikaci z polityką adaptacji

6.2.3 Podsumowanie

Środowisko Adaptive VESB zostało pomyślnie wykorzystane w procesie adaptacji z wykrozystaniem statystycznego modelu wykonania serwisu złożonego. Przedstawione strategie adaptacji mogą być wykorzystane niezależnie w różnych wirtualnych szynach ESB w zależności od zapotrzebowania.

7 Podsumowanie

Autor pracy postawił tezę, że jest możliwe zbudowanie systemu, który wzbogaci środowisko uruchomieniowe dla aplikacji SOA o elementy adaptowalności w taki sposób, że uruchomiona aplikacja nie będzie świadoma działającej polityki adaptacji. W rozprawie pokazano, że statystyczny model wykonania serwisu złożonego może być z powodzeniem zastosowany w procesie adaptacji. Jest on wysokopoziomowym odzwierciedleniem aplikacji SOA i jest uaktualniany w czasie rzeczywistym. Opisywany model posłużył do sformułowania koncepcji kompozycji i wykonania aplikacji SOA i został przedstawiony w Rozdziale 3. Główna idea przedstawionej koncepcji polega na selekcji odpowiedniego zbioru instancji serwisów w trakcie wykonania co wpływa na całość kompozycji

Omówiona koncepcja posłużyła do zaprojektowania mechanizmów, które mogą być w sposób przezroczysty dodane do warstwy integracji SOA wzbogacając ja o elementy adaptowalności. Autor pokazał, że takie mechanizmy mogą być zaimplementowane dla nowoczesnych rozwiązań SOA bez ich modyfikacji. Architektura takiego rozwiązania została przedstawiona w Rozdziale 4, a szczegóły implementacyjne w Rozdziale 5. Prototypowa implementacja została zweryfikowana za pomocą szeregu testów. Bazując na ich wynikach, autor rozprawy jest przekonany że postawiona teza jest prawdziwa, tzn. że da się zbudować system o przedstawionych własnościach.

7.1 Nowatorstwo pracy

Jednym z głównych osiągnięć rozprawy jest wprowadzenie nowatorskich koncepcji kompozycji i wykonania aplikacji SOA. Zaproponowane rozwiązanie wykorzystuje statystyczne modele wykonania serwisów złożonych w procesie adaptacji. Autor proponuje również mechanizm *Virtual ESB*, który jest logiczną szyną ESB w obrębie fizycznej instalacji szyny ESB. Pozwala to administratorowi systemu na precyzyjne zaaplikowanie polityki adaptacji do odpowiedniego logicznego fragmentu systemu. Pozwala również na grupowanie użytkowników w klasy bazując na ich SLA czy tez oczekiwanej jakości usług.

Przedstawione koncepcje zostały wykorzystane do zaproponowania dwóch transformacji, które mogą być zaaplikowane do szyn ESB zgodnych ze standardem JBI. Pierwsza transformacja dodaje elementy adaptowalności, a druga rozszerza ESB o koncepcje wirtualizacji. Obydwie transformacje składają się na środowisko wykonawcze Adaptive ESB.

Autor proponuje algorytm *MapGraphReduce*, który służy do efektywnego wyliczania wartości metryk dla statystycznego modelu wykonania serwisów złozonych. Algorytm służy do przetwarzania danych opisanych za pomocą acyklicznych grafów skierowanych. Obliczenia definiowane są w postaci funkcji MAP oraz funkcji $GRAPH\_REDUCE$, które są następnie efektywnie przeliczane przez zaimplementowaną bibliotekę. Algorytm może być wykorzystany do innych zastosowań, przykładowo do przeliczania wydajności dróg i autostrad mając zdefiniowany graf połączeń pomiędzy miastami.

7.2 Dalsze prace

Dalsze prace w kierunku rozwoju przedstawionego systemu obejmują kilka jego aspektów. W obecnej wersji, dostawca serwisu musi stworzyć i dostarczyć model wykonania serwisu złożonego. System powinien być rozbudowany o mechanizmy automatycznego generowania modelu wykonania serwisu złożonego. Taki mechanizm mógłby działać w oparciu o obserwację wiadomości przesyłanych przez szynę ESB i odpowiednio je agregując stworzyć statystyczny model wykonania serwisu.

W obecnej wersji system nie wykorzystuje żadnych informacji o rekomendacjach użytkowników dotyczących konkretnych instancji serwisów. Serwis może spełniać różnego rodzaju wymagania niefunkcjonalne, ale może przykładowo dostarczać mało dokładne dane. Rekomendacje użytkowników mogłyby w znacznym stopniu ograniczyć stosowanie mało przydatmych serwisów w procesie kompozycji i wykonania.

Ostatnim obszarem dalszego rozwoju powinno być rozszerzenie implementacji algorytmu *MapGraphReduce*. Obecna implementacja wykazała swoją przydatność do przetwarzania grafów skierowanych acyklicznych i powinna zostać rozbudowana o obsługę procesorów wielordzeniowych.

Literatura

[1] *OSGi Service Platform, Core Specification, Release 4, Version 4.1.* aQute.

[2] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. Business Process Execution Language for Web Services Version 1.1. Technical report, BEA, IBM, Microsoft, SAP, Siebel, 2003.

[3] Gordon Blair, Nelly Bencomo, and Robert B. France. Models@ run.time. *Computer*, 42:22–27, 2009.

[4] Mark Burstein, Jerry Hobbs, Ora Lassila, Drew Mcdermott, Sheila Mcilraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara. OWL-S: Semantic Markup for Web Services. Website, November 2004.

[5] G. Chafle, K. Dasgupta, A. Kumar, S. Mittal, and B. Srivastava. Adaptation in Web Service Composition and Execution. In *Web Services, 2006. ICWS '06. International Conference on*, pages 549–557, Sept. 2006.

[6] Soo Ho Chang, Hyun Jung La, Jeong Seop Bae, Won Young Jeon, and Soo Dong Kim. Design of a dynamic composition handler for esb-based services. In *ICEBE '07: Proceedings of the IEEE International Conference on e-Business Engineering*, pages 287–294, Washington, DC, USA, 2007. IEEE Computer Society.

[7] I.-Y. Chen, G.-K. Ni, and C.-Y. Lin. A runtime-adaptable service bus design for telecom operations support systems. *IBM Syst. J.*, 47(3):445–456, 2008.

[8] K-WfGrid Project Consortium. http://www.kwfgrid.eu/, visited: 2010.

[9] Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns : Designing, Building, and Deploying Messaging Solutions.* Addison-Wesley Professional, October 2003.

[10] IEEE, `http://www.ieee802.org/1/pages/802.1Q.html`. *IEEE 802.1q: VLAN*, 2005.

[11] David Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems.* Addison-Wesley Professional, May 2002.

[12] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia Sycara. Semantic Matching of Web Services Capabilities, 2002.

[13] Mark Proctor. Relational Declarative Programming with JBoss Drools. *Symbolic and Numeric Algorithms for Scientific Computing, International Symposium on*, 0:5, 2007.

[14] ISP MATinternet s.c. usage statistics. http://www.linux.zakopane.biz/, visited: 2010.

[15] Oracle's Sun Server and Storage Systems. http://www.oracle.com/, visited: 2010.

[16] Event Stream and Complex Event Processing for Java. http://www.espertech.com/, visited: 2009.

[17] Tomasz Szydlo and Krzysztof Zielinski. Method of Adaptive Quality Control in Service Oriented Architectures. In *ICCS '08: Proceedings of the 8th international conference on Computational Science, Part I*, pages 307–316, Berlin, Heidelberg, 2008. Springer-Verlag.

[18] Maja Vukovic. Context Aware Service Composition. Technical report, University of Cambridge, 2006.