AUTOMATYKA • 2006 • Tom 10 • Zeszyt 3

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

Zastosowanie otwartej platformy transportowej *Peer-to-Peer* do połączenia i integracji obiektowych baz danych w architekturze DataGrid**

1. Wprowadzenie

Technologie rozproszone osiągają coraz większe znaczenie w procesie przetwarzania informacji biznesowej. Najczęściej stawianymi wymaganiami dla systemów wspierających procesy i operacje biznesowe są dostępność do informacji, aktualność informacji oraz prostota dostępu do informacji. Takie wymagania wypływają nie tylko od korporacji komercyjnych, lecz coraz częściej pojawiają się także w segmencie organizacji rządowych. Wychodząc naprzeciw jednym i drugim, skupiamy nasze badania nad realizacją prototypowej platformy wspomagającej działanie systemów bazodanowych o charakterze biznesowym. Nasze podejście do problemu opiera się o prototyp obiektowej bazy danych, opartej o teorię podejścia stosowego do języków zapytań (*Stack Based Approach*) [12] oraz obiektowe aktualizowalne perspektywy (*Updateable Object Views*) [3].

Badania prezentowane w niniejszej publikacji są praktycznym zastosowaniem założeń teoretycznych precyzujących aspekty budowy sieci gridowych oraz prototypu DataGrid [7, 9, 10, 11]. Koncepcja integracji źródeł rozproszonych została szeroko opisana w [4, 5, 7, 8, 9, 10, 11].

Zastosowanie prototypu architektury opisanej w [7, 9, 10, 11] opiera się o prototyp otwartej platformy transportowej realizującej swobodną komunikację sieciową (Ethernet oraz Internet) za pomocą architektury sieciowej „każdy z każdym" P2P (*Peer-to-Peer*) [6, 8]. Zaimplementowany prototyp platformy transportowej pozwala na realizację komunikacji niezależnie od reguł działania sieci Ethernet i Internet (NAT, Firewall), tworzy wirtualną sieć, w której współuczestnicy są niezależnie indeksowani, i których sieć organizuje w społeczność otrzymującą możliwość komunikacji zgodnie z odpowiednimi założeniami pro-

* 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)

jektantów sieci grid [1, 2]. Platforma transportowa wykorzystuje rozwiązanie Open Source JXTA [13].

W dalszej części artykułu omówione są następujące tematy: w rozdziale drugim opisano szczegóły integracji baz danych za pomocą platformy transportowej, przetwarzanie komunikatów sieciowych, zadania serwera centralnego wraz z indeksem zasobów, rozdział trzeci to podsumowanie wykonania prototypu.

2. Koncepcja integracji baz danych za pomocą platformy transportowej

Prototypowe rozwiązanie integracji łączy ze sobą dwa niezależne systemy przetwarzania danych;

1) bazę danych Odra (*Object Database for Rapid Application development*) – prototypową bazę danych zbudowaną na bazie koncepcji SBA [12],
2) otwartą platformę transportową.

Połączenie aplikacji zastało zrealizowane zgodnie z założeniami architektury DataGrid [7, 9, 10, 11]. W wyniku implementacji powstały w lokalnym środowisku komputera dwie aplikacje, z których jedna jest odpowiedzialna za przezroczystą komunikację sieciową z innymi aplikacjami współpracującymi w ramach sieci wirtualnej platformy transportowej. W wyniku tego tworzona jest niezależna warstwa komunikacyjna, którą zarządzają aplikacje bazodanowe, będące zarówno interfejsami użytkowników sieci oraz zasobami danych dla gridu. Tak utworzona sieć charakteryzuje się scentralizowaną architekturą P2P, jest przezroczysta dla współuczestników gridu, pozwala na implementację dodatkowych technologii jak np. infrastruktury zaufania (*trust infrastructure*).

W sieci mogą działać dwa rodzaje współuczestników:

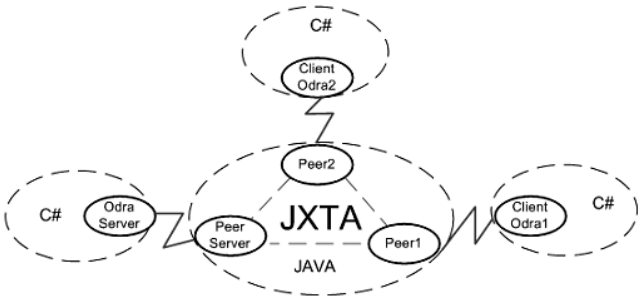
1) serwer centralny odpowiedzialny za utworzenie i utrzymanie sieci,
2) współużytkownicy sieci przetwarzający dane [7].

Uproszczony schemat sieci prezentuje rysunek 1. Na rysunku widać, że implementacja platformy transportowej opiera się o środowisko Java oraz technologię JXTA, a systemy baz danych napisane są w środowisku C#. Otwarta platforma transportowa została tak zaprojektowana, aby możliwe było jej wykorzystanie do tworzenia wszelkiego rodzaju sieci równoległych.

Opracowane zostały dwa protokoły komunikacyjne:

1) do komunikacji wewnątrz wirtualnej sieci,
2) do komunikacji z zewnętrznymi aplikacjami użytkownika (w tym wypadku z bazami danych Odra).

W dalszej części artykułu wyjaśnione zostanie działanie komunikacji pomiędzy aplikacjami bazy danych oraz platformy transportowej w środowisku lokalnym komputera. Mechanizm działania platformy jest obszernym problemem, któremu został poświęcony oddzielny artykuł.

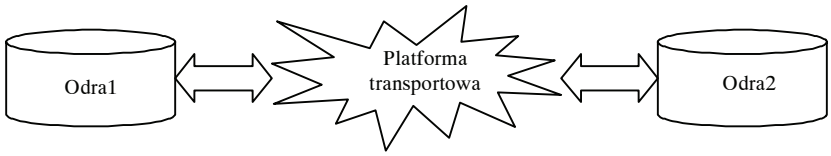


Rys. 1. Poglądowy schemat komunikacji elementów architektury DataGrid

2.1. Komunikacja na poziomie baz danych w prototypie DataGrid

Komunikacja pomiędzy obiektową bazą danych a aplikacją sieci wirtualnej odbywa się z wykorzystaniem gniazd sieciowych (*sockets*) języka C# plaftromy .NET od strony obiektowej bazy danych i gniazd sieciowych języka Java od strony aplikacji sieci P2P.

Zrealizowana wirtualna sieć P2P jest w pełni przezroczysta dla połączonych za jej pomocą rozproszonych obiektowych baz danych Odra. Użytkownik pracujący z interfejsem obiektowej bazy nie ingeruje w aplikację P2P, nie musi nawet wiedzieć o jej istnieniu. Jedyną informacją, jaką musi posiadać, jest nazwa innej bazy danych, z którą chce współpracować (zgodnie z semantyką SBA). Podczas budowy zapytania określa on tylko bazę i zasób, o którym chce uzyskać informacje. Przykładowe zapytanie wygląda następująco: *Odra1.Pracownik*. Taka konstrukcja ułatwia przetwarzanie danych – informacje na temat architektury i struktury sieci są zbyteczne podczas realizacji zleceń na danych. Rozwiązanie to znacznie ułatwia i przyspiesza pracę z rozproszoną obiektową bazą danych. Zlecenie realizacji zapytania przedstawia rysunek 2.



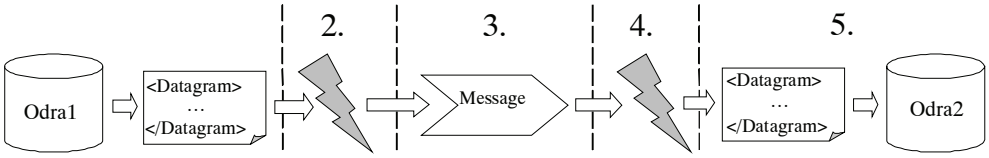
Rys. 2. Ewaluacja zapytań pomiędzy OODBMS poprzez platformę transportową

Transport zapytania wysłanego przez obiektową bazę danych można podzielić na pięć faz:

1. obiektowa baza danych Odra wysyła zapytanie do sieci;
2. zapytanie trafia do aplikacji platformy transportowej związanej z nadającą bazą danych, moduł ten odczytuje informacje o nadawcy i odbiorcy;
3. aplikacja platformy transportowej przesyła całą zawartość zapytania do odpowiedniego odbiorcy w sieci wirtualnej P2P, z którym związana jest określona w zapytaniu docelowa baza danych Odra;

4. aplikacja platformy transportowej odbiorcy po otrzymaniu danych od aplikacji nadawcy rozpoznaje docelową bazę danych, do której jest adresowanie zapytanie i przekazuje je do niej;
5. Odra docelowa odbiera zaadresowane do niej zapytanie.

Cały proces przedstawia rysunek 3



Rys. 3. Pięć faz procesu transportu zapytania w DataGrid

Aby powyższy proces mógł być zrealizowany, zaimplementowano wspomniany wcześniej protokół pozwalający na transport dowolnych danych pomiędzy bazami danych a aplikacją sieci wirtualnej zachowując jednocześnie informacje na temat ich lokalizacji i przeznaczenia.

Pojedyncza wiadomość przekazywana między aplikacjami w sieci grid ma postać dokumentu XML. Cała wiadomość jest zamknięta w elemencie o nazwie *Datagram* np.:

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

<!--Informacje na temat odbiorcy-->
   <Recipient Address=”Odra1@Peer1" />

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

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 odbiorcy zapisane są w atrybutach *Address*. Przykładowy adres ma następującą postać: *Odra1@Peer1*. Składa się z dwóch części: pierwsza identyfikuje bazę Odra, druga to identyfikator współuczestnika sieci JXTA, do którego podłączona jest baza. W kolejnym elemencie *Data* umieszczone są dane do przesłania. W atrybutach tego elementu zapisane są dwie informacje.

1) Pierwsza dotyczy typu przesyłanych danych – informacje różnią się dla serwera i klienta. Powodem tych rozbieżności są różne zadania, jakie pełnią obie aplikacje. Klient może wysyłać następujące typy wiadomości:

Query – zapytanie do innej bazy z sieci grid,
Response – odpowiedź na zapytanie z innej bazy z sieci grid,
GetMetaschema – prośba o przesłanie metaschematu przez serwer,
IAmAlive – potwierdzenie zwrotne dla serwera, że baza działa,
DeleteLocalSchema – odłącza schemat lokalny od schematu globalnego,
Message – wiadomość tekstowa.

Odpowiednio serwer wysyła następujące typy:

SetLocalSchema – dołącza schemat lokalny do schematu globalnego,
SetMetaschema – ustawia schemat globalny baz Odra,
SchemaDeleted – informacja potwierdzająca odłączenie schematu lokalnego od schematu globalnego,
AreYouAlive – zapytanie serwera o dostępność klienta,
Message – wiadomość tekstowa.

2) Drugi atrybut o nazwie *Part* ma znaczenie, gdy przesyłana ramka danych ma wielkość większą od wielkości bufora wysyłkowego protokołu TCP/IP. Wtedy wysyłane dane są dzielone na poziomie protokołu i umieszczane w odrębnych datagramach. Dodatkowo w atrybucie *Part* umieszczona zostaje informacja która część danych zawarta jest w ramce oraz ile jest wszystkich części danych. Dla przykładu: *Part=”5/10"* oznacza, że w ramce przesyłana jest piąta część danych, a liczba wszystkich części wynosi 10. Atrybut ten znacząco ułatwia składanie danych w całość z pojedynczych ramek.

Do zapisywania dokumentu w formacie XML użyto klasy *XmlTextWriter* z biblioteki .NET. Klasa ta wspomaga szybkie, bezbuforowe, zapisywanie przez dodawanie słów do strumienia. Zapisany dokument jest zgodny z wymaganiami *W3C Extensible Markup Language (XML) 1.0*.

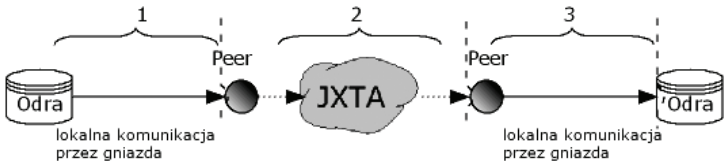
Za nasłuchiwanie i odbieranie ramek z sieci odpowiedzialny jest osobny wątek. Po odebraniu ramki sprawdzany jest jej format, jeżeli ramka posiada nieprawidłową budowę jest ignorowana. Wątek obsługuję sytuację zapełniania bufora odbioru przez większą ilość datagramów. Protokół wyodrębnia pojedyncze ramki identyfikując je po elementach początkowych i końcowych (znacznik *Datagram*). Po odebraniu ramki, aplikacja analizuje ją przy użyciu interfejsu SAX (*Simple API for XML*) zaimplementowanego w bibliotekach .NET.

2.2. Aspekty integracji aplikacji bazy danych oraz aplikacji platformy transportowej w ramach rozproszonego systemu baz danych

Transport komunikatów między bazami danych zrealizowany jest w trzech etapach (rys. 4):

1. przesyłanie informacji do platformy JXTA,
2. przesyłanie w ramach współuczestników wirtualnej sieci JXTA,
3. przekazanie danych do celu przeznaczenia.

Każda operacja wymaga opakowania informacji w odpowiednie ramki i przekazania dalej. Kolejne kroki przepływu informacji prezentuje **Błąd! Nie można odnaleźć źródła odsyłacza.** W miejscach oznaczonych pionowymi przerywanymi liniami dochodzi do przetwarzania danych.



Rys. 4. Etapy integracji baz danych

W projekcie założono, że komunikacja między aplikacją sieci wirtualnej (platformy transportowej), zewnętrzną aplikacją użytkownika (baza danych Odra) – co jest równoznaczne z realizacją komunikacji pomiędzy różnymi językami programowania – odbywać się będzie za pomocą mechanizmu gniazd. Po stronie JXTA utworzono serwer jednowątkowy oczekujący na połączenia ze strony bazy danych Odra zarówno dla aplikacji klienckich, jak i dla serwera centralnego. Po nawiązaniu połączenia tworzony jest kanał umożliwiający dwukierunkową komunikację. Ta część aplikacji ma typową architekturę klient-serwer. Założenie, że serwer jest jednowątkowy określa możliwość połączenia dokładnie z jednym klientem. Takie rozwiązanie pozwala traktować parę „baza – sieć P2P" jako jedną aplikację chociaż zaimplementowaną w różnych językach programowania, umożliwia dodatkowo przekazanie odpowiedzialności obsługi dostępności klientów w sieci grid na poziom sieci wirtualnej JXTA. Odłączenie klienta po stronie bazy danych jest równoznaczne z odłączeniem współuczestnika sieci JXTA (w sieci P2P).

Przetwarzanie ramek danych przychodzących od baz danych do aplikacji platformy transportowej

Aplikacja sieci wirtualnej połączona z aplikacją bazy danych może otrzymywać tylko jeden rodzaj wiadomości od bazy danych. Są to *datagramy* zapisane w formacie XML, które pomiędzy znacznikami <*Datagram*> oraz <*/Datagram*> zawierają dowolną informację. Wnętrze *datagramu* jest przesyłane tylko pomiędzy bazami danych znajdującymi się w sieci grid.

Budując wiadomość dedykowaną innej bazie danych, składa się ją z elementów opisanych w podrozdziale 2.1. Tylko wtedy informacja przenoszona przez tak złożony datagram może być prawidłowo odczytana i zinterpretowana przez inną bazę danych. Z punktu widzenia platformy transportowej datagram posiada jedynie informacje o współużytkowniku źródłowym oraz docelowym, inne informacje dotyczące platformy transportowej nie są przenoszone. Przykład datagramu jest zamieszczony w podrozdziale 2.1.

Po odebraniu wiadomości od bazy danych aplikacja sieci wirtualnej analizuje otrzymaną wiadomość i pobiera z niej dane nadawcy i odbiorcy. Na podstawie przykładowej wiadomości z podrozdziału 2.1 możemy określić:

– nadawcą wiadomości jest baza *Server* podłączona do aplikacji sieci wirtualne o identyfikatorze `Server <Sender Address="Server@Server"/>`,
– adresatem jest baza danych *Odra1* podłączona do aplikacji sieci wirtualnej o identyfikatorze `Peer1 <Recipient Address="Odra1@Peer1"/>`.

Aplikacja platformy transportowej po określeniu, kto jest adresatem wiadomości wysyła element *Datagram* do miejsca przeznaczenia. Jeżeli zdarzyłaby się sytuacja, w której baza docelowa będzie niedostępna, wówczas dostarczenie wiadomości nie będzie możliwe i baza wysyłająca wiadomość (*Server*) otrzyma od aplikacji sieci wirtualnej informacje o niedostępności bazy zdalnej.

Przetwarzanie ramek danych
przesyłanych pomiędzy aplikacjami platformy transportowej

Aplikacja sieci wirtualnej może odbierać i wysyłać komunikaty do innych aplikacji sieci wirtualnej, dla takiego scenariusza zaimplementowano dwa rodzaje wiadomości:

1) *Datagram* – przykład przedstawiony w podrozdziale 2.1;
1) *Command* – wiadomość sterująca siecią P2P, może być przesyłana przez aplikację serwera centralnego sieci P2P, bądź aplikację współuczestnika sieci; np. dodanie do sieci wirtualnej nowego współużytkownika sieci, realizowane przez następującą wiadomość wysłaną od aplikacji współużytkownika sieci wirtualnej do aplikacji serwera centralnego sieci wirtualnej:

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

Aplikacja sieci wirtualnej współużytkownika sieci po odebraniu wiadomości XML typu przedstawionego powyżej, dokonuje jej analizy. Jeżeli odebrana wiadomość jest typu *Datagram* wówczas aplikacja przesyła ją do wyższej warstwy – aplikacji bazodanowej. Jeżeli wiadomość jest typu *Command* rozpoznaje ją jako jeden z wewnętrznych komunikatów sterujących, bądź monitorujących sieć wirtualną. Komunikat zostaję przetworzony zgodnie z jego przeznaczeniem.

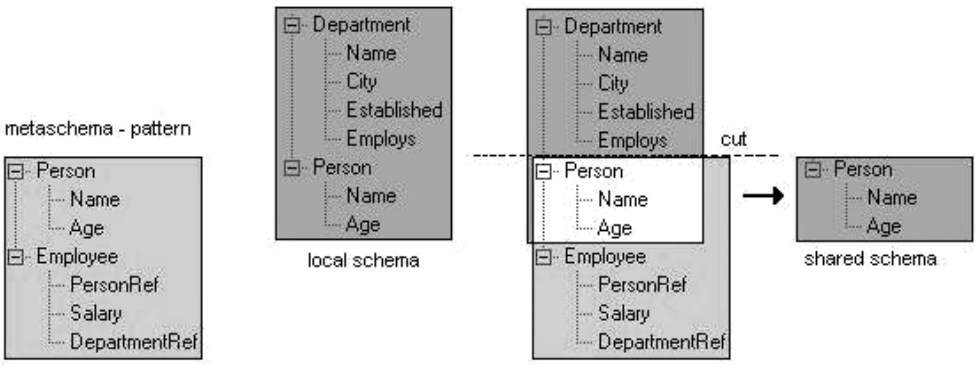
2.3. Integracja schematów baz danych współuczestników sieci grid

Kluczowym aspektem realizacji gridu dla baz danych jest odpowiednie zarządzanie lokalnymi schematami danych. Każda prototypowa baza danych działająca w DataGrid posiada swój schemat lokalny danych zapisany w strukturze podobnej do stosu. Aby zintegrować ten schemat z globalnym schematem gridu, należy zrealizować odpowiednie mapowanie zasobów lokalnych:

– Struktura bazy danych musi zostać przetransformowana do postaci łatwej do przesyłania przez jakikolwiek protokół komunikacyjny – w tym przypadku zastosowano XML

Schema, który jest standardem opisu struktury dokumentu XML. Umożliwia on dokładne opisanie drzewiastej struktury dokumentu, czyli typów danych znajdujących się w elementach i atrybutach. Ponieważ struktura bazy to również drzewo, wykorzystanie tego standardu spełnia stawiane wymagania. Dodatkowo, w przyszłości może przydać się do walidacji poprawności danych wprowadzanych lub odczytywanych z bazy danych do dokumentu XML. Taka konieczność może wystąpić np. przy przenoszeniu danych z jednej bazy do drugiej.

– Odrzucenie danych lokalnych na podstawie schematu lokalnego, które nie mogą zostać udostępnione w sieci. Są to dane prywatne, chronione przed dostępem z zewnątrz. Mechanizm jest analogiczny do udostępniania zasobów w sieci komputerowej. Należy postawić widoczną granicę między danymi udostępnionymi w sieci grid a pozostałymi zasobami. Takiego podziału można dokonać przy odpowiedniej wiedzy o danych publicznych. W tym przypadku realizuje się udostępnianie na podstawie schematu globalnego gridu. Schemat globalny przesyłany jest do współuczestnika sieci na żądanie wysłane do serwera centralnego. Mapowanie informacji jest podobne do wycinania kształtu z kartki papieru po przyłożeniu wzorca. Po wycięciu zostaje jedynie część pasująca do wzorca. Odrzuca się wszystkie dane niepasujące do wzorca, proces ten jest przedstawiony na rysunku 5.



Rys. 5. Proces mapowania schematu lokalnego do gridu na podstawie schematu globalnego

Implementacja mechanizmu z rysunku 5 realizuje analizę schematu lokalnego i metaschematu (schematu globalnego) za pomocą interfejsu DOM XML oraz porównywaniu poszczególnych elementów. Standard DOM reprezentuje dokument XML jako drzewo składające się z określonej hierarchii węzłów (obiektów *Node* lub po nich dziedziczących). W odróżnieniu od SAX, pozwala na tworzenie w pamięci nowych dokumentów. W mechanizmie wykorzystano zapis do pamięci elementów dokumentu XML, następnie dokumenty są porównywane względem zawartości elementów – jeżeli dany węzeł znajduje się w schemacie lokalnym, a nie występuje w schemacie globalnym zostaje usunięty. W re-

zultacie powstaje publiczny schemat, który jest przesyłany do serwera centralnego. Następnie po stronie klienta tworzony jest osobny schemat w bazie danych Odra. Schemat ten odpowiedzialny jest za sprawdzanie poprawności składniowych zapytań pochodzących z sieci grid. Zapobiega to sytuacji, w której zapytanie od innego klienta w sieci mogłoby dotyczyć nieudostępnionych danych.

2.4. Odpowiedzialność centralnego serwera gridu w zarządzaniu rozproszonymi bazami danych

W architekturze DataGrid serwer centralny pełni rolę integratora danych w odróżnieniu od architektury klient-serwer, w której pełni rolę pośrednika komunikacyjnego. Najważniejszym jego zadaniem jest przechowywanie metaschematu gridu, czyli opisu struktury, jaką mają tworzyć połączone bazy danych. Metaschemat przechowywany jest w takiej samej bazie danych, jak te, które pracują w aplikacjach klienckich gridu. Struktura schematu lokalnego jest tworzona przez administratora i tylko przez niego może być zmieniona z poziomu serwera. Metaschemat jest zapisany w postaci XML Schema i jest całkowicie zgodny ze standardem. W przypadku zapisu zawartości bazy danych do postaci dokumentu XML z powodzeniem można zastosować walidator poprawności składniowej. Dokument metaschematu można utworzyć za pomocą edytorów wspierających XML (np. Altova) lub utworzyć wzorcową bazę i zapisać jej strukturę do XML Schema.

Proces mapowania danych jest bardzo prosty:

1. baza danych współuczestnika sieci wysyła żądanie do serwera centralnego o przesłanie zwrotne schematu globalnego gridu (komunikat *GetMetaschema*);
2. serwer centralny po otrzymaniu takiego schematu przygotowuje dane do wysyłki i odsyła je w odpowiednim komunikacie (*SetMetaschema*);
3. współużytkownik sieci, po otrzymaniu danych, dokonuje mapowania schematu lokalnego zgodnie z algorytmem z podrozdziału 2.3;
4. współużytkownik odsyła do serwera centralnego komunikat zawierający jego zintegrowany już gridem schemat;
5. serwer centralny indeksuje zawartość lokalną do dalszego przetwarzania przez innych współużytkowników gridu.

Serwer centralny zawsze otrzymuje od nadawców struktury po mapowaniu, stąd pewność, że ich zawartość pokrywa się ze schematem globalnym lub jest jego częścią. Znajdująca się na serwerze centralnym baza danych Odra nie stanowi zasobu gridu jako zbiór danych. Jest ona wykorzystywana do przechowywania struktury gridu oraz schematów lokalnych baz danych.

Indeks lokalnych schematów na serwerze centralnym posiada zawsze aktualną informację o rozproszeniu obiektów w gridzie. Indeks jest dokumentem XML i posiada następującą postać (gdzie *Odra1*/*2*/*3* to lokalizacje zasobów oznaczonych jako elementy drzewa):

```
<Odra>
   <Department>
      <Item>Odra1</Item>
      <Item>Odra2</Item>
      <Item>Odra3</Item>
   </Department>
   <Person>
      <Item>Odra1</Item>
      <Item>Odra2</Item>
   </Person>
   <House>
      <Item>Odra2</Item>
      <Item>Odra3</Item>
   </House>
</Odra>
```

Indeks w takiej postaci jest rozsyłany do wszystkich aktywnych klientów gridu. Na jego podstawie oraz w oparciu o schemat globalny współuczestnicy sieci są w stanie odtworzyć kompletną strukturę sieci oraz lokalizację zasobów w węzłach, jest więc jednym z najważniejszych elementów sieci DataGrid. Sam indeks jest niewielki i zawiera wiele informacji o gridzie. Zostało to osiągnięte poprzez wyeliminowanie zbędnych, powtarzających się informacji i odpowiednią strukturę zawartości indeksu.

3. Podsumowanie

Opisane prace implementacyjne pozwoliły na podsumowanie badań dotyczących realizacji sieci grid zorientowanej na przetwarzanie danych (*data-intensive*). Koncepcja budowy platformy transportowej opartej o architekturę sieci „każdy z każdym" (*Peer-to-Peer*) została potwierdzona budową uproszczonego modelu DataGrid opisanego w niniejszym artykule. Prototyp modelu realizuje rozproszoną komunikację dla obiektowych baz danych, jednak aby osiągnąć zamierzony cel podejścia DataGrid, należy zapewnić pełną przezroczystość [7]. Jest to trudne do osiągnięcia bez zastosowania odpowiedniego mechanizmu. Dalsze implementacje będą integrować do istniejącego rozwiązania obiektowe aktualizowalne perspektywy (*updateable object views*) [3]. Propozycje takiego rozwiązania są przedstawione w [7].

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.: *Updateable Object Views.* PhD Thesis, 2005, http://www.ipipan.waw.pl/~subieta/, Finished PhD-s

[4] Kozankiewicz H., Stencel K., Subieta K.: *Implementation of Federated Databases through Updateable Views.* Proc. EGC 2005 – European Grid Conference, Springer LNCS, 2005
[5] 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
[6] Kuliberda K.: *Podstawowe mechanizmy architektury Grid Computing*. Półrocznik AGH, Automatyka, t. 8, z. 3, 2004, 505–515
[7] 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, 2006
[8] 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
[9] Kuliberda K., Wiślicki J.: *Architektura gridu bazodanowego – data grid.* Półrocznik AGH, Automatyka, t. 9, z. 3, 2005, 505–515
[10] 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
[11] 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
[12] Subieta K.: *Teoria i konstrukcja obiektowych języków zapytań.* Wydawnictwo PJWSTK, Warszawa 2004, ISBN 83-89244-28-4 (522 strony)
[13] Project JXTA Community: http://www.jxta.org