Autoreferat rozprawy doktorskiej

Architecture of a Framework for Dynamic, Semantics Based Deployment of Overlay Networks in Peer-to-Peer Environments

Architektura systemu dynamicznego tworzenia sieci nakładkowych w oparciu o informacje semantyczne w środowiskach peer-to-peer

mgr inż. Sławomir Zieliński

PROMOTOR: prof. dr hab. inż. Krzysztof Zieliński – Akademia Górniczo-Hutnicza, WEAIiE

RECENZENCI: prof. dr hab. inż. Wojciech Cellary – Uniwersytet Ekonomiczny w Poznaniu

prof. dr hab. inż. Jacek Kitowski – Akademia Górniczo-Hutnicza, WEAIiE

1. Wprowadzenie

Jednym z najważniejszych powodów popularności sieci węzłów równorzędnych (ang. *peer-to-peer*) jest łatwość ich użytkowania. W celu skorzystania z udostępnianych usług w typowej sieci tego rodzaju, przeznaczonej do współdzielenia zasobów, wystarczy zainstalować program kliencki i wykonać kilka prostych czynności konfiguracyjnych. Można zauważyć, że ukierunkowanie procesu tworzenia oprogramowania na łatwość jego obsługi wspomaga tworzenie grup użytkowników, posiadających podobne oczekiwania względem sieci.

Współpraca pomiędzy odległymi węzłami stanowi podstawę funkcjonowania systemów rozproszonych. W zależności od przeznaczenia systemu, węzły mogą pełnić rolę dostawców wyspecjalizowanych usług lub po prostu udostępniać swoje zasoby, takie jak moc obliczeniowa, pamięć operacyjna czy przestrzeń dyskowa. Współpraca pomiędzy węzłami jest często obsługiwana przez warstwę pośrednią (ang. *middleware*), odpowiedzialną m.in. za ukrywanie złożoności sieci komputerowej łączącej węzły systemu.

Prezentowana praca opisuje model oraz implementację prototypu platformy REMP (ang. **RE**mote **M**essage **P**rocessing framework), która wykorzystuje sieć węzłów równorzędnych do dostarczenia użytkownikowi – programiście – mechanizmów pozwalających na delegowanie przetwarzania strumieni wiadomości z maszyny użytkownika do węzłów zdalnych. Głównym celem autora było stworzenie platformy elastycznej oraz łatwej do użycia tak, by ułatwić tworzenie grup o wspólnych oczekiwaniach względem

wykonywania aplikacji rozproszonych. Platforma rozszerza możliwości współczesnych ram programowych dla sieci węzłów równorzędnych w kwestiach:

- integracji podsystemów generacji i rozmieszczania węzłów przetwarzających z podsystemem rejestracji zapotrzebowania na strumień wiadomości,
- wprowadzenia możliwości stosowania ekspresywnych zapytań bazujących na semantycznym opisie strumieni wiadomości emitowanych przez źródła,
- wprowadzenia łatwego w użyciu mechanizmu odwzorowania aplikacji użytkownika na sieć nakładkową, rozmieszczaną następnie na sieci węzłów równorzędnych.

1.1. Zarys funkcjonalności proponowanej platformy

Podstawowym celem opracowania platformy REMP jest dostarczenie ramy programowej przeznaczonej do tworzenia sieci węzłów równorzędnych zorientowanych na obsługę dynamicznie tworzonych sieci nakładkowych przetwarzających strumienie wiadomości. Tworzone i obsługiwane sieci nakładkowe składać się mają z węzłów odpowiadających za kolejne etapy przetwarzania.

Systemy tworzone na bazie proponowanej platformy obsługują dwie klasy aktorów – producentów i konsumentów wiadomości. Mechanizm obsługi zapytań pozwala nie tylko na wybór źródła wiadomości, ale także na przekazanie platformie REMP kodu przetwarzania wiadomości w ramach specyfikacji sieci nakładkowej. Specyfikacja taka stanowi bazę dla wygenerowania i rozmieszczenia węzłów sieci przez platformę. Podstawowe etapy przetwarzania specyfikacji to:

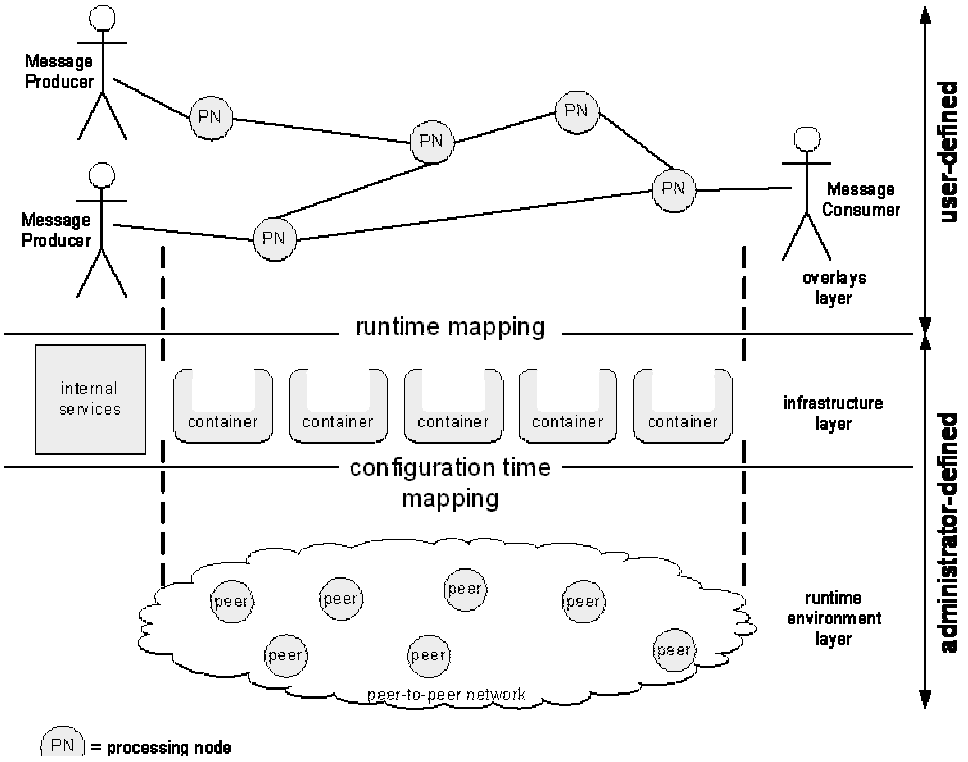
- wyszukiwanie producentów wiadomości,
- generacja lub wyszukanie węzłów przetwarzających,
- rozmieszczenie węzłów i połączenie ich kanałami komunikacyjnymi.

Pierwsze dwa z wymienionych etapów realizowane są wyłącznie na bazie specyfikacji dostarczonej przez konsumenta; realizacja trzeciego zależy od bieżącej konfiguracji instancji proponowanej platformy – informacje na ten temat zbierane są za pomocą wbudowanych mechanizmów introspekcji.

Generacja sieci nakładkowej składającej się z węzłów przetwarzających polega na transformacji kodu przetwarzającego wiadomości na możliwe do rozmieszczenia komponenty, odnalezieniu odpowiednich kontenerów oraz rozmieszczeniu w nich komponentów. Kontenery stanowią zatem podstawowy element infrastrukturalny zapewniany przez platformę.

Architektura systemów tworzonych z wykorzystaniem proponowanej platformy składa się z trzech warstw (rys. 1):

- warstwy środowiska wykonawczego,
- warstwy infrastruktury dostarczanej przez REMP,
- warstwy sieci nakładkowych.



Rys. 1. Zarys architektury systemu przetwarzania wiadomości

Warstwy środowiska wykonawczego oraz infrastruktury są tworzone przez administratorów, natomiast elementy warstwy sieci nakładkowych reprezentują aplikacje użytkowników. Zadaniem instancji proponowanej platformy jest dynamiczna implementacja sieci nakładkowych na bazie zasobów udostępnianych przez niższe warstwy. Po zaimplementowaniu sieci, zadaniem instancji platformy jest jej utrzymanie w warunkach zmiennego środowiska wykonawczego.

1.2. Teza i cele badawcze rozprawy

Teza prezentowanej rozprawy została sformułowana następująco:

Contemporary peer-to-peer middleware environments provide enough support for building a framework for dynamic deployment of message processing overlay networks. Augmentation of peer-to-peer middleware with semantic descriptions of messages improves the flexibility of the message processing delegation service.

W tłumaczeniu na język polski:

Współczesne środowiska sieci węzłów równorzędnych zapewniają wystarczające wsparcie dla potrzeb dynamicznego tworzenia sieci nakładkowych przetwarzających wiadomości. Wprowadzenie semantycznego opisu wiadomości zwiększa elastyczność usługi delegacji przetwarzania.

W celu wykazania tezy pracy sformułowano następujące cele badawcze:

- opracowanie architektury ramy programowej tworzenia systemów obsługujących dynamiczne tworzenie sieci nakładkowych zorientowanych na przetwarzanie strumieni wiadomości,

- wbudowanie w ramę programową mechanizmu realizacji odwzorowań pomiędzy semantycznymi opisami wiadomości generowanych przez producentów oraz konceptualnymi zapytaniami tworzonymi przez konsumentów,
- wyposażenie ramy programowej w mechanizmy łatwej konfiguracji, rekonfiguracji oraz rozszerzania funkcjonalności ramy,
- implementację rdzenia ramy programowej na bazie istniejącej technologii tworzenia sieci węzłów równorzędnych.

1.3. Osiągnięcia rozprawy

W ocenie autora, główne osiągnięcia prezentowanej rozprawy obejmują:

- wprowadzenie koncepcji implementacji komponentowych aplikacji przetwarzających strumienie wiadomości jako dynamicznie tworzonych, niezależnych sieci nakładkowych złożonych z elementów generowanych w czasie wykonania instancji platformy REMP,
- wprowadzenie koncepcji integracji podsystemów dynamicznej generacji i rozmieszczania węzłów z podsystemami odpowiedzialnymi za realizację konceptualnych zapytań dotyczących źródeł wiadomości,
- wprowadzenie częściowo deklaratywnego modelu tworzenia aplikacji dla sieci węzłów równorzędnych,
- opracowanie usługi dynamicznego rozmieszczania węzłów sieci nakładkowej, bazującej na wynikach realizacji konceptualnych zapytań o źródła wiadomości,
- opracowanie mechanizmów ponownego wykorzystania pośrednich wyników przetwarzania strumieni wiadomości,
- implementację prototypu ramy programowej obejmującego środowisko wykonawcze dla aplikacji opartych na proponowanym rozwiązaniu,
- praktyczną weryfikację proponowanych rozwiązań.

2. Prace związane z obszarem badań

Zagadnienia związane z budową heterogenicznych systemów rozproszonych są popularnym przedmiotem badań, których rezultatem są technologie warstwy pośredniej (ang. *middleware*). Różnorodność warstw pośrednich jest do pewnego stopnia odzwierciedleniem zróżnicowania zastosowań systemów rozproszonych [Mahmoud 2004]. Większość istniejących różnic jest wynikiem wyboru różnych rozwiązań typowych zagadnień projektowych, takich jak:

- wybór modelu przetwarzania,
- wybór mechanizmów dystrybucji i interpretacji metadanych,
- określenie schematów adresacji elementów systemu,
- określenie zakresu i sposobu realizacji wymagań niefunkcjonalnych.

Rozdział 2 rozprawy poświęcony jest przeglądowi typowych rozwiązań w zarysowanym zakresie.

2.1. Przegląd modeli przesyłania i przetwarzania wiadomości wykorzystywanych we współczesnych systemach rozproszonych

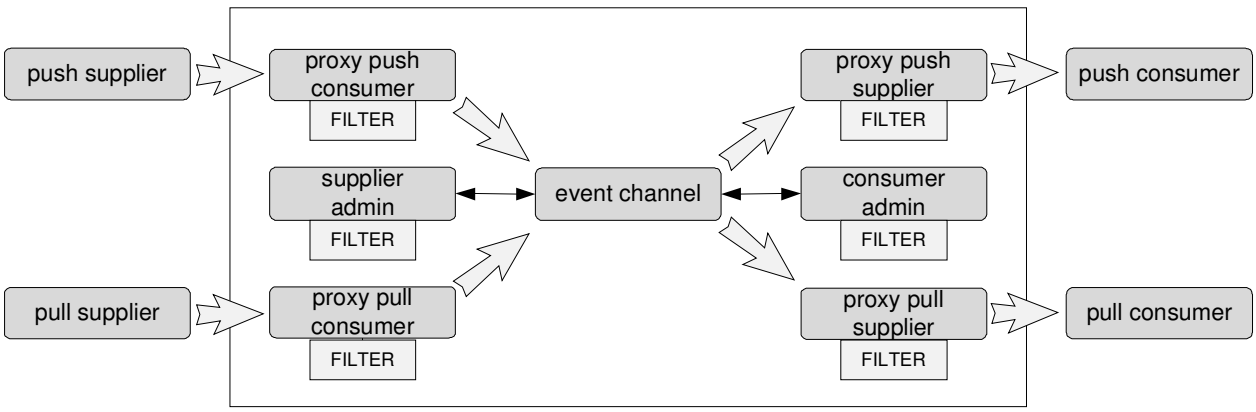
Współczesne systemy rozproszone zorientowane na przesyłanie i przetwarzanie wiadomości często bazują na współdzielonej infrastrukturze złożonej z brokerów wiadomości (stosowane jest także podejście scentralizowane, bazujące na jednym serwerze). Przetwarzanie wiadomości jest zazwyczaj organizowane na bazie modelu kanałowego bądź grafowego (por. p. 2.1.1 i 2.1.2). Jedną z podstawowych funkcji brokerów, jako jednostek implementujących wymienione modele, jest wybór tras dla wiadomości. Rozdział 2 rozprawy zawiera również – skrótowo przedstawiony w punkcie 2.1.3 niniejszego opracowania – przegląd najbardziej popularnych koncepcji w tym zakresie.

2.1.1. Przetwarzanie kanałowe (channel-based processing)

Proste serwisy przesyłania wiadomości, takie jak CORBA Event Service [OMG 2000-1] oparte są na koncepcji kanału przesyłania wiadomości, który stanowi element pośredniczący pomiędzy producentami i konsumentami. Koncepcja ta, choć rozszerzona, jest często stosowana w technologiach integracyjnych [SAS 2008].

W celu zmniejszenia obciążenia konsumentów przetwarzaniem nieistotnych wiadomości oraz redukcji zapotrzebowania na przepustowość sieci, wiele technologii pozwala na definiowanie filtrów wiadomości, rozmieszczanych blisko producentów. Przykładem usługi pozwalającej na filtrowanie wiadomości jest CORBA Notification Service[OMG 2004]. Wykorzystuje ona specyficzny język opisu filtrów - Trader Constraint Language (TCL) [OMG 2000-2].

Na rysunku 2 zilustrowano przepływ komunikatów w CORBA Notification Service. Wiadomości przechodzą kilka etapów filtrowania, jako że – poza konsumentami – także producenci oraz administratorzy usługi mają możliwość definiowania ograniczeń odnoszących się do przesyłania wiadomości.



Rys. 2. Przepływ komunikatów w kanałach CORBA Notification Service

Nawet na przykładzie stosunkowo prostej usługi, jaką jest CORBA Notification Service, można zaobserwować, że efekt działania filtra zależy od jego umieszczenia. Umieszczenie filtra na wejściu kanału spowoduje, że usunięte wiadomości nie będą dostarczone do

żadnego z konsumentów, podczas gdy umieszczenie tego samego filtra na wyjściu poskutkuje brakiem doręczania wiadomości tylko jednemu konsumentowi.

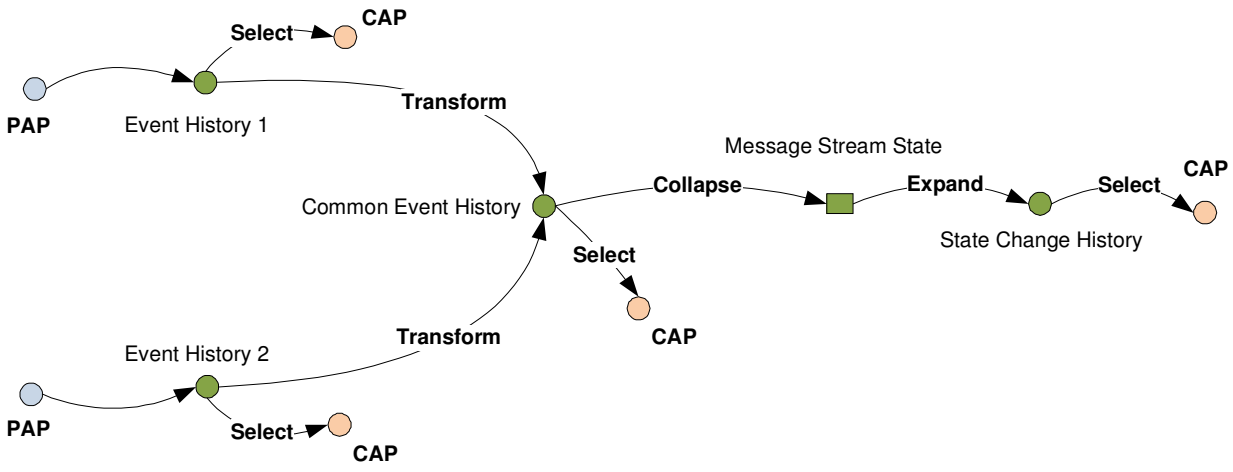
2.1.2. Przetwarzanie grafowe

Grafowy model przetwarzania jest kluczową alternatywą dla modelu kanałowego. Jednym z najbardziej popularnych środowisk wykorzystujących ten model jest IBM WebSphere MQ. Mechanizmy federowania brokerów wiadomości bazują na osiągnięciach projektu Gryphon[Banavar 1999-1]. Podstawową koncepcją opracowaną w ramach tego projektu jest graf przepływu informacji (ang. *information flow graph, IFG*), który składa się z:

- wierzchołków reprezentujących rejestry zdarzeń (ang. *event histories*) lub stany (ang. *event states*) związane z przetwarzaniem wiadomości o zdarzeniach,
- łuków reprezentujących kanały przepływu wiadomości oraz podstawowe operacje na nich wykonywane, a mianowicie:
  - selekcję (ang. *select*),
  - transformację (ang. *transform*),
  - zamianę strumienia informacji na stan węzła (ang. *collapse*),
  - generację strumienia wiadomości w reakcji na zmianę stanu (ang. *expand*).

Wymienione operacje bazują na syntaktycznym opisie wiadomości. Warto przy tym zauważyć, że w IFG przetwarzanie wiadomości reprezentowane jest wyłącznie przez łuki. Stosunkowo krótka lista dozwolonych operacji, w połączeniu z prostym językiem opisu transformacji wiadomości jest wynikiem decyzji projektowej o implementacji mechanizmów automatycznej optymalizacji IFG.

Rysunek 3 ilustruje przykładowy graf przepływu informacji. Poza IFG uwidoczniono na nim miejsca, w których możliwe jest dołączenie producentów i konsumentów.



Rys. 3. Przykład grafu przesyłu informacji (IFG) obsługiwanego przez system Gryphon.
(CAP: *consumer access point* – punkt przyłączenia konsumenta,
PAP: *publisher access point* – punkt przyłączenia producenta)

Grafy przepływu informacji są implementowane na bazie sieci brokerów obsługujących m.in. wybór ścieżek przesyłania bazujący na zawartości wiadomości (ang. content-based routing). Brokery są ponadto odpowiedzialne za obsługę żądań konsumentów

oraz zagwarantowanie podstawowych parametrów jakości usługi (ang. *Quality of Service, QoS*), takich jak eliminacja powtórzonych wiadomości, szeregowanie wiadomości w kolejności wysyłania itp. [Zhao 2001]

Grafowy model przetwarzania, mimo iż trudniejszy w implementacji i utrzymaniu, pozwala na bardziej elastyczne definiowanie aplikacji przetwarzających wiadomości, będących w centrum uwagi rozprawy. Ponadto pozwala na bardzo elastyczne podejście do ponownego wykorzystania pośrednich wyników przetwarzania.

2.1.3. Algorytmy wyboru tras dla wiadomości

Wybór trasy realizowany na podstawie zawartości wiadomości jest koncepcyjnie podobny do trasowania na warstwie sieciowej modelu OSI. Zasady przekazywania wiadomości są podobne do obowiązujących przy realizacji trasowania pakietów skierowanych na adres grupowy [Deering 1990]. Zadaniem brokera (będącego podstawowym elementem systemu przekazywania wiadomości) jest przekazanie wiadomości do właściwych sąsiadów (konsumentów lub innych brokerów). W celu optymalizacji przesyłania wiadomości brokery współpracują ze sobą z pośrednictwem dedykowanych protokołów, które pozwalają także na reakcję na zdarzenia dołączenia lub odłączenia brokerów [Baldoni 2004][Zhou 2006].

Podstawowe rozwiązania w zakresie wyboru tras dla wiadomości obejmują:

- algorytm zalewania,
- zalewanie z prostą filtracją bazującą na subskrypcjach,
- zalewanie z filtracją bazującą na agregowanych subskrypcjach,
- zalewanie z filtracją i agregacją subskrypcji na bazie ogłoszeń emitowanych przez źródła wiadomości.

Algorytm zalewania jest najprostszą z technik zapewniających doręczenie wiadomości do wszystkich zainteresowanych konsumentów. Konsekwencją braku filtracji jest jednak dostarczanie także niepotrzebnych wiadomości.

Wprowadzenie filtracji może skutkować znaczną poprawą efektywności działania sieci [Denning 2006] bez znaczących konsekwencji dla wygody jej użytkowania. Informacje potrzebne do konfiguracji mechanizmów filtracji pozyskiwane są przez system na etapie rejestracji subskrypcji konsumenta. W większości współczesnych systemów przesyłania wiadomości operacja filtracji dokonywana jest na bazie informacji przekazywanych przez konsumenta na etapie rejestracji. Żądanie konsumenta jest propagowane w sieci brokerów zgodnie z algorytmem zalewania ograniczanym ewentualnie przez operację agregacji. Ciekawym rozszerzeniem jest wykorzystanie ogłoszeń źródeł danych rozgłaszanych przez algorytm zalewania w sieci brokerów. Wprowadzenie tej funkcjonalności, przy założeniu, że ilość ogłoszeń źródeł jest istotnie mniejsza od ilości subskrypcji, pozwala na zmniejszenie ilości przekazywanych subskrypcji poprzez wybór tras propagacji w kierunku właściwych źródeł i ograniczenie w ten sposób rozmiarów tablic routingu będących podstawą funkcjonowania brokerów[Carzaniga 2004][Cugola 2002][Mühl 2002]. Ponadto pozwala na informowanie potencjalnych konsumentów o źródłach informacji dostępnych w sieci.

Z uwagi na to, że proponowana platforma ma w założeniu bazować na istniejącej warstwie pośredniej przeznaczonej do organizowania współpracy w sieciach węzłów równorzędnych, funkcja wyboru tras raczej nie będzie implementowana przez platformę.

Zamiast tego, wykorzystany zostanie mechanizm dostarczony przez rozszerzane środowisko. Wydaje się jednak, że propagacja ogłoszeń źródeł wiadomości do potencjalnych konsumentów może zwiększyć komfort korzystania z platformy.

2.2. Reprezentacja i dystrybucja metadanych

Metadane są podstawą przetwarzania w warstwie pośredniej, w szczególności w kontekście:

- semantyki i syntaktyki przekazywanych danych,
- funkcjonowania elementów wewnętrznych warstwy pośredniej.

Z uwagi na cele rozprawy, zaprezentowany w niej przegląd metod reprezentacji metadanych skupiony jest na pierwszym z wymienionych aspektów.

2.2.1. Reprezentacja metadanych opisujących składnię dokumentów

Podstawowym kryterium wyboru metody reprezentacji składni przesyłanych wiadomości jest łatwość integracji ich obsługi z oprogramowaniem przetwarzającym te dokumenty. Współczesne systemy przetwarzania wiadomości opierają się na podejściu opisującym dokumenty za pomocą ich prototypów (vide *Document Object Model, DOM*) lub wyodrębnionych schematów (np. XML Schema).

Document Object Model jest interfejsem programistycznym przeznaczonym do integracji obsługi zawartości i struktury dokumentu z aplikacjami użytkownika. Dokumenty reprezentowane są jako drzewa konstrukcji programowych, jedyną możliwością eksternalizacji struktury dokumentu jest utworzenie jego prototypu.

Podejście oparte na zestawie standardowych języków związanych z eksternalizacją schematów dokumentów XML (XML Schema, XPath, XQuery) pozwala na łatwiejszą integrację aplikacji. W odróżnieniu od DOM, nie definiuje ono interfejsu programisty, choć jest obsługiwane przez dostępne pakiety oprogramowania (np. XMLBeans[McLaughlin 2006], JAXB[Ort 2003]). XML Schema pozwala na definiowanie złożonych struktur danych, dziedziczenie typów, ograniczanie zakresów dozwolonych wartości itp. Standardowe towarzyszące języki pozwalają na wykonywanie prostych (XPath[Clark 1999]), bądź bardziej złożonych (XQuery[Boag 2007]) zapytań.

Proponowana platforma powinna pozwalać na stosowanie wielu technik opisu metadanych, tym niemniej zestaw standardów związany z językiem XML ze względu na elastyczność opisu wydaje się być najważniejszą z nich.

2.2.2. Reprezentacja metadanych opisujących semantykę dokumentów

Znajomość semantyki wiadomości jest kluczowa dla zapewnienia poprawnych rezultatów ich przetwarzania. W wielu systemach rozproszonych wiedza ta jest zawarta w kodzie przetwarzającym. W celu zapewnienia obsługi konceptualnych zapytań przez proponowaną platformę konieczna jest jednak jej eksternalizacja.

Resource Description Framework (RDF) [Klyne 2004] definiuje standardowy język eksternalizacji metadanych dotyczących semantyki. Bazuje on na składni XML oraz typach danych zdefiniowanych przez XML Schema. Zdefiniowano również towarzyszące RDF języki zapytań, takie jak SPARQL (zestandaryzowany przez W3C [Prud'hommeaux 2008]).

Rozszerzeniem możliwości RDF jest *Web Ontology Language* (OWL) [Patel-Schneider 2004]. Wprowadza on m.in. większe możliwości w zakresie definicji typów, możliwość definiowania relacji pomiędzy klasami oraz atrybuty dotyczące liczebności w opisie relacji. Spośród języków zapytań towarzyszących OWL (np. OWL-QL[Fikes 2003], żaden nie został ustandaryzowany). Tym niemniej – jako że dokumenty OWL są zarazem dokumentami RDF – w celu opisywania zapytań można stosować np. SPARQL.

Podobnie, jak w przypadku opisu syntaktycznego, zamiarem autora jest wyposażenie proponowanej platformy w możliwość obsługi wielu standardów opisu semantyki wiadomości. Standardowe techniki – OWL i SPARQL – powinny być jednak w centrum uwagi.

2.3. Wirtualizacja środowiska uruchomieniowego

Miejsca dostarczenia oraz trasy przebyte przez wiadomości są determinowane m.in. przez adresację. Adresy docelowe mogą być wyrażone explicite bądź też zaszyte w ciele wiadomości (w przypadku wyboru schematu adresacji opartego na zawartości przesyłanej wiadomości – ang. *content-based addressing*).

Wprowadzenie schematu adresacji jest częścią procesu wirtualizacji środowiska uruchomieniowego. W celu uproszczenia procesu tworzenia aplikacji rozproszonych, środowiska programistyczne ukrywają złożoność sieci komputerowej oraz mechanizmów w niej pracujących poprzez dostarczenie jej uproszczonej reprezentacji, na bazie której operują programiści.

Tabela 1 prezentuje schematy adresacji wybrane dla elementów proponowanej platformy.

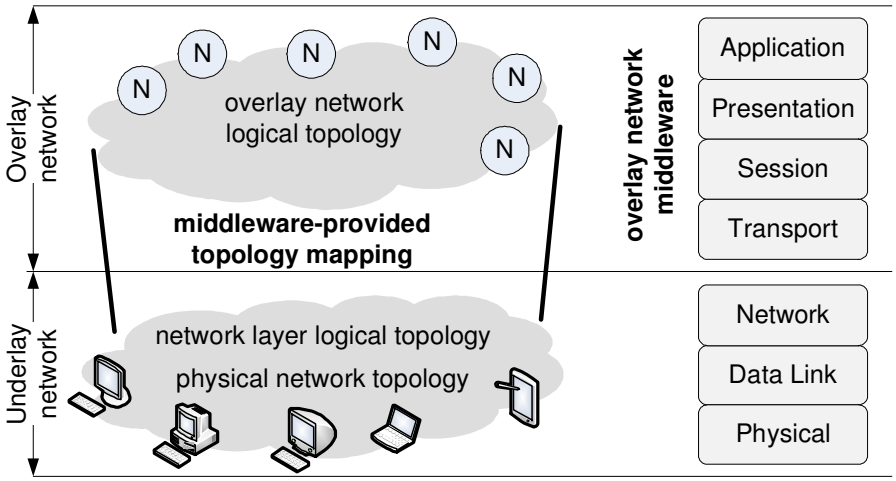
Tabela 1. Zestawienie typów adresów wykorzystywanych przez węzły instancji REMP

element platformy REMP	**typ adresu**
adresy wykorzystywane przez zewnętrzne jednostki	
adres docelowy zapytania	adres dowolnej jednostki (*anycast*), konceptualny, oparty na zawartości lub referencja
adres źródłowy zapytania	adres jednostki, referencja
adres docelowy ogłoszenia źródła	adres dowolnej jednostki, referencja
adres źródła	adres jednostki, referencja
adresy wykorzystywane przez jednostki wewnętrzne	
węzeł przetwarzający	adres jednostki, referencja
redundantne węzły wewnętrzne	adres dowolnej jednostki, referencja
nie-redundantne węzły wewnętrzne	adres jednostki, referencja
adres docelowy wiadomości	adres jednostki lub grupy, referencja
adres źródłowy wiadomości	adres jednostki, referencja

Głównymi rozszerzeniami w stosunku do istniejących systemów są:

- możliwość stosowania wielu, w tym konceptualnych, schematów adresacji dla zapytań konsumentów,
- zastosowanie adresów dowolnej jednostki z grupy równoważnych zarówno przez jednostki zewnętrzne, jak i wewnętrzne.

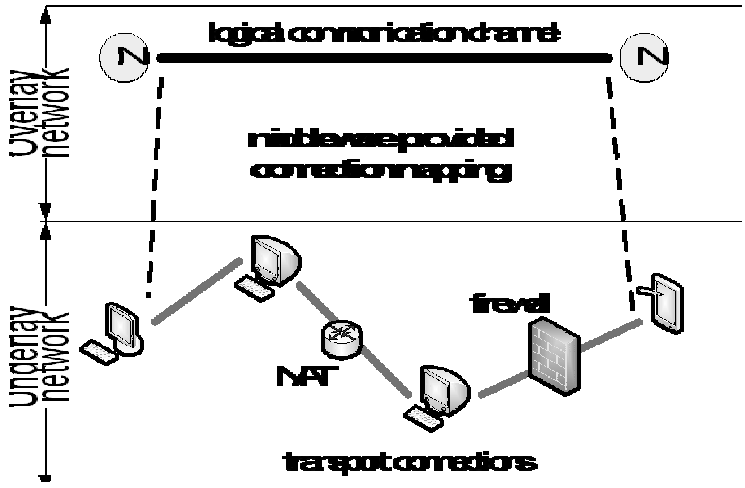
Sieci nakładkowe, których tworzenie jest przedmiotem prezentowanej rozprawy mogą być zdefiniowane jako wirtualne sieci składające się z węzłów i połączeń logicznych, zbudowane na bazie istniejących sieci; celem ich istnienia jest dostarczenie usługi, która nie jest dostępna w istniejących sieciach [Stoica 2003]. Większość wysiłków badawczych ukierunkowana jest na tworzenie sieci dostarczających nowe usługi, działających na bazie Internetu. Z punktu widzenia referencyjnego modelu OSI, sieci te są implementowane w warstwach 4-7 (por. rys. 4).



Rys. 4. Schemat implementacji sieci nakładkowych przez warstwę pośredniczącą (ang. *middleware*)

Podstawowym elementem sieci nakładkowej jest węzeł. Warto zauważyć, że mimo iż pojedynczy komputer może być współdzielony przez wiele węzłów logicznych, pozostają one odosobnione w logicznej topologii sieci nakładkowej. Węzły stanowią środowisko uruchomieniowe dla usług sieciowych oraz aplikacji użytkownika.

Węzły sieci nakładkowej są połączone wirtualnymi kanałami komunikacyjnymi. Celem implementacji takich kanałów jest ukrycie złożoności rzeczywistej sieci komputerowej przed programistą poprzez zapewnienie środków komunikacji pokonujących przeszkody takie jak translatory adresów, czy filtry ruchu (ang. *firewall*) – por. rys. 5. Implementacje wirtualnych kanałów komunikacyjnych mogą korzystać z wielu tzw. połączeń transportowych.



Rys. 5. Implementacja logicznego kanału komunikacyjnego za pomocą wielu połączeń transportowych

2.4. Wymagania niefunkcjonalne

Kluczowymi, z punktu widzenia autora proponowanej platformy, wymaganiami niefunkcjonalnymi są te odnoszące się do możliwości (samo)adaptacji środowiska do zmieniających się warunków w sieci. Podstawowymi wymaganymi cechami platformy będą zatem:

- możliwość introspekcji jej stanu,
- możliwość zmiany sposobu jej funkcjonowania przez jednostkę zewnętrzną (adaptacji),
- możliwość samo adaptacji instancji platformy do typowych zmian w jej środowisku wykonawczym (np. odłączenie węzła obsługiwanej sieci nakładkowej),
- możliwość zwielokrotnienia niektórych węzłów wewnętrznych instancji proponowanej platformy w celu zapewnienia redundancji lub podziału obciążenia,
- zabezpieczenie komunikacji w obsługiwanych sieciach nakładkowych.

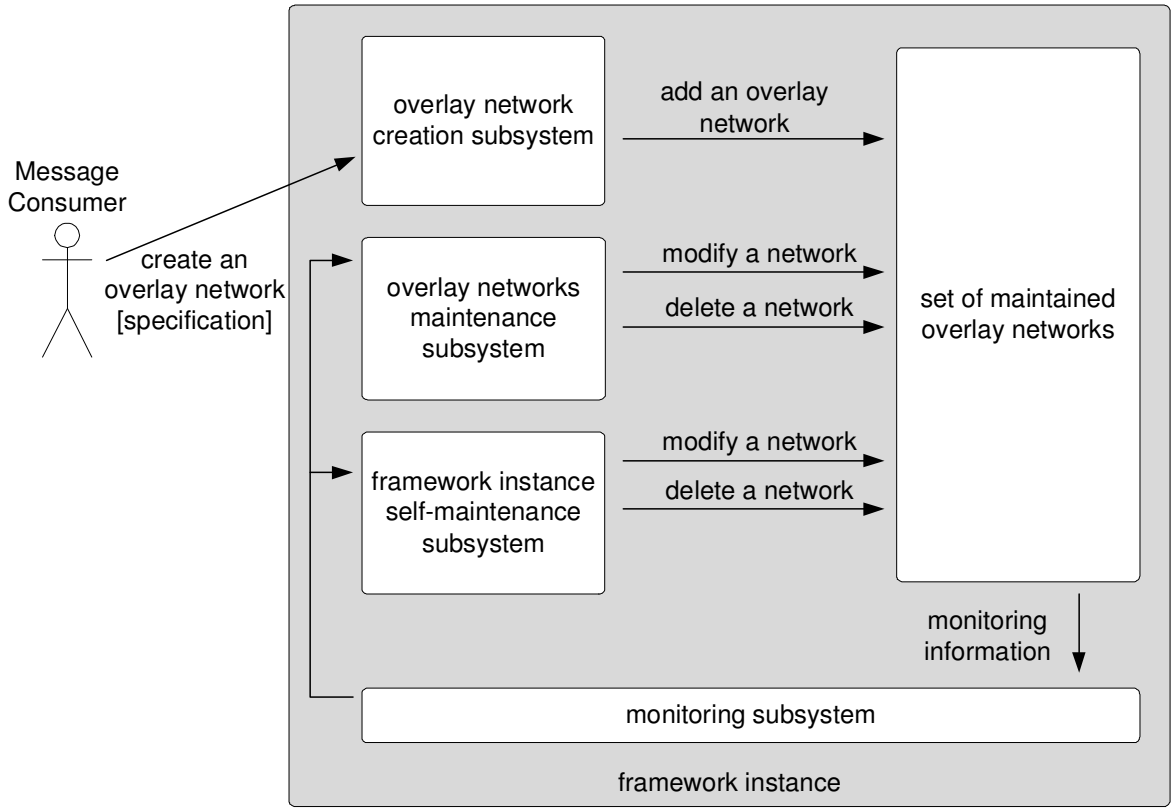
Zagadnienia te będą szerzej omówione w części autoreferatu poświęconej modelowi instancji platformy REMP.

3. Model instancji platformy oraz implementacja prototypu

Rozdział 3 rozprawy prezentuje model systemu przetwarzania wiadomości zbudowanego na bazie platformy REMP. Istotą modelu jest zdekomponowanie platformy na szereg współpracujących usług. Rozdział 3 rozprawy prezentuje proponowany podział funkcjonalności oraz sposoby realizacji podstawowych scenariuszy użytkowania.

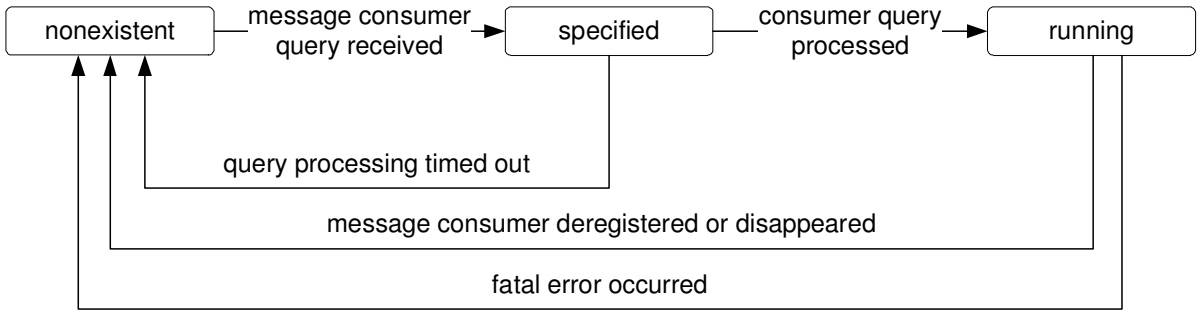
3.1. Podział na podsystemy

Podstawowe podsystemy, z których złożona jest platforma REMP, zaprezentowano na rys. 6.



Rys. 6. Podsystemy tworzące instancję platformy REMP

Żądania konsumentów wiadomości zawierają specyfikację źródeł, węzłów przetwarzających oraz połączeń pomiędzy wszystkimi elementami sieci nakładkowej. Obsłużenie żądania skutkuje zatem utworzeniem takiej sieci i dodaniem jej do zbioru aktualnie obsługiwanych sieci nakładkowych, których rozmieszczenie może ulegać modyfikacjom spowodowanym zmianami we współdzielonym przez wszystkie obsługiwane sieci środowisku wykonawczym. Cykl życia sieci nakładkowej zaprezentowano na rys. 7.



Rys. 7. Cykl życia sieci nakładkowej obsługiwanej przez instancję REMP

Proces tworzenia sieci nakładkowej może być długotrwały, szczególnie jeśli system nie może odnaleźć żądanych źródeł wiadomości - mogą być one zarejestrowane po wyspecyfikowaniu zapytania przez konsumenta, bądź – w skrajnym przypadku – nie być

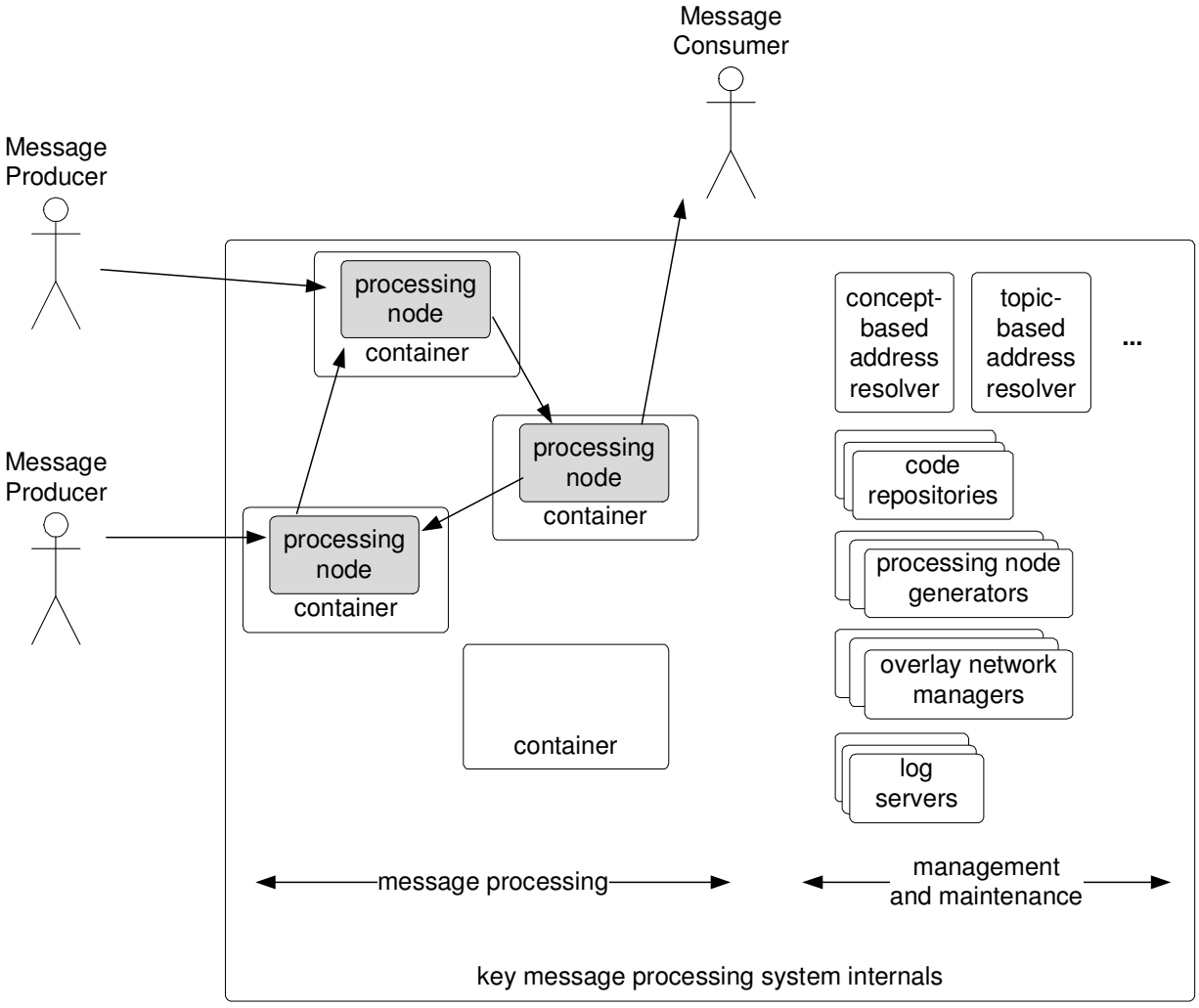
rejestrowane w ogóle. Z tego powodu należy przewidzieć możliwość określenia przez konsumenta maksymalnego czasu przetwarzania jego żądania.

Po rozmieszczeniu elementów sieci i jej uruchomieniu instancja platformy REMP powinna monitorować spójność sieci tzn. reagować na zerwanie logicznych połączeń lub odłączenie węzłów. W wypadku odłączenia konsumenta, który zdefiniował sieć, jest ona – po odczekaniu pewnego czasu – usuwana.

W celu utworzenia sieci nakładkowej instancja platformy REMP musi dysponować przede wszystkim odpowiednimi kontenerami, oraz węzłami realizującymi funkcje:

- rozwiązywania konceptualnych (bądź innych) zapytań o źródła,
- generowania kodu komponentów,
- przechowywania kodu komponentów,
- zarządzania procesami tworzenia i utrzymania sieci nakładkowych,
- przechowywania informacji służących introspekcji.

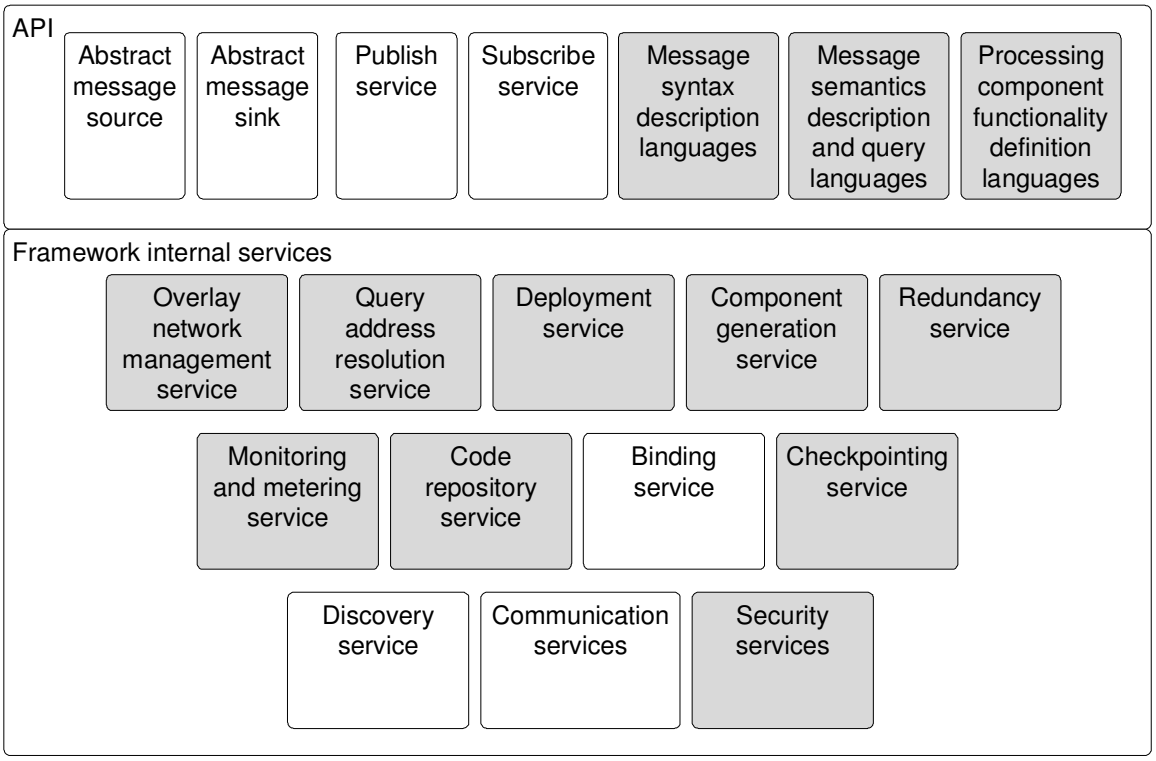
Minimalny zbiór węzłów tworzących instancję platformy REMP przedstawiono na rys. 8.



Rys. 8. Kluczowe elementy infrastruktury systemu przetwarzania wiadomości (instancji REMP)

Z programistycznego punktu widzenia platform REMP jest zbiorem usług dostępnych za pośrednictwem interfejsu programisty (ang. *Application Programmer's Interface, API*).

Rysunek 9 ilustruje elementy tego interfejsu oraz usługi wewnętrzne platformy, niedostępne bezpośrednio, a jedynie za pośrednictwem protokołów wewnętrznych.

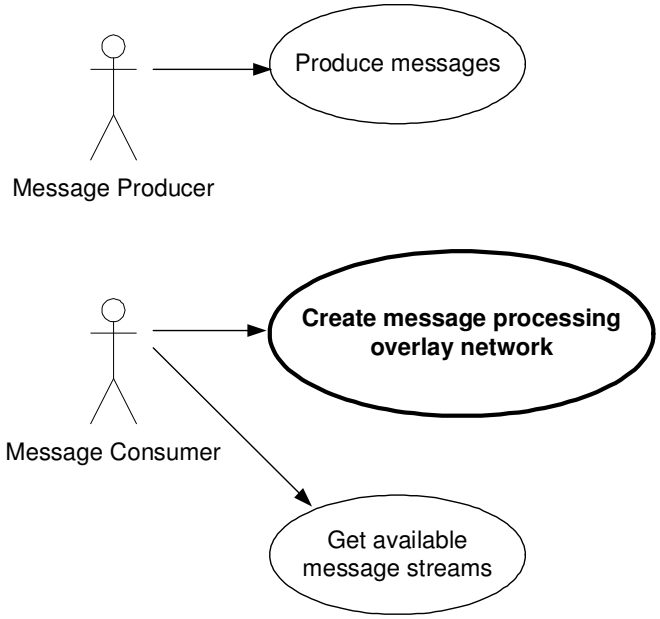


Rys. 9. Dekompozycja platformy REMP na usługi i elementy interfejsu programisty

Kolorem szarym oznaczono elementy, które powinny zapewniać możliwość rozszerzenia przez administratora lub programistę rozbudowującego REMP. Nie należą do nich usługi związane z wyszukiwaniem bądź też łączeniem węzłów, jako że w zamierzeniu mają być one dostarczone przez istniejącą technologię tworzenia sieci węzłów równorzędnych.

3.2. Podstawowe przypadki użycia platformy REMP

Podstawową funkcją oferowaną przez platformę REMP jest możliwość delegacji przetwarzania strumieni wiadomości z maszyny konsumenta na węzły zdalne. Tworzenie sieci nakładkowej jest zatem kluczowym przypadkiem użycia dowolnej instancji platformy. Spośród pozostałych można wyróżnić emitowanie wiadomości (przez producenta) oraz pobranie opisów dostępnych źródeł (przez konsumenta) – por. rys. 10.



Rys. 10. Podstawowe przypadki użycia instancji platformy

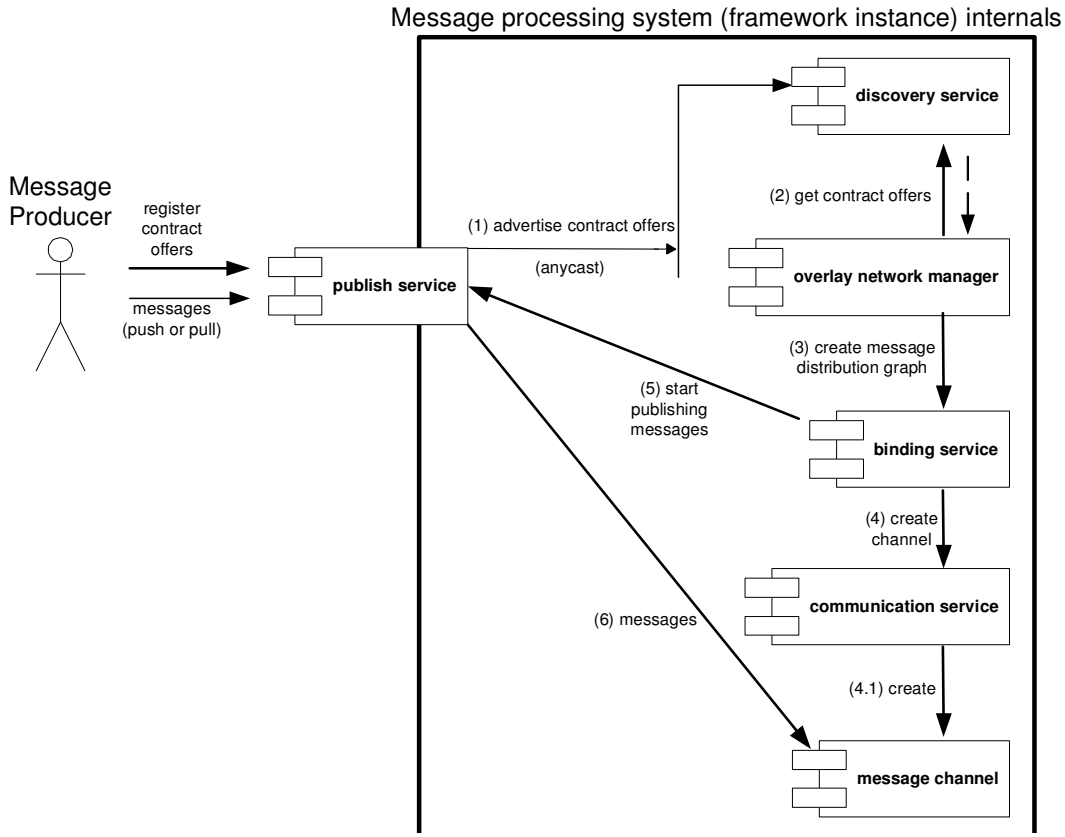
3.2.1. Emitowanie (publikowanie) wiadomości

Rolą producenta wiadomości jest dostarczenie instancji środowiska przynajmniej jednego strumienia wiadomości, jego opisu oraz propozycji kontraktów obejmujących m.in. podstawowe parametry jakości oferowanej usługi (ang. *Quality of Service, QoS*) w formie dedykowanego dokumentu – ogłoszenia. Ogłoszenie powinno oczywiście zawierać opis zawartości publikowanych wiadomości – zarówno semantyczny, jak i syntaktyczny.

Rysunek 11 ilustruje interakcję pomiędzy producentem wiadomości a wewnętrznymi elementami instancji REMP. Ogłoszenie producenta jest propagowane do usługi wyszukiwania. Jeśli istnieje (bądź zostanie zgłoszone później) zapotrzebowanie na usługę danego producenta, węzeł zarządzający tworzeniem i utrzymaniem sieci nakładkowych pobierze to ogłoszenie i przekaże jego elementy usłudze łączenia węzłów sieci nakładkowej w celu utworzenia kanału komunikacyjnego. Usługa ta skomunikuje się następnie z producentem w celu uruchomienia transmisji wiadomości.

Komunikacja z wewnętrznymi elementami systemu jest realizowana w sposób niewidoczny dla programisty aplikacji użytkownika przez instancję usługi publikowania wiadomości, w której rejestruje się producent. Daje to możliwość modyfikacji sposobu funkcjonowania wewnętrznych protokołów platformy bez wpływu na aplikacje użytkowników.

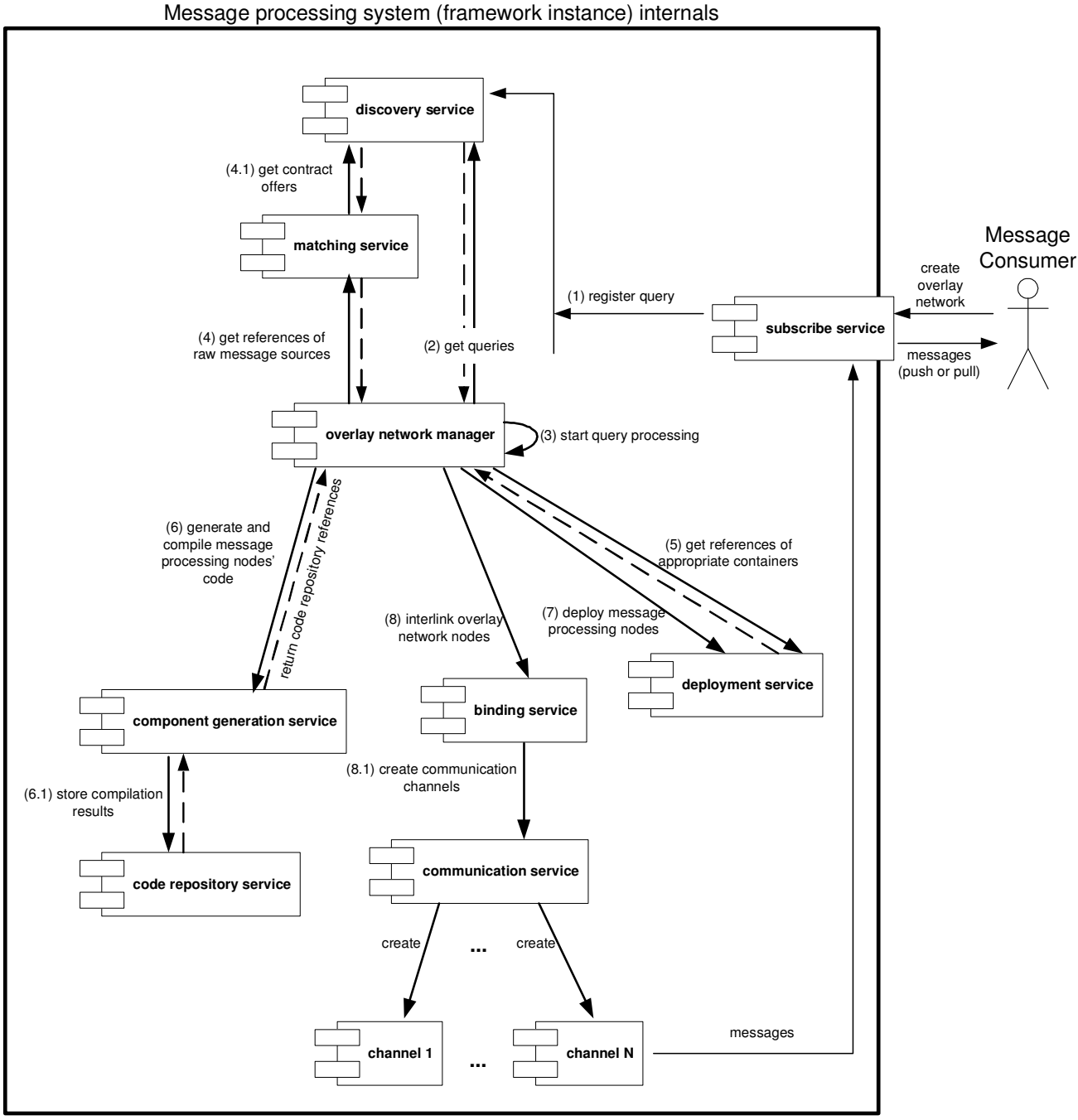
Zamknięcie kanału komunikacyjnego jest realizowane za pomocą podobnej sekwencji zdarzeń, również bez konieczności udziału użytkownika.



Rys. 11. Schemat realizacji publikowania wiadomości (uproszczony)

3.2.2. Tworzenie sieci nakładkowej

Efektem obsługi żądania konsumenta wiadomości jest utworzenie sieci nakładkowej. Rysunek 12 przedstawia (uproszczony) proces tworzenia takiej sieci.



Rys. 12. Uproszczony schemat procesu tworzenia sieci nakładkowej

Żądanie konsumenta jest stosunkowo złożonym dokumentem. Zawiera ono:

- zapytania dotyczące ofert kontraktów publikowanych przez producentów wiadomości,
- specyfikacje transformacji dokonywanych na wiadomościach z poszczególnych strumieni przez węzły przetwarzające,
- specyfikację grafu połączeń pomiędzy węzłami przetwarzającymi.

Podobnie, jak w przypadku producentów, interakcje z elementami wewnętrznymi platformy REMP prowadzi w imieniu aktora dedykowana usługa. Interfejs programisty usługi odbioru wiadomości sprowadza się praktycznie do rejestracji lub derejestracji konsumenta oraz do przetwarzania wiadomości otrzymywanych od stworzonej sieci nakładkowej.

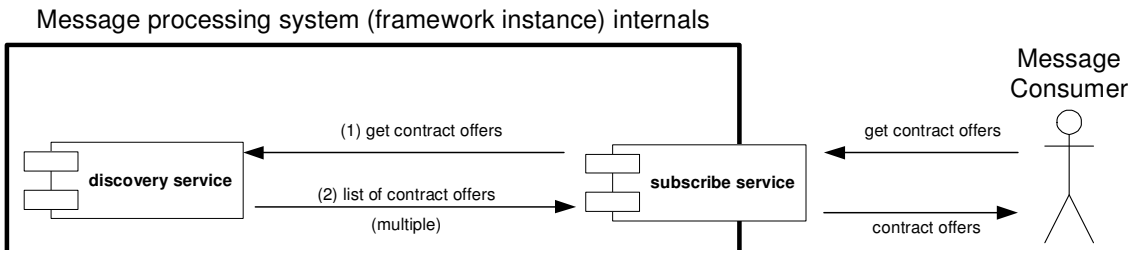
Po otrzymaniu żądania konsumenta (krok (1) na rys. 12), REMP propaguje je do usługi wyszukiwania, z której może je pobrać węzeł zarządzający tworzeniem i utrzymaniem sieci nakładkowych(2). Po zdekomponowaniu zawartości żądania(3), podejmuje on próbę znalezienia odpowiednich źródeł informacji (4) oraz kontenerów dla węzłów przetwarzających (5). Po odnalezieniu potrzebnych zasobów zleca generację i przechowanie kodu komponentów reprezentujących węzły (6). Następnym etapem jest rozmieszczenie węzłów (7) i połączenie ich kanałami komunikacyjnymi (8). Algorytm tworzenia sieci nakładkowej w postaci pseudokodu zaprezentowany jest na listingu 1.

```
loop_forever:
  ConsumerQuerySequence cqseq = DiscoveryService.getConsumerQueries();  (2)
  if( cqseq==null || cqseq.size()==0 ) goto loop_forever;

  foreach( ConsumerQuery cq : cqseq ) {                                 (3)
    MessageSourceSelectOperation[] sso = getSourceSelectOperations(sq);
    MessageSourceAdvertisement[] msadvs =
            MatchingService.findSources(sso);                           (4)

    ProcessingComponentSpecification[] pcspecs =
            getProcessingComponentSpecifications(cq);
    ContainerConfigurationAdvertisement[][] contadvs =
            DeploymentService.findContainersFor(pcspecs);               (5)
    ContainerConfigurationAdvertisement[] selcontainers =
            selectContainersFrom(pcspecs,contadvs);

    CodeReference[] coderefs =
            ComponentGenerationService.createComponents(                (6)
                selcontainers,pcspecs);
    DeploymentService.deployComponents( coderefs,selcontainers );       (7)

    LinkOperations lops = pcspecs.getLinkOperations();
    BindingService.linkNodes( lops,selcontainers );                     (8)

    DiscoveryService.publish( createOverlayNetworkAdvertisement(cq) );
  }

goto loop_forever;
```

Listing 1. Operacje wykonywane przez węzeł zarządzający sieciami w celu utworzenia sieci nakładkowej

3.2.3. Pobranie opisów dostępnych źródeł wiadomości

Część z aplikacji klienckich przed wysłaniem żądania utworzenia sieci nakładkowej może oczekiwać od instancji platformy REMP dostarczenia opisów dostępnych źródeł (np. w celu dokonania introspekcji stanu systemu). Z punktu widzenia REMP jest to najprostsza interakcja, która sprowadza się do propagacji zapytania do usługi wyszukiwania (rys. 13).



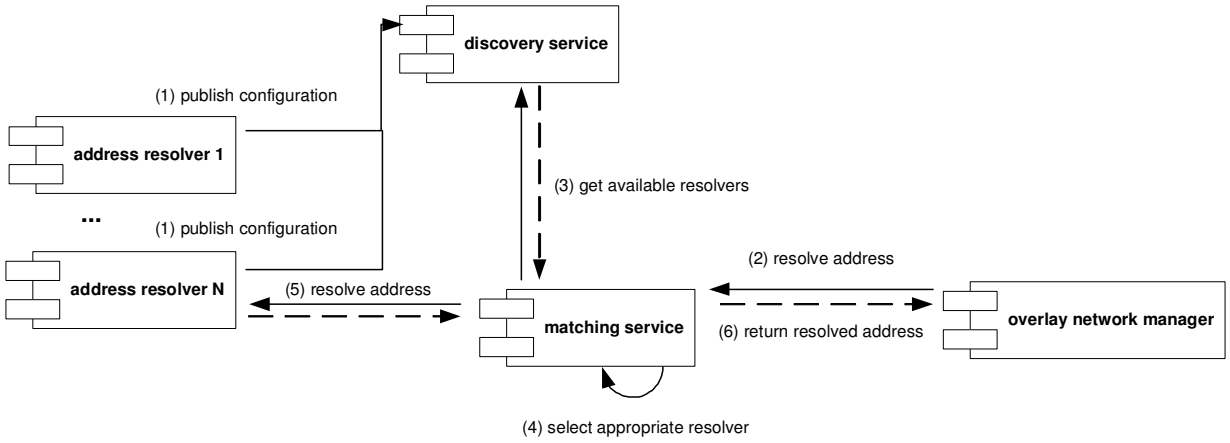
Rys. 13. Schemat procesu pobierania informacji o dostępnych ofertach kontraktów

3.2.4. Funkcjonowanie kluczowych usług wewnętrznych

Niniejszy podrozdział poświęcony jest przedstawieniu sposobu funkcjonowania podstawowych wewnętrznych usług składających się na platformę REMP.

Usługa rozwiązywania zapytań

Usługa ta jest odpowiedzialna na znalezienie ogłoszeń producentów wiadomości odpowiadających zapytaniom konsumentów. Funkcjonowanie usługi schematycznie przedstawiono na rys. 14.

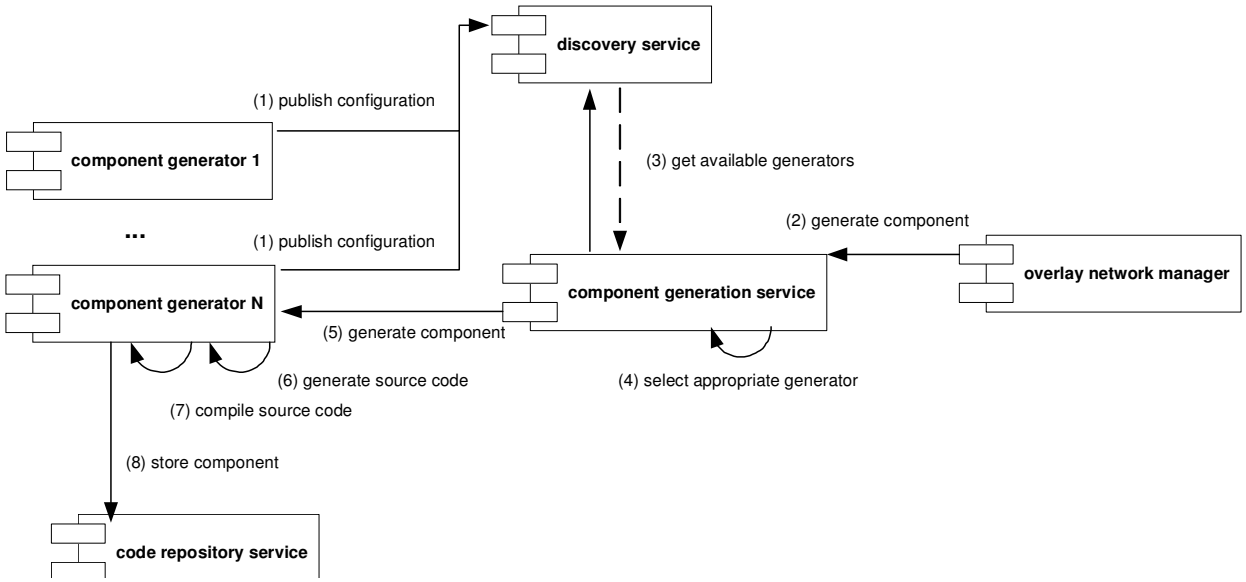


Rys. 14. Uproszczony schemat działania usługi rozwiązywania zapytań

Zapytania trafiają do części interfejsowej usługi, która decyduje, do których spośród węzłów wykonujących odwzorowanie należy je przekazać. Warto zwrócić uwagę, że każdy z węzłów wykonujących dopasowanie może oferować inną funkcjonalność, np. poprzez obsługę innego języka zapytań. Dodawanie i usuwanie węzłów wykonawczych może odbywać się dynamicznie w trakcie działania systemu.

Usługa generacji kodu węzła przetwarzającego

Zasadnicza (z funkcjonalnego punktu widzenia) część kodu realizującego przetwarzanie wiadomości dostarczana jest przez konsumenta w jego zapytaniu. W celu umożliwienia umieszczenia jej w jednym z kontenerów, konieczna jest jednak generacja dodatkowego kodu integrującego komponent ze środowiskiem wykonawczym. Jest to zadanie usługi generacji kodu. Funkcjonowanie tej usługi przedstawiono schematycznie na rys. 15.

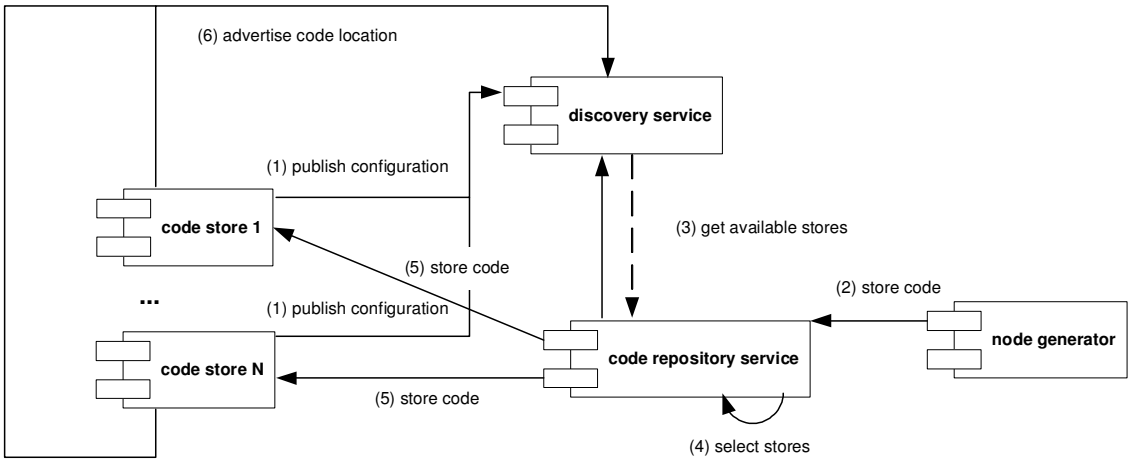


Rys. 15. Schemat procesu generacji kodu komponentu przetwarzającego

Podobnie, jak w przypadku usługi rozwiązywania zapytań, wprowadzono tu podział na część interfejsową i wykonawczą. Węzły wykonawcze mogą różnić się funkcjonalnością, np. obsługiwanym językiem wysokiego poziomu.

Repozytorium kodu

W celu zapewnienia możliwości ponownego wykorzystania, kod wygenerowanego węzła jest zapisywany w odpowiednim repozytorium w kilku kopiach. Proces zapisywania kodu zilustrowano na rys. 16.



Rys. 16. Schemat procesu zapisywania kodu komponentu przetwarzającego w repozytorium kodu

Po zapisaniu kodu komponentu, jego lokalizacja jest ogłaszana za pomocą usługi wyszukiwania.

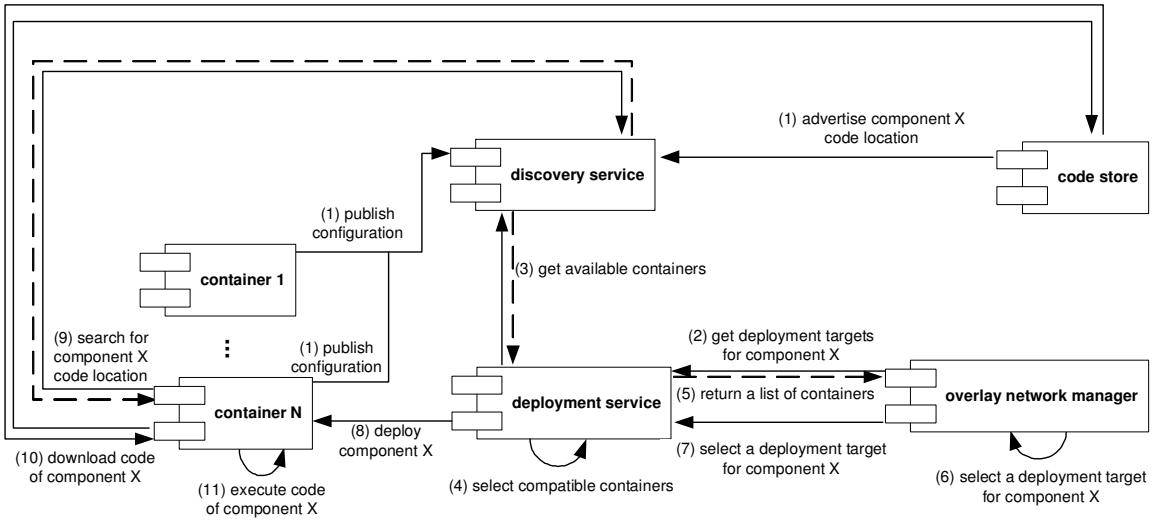
Usługa rozmieszczania węzłów sieci nakładkowej

Zadania usługi rozmieszczania węzłów sieci nakładkowej są następujące:

- dostarczenie węzłowi zarządzającemu rozmieszczeniem sieci informacji nt. możliwych do zrealizowania wariantów rozmieszczenia,

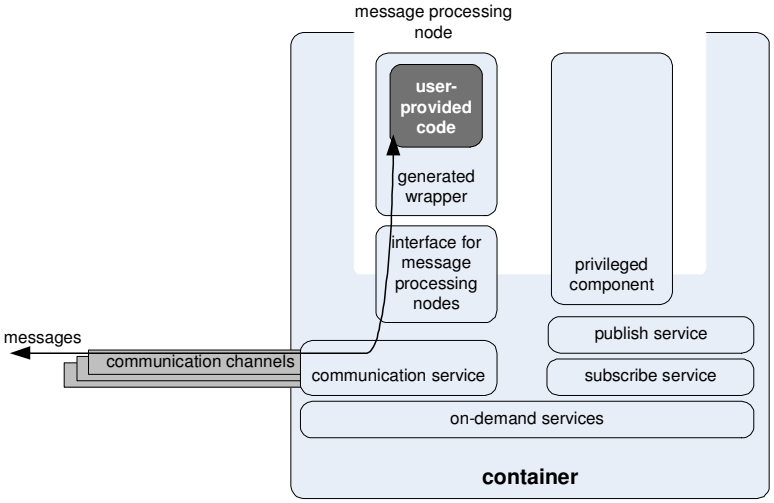
- realizacja wybranego przez zarządcę wariantu.

Usługa rozmieszczania nie podejmuje autonomicznych decyzji, gdyż ma mniejszą wiedzę o stanie całości instancji REMP, niż węzeł zarządzający. Przyjęcie takiego podejścia ułatwia też modyfikację instancji w zakresie strategii rozmieszczania. Funkcjonowanie usługi rozmieszczania węzłów zilustrowano na rys. 17.



Rys. 17. Schemat procesu umieszczenia i uruchomienia komponentu przetwarzającego w kontenerze

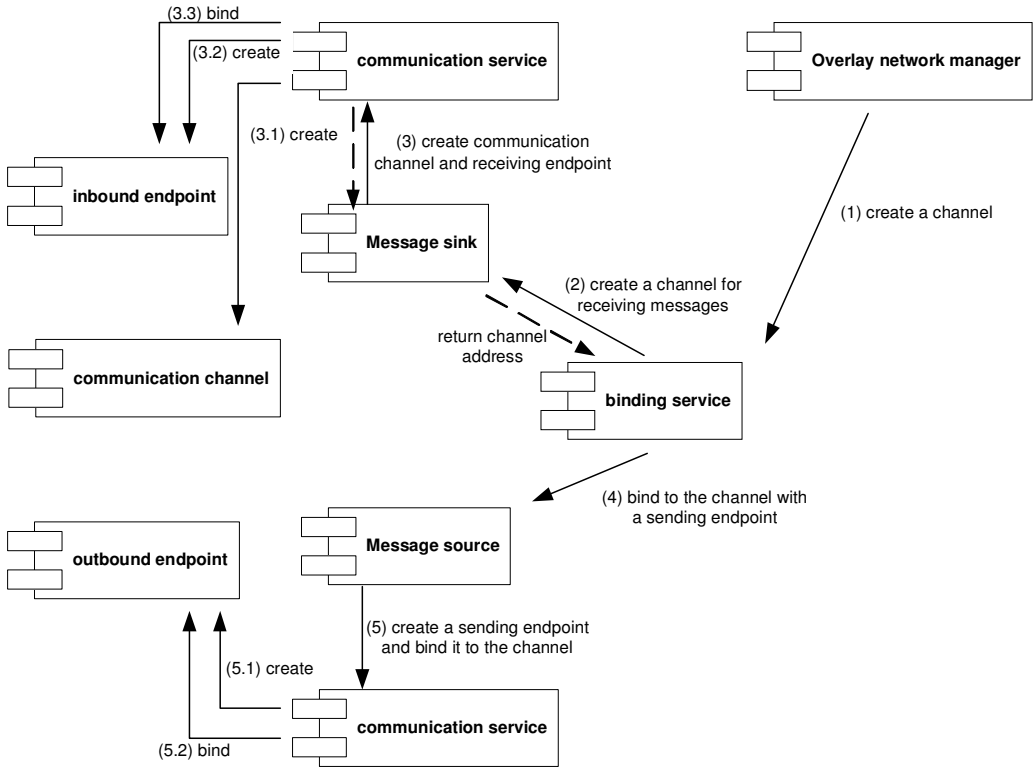
Kontenery zapewniają środowisko uruchomieniowe dla komponentów. Węzły sieci nakładkowych wygenerowanych na żądanie użytkownika mogą korzystać tylko z usług publikacji i odbierania wiadomości i to wyłącznie za pomocą wygenerowanego kodu pośredniczącego. Wydaje się jednak celowe, aby części wykonawcze przynajmniej niektórych z usług wewnętrznych REMP zaimplementować jako komponenty możliwe do uruchomienia w tych samych kontenerach. Komponenty takie powinny mieć możliwość korzystania z wszystkich usług wewnętrznych systemu, łącznie z dynamicznym ładowaniem komponentów do kontenera, w którym się znajdują. Konieczne jest zatem zróżnicowanie sposobu obsługi dwóch typów komponentów – co schematycznie zilustrowano na rys. 18.



Rys. 18. Prosty kontener – węzeł wewnętrzny instancji platformy REMP

Usługa łączenia węzłów sieci nakładkowej

Zadaniem usługi łączenia węzłów jest utworzenie kanałów komunikacyjnych pomiędzy węzłami sieci nakładkowej. W celu uniknięcia konieczności buforowania przez komponenty publikowanych wiadomości, kanały te powinny być tworzone począwszy od konsumenta, aż do producentów wiadomości. Schemat funkcjonowania usługi łączenia węzłów w zakresie tworzenia pojedynczego kanału przedstawiono na rys. 19.



Rys. 19. Schemat procesu utworzenia kanału komunikacyjnego

Warto zauważyć, że usługa łączenia węzłów komunikuje się z elementami reprezentującymi aktorów, tzn. instancjami usług publikowania i odbioru wiadomości.

3.3. Utrzymywanie sieci nakładkowych

Komunikacja w środowiskach węzłów równorzędnych charakteryzuje się większą spontanicznością nawiązywania i rozwiązywania połączeń różnych typów. Środowiska te są tworzone przez węzły dołączane i odłączane w dowolnym czasie, nie mają ponadto wspólnej polityki zarządzania. Stanowią zatem stosunkowo trudne otoczenie dla uruchamiania sieci nakładkowych – szczególnie takich, które są przeznaczone do działania przez stosunkowo długi czas. Instancje platformy REMP powinny zatem posiadać możliwość reakcji na zmiany w środowisku uruchomieniowym, w szczególności te polegające na odłączeniu węzłów (np. kontenerów).

3.3.1. Sprawdzanie funkcjonowania sieci nakładkowych

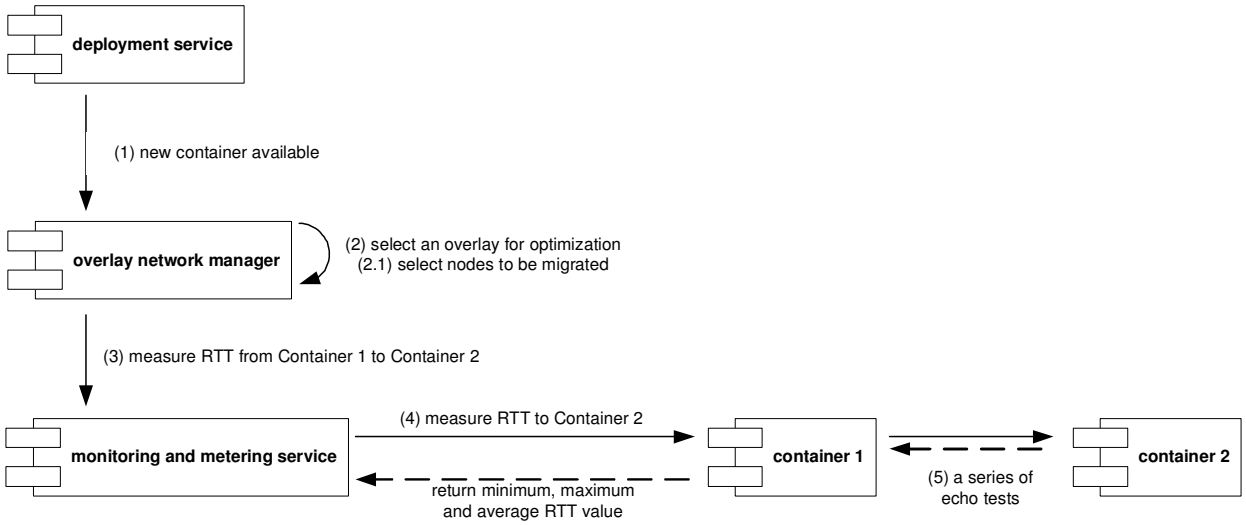
Semantyka przetwarzania dokonywanego przez sieci nakładkowe, ze względu na konieczność zapewnienia użytkownikom poufności, nie powinna być przedmiotem zainteresowania instancji REMP. Utrzymywanie sieci nakładkowej powinno polegać raczej na

pilnowaniu jej spójności, monitorowaniu spełniania warunków oferowanych kontraktów itp. W celu zapewnienia tej funkcjonalności zaproponowano utworzenie usługi monitoringu i dokonywania pomiarów (ang. *monitoring and metering*). Na bazie informacji dostarczanej przez tę usługę węzeł zarządzający sieciami nakładkowymi może wpływać na ich rozmieszczenie lub realizować określoną politykę odnośnie rozmieszczania nowopowstających sieci.

Pomiary dokonywane przez proponowaną usługę powinny obejmować:

- pobieranie wartości kluczowych z punktu widzenia środowiska zmiennych charakteryzujących poszczególne węzły,
- dokonywanie aktywnych pomiarów w trakcie działania środowiska.

Przykład scenariusza wykorzystania proponowanej usługi w zakresie aktywnych pomiarów zilustrowano na rys. 20. W scenariuszu tym zakłada się, że zarządca sieci nakładkowych chce wykorzystać nowo dołączony kontener w celu umieszczenia na nim węzła jednej z obsługiwanych przez niego sieci. W tym celu zleca pomiar czasu propagacji wiadomości pomiędzy nowo dołączonym kontenerem, a jego potencjalnym sąsiadem.



Rys. 20. Prosty przykład zastosowania usługi monitoringu i dokonywania pomiarów

Innym przykładem zastosowania pomiarów aktywnych jest periodyczne sprawdzanie dostępności węzłów sieci nakładkowej dla węzłów je poprzedzających. Na bazie raportów zarządca sieci może podjąć decyzję o realokacji części węzłów z uwagi np. na stosunkowo długie przerwy w komunikacji.

3.3.2. Reakcja na awarie

Awarie instancji platformy REMP mogą dotyczyć bądź to zestawu sieci nakładkowych, bądź też węzłów wewnętrznych – organizujących tworzenie i nadzorujących pracę tych sieci. W przypadku odłączenia części sieci nakładkowej zarządca sieci powinien mieć możliwość dokonania introspekcji stanu systemu oraz odtworzenia fragmentów, które uległy zniszczeniu. Może przy tym bazować na usługach omawianych wcześniej, takich jak repozytorium kodu komponentów implementujących funkcjonalność węzłów przetwarzających lub np. usługi wyszukiwania. Dla zapewnienia obsługi przypadku, gdy komponenty sieci nakładkowej charakteryzują się posiadaniem stanu, konieczne wydaje się dostarczenie usługi przechowywania stanu (ang. *checkpointing*).

Odłączenie konsumenta wiadomości jest – jak już wspomniano – przesłanką do zlikwidowania zdefiniowanej przez niego sieci nakładkowej. Jedyną akcją, jaką może podjąć instancja REMP jest odczekanie określonego czasu i sprawdzenie, czy konsument nie pojawił się ponownie.

Reakcja na odłączenie źródła wiadomości może być trojaka – zignorowanie zdarzenia, usunięcie zasilanych przez nie sieci nakładkowych lub wyszukanie alternatywnego, równoważnego źródła wiadomości. Przesłanką do podjęcia decyzji przez zarządcę powinno być w tym wypadku zapytanie konsumenta specyfikujące sieć – powinien się w nim znaleźć podział na kluczowe i opcjonalne źródła wiadomości.

Odłączenie węzłów wewnętrznych, takich jak np. zarządca sieci nakładkowych, wymaga od instancji działania samonaprawczego, polegającego np. na stworzeniu nowej instancji potrzebnego węzła – przydatna staje się tu koncepcja implementacji węzłów wewnętrznych jako komponentów możliwych do umieszczenia w którymś z dostępnych kontenerów. Alternatywne podejście może polegać na zapewnieniu redundancji w zbiorze kluczowych węzłów i monitorowaniu obecności aktywnej instancji.

Dalsza część tego podrozdziału skrótowo omawia usługi kluczowe dla zapewnienia reakcji na awarie.

Usługa wyszukiwania (ang. discovery service)

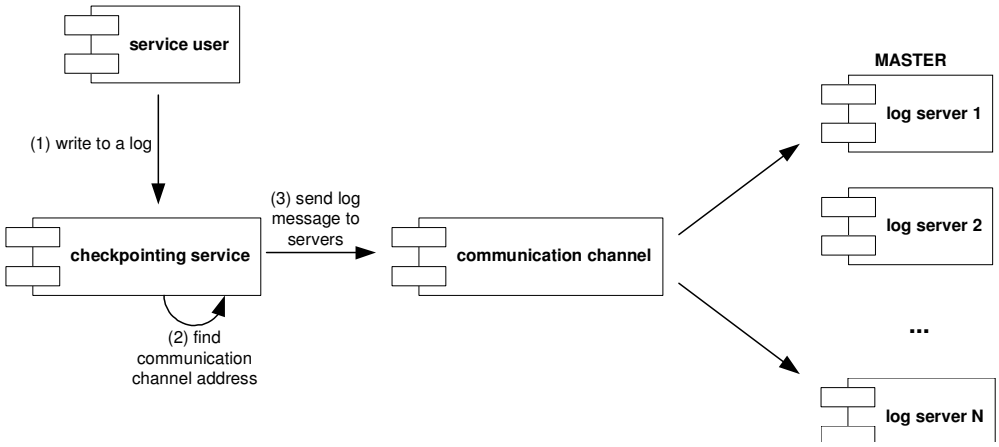
Mechanizmy introspekcji stanu instancji REMP bazują na generycznej usłudze wyszukiwania, akceptującej różnego rodzaju ogłoszenia, replikowane w rozproszonej bazie danych. Usługi takie (operujące np. na rozproszonej tablicy mieszającej – ang. *distributed hash table, DHT*) są popularnym sposobem zapewnienia możliwości ogłoszenia i wykrycia zasobu. Spośród informacji przechowywanych przez usługę wyszukiwania najważniejsze wydają się:

- informacje o istnieniu i konfiguracji węzłów wykonawczych poszczególnych serwisów,
- informacje o ofertach kontraktów pochodzące od producentów wiadomości,
- informacje o zapytaniach konsumentów oczekujących na realizację,
- informacje o kodzie komponentów, przechowywanym w repozytorium kodu,
- informacje o istnieniu, specyfikacji i sposobie rozmieszczenia sieci nakładkowych.

Zawartość bazy danych usługi wyszukiwania jest uzupełniana przez informacje dostępne na poszczególnych węzłach za pośrednictwem usługi monitorowania.

Usługa zapamiętywania stanu (ang. checkpointing)

Odzyskiwanie stanu elementów sieci nakładkowych musi bazować na ich uprzednim zapamiętaniu. W przypadku platformy REMP kluczowe jest ponadto replikowanie takich zapisów tak, by zmniejszyć prawdopodobieństwo utraty wszystkich kopii. Funkcjonowanie usługi zapisu stanu węzłów sieci nakładkowych schematycznie zaprezentowano na rys. 21.

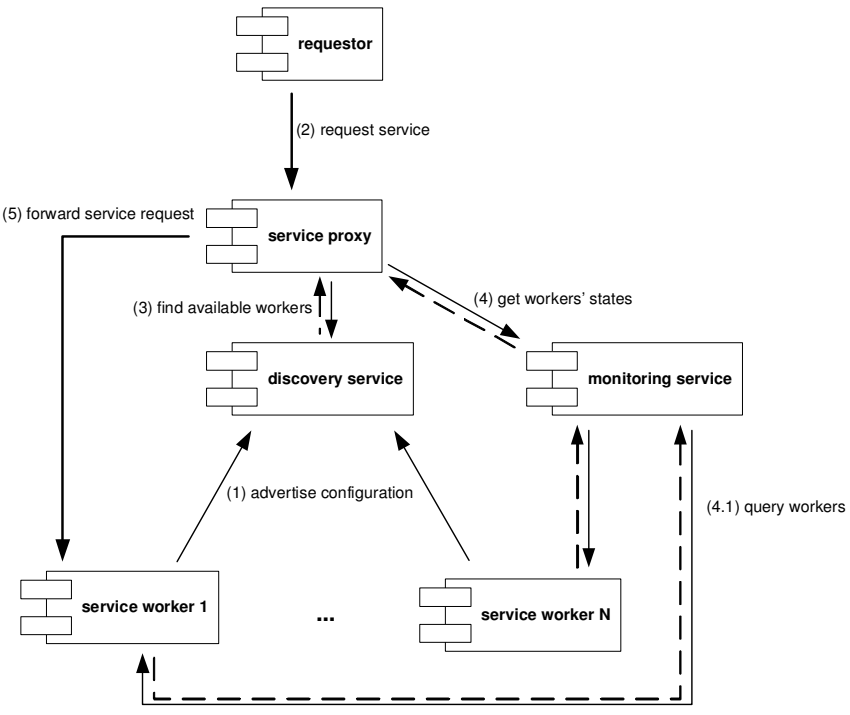


Rys. 21. Schemat funkcjonowania usługi zapamiętywania stanu

Dostępność węzłów pełniących role serwerów logów przechowujących stany węzłów powinna być na bieżąco monitorowana. Wydaje się ponadto, że zasadne byłoby korzystanie z usługi zapewniającej organizację pracy węzłów redundantnych w celu równoważenia obciążenia bądź umożliwienia przejęcia zadań odłączonego węzła przez inny – aktywny.

Usługa koordynacji pracy węzłów redundantnych

Platforma REMP zapewnia dwa sposoby organizacji pracy węzłów wykonujących te same funkcje. Pierwszym z nich – wykorzystywanym przez wiele usług opisanych w rozprawie – jest wzorzec projektowy polegający na wykorzystaniu usługi wyszukiwania do odnalezienia węzłów wykonawczych usługi oraz wyborze jednego z nich na bazie specyficznych dla danej usługi kryteriów odnoszących się do konfiguracji węzła wykonawczego oraz informacji dostępnych za pośrednictwem usługi monitorowania i dokonywania pomiarów. Wzorzec ten (zilustrowany na rys. 22) jest stosowany w przypadkach, gdy zadanie do wykonania ma charakter wsadowy.

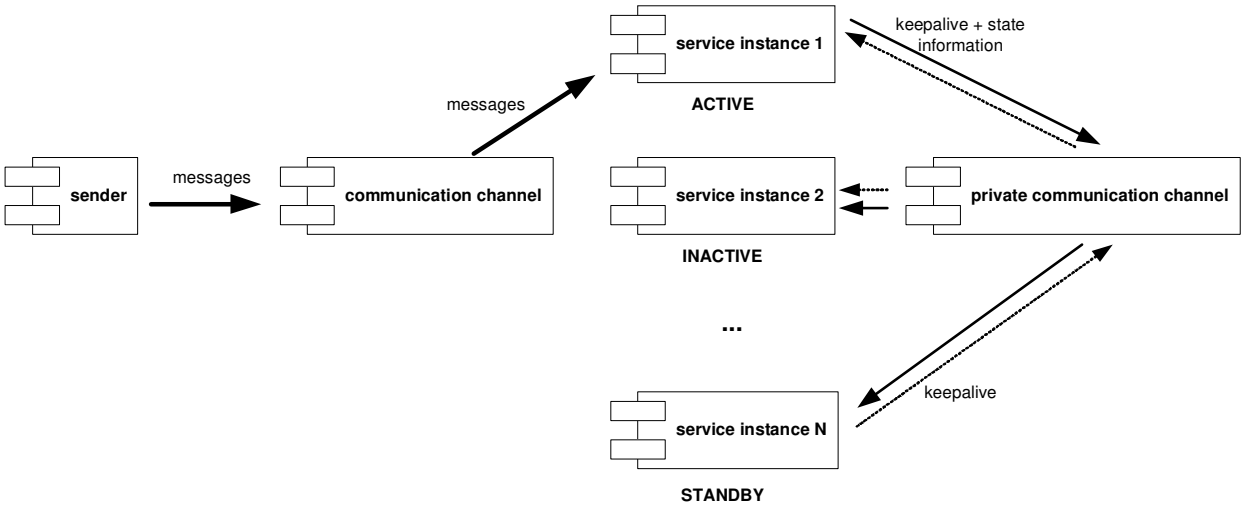


Rys. 22. Schemat procesu wyboru węzła wykonawczego

Ze względu na brak współpracy pomiędzy instancjami reprezentantów usługi (ang. *service proxy*), możliwe jest wystąpienie zjawiska synchronizacji i periodyczne przeciążanie niektórych węzłów wykonawczych. Z tego względu wzorzec ten jest stosowany w usługach wywoływanych stosunkowo rzadko.

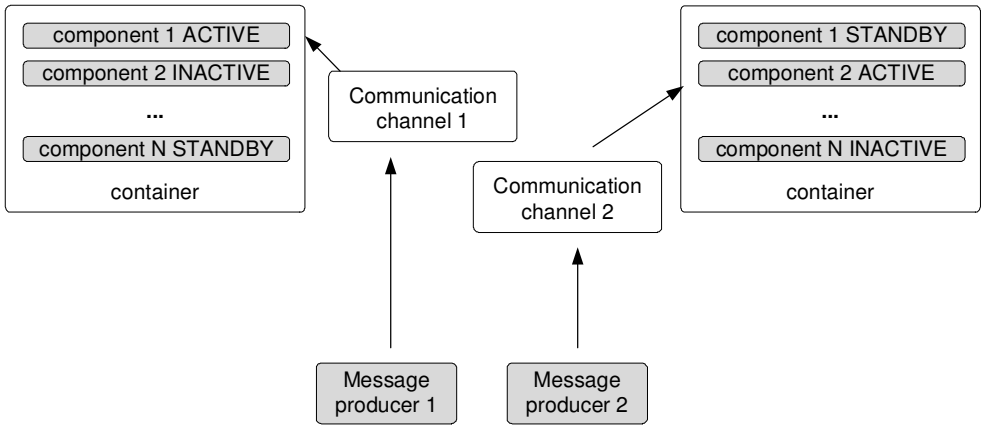
Przykładem kluczowej klasy węzłów, dla których wprowadzenie redundancji wydaje się konieczne, są zarządcy sieci nakładkowych. W celu koordynacji podejmowanych przez nie działań korzystają one z usługi pozwalającej na synchronizację stanów i koordynację pracy węzłów. Z implementacyjnego punktu widzenia usługa taka może przypominać mechanizmy znane z „tradycyjnych" sieci IP, takie jak *Hot Standby Routing Protocol*, *Gateway Load Balancing Protocol*, czy *Virtual Router Redundancy Protocol*.

Z uwagi na to, że rozmiar reprezentacji stanu węzła jest zazwyczaj mniejszy od rozmiaru wiadomości wprowadzających węzeł w dany stan, przyjęto, że strumień wiadomości nie będzie replikowany – zamiast tego synchronizacja stanu będzie dokonywana za pomocą dedykowanego połączenia pomiędzy węzłami (rys. 23).



Rys. 23. Współpraca pomiędzy redundantnymi instancjami usług

Model REMP dopuszcza także replikację węzłów sieci nakładkowych – por. rys. 24. Warto przy tym dodać, że replikacja taka jest o wiele trudniejsza w realizacji z uwagi na przyjęte założenie o braku bezpośredniej współpracy komponentów reprezentujących węzły z wewnętrznymi usługami instancji platformy.



Rys. 24. Schemat prostego równoważenia obciążenia pomiędzy kontenerami zawierającymi redundantne instancje komponentów

3.4. Implementacja prototypu platformy REMP

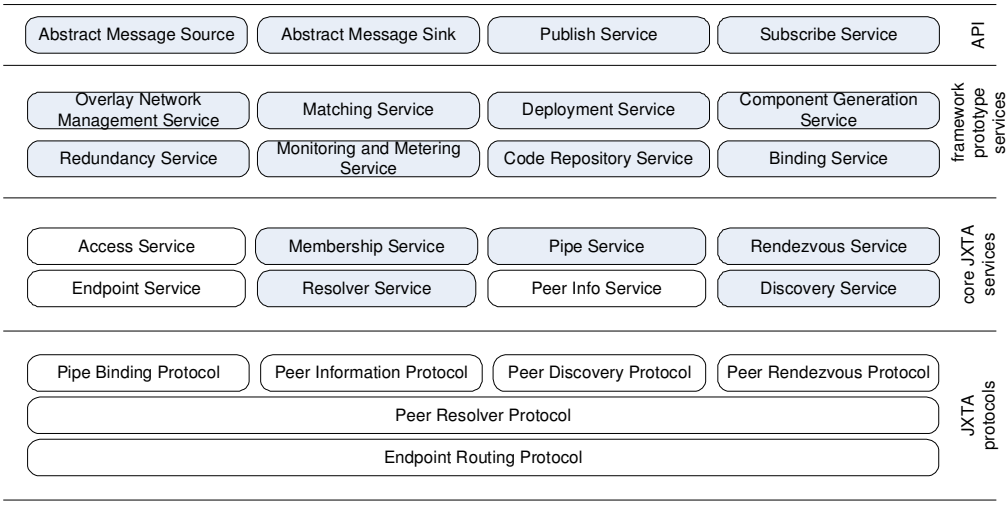
Najważniejsze z technologii wykorzystanych do konstrukcji prototypu platformy to:

- JXTA – platforma konstrukcji sieci węzłów równorzędnych,
- IBM Ontology Development Toolkit – w celu implementacji węzłów rozwiązujących zapytania konceptualne sformułowane w języku SPARQL,
- XML Beans – pakiet wspomagający obsługę dokumentów XML.

Platforma JXTA dostarcza implementacje podstawowych usług i protokołów organizujących współpracę w sieciach węzłów równorzędnych. Najważniejsze z nich, z punktu widzenia implementacji REMP, to:

- usługa i protokół wyszukiwania zasobów (*discovery service*),
- usługi komunikacyjne (*pipe service* oraz *resolver service*),
- dynamiczne ładowanie modułów (funkcja referencyjnej implementacji platformy opartej na Java 2).

Rysunek 25 przedstawia architekturę implementacji prototypu platformy. Kolorem szarym oznaczono usługi wyspecyfikowane w opisie modelu.

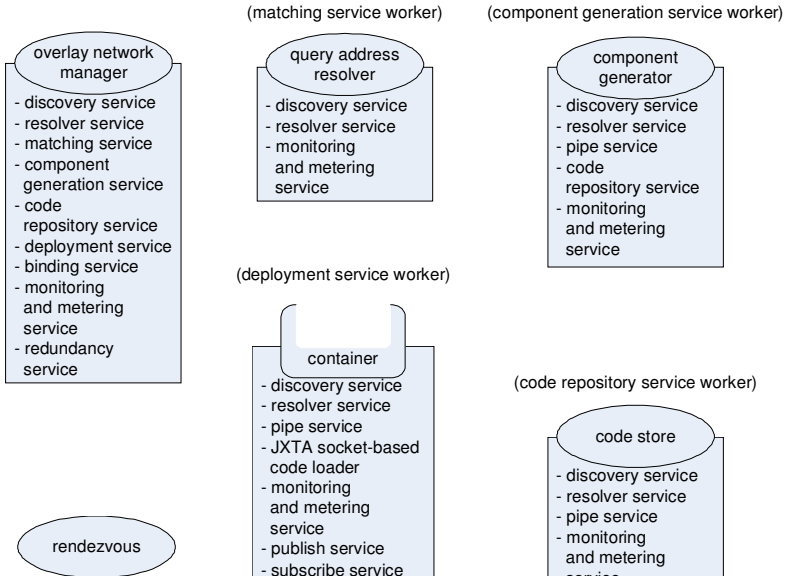


Rys. 25. Architektura prototypowej implementacji platformy REMP

Minimalna instancja prototypu platformy REMP składa się z sześciu węzłów (por. rys. 26). W celu umożliwienia funkcjonowania platformy konieczne jest bowiem uruchomienie:

- zarządcy sieci nakładkowych,
- węzła rozwiązującego zapytania konsumentów,
- węzła generującego kod komponentów,
- węzła przechowującego kod komponentów,
- kontenera.

Dodatkowy węzeł pełni funkcję *rendezvous* – wyróżnionego węzła instancji platformy JXTA.



Rys. 26. Minimalny zestaw węzłów zapewniających poprawne funkcjonowanie instancji prototypu

3.5. Podsumowanie

Niniejszy rozdział autoreferatu poświęcony został streszczeniu zawartości rozdziałów 3 i 4 rozprawy, opisujących odpowiednio model i implementację prototypu platformy REMP. Zaprezentowano jedynie najważniejsze fragmenty tych rozdziałów. W szczególności w celu dokładniejszego zapoznania się z implementacją prototypu platformy (protokołów, formatów wiadomości, itp.), konieczne jest sięgnięcie do pełnego tekstu rozprawy.

4. Przykładowe aplikacje prototypu platformy

W celu zilustrowania funkcjonalności prototypu, w rozdziale 5 rozprawy zaprezentowano trzy przykładowe scenariusze jego wykorzystania, z których jeden zaimplementowano i poddano pomiarom odnoszącym się do zapotrzebowania na przepustowość, pamięć operacyjną i moc obliczeniową wykorzystywanych maszyn.

Zaprezentowane scenariusze to:

- system informatycznej obsługi biegu maratońskiego (zaimplementowany),
- system raportowania informacji giełdowych,
- system wspomagania organizacji akcji ratunkowej.

Niniejszy rozdział autoreferatu poświęcony jest skrótowemu przedstawieniu pierwszej z wymienionych aplikacji.

4.1. Model systemu informatycznej obsługi biegu maratońskiego

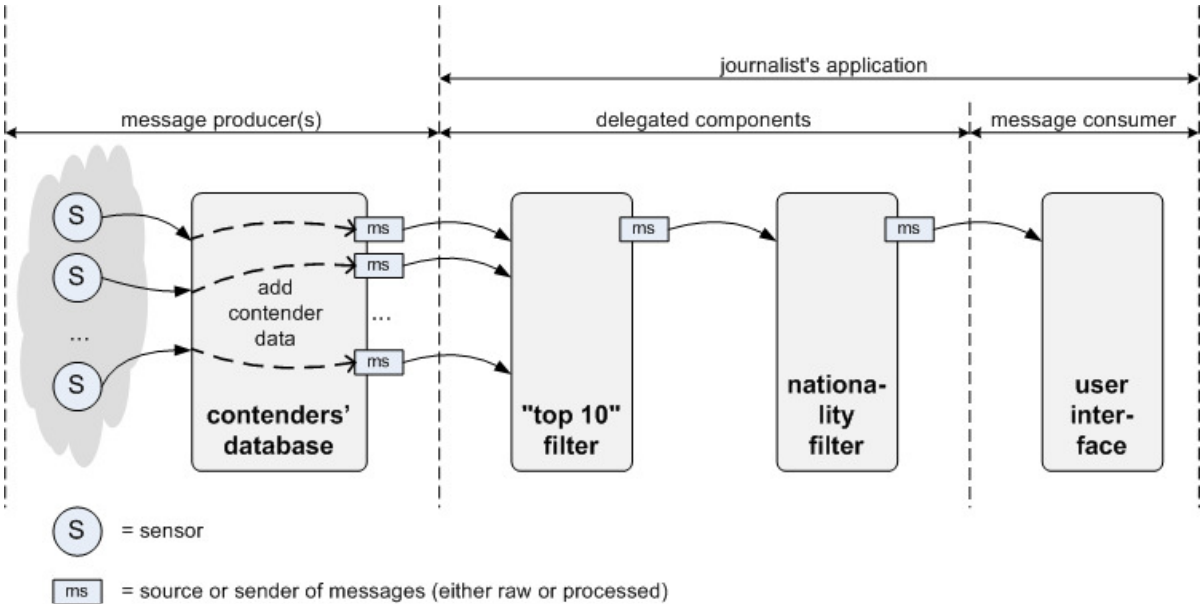
Typowe usługi oferujące śledzenie wyników zawodów sportowych „na żywo”, przekazują użytkownikom szczegółowe dane w odstępach wynoszących 45-90 sekund.

Większość spośród tych informacji jest zbędna dla użytkownika końcowego i mogłaby nie być przesyłana. Znakomitą ilustracją takiej konkurencji sportowej jest bieg maratoński. W szczególnym przypadku, użytkownik zainteresowany śledzeniem wyników członka rodziny uczestniczącego w biegu razem z 20 tysiącami innych osób, których rezultaty są raportowane przez 20 sensorów rozmieszczonych na trasie biegu, może otrzymać 400 000 wiadomości, spośród których jedynie 20 (0,005%) będzie dla niego istotnych.

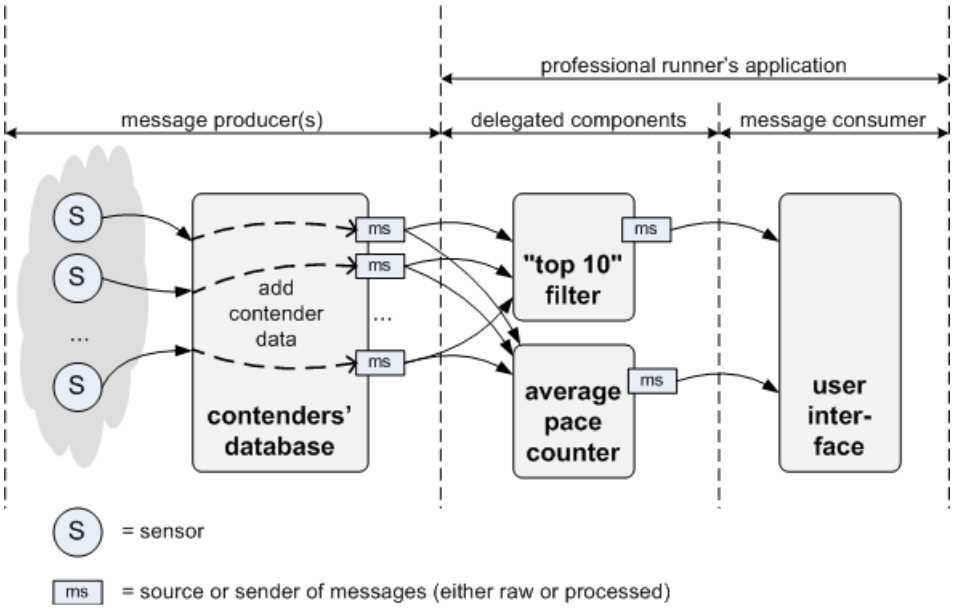
Implementacja śledzenia rezultatów pojedynczego uczestnika jest stosunkowo prosta i nie pozwala na zademonstrowanie funkcjonalności platformy REMP. Z tego powodu zaproponowano również trzy dodatkowe aplikacje:

- aplikację dla dziennikarza zainteresowanego, czy któryś z reprezentantów jego kraju osiągnął wynik kwalifikujący go do czołowej dziesiątki w jakiejkolwiek klasyfikacji,
- aplikację dla zawodnika zainteresowanego śledzeniem wyników czołowych dziesiątek we wszystkich klasyfikacjach oraz przeciętnego tempa biegu pierwszych 10, 100 i 1000 zawodników,
- aplikację dla trenera zainteresowanego wynikami zawodników reprezentujących jego klub.

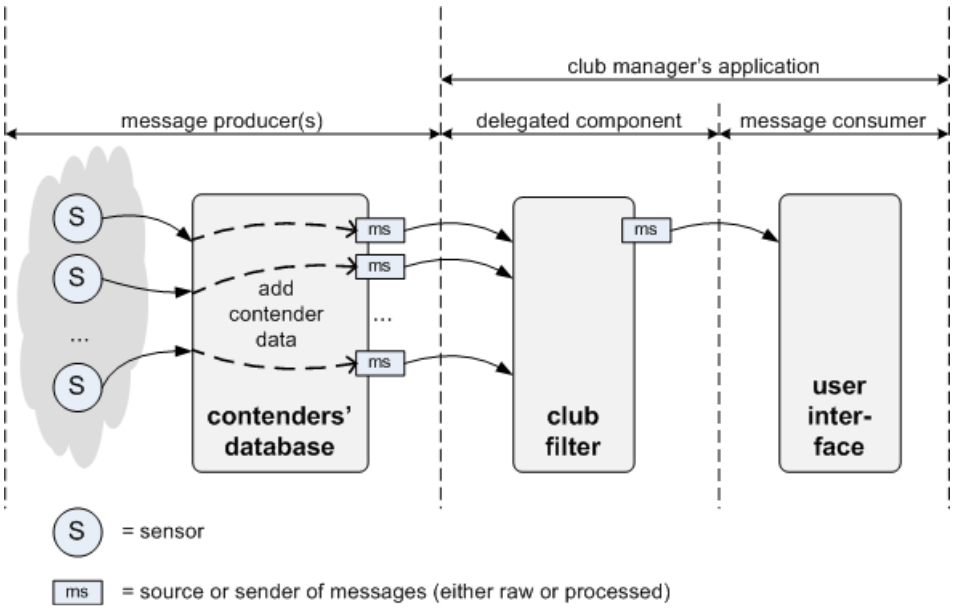
Rysunki 27-30 ilustrują komponenty wchodzące w skład poszczególnych aplikacji.



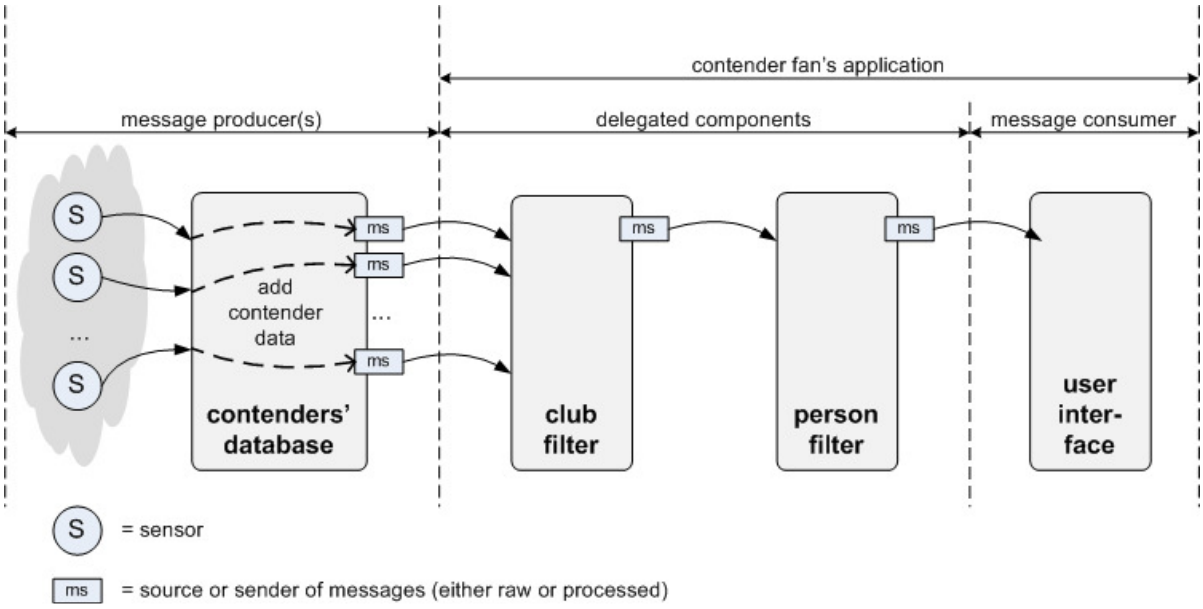
Rys. 27. Schemat budowy aplikacji obsługującej dziennikarzy



Rys. 28. Schemat budowy aplikacji obsługującej zawodników

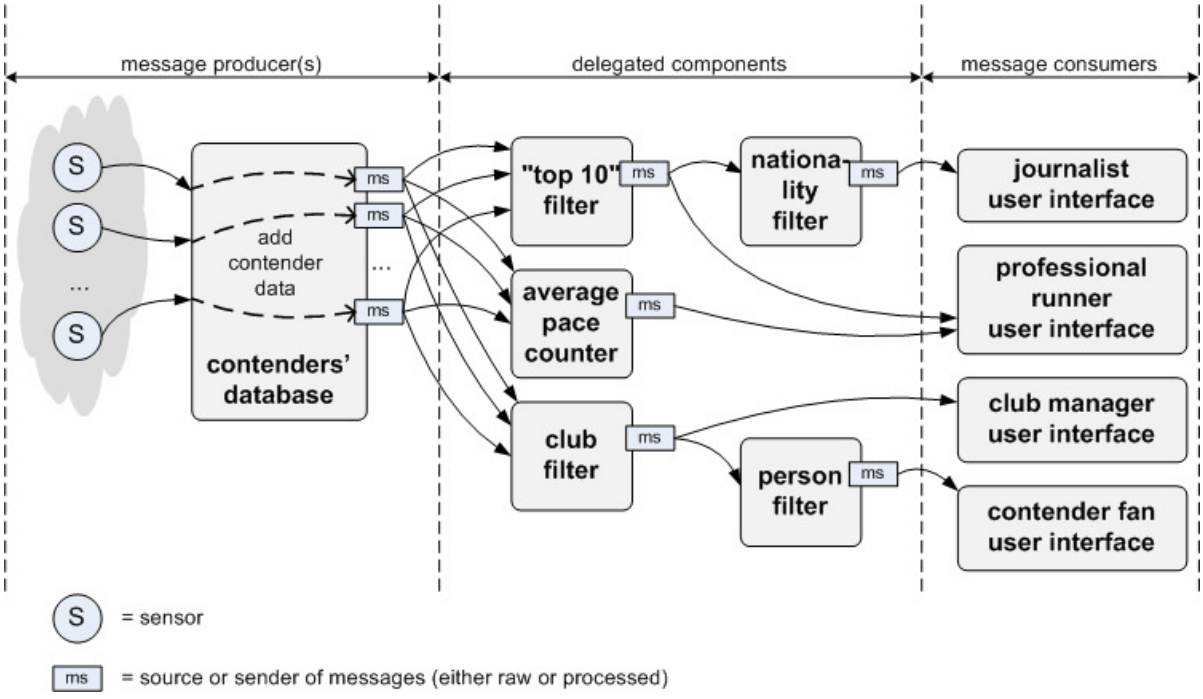


Rys. 29. Schemat apliakacji obsługującej trenera



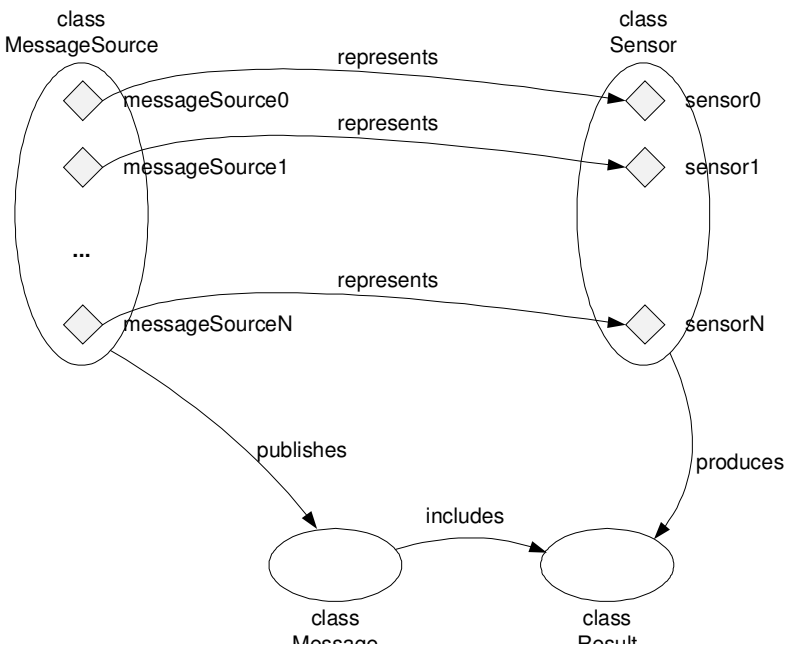
Rys. 30. Schemat aplikacji obsługującej użytkownika śledzącego wyniki pojedynczego zawodnika

Platforma REMP pozwala na ponowne wykorzystanie pośrednich wyników obliczeń. Z tego względu, aby zaprezentować zmniejszenie sumarycznego zapotrzebowania na zasoby, implementacji aplikacji użytkowników dokonano tak, by maksymalnie wykorzystać przetwarzanie dokonywane w poszczególnych komponentach. Złożenie sieci nakładkowych specyfikowanych przez poszczególne aplikacje zilustrowano na rys. 31.



Rys. 31. Złożenie testowych sieci nakładkowych

Semantyczny opis wiadomości publikowanych przez źródła jest stosunkowo prosty. Pojedyncze źródło wiadomości reprezentuje sensor produkujący rezultaty, umieszczane w wiadomościach – por. rys. 32.



Rys. 32. Ontologie publikowane przez źródła wiadomości

Składnia przesyłanych wiadomości odpowiadała schematowi zaprezentowanemu na listingu 2. Rozmiar pojedynczej przesyłanej wiadomości wynosił około 750B, narzuty wnoszone przez JXTA zwiększały go do około 2kB.

Listing 2. Opis składni pojedynczej wiadomości publikowanej przez źródło

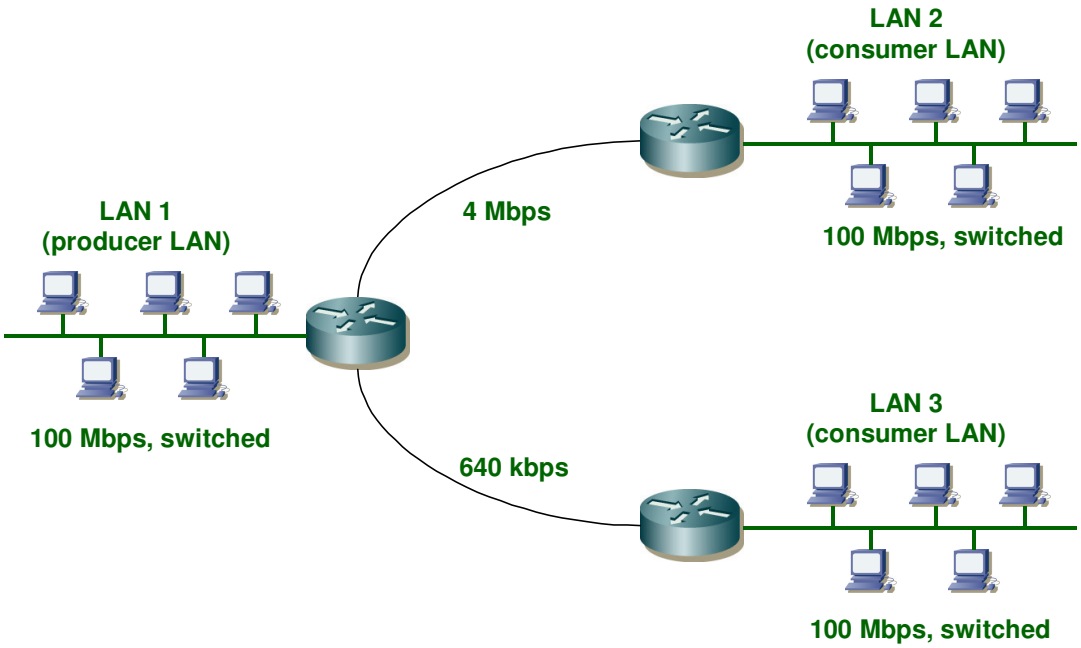
```
<xs:complexType name="ContenderResult">
  <xs:sequence>
    <xs:element name="startNumber" type="xs:int"/>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="surname" type="xs:string"/>
    <xs:element name="club" type="xs:string"/>
    <xs:element name="nationality" type="xs:string"/>
    <xs:element name="MW" type="marathon:ManWoman"/>
    <xs:element name="category">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="TM"/>
          <xs:enumeration value="TF"/>
          ...
          <xs:enumeration value="MM80"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="distanceInMeters">
      <xs:simpleType>
        <xs:restriction base="xs:int">
          <xs:minInclusive value="0"/>
          <xs:maxInclusive value="42200"/> <!-- should be 42195 -->
        </xs:restriction>
```

```
      </xs:simpleType>
    </xs:element>
    <xs:element name="position" type="xs:int"/>
    <xs:element name="timeInMillis" type="xs:long"/>
    <xs:element name="realTimeInMillis" type="xs:long"/>
  </xs:sequence>
</xs:complexType>
```

Implementacja poszczególnych elementów aplikacji jest bardziej szczegółowo opisana w rozdziale 5 rozprawy.

4.2. *Środowisko testowe*

Prototyp platformy REMP został poddany testom w sieci laboratoryjnej składającej się z 20 identycznych sprzętowo i systemowo komputerów PC podzielonych na trzy odrębne sieci lokalne połączone za pośrednictwem trzech routerów i dwóch łączy WAN (rys. 33).



Rys. 33. Środowisko testowe

Tabela 2 zawiera specyfikację najważniejszych parametrów wykorzystywanych komputerów.

Tabela 2. Sprzętowa I systemowa konfiguracja komputerów tworzących środowisko testowe

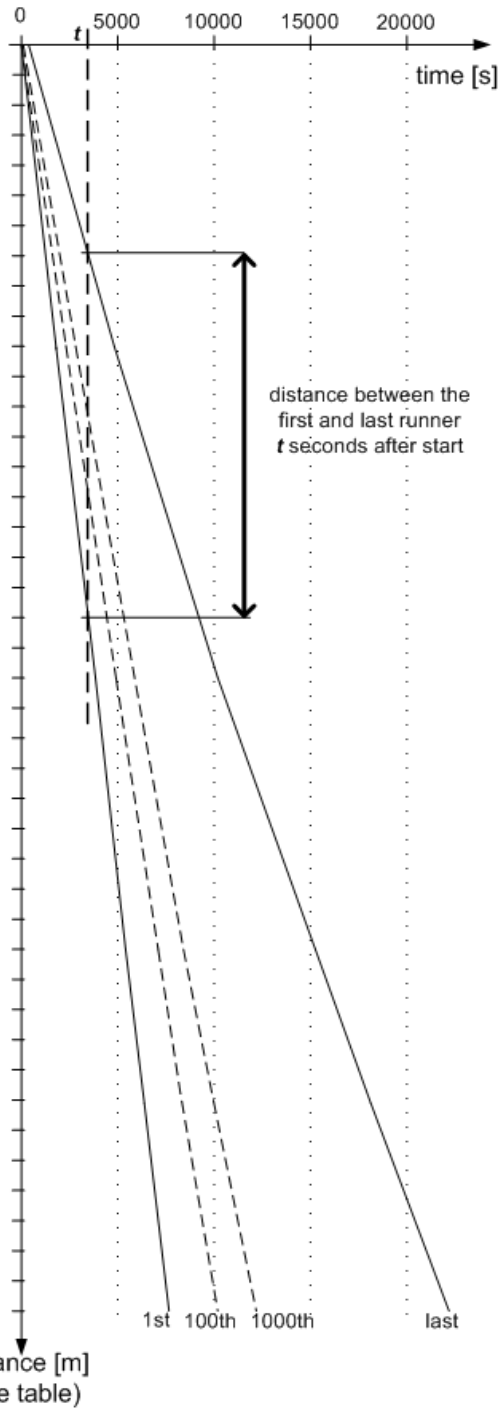
Element	**Opis**
Konfiguracja sprzętowa	
procesor	Intel® Celeron® 2.8GHz, 256 kB L2 cache
płyta główna, chipset	ASUS P5-P800, Intel® 865PE

RAM	1GB DDR-400
HDD	Samsung SP0812N, 80GB, ATA-133, 7200 rpm, 8MB cache
karta sieciowa	Marvell Technology Group Ltd. 88E8001 Gigabit Ethernet
Konfiguracja programowa	
System operacyjny	Fedora Core 6 Linux, kernel 2.6.18
Java VM	Sun JDK 1.6.0_01-b06
JXTA	2.4.1
XML Beans	2.4.0
IODT Minerva	1.1.2
REMP	0.6

Jako dane dla potrzeb testów wykorzystano rezultaty Milano City Marathon 2008. Ze względu na to, że dostępne w Internecie dane [MCM 2008] obejmują pomiary jedynie z 6 sensorów, dokonano ich przedziałami liniowej interpolacji w celu symulacji działania 43 sensorów. Ponadto zasymulowano udział 5000 uczestników (w rzeczywistych danych figuruje 4098) i przyjęto, że wszyscy z nich ukończyli bieg. Tempo biegu ostatniego uczestnika zostało ustalone na dopuszczalne przez organizatorów biegu maksimum.

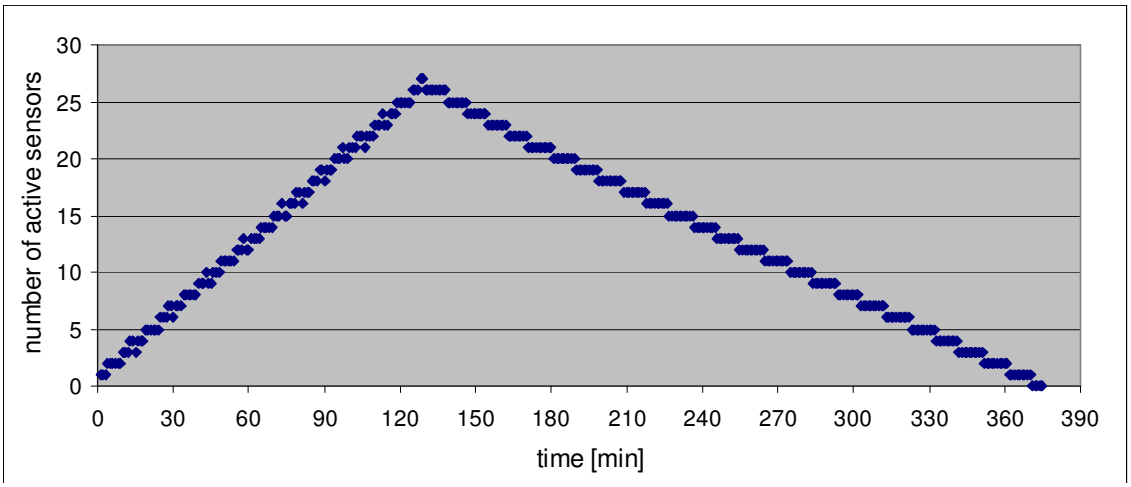
Konfiguracja źródeł danych została zaprezentowana na rys. 34.

sensor number	distance [m]	contender mid-time/result [s]			
		1	100	1000	5000
0	0	0	7	72	360
1	1000	180	237	346	804
2	2000	360	467	621	1248
3	3000	540	697	895	1692
4	4000	720	928	1170	2136
5	5000	900	1158	1445	2580
6	6000	1080	1388	1720	3024
7	7000	1260	1619	1994	3468
8	8000	1440	1849	2270	3912
9	9000	1620	2079	2545	4356
10	10000	1800	2310	2820	4800
11	11000	1985	2547	3098	5290
12	12000	2171	2784	3376	5780
13	13000	2356	3021	3654	6270
14	14000	2542	3258	3932	6760
15	15000	2727	3495	4210	7250
16	16000	2913	3732	4488	7740
17	17000	3098	3969	4766	8230
18	18000	3284	4206	5044	8720
19	19000	3469	4443	5322	9210
20	20000	3655	4680	5600	9700
21	21095	3840	4920	5880	10200
22	22000	4013	5160	6160	10760
23	23000	4186	5400	6440	11320
24	24000	4359	5640	6720	11880
25	25000	4532	5880	7000	12440
26	26000	4705	6120	7280	13000
27	27000	4878	6360	7560	13560
28	28000	5051	6600	7840	14120
29	29000	5224	6840	8120	14680
30	30000	5400	7080	8400	15240
31	31000	5592	7332	8712	15816
32	32000	5784	7584	9024	16392
33	33000	5976	7836	9336	16968
34	34000	6168	8088	9648	17544
35	35000	6360	8340	9960	18120
36	36000	6540	8595	10285	18702
37	37000	6720	8850	10610	19284
38	38000	6900	9105	10935	19866
39	39000	7080	9360	11260	20448
40	40000	7260	9615	11585	21030
41	41000	7440	9870	11910	21612
42	42195	7620	10200	12240	22200



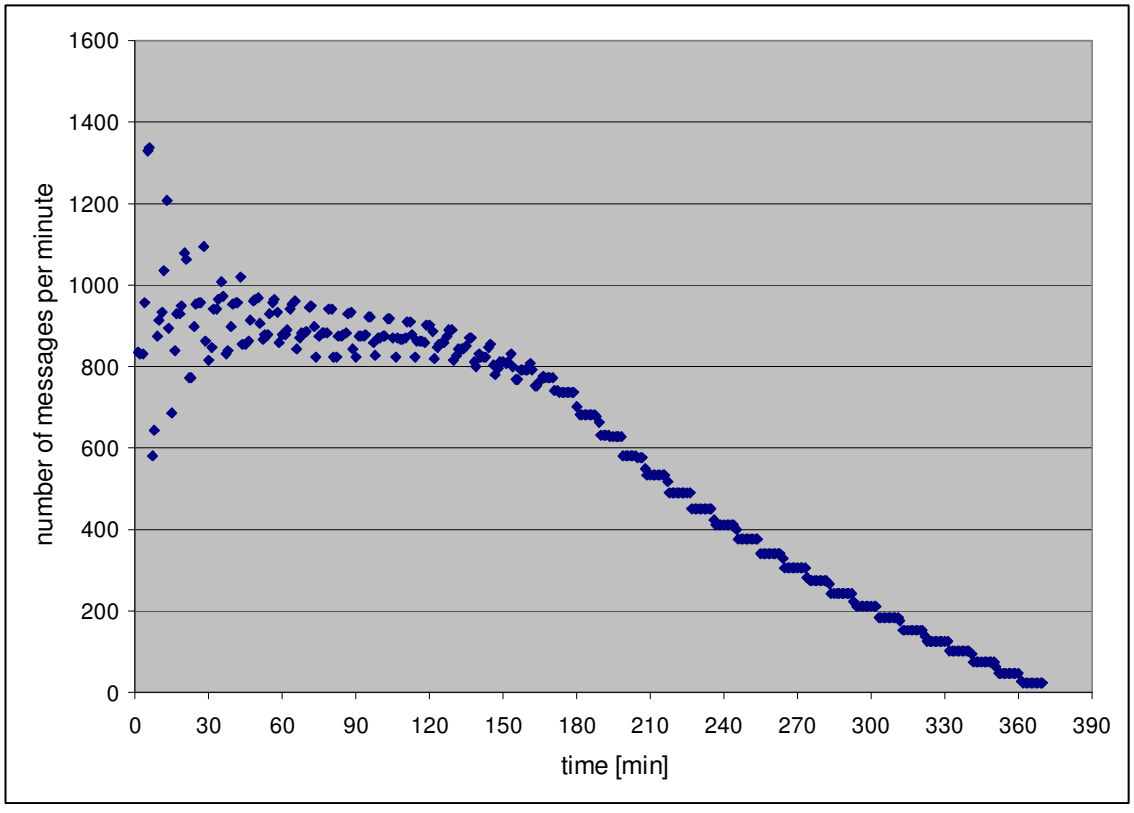
Rys. 34. Konfiguracja źródeł informacji dla prezentowanego scenariusza testowego

Dla celów analizy działania systemu informatycznej obsługi biegu warto zauważyć, że ilość aktywnych, tzn. emitujących wiadomości, źródeł z początku rośnie wraz ze wzrostem odległości pomiędzy pierwszym a ostatnim zawodnikiem, a następnie – po przybyciu pierwszego zawodnika na metę – maleje, jako że zaczyna być determinowana przez odległość ostatniego zawodnika od mety biegu. Tendencje te zilustrowane są na rys. 35. Z uwagi na przyjęte uproszczenia, m.in. liniową interpolację wyników, tendencje te mają również charakter liniowy.

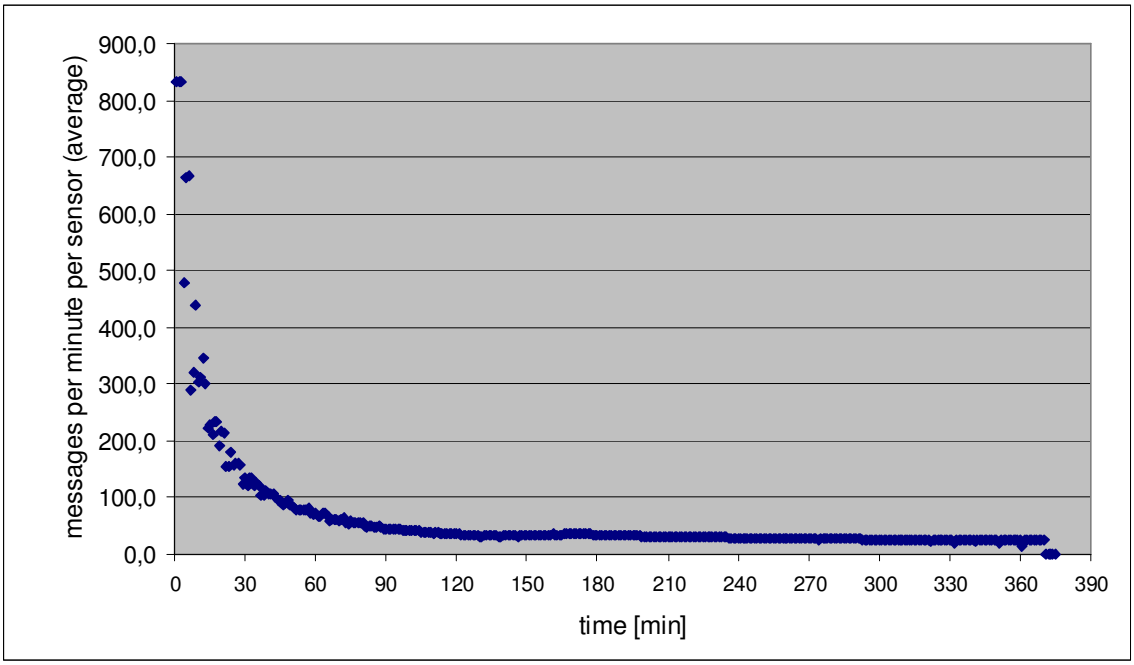


Rys. 35. Zmiany ilości aktywnych źródeł wiadomości w trakcie trwania eksperymentu

Rysunek 36 ilustruje sumaryczną ilość wiadomości do opublikowania w kolejnych minutach symulacji. W przeprowadzonych testach czas większego obciążenia instancji prototypu REMP wynosił około trzech godzin. Obciążenie to miało jednak niejednorodny charakter – z początku było generowane głównie przez dwa źródła reprezentujące sensory umiejscowione blisko startu wyścigu, później przez wiele sensorów emitujących mniej wiadomości (rys. 37).



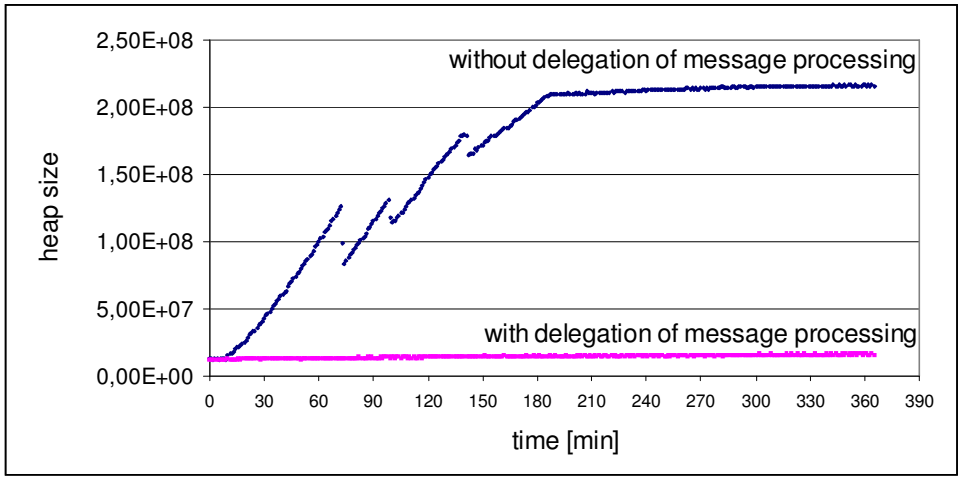
Rys. 36. Zmiany ilości publikowanych wiadomości na minutę w trakcie trwania eksperymentu



Rys. 37. Zmiany średniej ilości wysyłanych wiadomości przypadającej na pojedynczy aktywny sensor w czasie trwania eksperymentu

4.3. Najważniejsze wyniki pomiarów

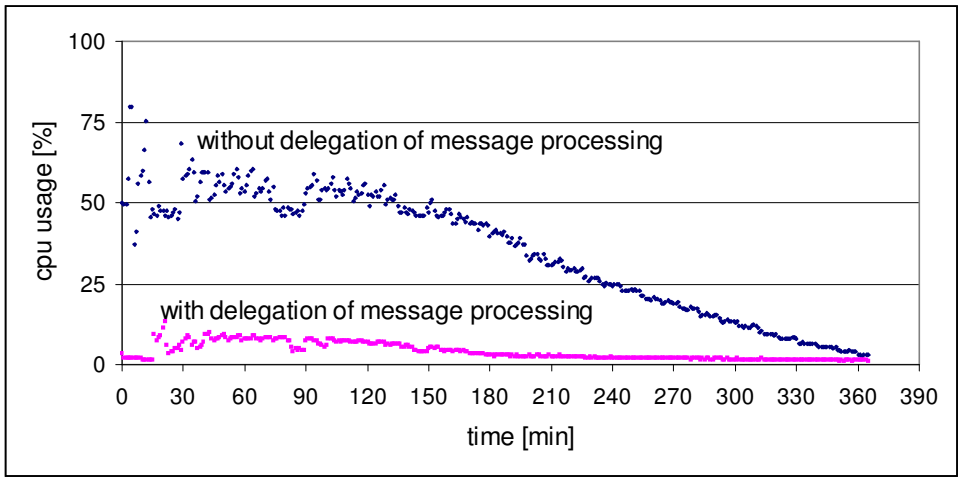
W ramach eksperymentu mierzono wartości reprezentujące zapotrzebowanie aplikacji użytkowników na zasoby takie, jak pamięć operacyjna, procesor oraz przepustowość łącz sieciowych. Z punktu widzenia użytkownika aplikacji końcowej, najłatwiejszym do zaobserwowania jest wpływ delegacji przetwarzania na zapotrzebowanie na pamięć operacyjną (rys. 38), jako że jest ono „delegowane" wraz z przetwarzaniem strumieni wiadomości. W celu osiągnięcia zmniejszenia sumarycznego zapotrzebowania na pamięć konieczne jest jednak zastosowanie mechanizmów ponownego wykorzystania wyników pośrednich.



Rys. 38. Wpływ delegowania przetwarzania wiadomości na zużycie pamięci operacyjnej konsumenta

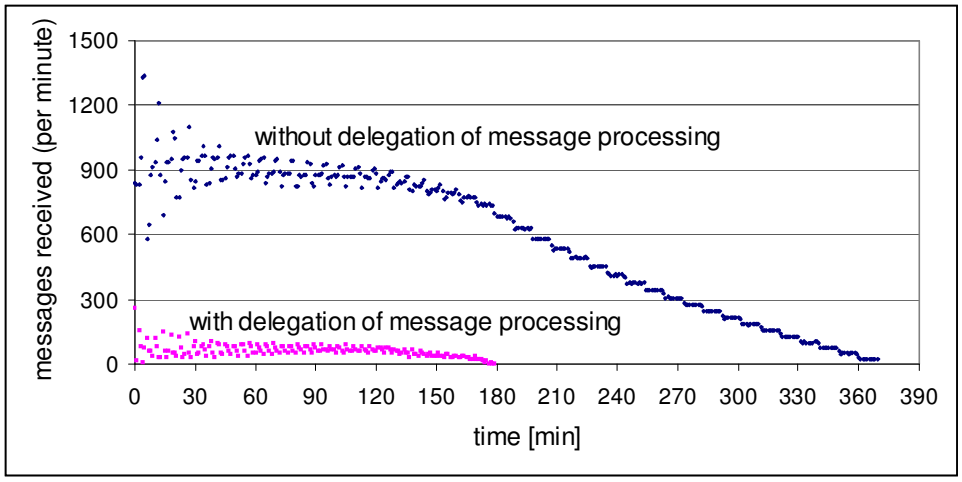
Konsument wiadomości nie jest jedyną stroną odnoszącą korzyści z wykorzystania udostępnianych przez prototyp REMP mechanizmów ponownego wykorzystania wyników

pośrednich. Rys. 39 ilustruje wpływ ponownego wykorzystania jego wyników przez konsumentów na obciążenie procesora po stronie producenta wiadomości. Obciążenie to jest generowane przede wszystkim przez konieczność wielokrotnego tworzenia dokumentów XML przez producentów. Ponowne wykorzystanie wyników przetwarzania zmniejsza ilość kanałów komunikacyjnych zasilanych przez producentów, przyczyniając się w ten sposób do zmniejszenia obciążenia procesora.



Rys. 39. Wpływ delegowania przetwarzania wiadomości na obciążenie procesora producenta

Rysunek 40 ilustruje ilość wiadomości odbieranych przez konsumenta w warunkach stosowania oraz braku stosowania delegacji przetwarzania. W przypadku testowanej aplikacji wiadomości charakteryzują się niewielkimi wahaniami rozmiarów, a zatem ilustracja ilości odbieranych wiadomości jest zarazem ilustracją zapotrzebowania aplikacji klienckiej na przepustowość.



Rys. 40. Wpływ delegowania przetwarzania wiadomości na zapotrzebowanie konsumenta na przepustowość sieci

4.4. Podsumowanie

Przeprowadzone testy pozwoliły na sprawdzenie kluczowych elementów prototypu proponowanej platformy. W prezentowanym scenariuszu zastosowania charakteryzującego się odpowiednio długim czasem działania i dużą ilością przesyłanych wiadomości delegacja

przetwarzania strumieni wiadomości można zauważyć znaczące zmniejszenie zapotrzebowania na zasoby obliczeniowe i przepustowość sieci zarówno po stronie producenta, jak i konsumenta wiadomości.

5. Wnioski końcowe. Kierunki rozwoju platformy

Celem prezentowanej rozprawy było zaproponowanie architektury oraz częściowa implementacja prototypu platformy służącej tworzeniu w środowiskach węzłów równorzędnych sieci nakładkowych reprezentujących komponentowe aplikacje przetwarzające strumienie wiadomości. Niniejszy rozdział autoreferatu poświęcony jest przedstawieniu zaprezentowanych osiągnięć w kontekście tezy rozprawy i wyspecyfikowanych celów badawczych.

5.1. Osiągnięcia badawcze

Podstawowym, z punktu widzenia autora rozprawy, osiągnięciem badawczym rozprawy jest stworzenie działającego i stabilnego prototypu platformy służącej dynamicznemu tworzeniu sieci nakładkowych w środowisku węzłów równorzędnych, co jest bezpośrednio związane z wykazaniem tezy rozprawy.

W celu zapewnienia elastyczności użytkowania, funkcjonalność platformy została podzielona na odrębne usługi, konfigurowalne i rozszerzalne. Współpraca pomiędzy instancjami poszczególnych usług organizowana jest przez szereg dedykowanych protokołów udostępnianych użytkownikom za pomocą odpowiednich interfejsów programistycznych. Interfejsy dla aktorów charakteryzują się przy tym maksymalną prostotą budowy. Elastyczność użytkowania prototypu platformy jest dodatkowo zwiększona przez zapewnienie obsługi zapytań konceptualnych.

Implementacja prototypu była dużym wyzwaniem ze względu na złożoność ich kodu. Realizacja tego zadania była możliwa dzięki przejrzystemu projektowi i precyzyjnemu zdefiniowaniu funkcjonalności usług. Warto przy tym zauważyć, że architektura platformy jest zdefiniowana w sposób niezależny od środowiska implementacji i uruchomienia. Oparcie implementacji prototypu na JXTA było decyzją niezależną od projektu platformy.

5.2. Podstawowe kierunki rozwoju

Jednym z podstawowych kierunków dalszego rozwoju platformy powinna być implementacja mechanizmów związanych z aspektami bezpieczeństwa. Implementacja tej funkcjonalności może być oparta na usługach dostarczanych przez platformy tworzenia sieci węzłów równorzędnych. W przypadku prototypu REMP opartego na JXTA wykorzystano mechanizmy kontroli dostępu do grupy węzłów oraz szyfrowania zawartości wiadomości przesyłanych przez kanały komunikacyjne. W przyszłości mechanizmy związane z bezpieczeństwem powinny być rozwinięte szczególnie w kierunku dostarczenia narzędzi do administrowania węzłami wewnętrznymi platformy.

Stosowanie własnych polityk rozmieszczania z wykorzystaniem prototypowej aplikacji jest możliwe, choć stosunkowo trudne. Zapewnienie wygodniejszego w użyciu interfejsu programisty umożliwiającego definiowanie własnych polityk jest kolejnym kierunkiem rozwoju implementacji REMP.

Prototyp platformy nie obsługuje migracji węzłów sieci nakładkowych, choć funkcjonalność ta jest przewidziana przez projekt platformy. Uzupełnienie implementacji prototypu w tym zakresie zwiększy możliwości samoadaptacji instancji REMP. Interesującym uzupełnieniem będzie ponadto rozbudowanie serwisu obsługi instancji redundantnych do obsługi równoważenia obciążenia.

Z punktu widzenia łatwości użytkowania kluczowym kierunkiem rozwoju proponowanego oprogramowania będzie wspomaganie specyfikacji sieci nakładkowych za pomocą graficznego interfejsu użytkownika.

Literatura

1. [Baldoni 2004] R. Baldoni, R. Beraldi, L. Querzoni, A. Virgillito, *A Self-Organizing Crash-Resilient Topology Management System for Content-Based Publish/Subscribe*, in A. Carzaniga, P. Fenkam (eds.), Proceedings of the 3rd International Workshop on Distributed Event-Based Systems (DEBS'04), IEEE, 2004
2. [Banavar 1999-1] G. Banavar, T. D. Chandra, R. E. Strom, D. C. Sturman, *A Case for Message Oriented Middleware*, Proceedings of the 13th ACM International Symposium on Distributed Computing, Springer-Verlag Lecture Notes in Computer Science, vol. 1693, 1999, p. 1-18
3. [Boag 2007] S. Boag, D. Chamberlin, M. F. Fernandez, D. Florescu, J. Robie, J. Simeon, XQuery 1.0: An XML Query Language, W3C Recommendation, W3C Consortium, 2007, http://www.w3.org/TR/xquery/
4. [Carzaniga 2004] A. Carzaniga, M. J. Rutherford, A. L. Wolf, *A Routing Scheme for Content-Based Networking*, Proceedings of IEEE INFOCOM 2004, Hong Kong, China, 2004
5. [Clark 1999] J. Clark, S. DeRose, XML Path Language (XPath) Version 1.0, W3C Recommendation, W3C Consortium, 1999, http://www.w3.org/TR/xpath
6. [Cugola 2002] G. Cugola, H.-A. Jacobsen, *Using publish/subscribe middleware for mobile systems*, ACM SIGMOBILE Mobile Computing and Communications Review, Vol. 6, Issue 4, 2002
7. [Deering 1990] S. E. Deering, D. R. Cheriton, Multicast routing in datagram networks and extended LANs, *ACM Transactions on Computer Systems*, vol. 8, no. 2, 1990, pp. 85-111
8. [Denning 2006] Infoglut, Communications of the ACM, July 2006, vol. 49, no. 7
9. [Fikes 2003] R. Fikes, P. Hayes, I. Horrocks, *OWL-QL - A Language for Deductive Query Answering on the Semantic Web*, Knowledge Systems Laboratory, Stanford University, Stanford, USA, 2003.
10. [Klyne 2004] G. Klyne (ed.), J. Carroll (ed.), *Resource Description Framework (RDF): Concept and Abstract Syntax*, W3C Recommendation, W3C, 2004
11. [Mahmoud 2004] Q. S. Mahmoud (ed.), Middleware for Communications, John Wiley & Sons, 2004
12. [McLaughlin 2006] B. McLaughlin, J. Edelson, Java and XML, O'Reilly Media, Inc., 2006
13. [MCM 2008] 9th Milano City Marathon Timing Data Service website, http://www.tds-live.com/wtrpg/race.jsp?id=2175

14. [Mühl 2002] G. Mühl, *Large-Scale Content-Based Publish/Subscribe Systems*, PhD Dissertation, Technische Universität Darmstadt, 2002
15. [OMG 2000-1] [OMG 2000-1] CORBA Event Service Specification, Version 1.1, Object Management Group, 2000
16. [OMG 2000-2] CORBA Trading Object Service Specification, version 1.0, OMG, 2000
17. [OMG 2004] CORBA: Notification Service, Version 1.1. Specification, Object Management Group, 2004
18. [Ort 2003] E. Ort, B. Mehta, Java Architecture for XML Binding, Sun Microsystems, 2003, http://java.sun.com/developer/technicalArticles/WebServices/jaxb/
19. [Patel-Schneider 2004] P.F. Patel-Schneider, P. Hayes, I. Horrocks (eds.), *OWL Web Ontology Language Semantics and Abstract Syntax*, W3C Recommendation, W3C Consortium, 2004, http://www.w3.org/TR/owl-semantics/
20. [Prud'hommeaux 2008] E. Prud'hommeaux, A. Seaborne, SPARQL Query Language for RDF, W3C Recommendation, W3C Consortium, 2008, http://www.w3.org/TR/rdf-sparql-query/
21. [SAS 2008] *SAS Integration Technologies: Developer's Guide*, SAS Institute, http://support.sas.com/rnd/itech/doc9/dev_guide/
22. [Stoica 2003] Ion Stoica, *Overlay Networks*, in J. Liebeherr (ed.) Computer Networks lecture notes, University of Virginia, 2003, http://www.cs.virginia.edu/~cs757/slidespdf/757-09-overlay.pdf
23. [Zhao 2001] Y. Zhao, R. E. Strom, *Exploiting Event Stream Interpretation in Publish-Subscribe Systems*, Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC 2001), ACM, 2001, p. 219-228
24. [Zhou 2006] Y. Zhou, B. C. Ooi, K.-L. Tan, F. Yu, *Adaptive reorganization of coherency-preserving dissemination tree for streaming data*, in L. Liu, A. Reuter, K.-Y. Whang, J. Zhang (eds.), Proceedings of the 22nd International Conference on Data Engineering (ICDE'06), IEEE Computer Society, 2006