AUTOMATYKA • 2006 • Tom 10 • Zeszyt 3

Kamil Kuliberda*, Jacek Wiślicki*, Tomasz Kowalski*,
Piotr Błaszczyk*, Grzegorz Balcerzak*, Radosław Adamus*

Implementacyjne aspekty otwartej platformy transportowej opartej o *Peer-to-Peer* **

1. Wprowadzenie

Niniejszy artykuł jest kontynuacją badań związanych z realizacją równoległej sieci rozproszonej. Założenia teoretyczne dla budowy sieci DataGrid [6, 8, 9, 10] zostały szeroko opisane w [3, 4, 6, 7, 8, 9, 10]. Badania nad prototypem architektury opisane w [6, 8, 9, 10] znalazły swoje potwierdzenie w implementacji, której elementy są przedstawione w niniejszym artykule. Autorzy opisują proces realizacji wirtualnej sieci komunikacyjnej, bez mechanizmów integracyjnych dla prototypu DataGrid. Wynikiem takiego podejścia jest otwarta platforma transportowa, którą można adaptować jako mechanizm realizujący komunikację dla każdej aplikacji sieciowej.

Prezentowane podejście zakłada możliwość realizacji swobodnej komunikacji sieciowej (Ethernet oraz Internet) za pomocą architektury „każdy z każdym" P2P (*Peer-to-Peer*) [5, 7], co więcej, odpowiednio zaimplementowana platforma transportowa pozwala na realizację komunikacji niezależnie od reguł działania sieci Ethernet i Internet (NAT, Firewall). Omawiana platforma transportowa implementuje wirtualną sieć, w której współuczestnicy są niezależnie indeksowani i tworzą społeczność, która ma możliwość komunikacji zgodnej z odpowiednimi założeniami twórców sieci. Do realizacji wirtualnej sieci została wykorzystana platforma open-source JXTA [12].

Dalsza część artykułu porusza następujące tematy; w rozdziale drugim opisano platformę JXTA, w rozdziale trzecim zasady realizacji sieci P2P, protokół komunikacyjny wraz z prostym przykładem jego rozbudowy i integracji własnego oprogramowania z otwartą platformą transportową, rozdział czwarty zawiera podsumowanie.

---

* Katedra Informatyki Stosowanej, Politechnika Łódzka

** Praca współfinansowana ze środków Europejskiego Funduszu Społecznego oraz Budżetu Państwa w ramach programu Mechanizm WIDDOK (numer umowy Z/2.10/II/2.6/04/05/U/2/06)

2. Platforma JXTA

Założeniem projektu JXTA opracowanego i zaimplementowanego początkowo w laboratoriach firmy Sun Microsystems, obecnie rozwijanego przez społeczność *jxta.org* [12] jest wytworzenie wspólnej platformy umożliwiającej łatwe i szybkie budowanie rozproszonych serwisów i aplikacji, w których każde urządzenie jest adresowane jako współuczestnik takiej sieci. JXTA w znacznej mierze opiera się na języku XML, który jest dobrze znany, ma duże wsparcie i jest wygodny dla programistów.

Sieć komunikacyjna na bazie JXTA jest niezależna od platformy sprzętowej oraz od urządzenia na jakim może być zaimplementowana, posiada status *open-source* oraz udostępnia podstawowe funkcje do konstruowania sieci P2P [5, 7]. Dostarcza szereg otwartych protokołów, dzięki którym tworząc sieć równoległą, można:

- odnajdywać i przekazywać informacje o usługach sieciowych,
- wzajemnie się komunikować,
- wzajemnie się odnaleźć,
- wzajemnie się monitorować,
- organizować się w grupy.

Protokoły JXTA są tak zaprojektowane, aby były niezależne od języka programowania i od protokołów transportowych. Mogą być implementowane w Javie, C/C++, Perlu i wielu innych językach. Mogą być także implementowane ponad stosem TCP/IP, protokołem HTTP, technologią Bluetooth lub podobnymi protokołami transportowymi. Takie rozwiązanie pozwala na zastosowanie platformy na różnych urządzeniach.

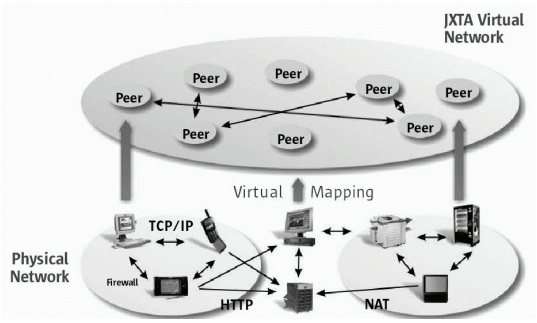
Aplikacje oparte na JXTA potrafią:

- znaleźć docelowy węzeł, nawet jeśli jest za ścianą ogniową (*firewall*);
- łatwo dzielić dokumenty z innymi współuczestnikami sieci;
- błyskawicznie znajdować dokumenty w sieci;
- tworzyć grupy węzłów (współuczestników) dostarczających różne usługi;
- monitorować zdalnie działanie węzła;
- bezpiecznie komunikować się z innym użytkownikami w sieci.

2.1. Wirtualna sieć oparta o JXTA

Sieć komunikacyjna (platforma) oparta o JXTA tworzy wirtualną sieć połączonych ze sobą węzłów (współuczestników sieci). Realizuje ona logiczną warstwę sieciową, która swobodnie działa ponad istniejącą już siecią fizyczną. Takie rozwiązanie jest bardzo elastyczne, pozwala na tworzenie sieci opartych na założeniach ich twórców, co więcej pozwala na zastosowanie oraz zaimplementowanie wewnątrz platformy wielu mechanizmów dedykowanych konkretnym potrzebom współuczestników oraz zgodnych z założeniami twórców sieci.

Na rysunku 1 [Project JXTA] przedstawiono ideę wirtualnej sieci.



Rys. 1. Wirtualna sieć oparta o JXTA

2.2. Architektura JXTA

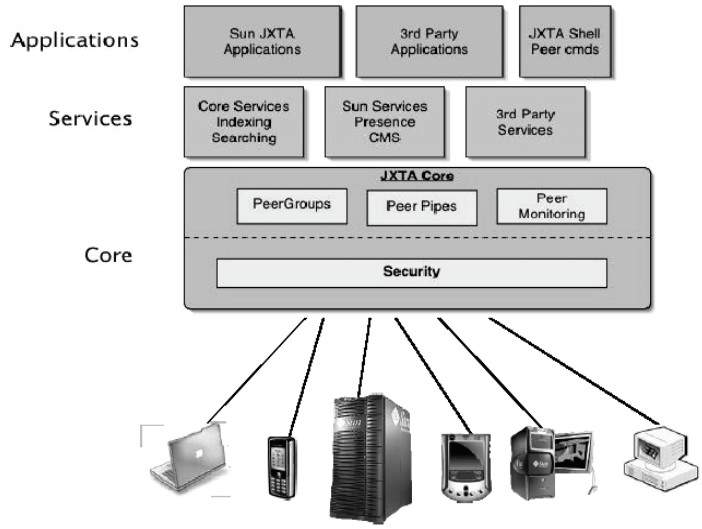
Architektura JXTA podzielona jest na trzy warstwy, ich rozmieszczenie i zależności przedstawia rysunek 2.

1) Warstwa Core (JXTA Core) integruje cztery bloki, które zawierają kluczowe mechanizmy dla aplikacji *Peer-to-Peer*. Blok Peer Groups umożliwia tworzenie i usuwanie grup współużytkowników, ogłasza ich istnienie potencjalnym użytkownikom sieci, wyszukuje, dołącza oraz odłącza ich od grup. Peer Pipes realizuje kanały komunikacyjne pomiędzy aplikacjami współuczestników. Komunikacja odbywa się za pośrednictwem formatu XML, przy zapewnieniu bezpieczeństwa i prywatności użytkowników. Peer Monitoring sprawuje kontrolę nad zachowaniem aplikacji współużytkowników i grup – kontrola dostępu, ustawianie priorytetów, kontrolowanie przepustowości itp.
2) Warstwa usług (JXTA Services) zawiera usługi sieciowe, które są niezbędne dla działania sieci P2P. Między innymi realizuje; wyszukiwanie i indeksowanie, katalogowanie, współdzielenie plików, zawiera protokoły translacji, realizuje poświadczenia (*authentication*) oraz zarządza infrastrukturą klucza publicznego – PKI (*Public Key Infrastructure*).
3) Warstwa aplikacji (JXTA Application Layer) zawiera implementacje integralnych aplikacji dla użytkownika, takich jak P2P instant messaging, współdzielenie dokumentów i zasobów, dostarczanie i zarządzanie treścią, P2P E-mail system, systemy dystrybucji i wiele innych.

Cała platforma JXTA jest zaprojektowana modułowo, co pozwala programistom wybierać tylko te usługi, które są im potrzebne.

Innowacyjność rozwiązania określają trzy kluczowe aspekty architektury JXTA, które wyróżniają ją od innych modeli sieci P2P:

1) do opisu źródeł sieciowych używane są dokumenty XML nazwane ogłoszeniami (*advertisement*);
2) połączenia między aplikacjami użytkowników realizowane są z wykorzystaniem abstrakcyjnego kanału komunikacyjnego (*pipe*), dodatkowo współuczestnicy łącząc się ze sobą, nie używają centralnego schematu nazewnictwa (np. takiego jak w DNS);
3) posiadają unikalny schemat adresowania oparty na PeerID [12].



Rys. 2. Architektura JXTA

2.3. Organizacja wirtualnej sieci w JXTA

Współuczestnicy za pomocą swoich aplikacji mogą włączać i odłączać się od sieci w dowolnym czasie, w efekcie czego drogi transportu wiadomości często się zmieniają, a ich czas jest niedeterministyczny. Aplikacje sieciowe JXTA mogą udostępniać różne usługi, w praktyce używane są cztery, niżej opisane rodzaje aplikacji współużytkowników sieci.

1) Minimal Edge Peer – może wysyłać i odbierać komunikaty, nie przechowuje ogłoszeń (*advertisements*) i nie przekazuje komunikatów do innych węzłów.
2) Full-featured Edge Peer – może wysyłać, odbierać komunikaty i zazwyczaj przechowuje otrzymane rozgłoszenia, ale odpowiada wskazując jedynie kierunek dalszego poszukiwania węzła, natomiast nie przekazuje wiadomości dalej.
3) Rendezvous Peer – jest podobny do innych współuczestników sieci, dodatkowo przechowuje otrzymane rozgłoszenia oraz przekazuje je dalej. W momencie, gdy nowy współuczestnik włącza się do grupy, automatycznie poszukuje rendezvous peera, jeśli żadnego nie znajdzie, uruchamia dodatkowe procesy i sam zostaje rendezvous peerem dla grupy, do której dołączył. Każdy taki współuczestnik sieci przechowuje listę innych znanych mu rendezvous peerów. Każda grupa współuczestników musi posiadać przynajmniej jeden rendezvous peer, może posiadać ich więcej w zależności od potrzeb.
4) Relay Peer – przechowuje informacje o trasach do innych współużytkowników sieci. Gdy węzeł chce się połączyć z innym węzłem poszukuje informacji o trasie we własnej pamięci podręcznej. Gdy tam jej nie zlokalizuje, odpytuje relay peer. Specjalną funkcją opisywanego współuczestnika sieci jest przekazywanie komunikatów w imieniu współużytkowników znajdujących się za NATem lub ścianą ogniową (*firewall*), ponieważ nie mogą one połączyć się bezpośrednio z innymi węzłami sieci.

Podczas realizacji połączeń między współuczestnikami sieci (Peer Edge), aplikacje JXTA wysyłają zgłoszenia o wyszukanie i zlokalizowanie innych współuczestników do randezvous peerów. Jeżeli najbliższy randezvous peer nie posiada żądanej informacji zwrotnej, przesyła zgłoszenie dalej do innych znanych mu randezvous peerów. Proces lokalizowania (*discovery process*) trwa do momentu otrzymania przez zgłaszającego żądanej odpowiedzi lub czas życia zgłoszenia dobiegnie końca. Każdy komunikat posiada domyślny czas życia TTL (*time-to-live*) – siedem skoków (*hops*).

Dowolna aplikacja peeru w JXTA może mieć zaimplementowane mechanizmy, jakie dedykowane są relay peerom lub randevous peerom. Usługi relay i randevous mogą być implementowane w ramach jednego peeru.

JXTA definiuje szereg XML formatów komunikatów i protokołów do wzajemnej komunikacji pomiędzy współuczestnikami sieci. Aplikacje użytkowników używają tych protokołów do odnajdywania się, rozgłaszania i lokalizowania zasobów sieciowych, komunikacji oraz wyszukiwania tras [12].

Rdzeń platformy stanowi sześć protokołów:

1) Peer Discovery Protocol (PDP) – stosowany do odnajdywania innych węzłów;
2) Peer Information Protocol (PIP) – wydobywa informacje o stanie innego węzła;
3) Peer Resolver Protocol (PRP) – umożliwia odwołanie się do danego zasobu za pośrednictwem rozgłaszanych przez węzły opisów;
4) Pipe Binding Protocol (PBP) – pozwala na dołączenie się do kolejki komunikatów;
5) Endpoint Routing Protocol (ERP) – wyszukuje trasę do węzła docelowego;
6) Randezvous Protocol (RVP) – umożliwia propagowanie komunikatu wewnątrz grupy.

Wszystkie protokoły JXTA są asynchroniczne i bazują na modelu żądanie/odpowiedź.

2.4. Kanały komunikacyjne w JXTA

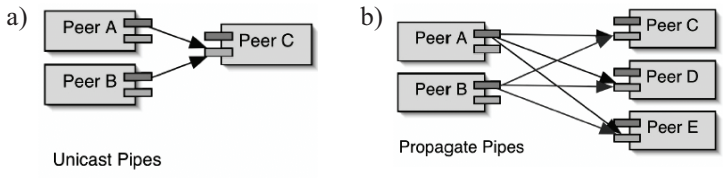
W JXTA występuje kanał komunikacyjny (*pipe*) [12], który umożliwia połączenie dwóch współużytkowników i przesyłanie między nimi wszystkich rodzajów danych w sposób asynchroniczny i wielokierunkowy. Końce kanałów (*pipe endpoints*) składają się z:

– końcówki otrzymującej dane (*input pipe*),
– końcówki wysyłającej dane (*output pipe*).

Kanał jest wiązany z określonymi punktami końcowymi (*pipe endpoints*), wtedy port TCP jest kojarzony z adresem IP (w warstwie fizycznej sieci).

W JXTA występują trzy rodzaje kanałów [12]:

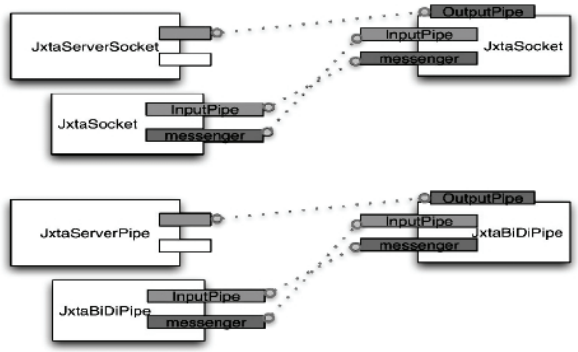
1) Point-to-point pipe – łączący bezpośrednio dwa końce input pipe i output pipe (rys. 3a),
2) Propagate Pipe – umożliwiający wysyłanie wiadomości do wielu węzłów jednocześnie (rys. 3b),
3) Secure Unicast Pipe – to rodzaj Point-to-point pipe, przy zapewnieniu bezpiecznego kanału komunikacyjnego.



Rys. 3. Kanały: a) Point-to-point/Unicast; b) Propagate

Ponadto JXTA jest zaopatrzona w dwukierunkowy niezawodny kanał komunikacyjny JxtaBiDiPipe oraz mechanizm komunikacyjny JxtaSocket oparty na gniazdach (*sockets*) (rys. 4) [12]. Rozwiązania te:

– dostarczają dwukierunkowość komunikacji,
– operują na strumieniach danych,
– zapewniają buforowanie przesyłanych danych,
– zapewniają wielowątkowość komunikacji.



Rys. 4. JXTA Socket oraz JXTABiDiPipe

Mechanizmy JxtaServerSocket oraz JxtaBiDiServerPipe udostępniają kanał Input pipe do odbierania żądań inicjujących połączenie (*connection request*) i negocjacji parametrów komunikacji. Natomiast za pomocą JxtaSocket i JxtaBiDpipe budowane są dedykowane kanały komunikacyjne niezależnie od żądania inicjującego połączenie [12].

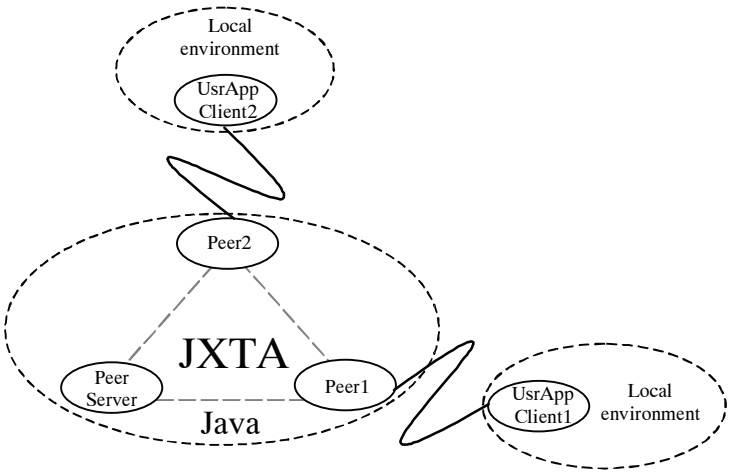
3. Elementy konstrukcji sieci „każdy-z-każdym" opartej na JXTA

W prezentowanym prototypie przyjęto założenie [6], że musi występować *zewnętrzna aplikacja użytkownika* (może to być dowolna aplikacja wykorzystująca platformę transportową; np. komunikator, baza danych), która:

– komunikuje się z aplikacją wirtualnej sieci,
– jest w niej rozpoznawana,

– posiada swój unikalny identyfikator w ramach sieci wirtualnej,
– posiada interfejs komunikacyjny wraz z protokołem do sterowania aplikacją sieci wirtualnej.

Tylko w takim wypadku istnieje możliwość uruchomienia sieci i udostępnienia platformy transportowej dla komunikacji pomiędzy innymi aplikacjami. Reasumując, pojedynczy współuczestnik sieci to para aplikacji, z których bezpośrednio dostępna dla użytkownika jest ta, która wykorzystuje aplikację sieci wirtualnej (Peer Edge), a sama aplikacja sieci wirtualnej jest przezroczysta dla użytkownika (rys. 5).



Rys. 5. Komunikacja zewnętrznych aplikacji użytkownika
za pomocą platformy transportowej

Architektura omawianej platformy transportowej implementuje rozwiązanie zcentralizowanej sieci równoległej [5, 7]. Przy takim rozwiązaniu wewnątrz sieci znajduje się serwer tzw. centralny, który jest odpowiedzialny za utrzymanie i zarządzanie siecią [5, 7]. Dzięki takiemu podejściu uzyskuje się lepszą wydajność przetwarzania danych wewnątrz sieci, lepszą skalowalność oraz wzrost szybkości wyszukiwania zasobów. Kolejnym argumentem przemawiającym za implementowaną architekturą jest kwestia zaufania w sieci (*trust*), której nie da się osiągnąć bez serwera centralnego. Jest to jednak znacznie większy problem, który można opisać w oddzielnym artykule.

W proponowanym rozwiązaniu zadaniem centralnego serwera jest przechowywanie informacji o aktualnie dostępnych współuczestnikach sieci, funkcjonalność ta jest realizowana poprzez:

– zarejestrowanie nowego współuczestnika sieci, gdy pojawi się on w sieci;
– wyrejestrowanie współuczestnika sieci, gdy zakończy on pracę w sieci;
– bieżący monitoring dostępności współuczestników w sieci;
– przechowywanie informacji o aktualnie dostępnych współuczestnikach sieci.

Omawiana i prototypowo zaimplementowana platforma transportowa jest typową aplikacją P2P, tzn. każdy współużytkownik może jednocześnie zgłaszać żądanie i odpowiadać na żądania innych współużytkowników (pełni rolę klienta i serwera jednocześnie).

Ważnym zagadnieniem implementacyjnym okazał się wybór i realizacja kanału komunikacyjnego, który jest podstawowym mechanizmem przy przesyłaniu danych wewnątrz wirtualnej sieci. W rezultacie platformę zaopatrzono w mechanizm oparty na JXTA Socket (JXTA BiDiPipe nie posiada mechanizmu dzielenia dużych wiadomości i ogranicza wielkość przesyłanej wiadomości do 64 KB, co w przypadku przesyłania danych w postaci XML dyskwalifikuje jego wykorzystanie) [6, 12].

3.1. Protokół komunikacyjny dla platformy transportowej

Do realizacji poprawnego działania sieci oraz serwera centralnego zaprojektowano i zaimplementowano protokół komunikacyjny oparty na języku XML.

Protokół ten obsługuje następujące komunikaty:

– dodanie nowego współuczestnika do listy dostępnych współuczestników – po tym jak zewnętrzna aplikacja użytkownika zainicjalizuje połączenie z aplikacją sieci wirtualnej, realizowane jest włączenie do sieci P2P i zarejestrowanie współuczestnika sieci na serwerze centralnym; realizowane jest to poprzez wysłanie wiadomości w następującym formacie:

```
<?xml version=”1.0" ?>
<Command>
<!--Infromacja na temat Nadawcy-->
<Sender Address=”PeerName” />
<Add Name=”PeerName” />
</Command>
```

– usunięcie współuczestnika z listy dostępnych – w momencie gdy użytkownik kończy pracę aplikacja wysyła do serwera centralnego żądanie wyrejestrowania w następującym formacie:

```
<?xml version=”1.0" ?>
<Command>
<!--Infromacja na temat Nadawcy-->
<Sender Address=”PeerName” />
<Del Name=”PeerName” />
</Command>
```

– aktualna lista współuczestników sieci – w chwili gdy użytkownik rozpocznie lub zakończy pracę, bądź serwer centralny wykryje niedostępność danego współuczestnika sieci, aplikacja serwera centralnego rozsyła do wszystkich dostępnych użytkowników wiadomość o następującym formacie:

```
<?xml version=”1.0" ?>
<Command>
<!--Infromacja na temat Nadawcy-->
<Sender Address=”Server”/>
<NewList Name0=”Peer3" Name1=”Peer1" />
</Command>
```

Po otrzymaniu takiej wiadomości każda aplikacja sieci wirtualnej uaktualnia swoją lokalną listę dostępnych współużytkowników sieci.

Protokół zaimplementowany w obecnej formie jest bardzo prosty, ponieważ ilość komunikatów sterujących jest uzależniona od funkcjonalności aplikacji użytkowników. Powyższe komunikaty opisują działanie sieci wirtualnej oraz indeksację ilości użytkowników tej sieci.

3.2. Przykład rozbudowy protokołu

Aby obsłużyć aplikację użytkowników należy wprowadzić odpowiednie komunikaty charakteryzujące odpowiednią funkcjonalność tych aplikacji. Jako przykład można tutaj użyć identyfikacji zewnętrznych aplikacji mających swoją nazwę i związanych z aplikacją sieci wirtualnej; niech składnia nazwy aplikacji użytkownika w sieci wirtualnej będzie następująca: *AplikacjaUżytkownika@Współużytkownik*, gdzie ‘AplikacjaUżytkownika’ to nazwa sieciowa aplikacji użytkownika, a ‘Współużytkownik’ to nazwa aplikacji sieci wirtualnej w platformie transportowej [6]. Dla takiego przypadku można zdefiniować przykładowy komunikat, który będzie realizował sprawdzenie dostępności współużytkownika sieci:

```
<Datagram>
   <!--Informacje na temat nadawcy-->
   <Sender Address=”Server@Server” />

<!--Informacje na temat odbiorcy-->
   <Recipient Address=”AplikacjaUżytkownika@Współużytkownik” />

<!--Dane do przesłania-->
   <Data Type=”AreYouAlive” Part=”1/1">
   </Data>
</Datagram>

<Datagram>
   <!--Informacje na temat nadawcy-->
   <Sender Address=”Server@Server” />

<!--Informacje na temat odbiorcy-->
<Recipient Address=”AplikacjaUżytkownika@Współużytkownik” />

<!--Dane do przesłania-->
   <Data Type=”AreYouAlive” Part=”1/1">
   </Data>
</Datagram>
```

W powyższym przykładzie element *Datagram* zawiera miedzy innymi dwa podelementy *Sender* i *Recipient*. To w nich zawarte są informacje na temat nadawcy i odbiorcy. Potencjalny odbiorca może odczytać od kogo pochodzi wiadomość i czy była adresowana do niego. Adresy nadawcy i odbiory zapisane są w atrybutach *Address*.

Jak widać na przykładowych komunikatach, protokół jest bardzo elastyczny, co sprawia, że bardzo łatwo go rozbudować przez zdefiniowanie nowych komunikatów. Taka rozbudowa nie będzie miała negatywnych skutków w działaniu protokołu.

4. Podsumowanie

Artykuł w sposób skrótowy przedstawia aspekty realizacji i implementacji otwartej platformy transportowej opartej o sieć „każdy z każdym". Artykuł nawiązuje do wcześniejszych prac opisanych w szeregu publikacji i zaprezentowanych na licznych konferencjach. Publikowane dotąd materiały zawierają koncepcję i architekturę sieci grid [1, 2] opartej o obiektowe bazy danych i teorię podejścia stosowego do języków zapytań [11].

Autorzy w powyższych rozdziałach tego artykułu skupili się na opisie zaimplementowanej wirtualnej sieci oraz jej elementów, wraz z protokołem transportowym. Problem integracji zewnętrznej aplikacji użytkownika z prezentowaną siecią wirtualną jest oddzielnym procesem wymagającym złożonej realizacji. Wyniki badań nad postępami prac zostaną przedstawione w oddzielnej publikacji. Obecny stan badań obejmuje integrację działającego prototypu, współpracującego z obiektowymi bazami danych, gdzie celem jest utworzenie sieci DataGrid [6, 8, 9, 10].

Literatura

[1] Moore R., Merzky A.: *Persistent Archive Concepts.* Global Grid Forum GFD-I.026. December 2003

[2] Foster I., Kesselman C., Nick J., Tuecke S.: *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration.* Global Grid Forum, June 22, 2002

[3] Kozankiewicz H., Stencel K., Subieta K.: *Implementation of Federated Databases through Updateable Views.* Proc. EGC 2005 – European Grid Conference, Springer LNCS, 2005

[4] Kozankiewicz H., Stencel K., Subieta K.: *Integration of Heterogeneous Resources through Updatable Views.* Workshop on Emerging Technologies for Next generation GRID (ETNGRID-2004), 13th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE-2004), University of Modena and Reggio Emilia, Italy, June 14–16, 2004, Proceedings published by IEEE

[5] Kuliberda K.: *Podstawowe mechanizmy architektury Grid Computing.* Automatyka, Półrocznik AGH, t. 8, z. 3, 2004, 505–515

[6] Kuliberda K., Kaczmarski K., Adamus R., Błaszczyk P., Balcerzak G., Subieta K.: *Virtual Repository Supporting Integration of Pluginable Resources*, 17th DEXA 2006 and 2nd International Workshop on Data Management in Global Data Repositories (GRep) 2006, Proceedings in IEEE Computer Society, to appear

[7] Kuliberda K., Sekulska-Nalewajko J., Kuzański M.: *Technologia Grid Computing – nowa architektura sieci komputerowych.* XII Sieci i systemy informatyczne: teoria, projekty, wdrożenia, aplikacje, Politechnika Łódzka, t. 2, 2004, 100–111

[8] Kuliberda K., Wiślicki J.: *Architektura gridu bazodanowego – data grid*. Automatyka, Półrocznik AGH, t. 9, z. 3, 2005, 505–515
[9] Kuliberda K., Wiślicki J: *Przezroczysta platforma transportowa dla rozproszonego, równoległego przetwarzania zasobów bazodanowych.* Sieci i systemy informatyczne: teoria, projekty, wdrożenia, aplikacje, Politechnika Łódzka, t. 2, 2005, 100–111
[10] Kuliberda K., Wiślicki J., Adamus R.: *The Main Issues of The Database Grid Realization Based on Data Grid Concept.* I KKNTPD, Poznań, 2005, 100–111
[11] Subieta K.: *Teoria i konstrukcja obiektowych języków zapytań.* Wydawnictwo PJWSTK, Warszawa 2004, ISBN 83-89244-28-4 (522 strony)
[12] Project JXTA Community: http://www.jxta.org