AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Wydział Elektrotechniki, Automatyki,
Informatyki i Elektroniki

On demand Semantics Based Service Component Integration and Provisioning for SOA Applications

---

Integracja i dostarczanie na żądanie komponentów usług dla aplikacji zorientowanych na usługi w oparciu o informacje semantyczne

Autoreferat rozprawy doktorskiej

Robert Szymacha

Promotor:
prof. dr hab. inż. Krzysztof Zieliński, Akademia Górniczo-Hutnicza

Recenzenci:
dr hab. Stanisław Ambroszkiewicz, Instytut Podstaw Informatyki PAN
dr hab. inż. Grzegorz Dobrowolski, prof. AGH, Akademia Górniczo-Hutnicza

Kraków, 2010

1. Wprowadzenie

Aktualne prace nad integracją systemów koncentrują się wokół dwóch głównych nurtów: programowania komponentowego (ang. *Component-Based Software Engineering* (CBSE)) oraz architektur programowania zorientowanego na usługi (ang. *Service Oriented Architecture* (SOA)). Programowanie komponentowe jest wykorzystywane głównie w systemach małej i średniej wielkości, podczas gdy SOA jest wykorzystywane do integracji dużych systemów, często tworzonych przez różnych producentów oprogramowania.

Systemy zbudowane w oparciu o paradygmat SOA mogą być integrowane na różnych poziomach abstrakcji. Model *SOA Solution Stack* (S3), zaprezentowany w [1], specyfikuje następujące poziomy: *usługi atomowe* (ang. *atomic services*) są zbudowane z wykorzystaniem integracji komponentów, *usługi złożone* (ang. *composite services*) są zbudowane z usług atomowych, podczas gdy *procesy biznesowe* (ang. *business processes*) są zbudowane z wykorzystaniem usług atomowych i złożonych.

Każdy z tych poziomów integracji wymaga (przy obecnym stanie technologii) dużego nakładu pracy podczas tworzenia systemów informatycznych. Jednym z założeń rozprawy jest przedstawienie metod, które pozwalają na częściową automatyzację niektórych z tych procesów, a tym samym pozwalają na przyspieszenie tworzenia oprogramowania.

1.1. Motywacja i sformułowanie tezy rozprawy

W ostatnich latach paradygmat SOA stał się wiodącym wśród metodologii tworzenia aplikacji dla przemysłu. Większość istniejących rozwiązań bazuje na rozwiązaniach takich jak Enterprise Service Bus (ESB) czy też Service Component Architecture (SCA), pozwalających na integrację istniejących rozwiązań w jedną aplikację, a także na udostępnianie istniejących aplikacji w postaci usług. To podejście było wystarczające przez początkowe lata rozwijania SOA, jednakże ostatnio coraz większy nacisk kładzie się nie tylko na funkcjonalność usługi (czyli określenie „co?" usługa robi), ale także na wymagania pozafunkcjonalne (jakość usługi, ang. *Quality of Service* (QoS)).

Zapewnienie odpowiedniej jakości usługi jest obecnie kluczowe dla wielu aplikacji związanych z przemysłem, jednakże istniejące rozwiązania często nie są do tego przystosowane. Dostosowanie do określonej jakości w zmiennym środowisku (często komponenty i usługi są uruchomione na współdzielonych zasobach, które mogą ulec przeciążeniu, z usług może korzystać zmienna ilość użytkowników co może spowodować różne czasy odpowiedzi, np. w zależności od pory dnia, dnia tygodnia, etc.), a także do zmiennych wymagań klienta co do jakości (np. klient może stwierdzić, że określona usługa jest dla niego kluczowa i powinna odpowiadać w dużo krótszym czasie), wymaga nowego podejścia do tworzenia usług. W takim podejściu, usługa powinna się *adaptować* do zmieniających się warunków, aby jej jakość zadowalała klienta.

Adaptowalność usługi można zapewnić poprzez realizację systemu, którego działanie nawiązuje do, znanej z teorii sterowania, zamkniętej pętli sterowania. Podejście to opiera się na dokonywaniu pomiaru określonych wartości otrzymywanych na wyjściu (monitorowanie usługi) i porównywaniu ich z wartościami zadanymi. Na podstawie wyników

tego porównania należy podjąć określone decyzje, mówiące jakie akcje należy wykonać, aby wartości jakości usługi były jak najbardziej zbliżone do oczekiwanych. Jest to jedno z głównych zagadnień pracy.

Kolejnym istotnym problemem poruszanym w rozprawie jest tworzenie usług na żądanie użytkownika. Ze względu na fakt, iż obecnie dużą część prac programistycznych poświęca się na integrację istniejących rozwiązań, autor przedstawia pomysł, aby część z tych prac zautomatyzować poprzez zastosowanie automatycznej kompozycji. Oczywiście nie wszystkie obszary da się zautomatyzować, jednakże wśród części komponentów i usług, których funkcjonalność można opisać w sposób formalny, jest miejsce na zastosowanie takiego podejścia.

Na podstawie powyższych przemyśleń, teza pracy została sformułowana następująco:

> *Rozbudowanie istniejących technologii komponentowych o mechanizmy adaptacji w trakcie wykonania pozwala twórcom oprogramowania na budowę usług zgodnych z paradygmatem SOA, które potrafią dostosowywać się do pozafunkcjonalnych wymagań użytkownika usługi.*

1.2. Zakres badań i wyzwania naukowe

Przedstawiona praca skupia się na tworzeniu usług zgodnych z paradygmatem SOA, zbudowanych z komponentów. Komponenty są rozszerzone o opis semantyczny, pozwalający na specyfikację ich funkcjonalności.

Autor proponuje model usługi oparty o reprezentację grafową, która ułatwia jego wykorzystanie w dalszym procesie adaptacji. Reprezentacja grafowa składa się z dwóch części: abstrakcyjnej, opartej o opis semantyczny, oraz konkretnej, która wynika z istniejących implementacji komponentów i może być w dalszym etapie uruchomiona.

W rozprawie autor bada możliwości adaptacji usługi w oparciu o systemy regułowe. Reguły są opisywane za pomocą wysokopoziomowych polityk, które zapewniają warstwę abstrakcji pomiędzy użytkownikiem (klientem) a niskopoziomowym systemem adaptacji.

Jako środowisko wykonawcze autor analizuje szereg systemów komponentowych i wybiera jeden z nich do dalszych badań. Dzięki rozszerzeniu tego systemu poprzez implementację mechanizmów adaptacji, system pozwalaja na tworzenie usług, które dopasowują się do wymagań pozafunkcjonalnych użytkownika.

1.3. Główne osiągnięcia pracy

Do głównych osiągnięć pracy można zaliczyć:

- krytyczną analizę istniejących technologii komponentowych z punktu widzenia wymagań stawianych środowiskom wykorzystywanym w nowoczesnych aplikacjach SOA;
- wprowadzenie grafowego modelu usługi zbudowanej z komponentów, wykorzystywanego dla zapewnienia adaptowalności usługi do zmieniających się wymagań

pozafunkcjonalnych oraz możliwości dostarczyciela usługi; adaptowalność jest osiągnięta poprzez wybór określonych podgrafów zapewniających tą samą funkcjonalność, lecz inną jakość usługi;

- zaproponowanie mechanizmów adaptacji usługi w czasie wykonania, opartych na politykach; rozwiązanie to pozwala na oddzielenie wysokopoziomowych polityk adaptacji od niskopoziomowych decyzji, poprzez wprowadzenie dodatkowej warstwy abstrakcyjnych polityk i określenie ich odwzorowania na niskopoziomowe reguły;
- zaprojektowanie warstwowej architektury Systemu udostępniania usług na żądanie (ang. *On demand Service Provisioning System* (ODSPS)), oraz jego prototypowa implementacja;
- rozszerzenie modelu S3 poprzez dodanie mechanizmów adaptacji do warstwy *Service components*, co pozwala na tworzenie adaptowalnych usług;
- rozszerzenie specyfikacji Service Component Architecture (SCA) o mechanizmy adaptacji w czasie wykonania oraz implementacja tych rozszerzeń dla Apache Tuscany SCA; pozwala to na wykorzystanie istniejących usług zbudowanych z komponentów przygotowanych w różnych technologiach (co wynika z możliwości technologii SCA).

2. Prace związane z obszarem badań

Architektura zorientowana na usługi (ang. *Service Oriented Architecture* (SOA)) jest naturalnym następcą programowania obiektowego (ang. *Object Oriented Programming* (OOP)) oraz programowania komponentowego (ang. *Component-Based Software Engineering* (CBSE)). Konieczność powstania nowego paradygmatu powstała w wyniku tworzenia coraz większych systemów informatycznych, a także integracji istniejących systemów, bardzo często stworzonych przy użyciu różnych technologii implementacyjnych.

2.1. Założenia architektury zorientowanej na usługi

Paradygmat SOA wychodzi na przeciw nowym wymaganiom dotyczących tworzenia oprogramowania, poprzez zdefiniowanie *usługi* (ang. *service*), będącej podstawową jednostką tworzenia oprogramowania. Usługa musi spełniać osiem założeń [2]:

ponowne wykorzystanie (ang. *reusability*), tzn. usługa powinna reprezentować ogólną funkcjonalność, która może być wykorzystywana w różnych aplikacjach (nie powinna być „skrojona na miarę");

formalny kontrakt (ang. *formal contract*) jest jedynym elementem, który jest współdzielony pomiędzy usługami. Kontrakt ten reprezentuje usługę i zawiera informacje dotyczące sposobów interakcji z usługą;

luźne wiązanie (ang. *loose coupling*) pomiędzy usługami, wykorzystujące formalny kontrakt oznacza, że nie istnieje silna zależność pomiędzy usługami, dzięki czemu

w łatwy sposób istnieje możliwość podmiany jednej usługi na inną, dostarczającą tej samej funkcjonalności;

abstrakcja (ang. *abstraction*) od logiki usługi oznacza, że pomiędzy kontraktem, reprezentującym usługę na zewnątrz, a logiką usługi, istnieje dodatkowa warstwa, pozwalająca na odwzorowywanie wiadomości otrzymywanych przez kontrakt, na wiadomości, które są specyficzne dla określonej implementacji logiki usługi. Pozwala to na zwiększenie ponownego wykorzystania usługi, gdyż jedna usługa może być reprezentowana na zewnątrz poprzez inne kontrakty. Patrząc z drugiej strony, poprzez stworzenie specjalizowanego kontraktu, można wykorzystywać różne usługi posiadające zbliżoną funkcjonalność. Aby zapewnić abstrakcję od logiki usługi, konieczne jest dodanie warstwy Logiki Przetwarzania Wiadomości (ang. *Message Processing Logic* (MPL));

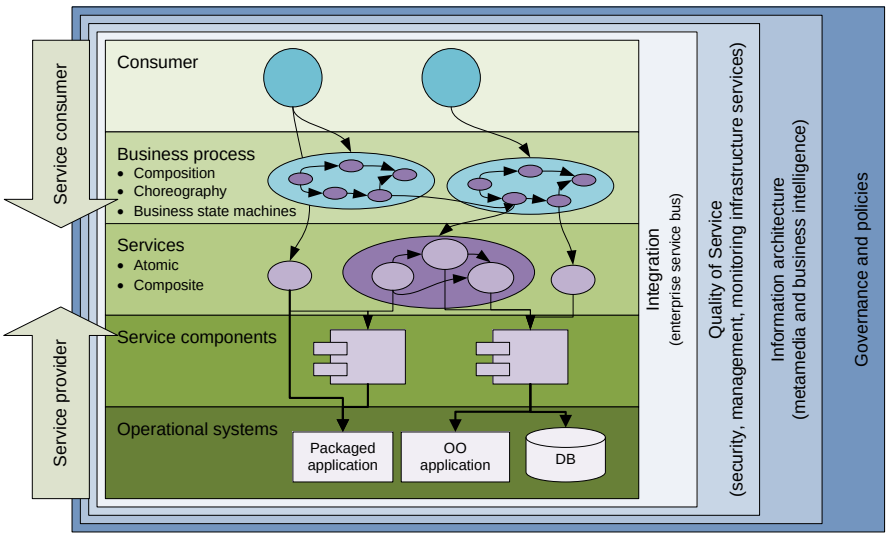
komponowalność (ang. *composability*) usług w złożone procesy. Aplikacje w oparciu o paradygmat SOA są tworzone poprzez komponowanie usług w większą funkcjonalność, zwaną procesem biznesowym (ang. *business process*). Usługi w SOA muszą pozwalać na ich komponowanie z wykorzystaniem istniejących języków opisu procesów biznesowych;

autonomia (ang. *autonomy*) usługi oznacza, iż każda z usług powinna być bytem niezależnym od pozostałych, dzięki czemu jest możliwe niezależne składanie usług i ich ponowne wykorzystanie;

bezstanowość (ang. *statelessness*) usługi minimalizuje ilość zasobów koniecznych do zarządzania usługą. W przypadku usług dla SOA, stan usługi powinien być dostarczany wraz z jej wywołaniem i przechowywany poza nią (przykładowo wewnątrz procesu biznesowego). Dzięki takiemu podejściu, znacznie zwiększa się skalowalność systemów SOA;

odkrywalność (ang. *discoverability*) usług pozwala na wyszukiwanie usług spełniających określone wymagania funkcjonalne i tym samym zwiększa możliwości ich ponownego wykorzystania. W celu zapewnienia odkrywalności usług konieczne jest stosowanie specjalizowanych języków opisu usługi (odnoszących się do semantyki usługi) i często repozytoriów usług (np. UDDI [3]).

Istnieje kilka liczących się architektur referencyjnych dla SOA, jak np. OASIS SOA-RA [4] lub SOA-RM [5]. Jedną z najbardziej kompletnych i ogólnych architektur, jest model opracowany przez IBM: SOA Solution Stack (S3) [1], zaprezentowany na Rys. 1.



Rysunek 1: SOA Solution Stack

Model S3 składa się z dziewięciu warstw, z których pięć (przedstawionych po lewej stronie rysunku) odpowiada za różny poziom abstrakcji i ziarnistości aplikacji SOA, a kolejne cztery (prawa strona rysunku) są położone ortogonalnie do pozostałych pięciu i odpowiadają za pozafunkcjonalne właściwości systemu, które są wykorzystywane przez inne warstwy. Szczegółowy opis modelu S3 znajduje się w Sekcji 2.1.3 rozprawy.

Rozprawa skupia się na analizie i rozszerzeniu warstw odpowiadających za Komponenty usług (ang. *Service components*), Jakość Usługi (ang. *Quality of Service*) oraz Polityki (ang. *Policies*).

2.2. Adaptacyjna integracja komponentów na żądanie

Pojęcie dostarczania oprogramowania na żądanie można zdefiniować jako zdolność do zapewnienia szybkiej odpowiedzi na zapotrzebowanie klienta dotyczące określonego oprogramowania. Oznacza to, że instytucja dostarczająca tego typu oprogramowanie potrafi dokonać adaptacji tworzonego oprogramowania w przypadku zmieniających się wymagań użytkownika (zarówno funkcjonalnych jak i pozafunkcjonalnych).

Z punktu widzenia paradygmatu SOA, termin *on-demand computing* dotyczy dostarczania na żądanie usług, które są dostosowane do określonych potrzeb użytkownika. W celu zapewnienia powyższej funkcjonalności, konieczne jest kilka kroków pośrednich.

Po pierwsze, aby odpowiedź na żądanie klienta była wystarczająco szybka, usługi powinny być tworzone automatycznie. Zgodnie z przedstawionym modelem S3, bazą do tworzenia usług są komponenty (warstwa *Service components*), tak więc konieczna jest automatyczna integracja (kompozycja) komponentów w usługi. Aby taka integracja była możliwa, potrzebny jest dodatkowy, wysokopoziomowy opis, zarówno dla komponentów (reprezentacja ich funkcjonalności), jak i usługi (czyli jaką funkcjonalność klient chce otrzymać). Tego typu opis musi opierać się nie tylko na składni (syntaktyce), ale także na zachowaniu (semantyce) komponentów i usług. W bieżących badaniach, do opisu semantycznego (głównie usług), powszechnie wykorzystywane są ontologie, wywodzące się z Logiki Opisowej (ang. *Description Logic*) [6].

W procesie automatycznej kompozycji należy brać pod uwagę zarówno wymagania funkcjonalne, jak i pozafunkcjonalne tak, aby zapewnić rozwiązanie akceptowalne przez klienta usługi. Wymagania pozafunkcjonalne odnoszą się głównie do parametrów jakości usługi (ang. *Quality of Service* (QoS)).

Jakość usługi jest terminem dobrze znanym z przemysłu IT i odnosi się do zapewnienia określonej jakości mierzonej przy użyciu określonych metryk, jak np. średni lub maksymalny czas odpowiedzi, zużycie pamięci, etc. Metryki jakości mogą być podzielone na dwie grupy: metryki proste, które są bezpośrednio mierzone w badanym systemie, i metryki złożone, które są wyliczane na podstawie metryk prostych (np. przy określonych warunkach).

Istnieją dwie perspektywy z punktu widzenia pomiaru jakości usług – dostarczyciela i klienta usługi. Pierwsza z nich, perspektywa dostarczyciela usługi, pokazuje zadeklarowane lub zmierzone wartości metryk, i jest określana jako Quality of Service (QoS). Druga z nich, perspektywa klienta usługi, pokazuje odczucia klienta dotyczące dostarczanej jakości i jest określana jako Quality of Experience (QoE). W niektórych publikacjach QoE odnosi się do wartości subiektywnych odczuwanych przez klienta (np. „strona WWW otwiera się zbyt wolno"), jednakże w rozprawie autor traktuje metryki QoE jako wartości obiektywne (tj. mierzalne). W idealnym rozwiązaniu, metryki QoE obserwowane przez klienta powinny być jak najbardziej zbliżone do wartości podanych przez dostarczyciela (QoS).

3. Koncepcja systemu i model usługi

Model usługi przedstawiony w pracy opiera się na grafowej reprezentacji budowy logiki usługi (ang. *service logic*). Jest on następnie wykorzystywany przy formalizacji transformacji, stosowanych dla zapewnienia adaptowalności usługi.

Transformacje są wykorzystywane przez dwie pętle adaptacji, które pracują na różnych poziomach abstrakcji. Pierwsza z nich operuje na semantycznym opisie komponentów oraz usługi i odnosi się do pojęcia Abstrakcyjnej Kompozycji (ang. *Abstract Composition*). Druga z nich działa na poziomie fizycznej reprezentacji usługi i odnosi się do terminów Kompozycji (ang. *Composition*) oraz Instancji Kompozycji (ang. *Composition Instance*). Terminy te są wyjaśnione szczegółowo w dalszej części autoreferatu.

Podczas analizy istniejących rozwiązań oraz wymagań dla systemów służących do udostępniania usług dla aplikacji zbudowanych w oparciu o paradygmat SOA, zostały przedstawione następujące wymagania dla tworzonego systemu oraz sposobu udostępniania usług:

- system musi dostarczać metod dla tworzenia usług na żądanie użytkownika (ang. *on customer's demand*), wykorzystując istniejące komponenty. Użytkownik specyfikuje wymagania funkcjonalne (ang. *functional requirements*) usługi, a także wymagania pozafunkcjonalne (ang. *non-functional requirements*);
- komponenty mogą być dostarczane przez różnych producentów oprogramowania,

przy użyciu różnych technologii (zarówno z punktu widzenia implementacji komponentu, jak i protokołów komunikacji). W związku z tym system (kontener komponentów) musi dostarczać mechanizmy pozwalające na komunikację pomiędzy tymi komponentami w sposób dla nich niewidoczny;

- kompozycja komponentów powinna być dokonywana automatycznie, na podstawie wymagań funkcjonalnych dostarczonych przez użytkownika, więc system musi zapewnić mechanizmy pozwalające na wykorzystanie semantycznego opisu komponentów i usług;
- system musi posiadać możliwość adaptacji stworzonej usługi do zmieniających się pozafunkcjonalnych wymagań użytkownika (jakość usługi uzgodniona w dokumencie *Service Level Agreement*) a także zmieniających się możliwości (ang. *capabilities*) dostarczyciela usługi, w trakcie wykonania.

Dalszy opis modelu usługi oraz koncepcja systemu wynika bezpośrednio z przedstawionych powyżej wymagań.

3.1. Grafowa reprezentacja usługi

Usługa w SOA jest zbudowana z [7]:

- logiki usługi (ang. *service logic*), odpowiadającej za funkcjonalność usługi;
- interfejsu usługi (ang. *service interface*), przedstawiającego programistyczny interfejs dostępu do logiki (zależny od technologii);
- kontraktu usługi (ang. *service contract*), prezentującego usługę użytkownikowi w sposób niezależny od technologii;
- logiki przetwarzania wiadomości (ang. *message processing logic*), służącej do translacji wiadomości otrzymywanych od użytkownika (poprzez kontrakt), na wiadomości obsługiwane przez interfejs usługi.

Takie podejście pozwala na zwiększenie poziomu abstrakcji i ponownego wykorzystania logiki usługi, poprzez udostępnianie jej przez abstrakcyjny, niezależny od technologii, kontrakt.

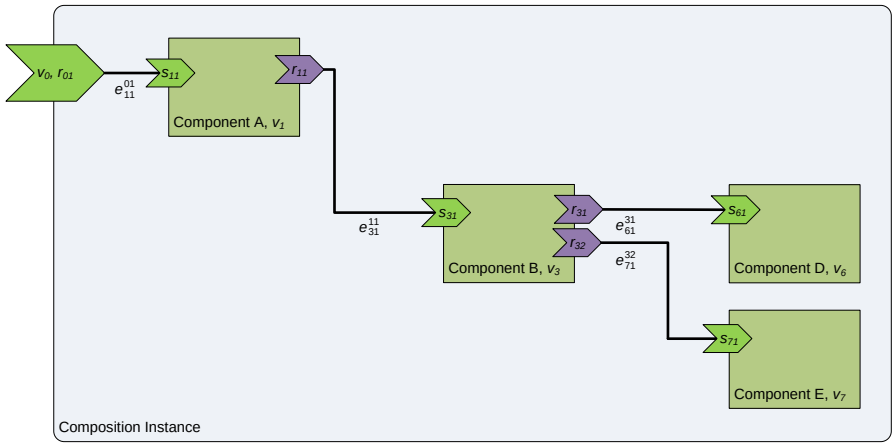
3.1.1. Instancja Kompozycji

Instancja Kompozycji (ang. *Composition Instance* (CI)) reprezentuje usługę złożoną z komponentów. Może być ona postrzegana jako graf skierowany, przy następujących założeniach:

- na węzły grafu składają się komponenty, które są wykorzystywane do implementacji danej logiki usługi. Każdy komponent składa się ze zbioru referencji (ang. *references*), reprezentujących funkcjonalność wymaganą przez komponent, oraz zbioru serwisów (ang. *services*), reprezentujących funkcjonalność udostępnianą przez komponent (dla innych komponentów lub dla użytkownika);

- węzły są połączone krawędziami, które odpowiadają fizycznym połączeniom między komponentami (ang. *wires*). Każde z połączeń jest skierowane od referencji jednego komponentu, do serwisu innego.
- istnieje jeden, wyróżniony wierzchołek grafu, reprezentujący funkcjonalność logiki całej usługi.

Dodatkowo została zdefiniowana funkcja b, przyporządkowująca każdej z krawędzi grafu, w zależności od aktualnego kontekstu, określoną właściwość pozafunkcjonalną. Przykładowo, może być ona wykorzystana do wyboru danego protokołu komunikacyjnego między komponentami. Przykładowa Instancja Kompozycji jest przedstawiona na Rys. 2.

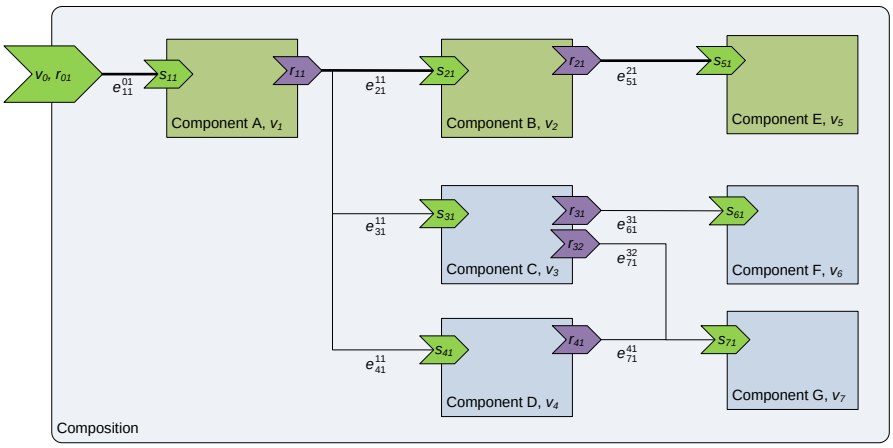


Rysunek 2: Przykładowy graf Instancji Kompozycji

3.1.2. Kompozycja

Pojęcie Kompozycji (ang. *Composition*) przedstawione w pracy, jest określone jako „zbiór Instancji Kompozycji, równoważnych pod względem spełnianych wymagań funkcjonalnych, lecz różniących się dostarczaną jakością usługi". Narzucająca się naturalnie reprezentacja Kompozycji jako zbioru grafów nie jest efektywna ze względu na możliwe duże wymagania pamięciowe (ilość Instancji Kompozycji rośnie wykładniczo w stosunku do ilości komponentów, które mogą być wykorzystane zamiennie). Stąd pomysł na reprezentację grafową Kompozycji, w której każda z Instancji Kompozycji będzie podgrafem grafu Kompozycji. Graf Kompozycji powstaje poprzez scalenie grafów wszystkich Instancji Kompozycji. Przykładowa Kompozycja jest przedstawiona na Rys. 3.

Dzięki takiej reprezentacji logiki usługi (w postaci Kompozycji) istnieje możliwość wyboru w czasie rzeczywistym funkcjonalności, która spełnia określone wymagania pozafunkcjonalne (tj. dostarcza usługę o zadanej jakości).



Rysunek 3: Przykładowy graf Kompozycji z wybraną Instancją Kompozycji

3.2. Integracja komponentów na żądanie

Przedstawiony w powyższych sekcjach grafowy model Kompozycji oraz Instancji Kompozycji pozwala na analizę logiki usługi po jej dostarczeniu w postaci opisu grafowego. Nie jest to jednak wystarczające dla tworzenia usług na żądanie (ang. *on demand*). W przypadku dostarczania usługi na żądanie, pojawiają się dodatkowe wymagania dla systemu, które opierają się na specyfikacji funkcjonalności wymaganej przez klienta.

Przedstawione w aktualnej literaturze prace koncentrują się na opisie semantycznym usług oraz komponentów [8–10]. Opis ten pozwala na uzyskanie informacji, nie tylko „w jaki sposób wykorzystać komponent/usługę?", ale także „co dany komponent/usługa robi?".

Jednym z najczęściej wykorzystywanych modeli opisu usługi jest opis uwzględniający wejścia usługi, jej wyjścia, warunki wstępne, jakie muszą spełniać wejścia usługi, i warunki końcowe opisujące stan wyjść i/lub zależności między wyjściami a wejściami usługi (ang. *Inputs, Outputs, Preconditions, Effects* (IOPE)). Model ten został w rozprawie zaadoptowany do opisu komponentów. Każdy z komponentów, który może być wykorzystywany w systemie musi posiadać dwa dodatkowe opisy – „wymagane IOPE", które mówi, jaką funkcjonalność należy komponentowi dostarczyć aby mógł funkcjonować, oraz „dostarczane IOPE", które mówi, jaką funkcjonalność komponent dostarcza. Te opisy są w dalszym etapie wykorzystywane podczas procesu semantycznego dopasowywania komponentów.

3.2.1. Abstrakcyjna Kompozycja

W wyniku opisanego powyżej procesu integracji komponentów na żądanie, otrzymujemy graf opisujący implementację logiki usługi, zwany w dalszej części pracy Abstrakcyjną Kompozycją (ang. *Abstract Composition*). Opis Abstrakcyjnej Kompozycji jest podobny do opisu Kompozycji, jednakże pojawia się tu jedna istotna różnica. Kompozycja to opis połączeń między konkretnymi instancjami komponentów, które mogą być bezpośrednio zainstalowane i uruchomione. Z drugiej strony, Abstrakcyjna Kompozycja opiera się na semantycznym, wysokopoziomowym opisie komponentów, co może powodować, iż

do stworzenia Kompozycji opartej na tej Abstrakcyjnej Kompozycji, będą wymagane dodatkowe działania. Przykładowo, mogą być konieczne dodatkowe komponenty, które będą usuwały niekompatybilności pomiędzy fizycznymi interfejsami komponentów. Te komponenty są zwane w dalszej części Mediatorami (ang. *Mediators*).

Ze względu na sposób tworzenia Abstrakcyjnej Kompozycji (na podstawie celu zawierającego wymagania funkcjonalne usługi), spełnia ona wymaganie stawiane usługom zgodnym z paradygmatem SOA, mówiące że usługi powinny być budowane w oparciu o kontrakt.

3.2.2. Wybór Instancji Kompozycji z Kompozycji

Abstrakcyjna Kompozycja jest podstawą do stworzenia Kompozycji, poprzez odwzorowanie węzłów abstrakcyjnego grafu na komponenty, które zostały dostarczone do systemu. Szczegóły tego procesu są opisane w dalszej części tego opracowania. Jednakże, aby usługa mogła być wykorzystana przez klienta, system musi dokonać wyboru Instancji Kompozycji z istniejącego zbioru, reprezentowanego przez Kompozycję. W tym procesie należy brać pod uwagę pozafunkcjonalne wymagania klienta oraz możliwości Instancji Kompozycji. Może to być osiągnięte poprzez wykorzystanie systemów regułowych (ang. *rule engines*), które na podstawie opisów Instancji Kompozycji oraz wymagań klienta, będą dokonywały odpowiedniego wyboru. Rozwiązanie to bazuje na informacjach, które są dostarczone przez twórców komponentów oraz klienta usługi, i są to:

- proste fakty, stworzone na podstawie opisów komponentów, zawierające informacje o dostarczanej jakości usługi (QoS);
- proste fakty, reprezentujące możliwości infrastruktury na której uruchomiona jest usługa, dostępnych zasobów, etc.;
- polityka klienta, dostarczona wraz z żądaniem usługi, zawierająca zbiór reguł jakie mają być wykorzystane podczas procesu wyboru Instancji Kompozycji;
- złożone fakty, które mogą być wywnioskowane z prostych faktów, stworzonej Kompozycji oraz dodatkowej wiedzy domenowej umieszczonej w systemie;
- oczekiwana jakość usługi, wyspecyfikowana przez klienta w żądaniu usługi.

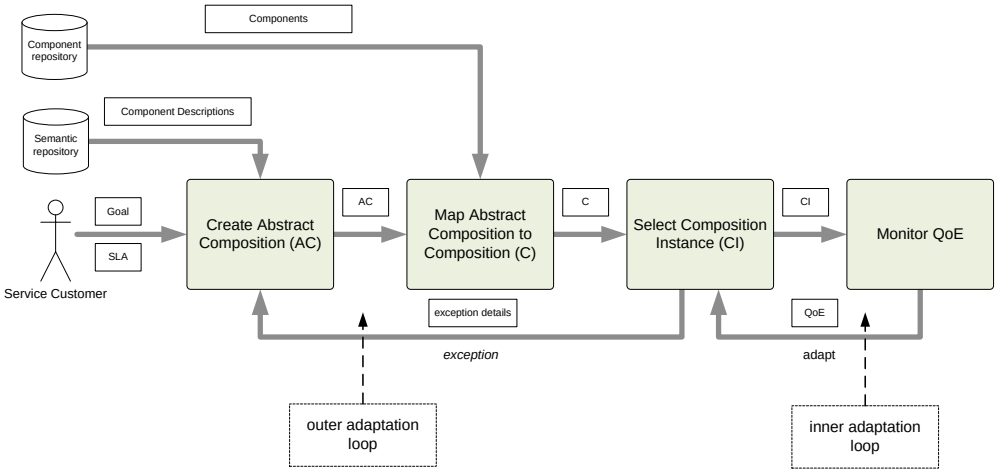
W celu odpowiedniej analizy powyższych danych, system musi być wyposażony w polityki wyboru Instancji Kompozycji (ang. *Composition Instance selection policies*).

3.3. Mechanizmy adaptowalności usługi

Jednym z większych wyzwań, jakie są stawiane współczesnym systemom udostępniania usług, jest spełnianie pozafunkcjonalnych wymagań użytkownika podczas działania usługi, mimo zmieniających się możliwości dostarczyciela usługi. Owe możliwości mogą się zmieniać przykładowo w wyniku chwilowego przeciążenia serwerów czy też zmniejszenia dostępności zasobów (np. pamięci) wynikającej z dostarczania usług dla

innych klientów. W typowych systemach udostępniania usług, takie zmiany pociągają za sobą deinstalację usługi, jej przebudowanie (np. przeniesienie jej na inne węzły) i ponowną instalację i uruchomienie. Pociąga to za sobą przerwę w dostępności usługi, co może nie być akceptowalne z punktu widzenia klienta.

Rozwiązanie przedstawione na Rys. 4 opiera się na ciągłym monitorowaniu jakości dostarczanej usługi i wyborze określonej Instancji Kompozycji, w przypadku gdy dostarczana usługa nie spełnia zadanych wymagań pozafunkcjonalnych.



Rysunek 4: Pętla adaptacji usługi

Rysunek przedstawia dwie pętle adaptacji (ang. *adaptation loops*) – wewnętrzną i zewnętrzną. Pierwsza z nich polega na zmianie wyboru Instancji Kompozycji, gdy dostarczana jakość usługi (QoE) nie jest zgodna z jakością wymaganą przez klienta (QoS). Proces ten zachodzi ciągle i nie powoduje przerwania pracy usługi. Z drugiej strony, istnieje możliwość, iż żadna z Instancji Kompozycji istniejących w Kompozycji nie spełnia określonych wymagań. W takim wypadku konieczne jest przebudowanie Kompozycji, co zachodzi w zewnętrznej pętli adaptacji. Wyjątek przesyłany do procesu tworzenia Abstrakcyjnej Kompozycji, zawiera informacje dotyczące parametrów jakości usługi, które nie zostały spełnione przez aktualną Kompozycję. W przypadku, gdy istnieje możliwość stworzenia nowej Abstrakcyjnej Kompozycji, jest ona następnie odwzorowywana na Kompozycję i system przechodzi do pracy w pętli wewnętrznej. Należy zaznaczyć, iż w wyniku ograniczeń istniejących systemów komponentowych, działanie pętli zewnętrznej może wiązać się z koniecznością odinstalowania usługi i zainstalowania jej ponownie, a co za tym idzie – chwilowej przerwy w jej działaniu. W związku z tym, jednym z założeń systemu jest, aby adaptacja była dokonywana przede wszystkim przy użyciu pętli wewnętrznej.

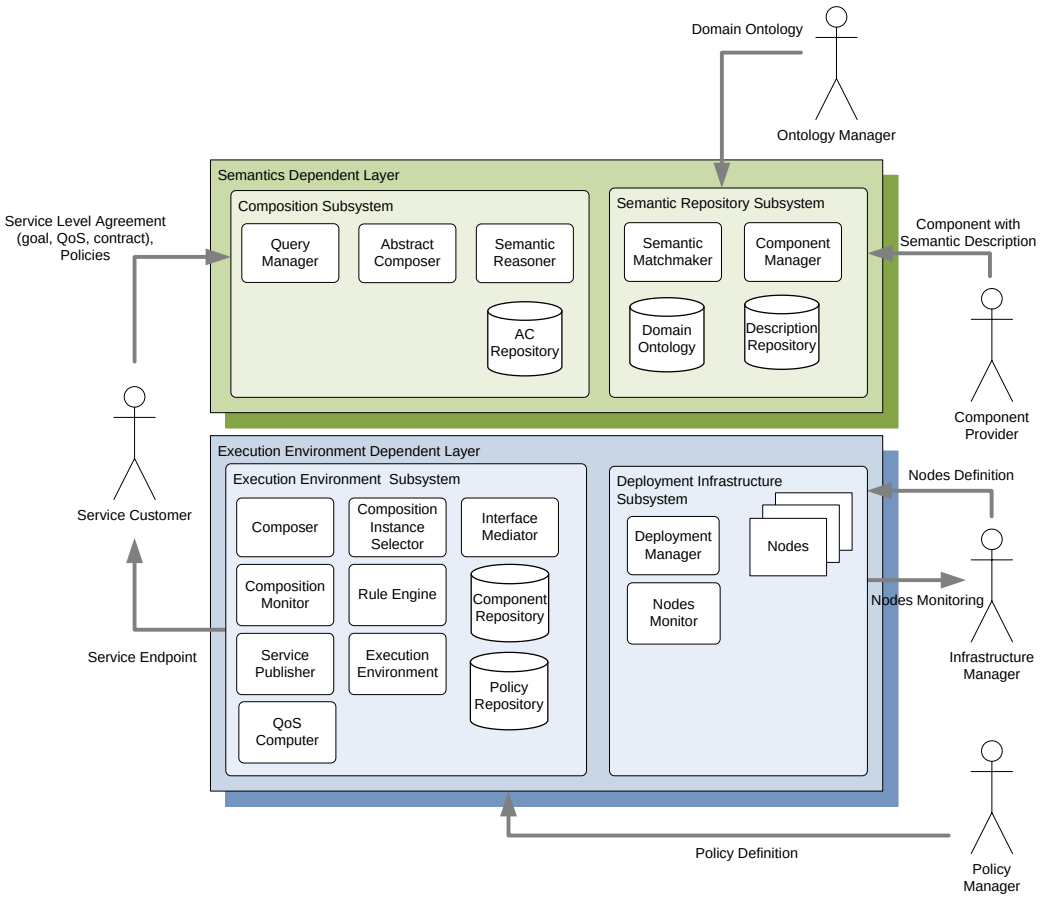
4. Projekt systemu

Projekt Systemu Udostępniania Usług Na Żądanie (ang. *On Demand Service Provisioning System* (ODSPS)) powstał na podstawie wymagań oraz modelu usługi przedstawionych w poprzedniej sekcji. Jest on bazą dla prototypowej implementacji przedstawionej

w dalszej części pracy. Poniższa sekcja przedstawia ogólną architekturę systemu wraz z podziałem na warstwy i komponenty.

4.1. Warstwowa architektura systemu ODSPS

Warstwowa architektura systemu ODSPS, przedstawiona na Rys. 5 wynika bezpośrednio z istnienia dwóch modeli usługi – modelu abstrakcyjnego oraz konkretnego. Górna warstwa systemu odpowiada za przetwarzanie informacji o komponentach i usłudze na poziomie semantycznym, stąd nazwa Warstwa Zależna od Semantyki (ang. *Semantics Dependent Layer* (SDL)). Dolna warstwa odpowiada na uruchomienie usługi na określonej infrastrukturze wykonawczej, z wykorzystaniem zastosowanego środowiska wykonawczego, stąd nazwa Warstwa Zależna od Środowiska Wykonawczego (ang. *Execution Environment Dependent Layer* (EEDL)).



Rysunek 5: Architektura systemu ODSPS

Komponenty przedstawione w architekturze systemu muszą dostarczyć funkcjonalności, której istnienie zostało wprowadzone w poprzednich sekcjach, tj.:

- integracji komponentów na żądanie;
- instalacji i uruchomienia usługi (ang. *deployment and execution*), biorąc pod uwagę konieczność adaptacji usługi do wymagań pozafunkcjonalnych;
- implementacji funkcji sterujących adaptacją: r, rc, ci, b (zaprezentowanych w rozprawie z Rozdziale 3);

- implementacji procesów wprowadzonych w modelu systemu, w szczególności dwóch pętli adaptacji.

4.1.1. Warstwa Zależna od Semantyki

Celem działania Warstwy Zależnej od Semantyki (SDL) jest dokonanie integracji komponentów na żądanie w celu otrzymania Abstrakcyjnej Kompozycji. SDL dostarcza tej funkcjonalności z wykorzystaniem semantycznego opisu komponentów oraz celu usługi (ang. *service goal*), który odpowiada za opis wymagań funkcjonalnych usługi. Warstwa ta odpowiada za interakcję z głównymi aktorami systemu, do których należą Klient (odbiorca usługi) oraz Dostawca Komponentów. Warstwa jest podzielona na dwa podsystemy – Podsystem Kompozycji (ang. *Composition Subsystem*) oraz Podsystem Semantycznego Repozytorium (ang. *Semantic Repository Subsystem*).

Podsystem Kompozycji odpowiada za przetworzenie żądania klienta i, na jego podstawie, stworzenie Abstrakcyjnej Kompozycji. Do tego celu wykorzystuje informacje uzyskane z Podsystemu Semantycznego Repozytorium, który zawiera opisy komponentów dostarczonych przez producentów komponentów. Głównym komponentem odpowiadającym za stworzenie Abstrakcyjnej Kompozycji jest Semantyczny Reasoner, wykorzystujący pośrednio komponent odpowiadający za dopasowanie semantycznych opisów komponentów (ang. *Semantic Matchmaker*). Szczegóły dotyczące algorytmu budowy kompozycji oraz wyszukiwania komponentów są zawarte w rozprawie.

4.1.2. Warstwa Zależna od Środowiska Wykonawczego

Celem działania Warstwy Zależnej od Środowiska Wykonawczego jest instalacja usługi, której opis jest dostarczony w postaci Abstrakcyjnej Kompozycji, z wykorzystaniem danego środowiska wykonawczego i dostarczenie tej usługi klientowi. Głównym założeniem jest fakt, że usługa musi być w stanie adaptować się do zmieniających się pozafunkcjonalnych wymagań użytkownika i możliwości infrastruktury.

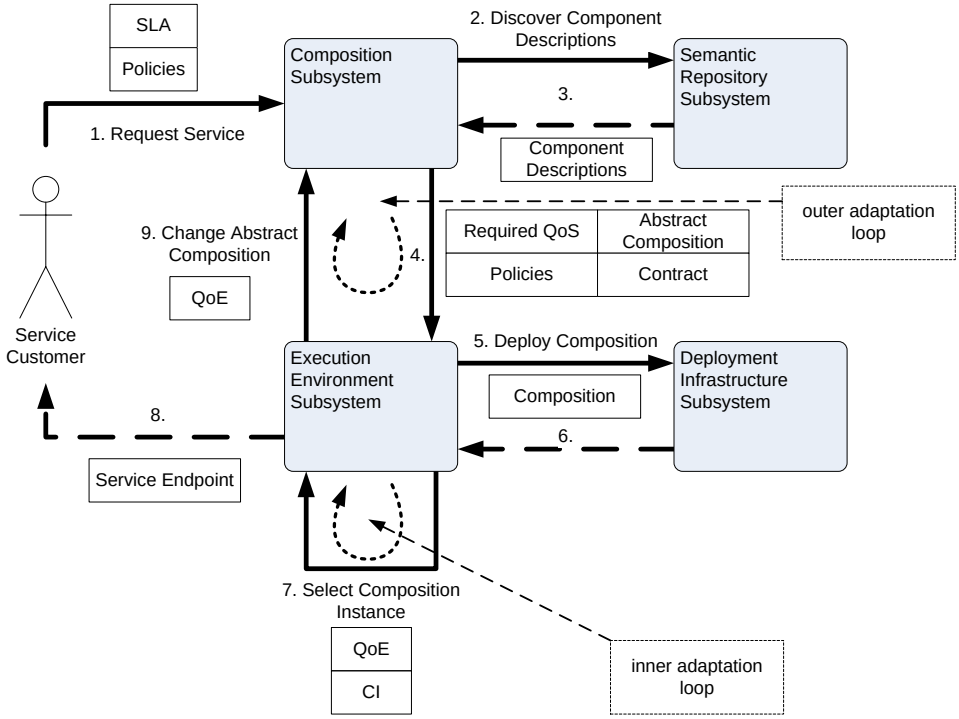
Warstwa ta podzielona jest na dwa podsystemy – Podsystem Środowiska Wykonawczego (ang. *Execution Environment Subsystem*), powiązany z określonym środowiskiem wykonawczym oraz Podsystem Infrastruktury Uruchomieniowej (ang. *Deployment Infrastructure Subsystem*), który odpowiada za rozmieszczenie komponentów usługi na istniejącej infrastrukturze dostarczyciela usługi.

Podsystem Środowiska Wykonawczego dokonuje odwzorowania Abstrakcyjnej Kompozycji na Kompozycję, a także odpowiada za wybór określonej Instancji Kompozycji na podstawie danych pochodzących z monitorowania usługi. Stworzony fizyczny opis rozmieszczenia komponentów, z których składa się usługa, jest przekazywany do Podsystemu Infrastruktury Uruchomieniowej, który dokonuje ich instalacji i rozmieszczenia z wykorzystaniem fizycznych węzłów infrastruktury.

4.2. Pętle adaptacji w architekturze systemu

Poniższa sekcja przedstawia odwzorowanie pętli adaptacji zaprezentowanych wcześniej, stanowiących projekt systemu. Szczegółowe przypadki użycia systemu są opisane dokładnie w rozprawie, w Rozdziale 4.

Na Rys. 6 został przedstawiony proces udostępniania usługi. Klient zgłasza do systemu żądanie dostarczenia usługi, zawierające *Service Level Agreement* (SLA) (składające się z kontraktu usługi oraz jej opisu semantycznego w postaci definicji celu), a także politykę adaptacji. Na podstawie celu zawartego w SLA, Podsystem Kompozycji dokonuje stworzenia Abstrakcyjnej Kompozycji i przekazuje ją do Podsystemu Środowiska Wykonawczego, który dokonuje uruchomienia tej usługi i dostarcza informacje o sposobie komunikacji z usługą (*Service Endpoint*) do klienta. Wewnętrzna pętla adaptacji jest uruchomiona wewnątrz Podsystemu Środowiska Wykonawczego i jest wykonywana w sposób ciągły, bez przerwy w trakcie działania usługi. Z drugiej strony, w przypadku gdy żądana jakość usługi nie może być dostarczona z wykorzystaniem wewnętrznej pętli adaptacji, do Podsystemu Kompozycji jest przekazywane żądanie zmiany Abstrakcyjnej Kompozycji. W związku z tym zewnętrzna pętla adaptacji znajduje się na styku tych dwóch podsystemów.



Rysunek 6: Proces udostępniania usługi

5. Implementacja systemu

Implementacja systemu została podzielona na dwie części. Pierwsza z nich, odpowiada głównie za obsługę semantycznej kompozycji, pozwalającej na automatyczne budowanie usług na żądanie. Z drugiej strony, ze względu na braki w istniejących technologiach komponentowych, konieczne było stworzenie środowiska wykonawczego, które spełni-

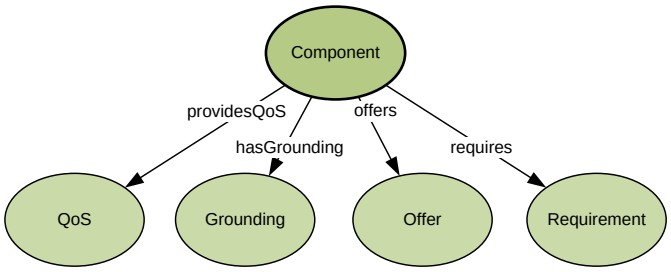
ałoby wymagania dotyczące adaptacji z punktu widzenia wewnętrznej pętli adaptacji. Poniższa sekcja opisuje obie części tej implementacji. Co więcej, w wyniku prac nad drugą z tych części powstało środowisko (tzw. Adaptacyjne SCA), które może być wykorzystanie niezależnie od całego systemu.

5.1. Warstwa semantycznej kompozycji

Warstwa semantycznej kompozycji dostarcza wysokopoziomowych mechanizmów budowy logiki usługi na podstawie definicji celu usługi (dostarczonego przez klienta) oraz semantycznego opisu komponentów (dostarczonego przez dostawców komponentów). Wynikiem działania tej warstwy jest Abstrakcyjna Kompozycja, przedstawiona we wcześniejszych częściach autoreferatu.

5.1.1. Semantyczny opis komponentów

Istniejące opisy semantyczne dedykowane dla usług, jak OWL-S [11] czy WSMO [12], posiadają mechanizmy specyficzne do opisu usług, a zwłaszcza dla technologii Web services. W związku z tym konieczne było opracowanie opisu, który mógłby zostać wykorzystywany dla opisu komponentów. Stworzona ontologia, wykorzystująca język OWL [13], bazuje na ontologii OWL-S, jednakże posiada specyficzne rozszerzenia. Ogólny zarys ontologii został przedstawiony na Rys. 7.



Rysunek 7: Ontologia opisu komponentów

Ontologia ta składa się z następujących części:

QoS, zawierającej szczegóły dotyczące pozafunkcjonalnych możliwości komponentu;

Grounding, zawierającej informacje dotyczące fizycznego dostępu do komponentu, sposób jego instalacji, etc.;

Offer, zawierającej informacje dotyczące funkcjonalnych możliwości oferowanych przez komponent. Opis ten jest przedstawiony w postaci IOPE;

Requirement, zawierającej informacje dotyczące funkcjonalnych wymagań komponentu (tj. takich, które muszą być zapewnione przez inne komponenty wchodzące w skład Kompozycji). Ten opis jest również przedstawiony w postaci IOPE.

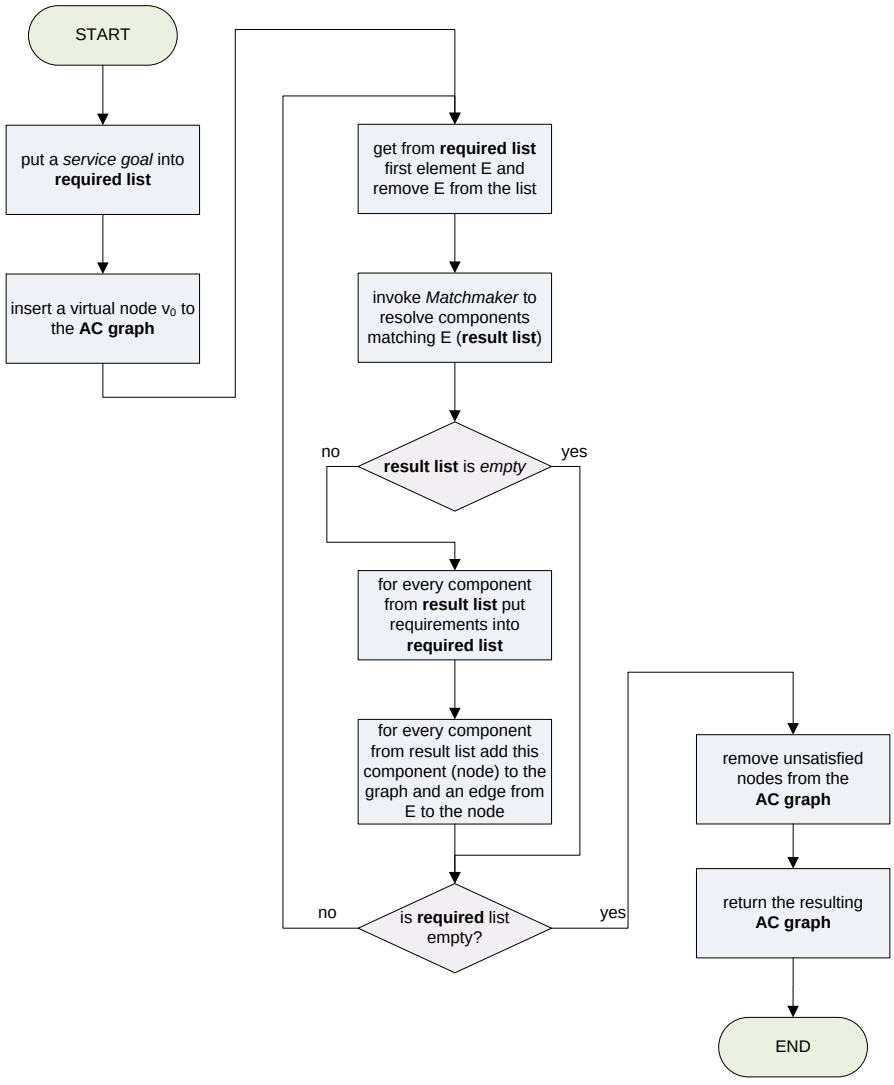
5.1.2. Semantyczne dopasowanie

Semantyczne dopasowanie (ang. *semantic matchmaking*) jest wykorzystywane w pracy w celu znalezienia komponentów, które spełniają wymagania (Requirement) stawiane przez inne komponenty, a tym samym znajdujące się razem z nimi w Kompozycji.

Podejście zastosowane w pracy jest zgodne z istniejącym w literaturze podejściem bazującym na różnych poziomach dopasowania (exact, plug-in, subsumes, intersection, disjoint). Istotnym elementem tego dopasowania jest zastosowanie języka SWRL [14] do opisu warunków wstępnych (ang. *preconditions*) oraz efektów (ang. *effects*) pracy komponentów, w postaci wyrażeń logicznych.

5.2. Semantyczne wnioskowanie

W pracy zastosowane zostało budowanie grafu Abstrakcyjnej Kompozycji poprzez wykorzystanie wnioskowania wstecz, przedstawione na Rys. 8.



Rysunek 8: Algorytm wnioskowania wstecz do budowy Abstrakcyjnej Kompozycji

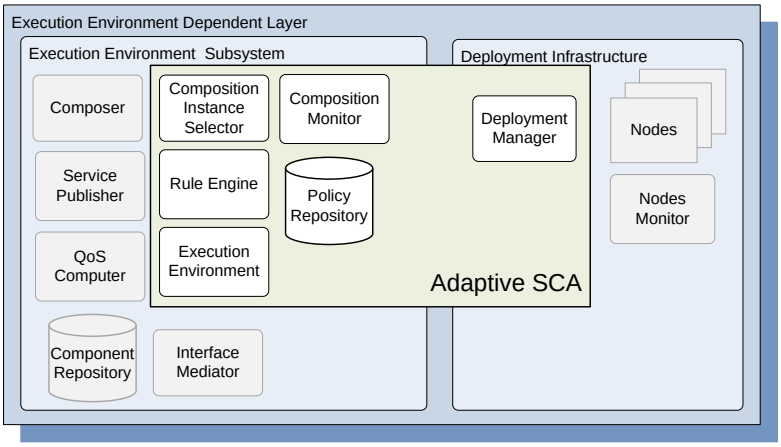
W pierwszym kroku do listy wymagań dodawany jest cel usługi (opisany w sposób analogiczny jak wymagania komponentu). Następnie, w pętli, wykonywane są operacje

semantycznego dopasowywania komponentów, które spełniają wymagania znajdujące się na tej liście. Ze względu na fakt, że takie komponenty również mogą posiadać dodatkowe wymagania, są one dołączane do listy. W przypadku gdy lista wymagań jest pusta, algorytm wychodzi z pętli. Tak zastosowany algorytm wymaga dodatkowo (na zakończenie) usunięcia komponentów, których wymagania nie zostały spełnione, a więc nie mogą zostać wykorzystane w Abstrakcyjnej Kompozycji.

5.3. Adaptacyjne SCA

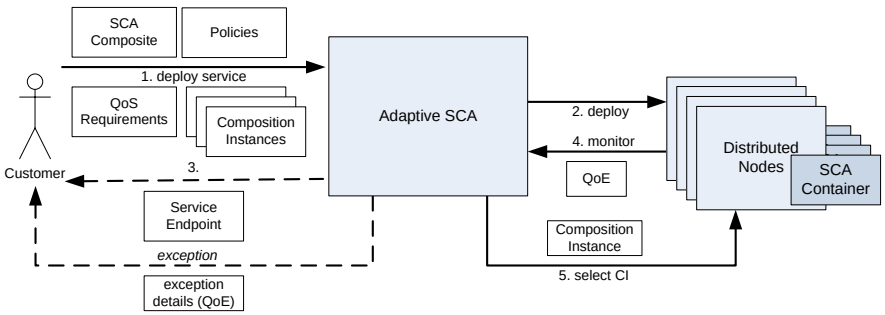
Adaptacyjne SCA (ang. *Adaptive SCA*, ASCA), jest rezultatem rozszerzenia specyfikacji SCA [15] o mechanizmy adaptacji, które wynikły z wymagań przedstawionych podczas analizy istniejących technologii komponentowych pod kątem ich zastosowania przy tworzeniu usług o zadanej jakości.

Z punktu widzenia systemu ODSPS, ASCA jest wykorzystywane jako środowisko wykonawcze rozszerzone o mechanizmy adaptacji, co pozwala na zastąpienie kilku komponentów architektury ODSPS, jak przedstawiono na Rys. 9.



Rysunek 9: Odwzorowanie komponentów z warstwy EEDL na Adaptacyjne SCA

Z drugiej strony, środowisko ASCA zostało zaprojektowane w sposób, który umożliwia jego niezależne wykorzystanie w sposób analogiczny do wykorzystania technologii SCA. W takim przypadku, to użytkownik systemu dostarcza deskryptor opisujący usługę (odpowiednik opisu Kompozycji), wraz z odpowiednimi politykami adaptacji oraz definicjami Instancji Kompozycji, które mogą być wykorzystane w procesie adaptacji. Proces ten jest przedstawiony na Rys. 10.

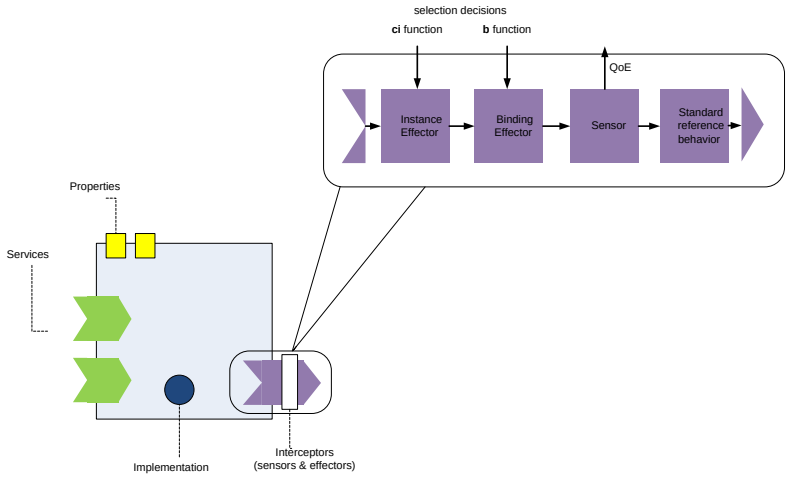


Rysunek 10: Uruchomienie i adaptacja kompozytu Adaptacyjnego SCA

5.3.1. Niskopoziomowe mechanizmy adaptacji dla SCA

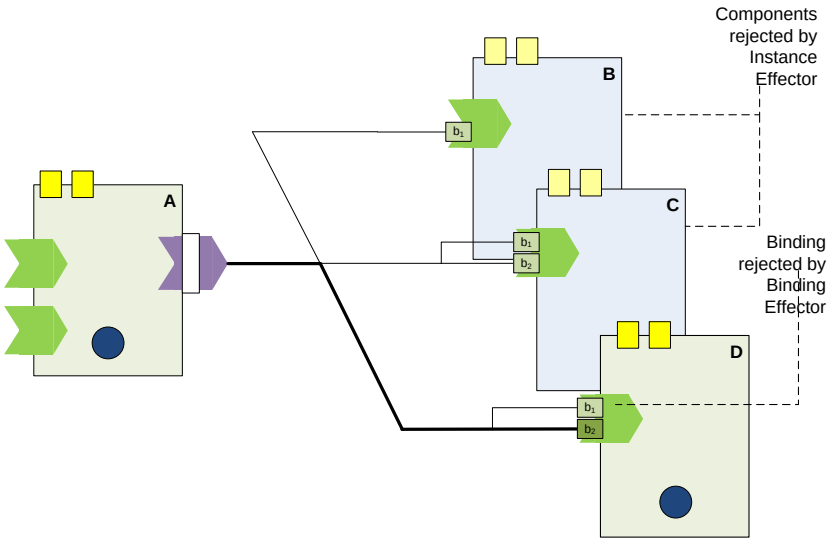
Mechanizmy adaptacji oparte na wyborze Instancji Kompozycji (o określonych parametrach QoS) z Kompozycji, przedstawione w poprzednich sekcjach, wymagają niskopoziomowych mechanizmów, które pozwolą na zaaplikowanie funkcji *ci* (dokonującej wyboru Instancji Kompozycji) oraz *b* (określającej pozafunkcjonalne wymagania dotyczące powiązań między komponentami). Istniejące implementacje SCA nie posiadają możliwości zmiany połączeń między komponentami w trakcie wykonania, w związku z czym autor dokonał rozbudowania implementacji Apache Tuscany SCA o ww. mechanizmy.

Aby istniała możliwość dynamicznej selekcji Instancji Kompozycji, a zarazem aby proces ten był niewidoczny dla komponentów, które są wykorzystywane do budowy usługi, referencje każdego z komponentów zostały wyposażone o mechanizm interceptorów. Referencje te są wykorzystywane przez SCA do przekazywania komunikacji do poszczególnych komponentów. Instalacja interceptorów (dwóch rodzajów: sensorów, pozwalających na monitorowanie przesyłanych danych, oraz efektorów, służących do aplikowania decyzji dotyczących funkcji *ci* i *b*), przedstawiona na Rys. 11, pozwala na przechwycenie standardowego zachowania referencji (tj. dokonywania transformacji wiadomości do określonego protokołu komunikacyjnego i przesłania jej do zadeklarowanego w pliku konfiguracyjnym komponentu) i tym samym na przekierowanie wiadomości do innego komponentu.



Rysunek 11: Komponent ASCA z zainstalowanymi interceptorami

Proces ten jest kilkuetapowy, jak przedstawiono na Rys. 12. Po pierwsze, za pomocą funkcji *ci*, określany jest komponent, który bierze udział w aktualnej Instancji Kompozycji i jest powiązany z aktualnie wykonywanym komponentem. Następnie, za pomocą funkcji *b*, określane są pozafunkcjonalne parametry połączenia między komponentami. Ze względu na możliwości SCA, aktualna implementacja funkcji *b* pozwala na wybór protokołu wiązania między komponentami (ang. *binding protocol*), który może zapewniać różne możliwości pozafunkcjonalne (takie jak transakcyjność, szyfrowanie, różne wymaganie na ilość przesyłanych danych, etc.). W dalszej części komunikacja przez referencję przechodzi przez sensor, który dokonuje jej analizy i przesyła wykryte parametry jakościowe QoE do systemu odpowiadającego za monitorowanie systemu.



Rysunek 12: Proces wyboru komponentu w referencji ASCA

Całość niskopoziomowej funkcjonalności ASCA jest udostępniona dla zarządzania poprzez interfejs Monitoringu i Zarządzania (ang. *Monitoring and Management Interface*, MMI), za pomocą technologii JMX. Szczegółowy opis tego interfejsu jest przedstawiony w rozprawie.

5.3.2. Adaptacja oparta o polityki

Zaprezentowane niskopoziomowe mechanizmy działające w oparciu o interceptory zainstalowane w referencjach komponentów pozwalają na zastosowanie zmiany Instancji Kompozycji na podstawie zadanych funkcji *ci* oraz *b*. Jednakże głównym wyzwaniem stawianym przy tworzeniu usług, które mogą się adaptować do zmieniających się wymagań jakościowych użytkownika oraz zmieniających się możliwości dostarczyciela usług, jest stworzenie schematu opisu, jaki może być wykorzystywany do definicji adaptacji.

Adaptacja przedstawiona w pracy opiera się na politykach (ang. *adaptation policies*), zapisanych z wykorzystaniem abstrakcyjnego języka, określających wysokopoziomowe zasady adaptacji pozwalającej na wybór określonej Instancji Kompozycji.

Polityki są następnie odwzorowywane na niskopoziomowe reguły, zdefiniowane w systemie przez Zarządcę Polityk (ang. *Policy Manager*). Reguły te operują na faktach, opisujących zarówno Instancje Kompozycji, jak i metryki jakości zmierzone w systemie (QoE) i deklarowane przez dostarczycieli komponentów i usługi (QoS). W przypadku zmiany wyboru Instancji Kompozycji, system ODSPS dokonuje odpowiednich wyborów poprzez wpływ na efektory wyboru instancji i protokołów przy użyciu interfejsów MMI poszczególnych komponentów w czasie ich wykonania.

Problem zaburzeń jakości usługi

W idealnym rozwiązaniu, jakość usługi deklarowana przez dostarczyciela usługi (QoS) jest identyczna jak jakość obserwowana przez użytkownika (QoE). Jednakże w rzeczywistych systemach może dochodzić do rozbieżności między tymi wartościami, przykładowo z powodu przeciążenia sieci lub chwilowego przeciążenia węzłów infrastruktury. W

wyniku takich rozbieżności, musi być możliwość zmiany Instancji Kompozycji na taką, która w danej chwili lepiej spełnia wymagania pozafunkcjonalne. Z drugiej strony należy pamiętać o fakcie, iż wartości QoE są dostępne wyłącznie wtedy, gdy Instancja Kompozycji jest wybrana (gdyż dane te pochodzą z jej monitorowania), w związku z czym nie można opierać się wyłącznie na wartościach QoE. Rozwiązaniem tego problemu jest wprowadzenie współczynnika *zaufania* (ang. *trust*), który opisuje „jak bardzo możemy zaufać wartościom QoS zadeklarowanym przez dostawcę usługi". Współczynnik zaufania można zdefiniować jako stosunek wartości QoS do QoE (w dalszej analizie autor zakłada, że wartość mniejsza metryki jest lepsza od wartości większej; w przypadku przeciwnym należy stosować odwrotności QoS i QoE). W takim przypadku, gdy współczynnik zaufania jest mniejszy od 1, obserwowana jakość jest „gorsza" od zadeklarowanej, a gdy jest większy od 1, jakość obserwowana jest lepsza od zadeklarowanej (w związku z czym nie należy zbytnio ufać zadeklarowanej wartości).

Po dokonaniu transformacji powyższej definicji, otrzymujemy wzór na jakość obserwowaną:

$$QoE = \frac{QoS}{trust}$$

Następnie wracając do powodów zdefiniowania współczynnika zaufania, możemy dokonać minimalizacji wpływu chwilowych zmian w jakości usługi, poprzez wprowadzenie funkcji, którą można zastosować na wartości tego współczynnika:

$$x_1 = f(x);$$

$$x_{n+1} = f(x_n);$$

$$\lim_{n\to\infty} x_n = 1.$$

Jeśli przedstawiona funkcja $f(x)$ będzie stosowana w kolejnych krokach czasowych na współczynniku zaufania dla Instancji Kompozycji, która nie jest wywoływana, po pewnym czasie współczynnik zbiegnie się do wartości 1. W takim wypadku, jeśli zaufanie dla danej Instancji Kompozycji spadło, np. ze względu na chwilowe przeciążenie systemu (i została ona odrzucona jako nie spełniająca jakości), to po pewnym czasie Instancja będzie mogła być wykorzystana ponownie.

Reguły i polityki

Jednym z głównych zadań dla Adaptacyjnego SCA jest dokonywanie adaptacji usługi z wykorzystaniem wysokopoziomowych polityk. Polityki te zawierają abstrakcyjne decyzje, które powinny być podjęte pod określonymi warunkami. Następnie decyzje i warunki są interpretowane przez Silnik Regułowy, na podstawie ich definicji znajdujących się w Repozytorium Polityk (ang. Policy Repository).

Proste polityki są przedstawione w formie deklaracji:

- utrzymaj wartość metryki M na jak najwyższym/najniższym poziomie;

- wartość metryki M musi być niższa/wyższa niż V;
- utrzymaj średnią wartość metryki M pomiędzy wartościami A i B;
- wartość metryki M nie powinna się zmienić o więcej niż V%.

Złożone polityki, przedstawione w postaci wyrażeń warunkowych, są wykorzystywane do opisu interakcji pomiędzy politykami, przykładowo:

- jeżeli wartość metryki M jest wyższa/niższa od V, nie stosuj polityki P.

Takie podejście pozwala na tworzenie rozbudowanych polityk adaptacji, w zależności od warunków, w jakich znajduje się uruchomiona usługa.

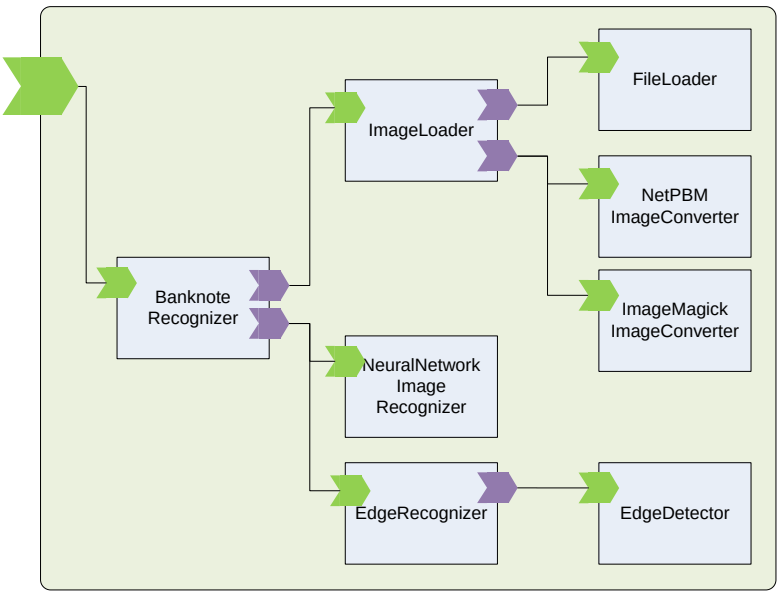
6. Badania eksperymentalne

W celu zbadania, czy stworzona prototypowa implementacja systemu ODSPS spełnia wymagania przedstawione w rozprawie, zaproponowany został szereg eksperymentów, skupiających się na różnych elementach. Zaproponowane studium przypadku prezentuje jedną, typową usługę biznesową i zachowanie systemu w celu zapewnienia jej jakości w różnych przypadkach, w oparciu o określone polityki.

Poniższe opracowanie przedstawia jedynie ogólny zarys testów i przykładowe wyniki niektórych z nich. Szczegółowe informacje na ich temat oraz dokładna analiza znajduje się w Rozdziale 7 rozprawy.

6.1. Ogólny zarys usługi

Usługa wykorzystywana w badaniach eksperymentalnych służy do rozpoznawania nominałów banknotów na podstawie zdjęć banknotów. W podstawowych eksperymentach wykorzystywane są cztery różne Instancje Kompozycji (zawierające różne komponenty służące do konwersji formatów obrazów, a także do rozpoznawania obrazów, np. oparte o sieci neuronowe lub wykrywanie krawędzi). W Eksperymencie 4 wykorzystywane są dodatkowe dwie Instancje Kompozycji, co zostało szczegółowo opisane w rozprawie. Rys. 13 przedstawia ogólny zarys wykorzystanej usługi (Kompozycji).



Rysunek 13: Kompozycja w badaniach eksperymentalnych

Jako środowisko testowe została przygotowana heterogeniczna infrastruktura wykorzystująca serwery Sun Blade 6000 oparte na procesorach SPARC oraz Xeon, a także komputery klasy PC. Jako system operacyjny został wykorzystany Solaris w wersji 10 (serwery Sun), Gentoo Linux 10.0 (kernel 2.6.32, komputer PC) oraz Microsoft Windows (komputer PC).

6.2. Eksperymenty

W pracy zostało przeprowadzonych siedem eksperymentów, z czego pięć z nich dotyczy testów funkcjonalnych przygotowanej prototypowej implementacji, a pozostałe dwa sprawdzają skalowalność systemu.

Przeprowadzone eksperymenty funkcjonalne są następujące:

Eksperyment 1 – stworzenie i uruchomienie usługi w oparciu o wymagania użytkownika. Klient dostarcza wymagania funkcjonalne w postaci celu usługi z wykorzystaniem zadanej ontologii. ODSPS tworzy Abstrakcyjną Kompozycję i na jej podstawie tworzy Kompozycję oraz instaluje komponenty biorące w niej udział. Na zakończenie wybierana jest Instancja Kompozycji, która spełnia pozafunkcjonalne wymagania klienta.

Eksperyment 2 – adaptacja usługi w wyniku zmian pozafunkcjonalnych możliwości systemu. Eksperyment 2 opera się na Eksperymencie 1, wprowadzając w pewnym momencie zaburzenie systemu poprzez przeciążenie węzłów. Eksperyment przedstawia dwa przypadki – jeden, w którym przeciążenie węzłów jest chwilowe, i drugi, w którym przeciążenie jest permanentne.

Eksperyment 3 – adaptacja usługi w wyniku zmian pozafunkcjonalnych wymagań klienta. Eksperyment ten jest podobny do Eksperymentu 2, jednakże inna jest geneza zmian i tym samym nieco inne jest zachowanie systemu w celu zapewnienia jakości usługi.

Eksperyment 4 – zachowanie systemu w przypadku, gdy wewnętrzna pętla adaptacji nie wystarczy dla utrzymania usługi o określonej jakości. W poprzednich eksperymentach w procesie adaptacji brała udział wyłącznie wewnętrzna pętla adaptacji. W Eksperymencie 4 zaburzenie jest tak duże, że wymagane są zmiany w Kompozycji. Dodatkowo pojawia się założenie, iż w trakcie pracy usługi pojawiły się dodatkowe komponenty, które mogą być wykorzystane w Kompozycji.

Eksperyment 5 – zastosowanie polityk adaptacji dla określenia wyboru protokołów komunikacji pomiędzy komponentami. Eksperyment ten jest podobny do Eksperymentu 1, jednakże dodatkowo zostają określone polityki określające zasady wyboru określonych protokołów komunikacji, pozwalając na komunikację szyfrowaną (z wykorzystaniem HTTPS) oraz otwartą (z wykorzystaniem HTTP).

Przeprowadzone eksperymenty sprawdzające skalowalność systemu:

Eksperyment 6 – skalowalność systemu podczas rosnącej ilości komponentów. Eksperyment ten skupia się na analizie wydajności komponentu odpowiadającego za wybór Instancji Kompozycji (Composition Instance Selector), gdyż rosnąca ilość komponentów tworzących Kompozycję powoduje wykładniczy wzrost ilości Instancji Kompozycji, które mogą być wybrane dla zapewnienia określonej jakości.

Eksperyment 7 – skalowalność systemu podczas rosnącej liczby metryk jakości. Eksperyment ten skupia się na analizie wydajności komponentu odpowiadającego za wybór Instancji Kompozycji (Composition Instance Selector) bazującego na silniku regułowym. Ze względu na to, że każda metryka jakości jest reprezentowana jako fakt dla silnika regułowego, ilość metryk powinna mieć znaczący wpływ na proces wyboru odpowiedniej Instancji Kompozycji.

6.3. Analiza Eksperymentu 2

Eksperyment 2 zaczyna się w 40 sekundzie, tuż po zakończeniu Eksperymentu 1, gdy Kompozycja jest uruchomiona na istniejącej infrastrukturze oraz określona Instancja Kompozycji, która spełnia wymagania pozafunkcjonalne, jest wybrana.

W istniejącej Kompozycji istnieją cztery Instancje Kompozycji (szczegółowy opis znajduje się w rozprawie), o następujących własnościach pozafunkcjonalnych (w dalszej części są one reprezentowane za pomocą nazw symbolicznych):

Nazwa	**czas wykonania** [s]	**koszt** [eurocenty]	**zużycie pamięci** [MB]
CI 1	5	2	2000
CI 2	8	5	1000
CI 3	2	9	1500
CI 4	5	4	1400

Tablica 1: Zadeklarowana jakość usługi (QoS) dla Instancji Kompozycji

Polityki adaptacja dostarczone przez klienta usługi są następujące:

- utrzymaj średnią wartość czasu wykonanie pomiędzy 4 a 6 sekund;

- utrzymaj średni koszt pojedynczego wywołania pomiędzy 3 a 6 eurocentów;
- maksymalny dozwolony czas wykonania usługi wynosi 11 sekund.

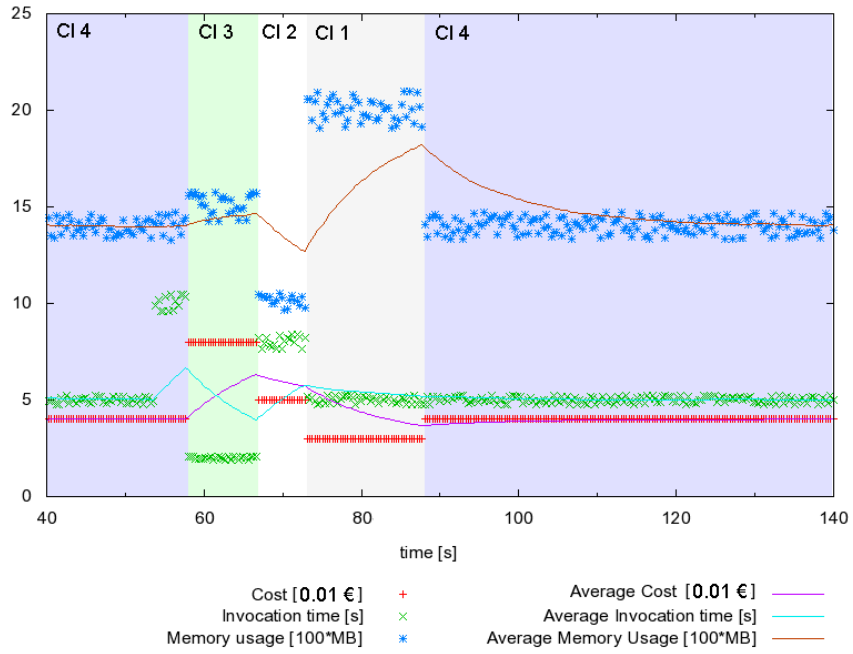
Natomiast polityki adaptacji dostarczone przez Zarządcę Polityk są następujące:

- utrzymaj średnią wartość zużycia pamięci pomiędzy 1300 MB a 1800 MB;
- jeżeli polityki dotyczące czasu wykonania i kosztu są spełnione przez kilka Instancji Kompozycji, wybierz tą, której koszt jest niższy.

W 58 sekundzie, czas wykonania Instancji Kompozycji CI 4 wzrasta do 10 sekund, co wynika z przeciążenia węzłów odpowiadających za tą instancję.

6.3.1. Chwilowe zaburzenie jakości usługi

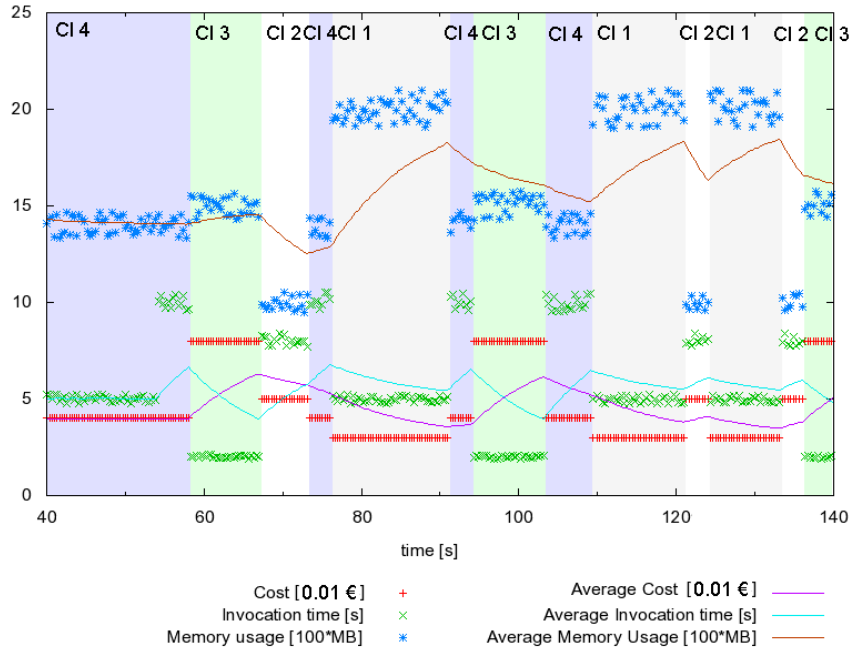
W pierwszym przypadku w Eksperymencie 2, zaburzenie jakości usługi jest chwilowe i znika po kilkunastu sekundach. Zwiększenie czasu wykonania usługi do 10 sekund nie jest akceptowalne przez klienta. Powoduje to zmniejszenie współczynnika zaufania (ang. *trust*), służącego do wyliczania jakości usługi, do wartości 0.50. W związku z tym zostaje wybrana inna Instancja Kompozycji, która posiada krótszy czas wykonania (w tym wypadku CI 3). Jednakże koszt użycia CI 3 jest zbyt wysoki i powoduje zwiększenie średniego kosztu powyżej wartości dopuszczalnej, co skutkuje wyborem Instancji Kompozycji CI 2. Ze względu na to, iż czas wykonania CI 2 jest zbyt wysoki, po chwili zostaje ona odrzucona i wybrana zostaje CI 1. W 84 sekundzie współczynnik zaufania dla czasu wywołania CI 4 jest już bliski 1.0, co pozwala na próbne wybranie tej Instancji Kompozycji. Ponieważ zaburzenie czasu wykonania było tylko chwilowe, CI 4 może już być wybrana na stałe. Całość procesu adaptacji można zobaczyć na Rys. 14.



Rysunek 14: Adaptacja przy chwilowym zaburzeniu jakości usługi

6.3.2. Permanentne zaburzenie jakości usługi

W drugim przypadku w Eksperymencie 2, zaburzenie jakości usługi jest permanentne. Różnica w adaptacji pojawia się w momencie, gdy współczynnik zaufania do Instancji Kompozycji CI 4 zbiega się do wartości 1.0 i CI 4 zostaje wybrana. Gdy okazuje się (po wykonaniu przez klienta operacji na usłudze), że czas odpowiedzi dalej nie jest akceptowalny (10 sekund), współczynnik zaufania z powrotem przyjmuje wartość 0.50. Całość tego procesu adaptacji jest przedstawiona na Rys. 15.



Rysunek 15: Adaptacja przy permanentnym zaburzeniu jakości usługi

7. Podsumowanie

Wynikiem prac i analiz przedstawionych w rozprawie jest System Udostępniania Usług na Żądanie. System został zaprojektowany biorąc pod uwagę wymagania i model usługi zaprezentowane w rozdziałach 2 i 3, a także przedstawiona została prototypowa implementacja w oparciu o technologię komponentową SCA. Prototyp jest przykładem systemu udostępniania usług, który bierze pod uwagę zarówno funkcjonalne, jak i pozafunkcjonalne wymagania dostarczone przez użytkownika (klienta). Implementacja ta nie rozwiązuje wszystkich problemów, jakie pojawiają się przy tworzeniu tego typu rozwiązań, lecz w niektórych miejscach jedynie wskazuje dalsze drogi badań i rozwoju.

Przedstawiona adaptacja w oparciu o polityki odwzorowywane na niskopoziomowe reguły powinna być traktowana jako punkt wyjściowy do dalszych prac w tym kierunku. Jak widać po przeprowadzonych badaniach, wyniki są dość obiecujące.

7.1. Weryfikacja tezy

Teza rozprawy, przedstawiona w Rozdziale 1, wynika z analizy istniejących systemów komponentowych, która pokazała, że obecne systemy nie pozwalają na adaptację usług

zbudowanych z komponentów do zmieniających się pozafunkcjonalnych wymagań klienta i możliwości infrastruktury.

Praca przedstawia system ODSPS, służący do tworzenia usług z uwzględnieniem jakości usługi. Przedstawione w prototypowej implementacji rozszerzenia dla środowiska komponentowego SCA (Adaptacyjne SCA), pozwalają na definiowanie polityk adaptacji, które sterują zmianą Instancją Kompozycji odpowiadającej za logikę usługi. Na podstawie metryk jakości usługi pochodzących z monitorowania usługi (QoE) oraz zadanych wartości jakości (QoS), system dokonuje (w czasie wykonania) wyboru takiej Logiki Usługi, która spełnia wymagania klienta.

Adaptacja przedstawiona w pracy jest opisywana abstrakcyjnymi politykami adaptacji, które są następnie odwzorowywane na niskopoziomowe reguły, uruchamiane przez specjalizowane silniki regułowe. Pozwala to na deklaratywne i elastyczne tworzenie nowych polityk adaptacji, a także na zmianę zachowania polityk adaptacji poprzez ich przedefiniowanie.

Zaprezentowane eksperymenty dowodzą, iż zastosowane podejście jest poprawne i dowodzą tezy na przykładzie technologii SCA. Usługa przedstawiona w studium przypadku, zbudowana z komponentów zintegrowanych z wykorzystaniem Adaptacyjnego SCA, dopasowuje się do pozafunkcjonalnych wymagań klienta, w zależności od różnych sytuacji, przedstawionych w eksperymentach.

7.2. Kierunki dalszego rozwoju

Przedstawiona prototypowa implementacja systemu powinna być postrzegana jako punkt wyjściowy dla systemów udostępniania usług opartych na paradygmacie SOA. Ponadto, autor dostrzega dalsze możliwości rozwoju tego systemu, których implementacja pozwoliłaby na szersze jego zastosowanie. Do najważniejszych rozszerzeń można zaliczyć:

- umożliwienie dynamicznej instalacji komponentów na węzłach SCA, co pozwoliłoby na ponowną instalację Kompozycji bez zatrzymywania pracy usługi. W obecnym rozwiązaniu, zmiana Kompozycji (wynikająca ze zmiany Abstrakcyjnej Kompozycji) powoduje chwilową niedostępność usługi dla klienta;
- integracja systemu z rozbudowanym systemem rozmieszczania komponentów, który rozmieszczałby komponenty na rozproszonych węzłach w oparciu o ich wymagania pozafunkcjonalne;
- implementacja komponentu QoS Computer. Ten komponent, przedstawiony w architekturze systemu ODSPS, odpowiada za obliczanie jakości poszczególnych Instancji Kompozycji, w oparciu o komponenty składające się na tą Instancję i specyfikację rozmieszczenia komponentów. Jest to bardzo trudne zadanie, jednakże może ono przynieść obiecujące wyniki;
- integracja z technologią zaawansowanego przetwarzania zdarzeń (ang. *Complex Event Processing* (CEP)), która mogłaby działać jako dodatkowa warstwa, służąca

do dostarczania faktów dla silnika regułowego. Technologia CEP pozwala na deklaratywne definiowanie złożonych zdarzeń na podstawie prostych. W systemie ODSPS może być wykorzystana jako źródło złożonych metryk QoS.

Dodatkowo, ciekawym kierunkiem rozwoju są także dalsze prace nad politykami adaptowalności, zwłaszcza w oparciu o analizę istniejących usług i kontraktów.

Wybrane pozycje literatury

[1] A. Arsanjani, L.-J. Zhang, M. Ellis, A. Allam, and K. Channabasavaiah, "S3: A Service-Oriented Reference Architecture," *IT Professional*, vol. 9, no. 3, pp. 10–17, 2007.

[2] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design.* Prentice Hall, 2005.

[3] *UDDI Version 3.0.2*, http://uddi.org/pubs/uddi˙v3.htm, UDDI Spec Technical Committee, October 2004.

[4] J. A. Estefan, K. Laskey, F. G. McCabe, and D. Thornton, *Reference Architecture for Service Oriented Architecture 1.0*, OASIS Standard, April 2008.

[5] C. M. MacKenzie, K. Laskey, F. G. McCabe, P. F. Brown, and R. Metz, *Reference Model for Service Oriented Architecture 1.0*, OASIS Standard, October 2006.

[6] D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, *The Description Logic Handbook: Theory, implementation, and applications.* Cambridge University Press, 2003.

[7] T. Erl, *SOA Design Patterns.* Prentice Hall, 2009.

[8] K. Baclawski and A. Simeqi, "Toward Ontology-based Component Composition," in *10th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, October 2001, pp. 1–11.

[9] A. Arsanjani, F. Curbera, and N. Mukhi, "Manners Externalize Semantics for On-demand Composition of Context-aware Services," in *ICWS '04: Proceedings of the IEEE International Conference on Web Services.* Washington, DC, USA: IEEE Computer Society, 2004, pp. 583–590.

[10] A. Hibner and K. Zielinski, "Semantic-based Dynamic Service Composition and Adaptation," in *Proceedings of the 2007 IEEE Congress on Services*, July 2007, pp. 213–220.

[11] *OWL-S: Semantic Markup for Web Services*, http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/, The World Wide Web Consortium (W3C), November 2004.

[12] "ESSI WSMO working group home," http://www.wsmo.org/, accessed on December 2008.

[13] *OWL Web Ontology Language. W3C Recommendation*, http://www.w3.org/TR/owl-features/, The World Wide Web Consortium (W3C), February 2004.

[14] *SWRL: A Semantic Web Rule Language Combining OWL and RuleML Specification*, http://www.w3.org/Submission/SWRL/, The World Wide Web Consortium (W3C), May 2004.

[15] *SCA Service Component Architecture. Assembly Model Specification, version 1.00*, OASIS, May 2007.