

KAMIL KULIBERDA
RADOSŁAW ADAMUS
TOMASZ MAREK KOWALSKI
JACEK WIŚLICKI

Wydział Elektrotechniki Elektroniki Informatyki i Automatyki
Politechniki Łódzkiej

ARCHITEKTURA GRIDU BAZODANOWEGO OPARTA O PODEJŚCIE PEER-TO-PEER

Recenzent: **prof. Dominik Sankowski**

Maszynopis dostarczono: 1. 10. 2010

W artykule autorzy przedstawiają, zaimplementowane i zweryfikowane poprzez w pełni funkcjonalny prototyp, podejście do realizacji obiektowego gridu bazodanowego przy wykorzystaniu wirtualnej sieci peer-to-peer omawiając szczegółowo najważniejszy element mechanizmu jakim jest proces integracji rozproszonych obiektów. W artykule przeprowadzono gruntowną analizę rozwiązań będących fundamentalnym zbiorem wiedzy na temat integracji danych. Zaprezentowano trójwarstwowy model integracyjny oparty o aktualizowalne obiektowe perspektywy oraz prototyp gridowej warstwy pośredniej wykorzystującej sieć wirtualną peer-to-peer.

1. WPROWADZENIE

Termin *grid* znany jest jako termin określający rozproszone sieci obliczeniowe, jednak szybka ewolucja Internetu, powiększanie się społeczności internetowych, globalny wzrost wymiany informacji przez Internet wytworzyły potrzebę przetwarzania danych w architekturze rozproszonej i tym samym rozwój systemów gridowych w stronę przetwarzania danych opisanych

modelami biznesowymi – danych o konkretnej strukturze. Obecnie popularne rozwiązania jak sieci P2P (*peer-to-peer*), w których możliwe jest równoległe przetwarzanie dużych ilości danych w postaci mediów, czy plików nie wspierają zarządzania danymi strukturalnymi. Taki typ danych przechowywany jest w bazach danych.

Przetwarzanie danych pochodzących ze źródeł, którymi są bazy danych, stwarza obecnie duże problemy. Są one związane z odpowiednim fizycznym traktowaniem tych danych oraz ze sposobem jak te dane widzi użytkownik – na co składa się dostęp do tych danych oraz sposoby ich przetwarzania. Główną trudnością w przetwarzaniu jest tutaj właśnie struktura takich danych. Dane biznesowe zazwyczaj charakteryzują się złożoną strukturą, przez co nie mogą być identyfikowane oraz przetwarzane jak zwykły ciąg bajtów. Często ich struktura może być zależna od innych struktur, stąd w systemie rozproszonym zarządzanie jest bardzo trudne, a w niektórych przypadkach niemożliwe. Przetwarzanie rozproszonych danych strukturalnych – opisanych modelem biznesowym, związane jest z budową modelu zdolnego realizować równoległe przetwarzanie rozproszonych danych, gdzie różnego typu dane oraz usługi znajdujące się w fizycznie odseparowanych od siebie lokalizacjach mogą być wirtualnie dostępne przez ich wirtualną reprezentację.

W systemach rozproszonych wymagane jest osiągnięcie *transparentności*, tzn. aby użytkownik pracując na danych mógł je przetwarzać bez względu na to czy są to dane lokalne znajdujące się na lokalnym komputerze użytkownika, czy też dane pobierane z lokalizacji zdalnych. Dodatkowo poszczególne lokalizacje skąd dane są pobierane zazwyczaj są systemami heterogenicznymi. Stawia to dodatkowe wyzwanie dla projektantów takich systemów w postaci realizacji mechanizmu integracji takich zasobów.

Mówiąc o przetwarzaniu danych mamy na myśli nie tylko ich odczyt, ale także swobodną ich aktualizację. Właśnie możliwość swobodnej modyfikacji danych rozproszonych przy założeniu, że użytkownik wprowadzający tą modyfikację nie jest nawet świadom, że działa na danych zdalnych jest najważniejszym problemem nierozwiązanym w innych istniejących systemach rozproszonych.

Przy powyższych założeniach rozproszony system baz danych, nazywany *data-intensive* lub *data-grid*, musi mieć zapewnioną ciągłość pracy i łatwość dostępu do danych, aby to zrealizować musi to być zapewnione już na poziomie architektury samego gridu. Dlatego wymaga to realizacji bardzo elastycznej łatwo skalowalnej architektury.

W niniejszym artykule, powyższe problemy poddane zostały dyskusji oraz zaproponowano ich rozwiązanie w kontekście architektury *data-grid*. Za cel ustalono realizację gridu baz danych cechującego się:

- przezroczystością dostępnych zasobów gridu przy ich przetwarzaniu,
- automatyczną integracją zasobów dołączających do gridu,
- wirtualną siecią łączącą współpracujące bazy danych.

2. PODSTAWY TECHNOLOGII GRIDOWYCH

Ewolucja systemów gridowych nakreśliła podejścia do przetwarzania, w sieci, danych rozproszonych w zależności od ich fizycznej postaci. Najbardziej skomplikowanym modelem danych są dane posiadające wewnętrzną strukturę odwzorowujące określony model biznesowy. Obecnie systemy gridowe które stosowane są do przetwarzania tego typu danych zaliczane są do systemów tzw. 3-ciej generacji gridów lub inaczej *Future Grids*. Systemy gridowe 3-ciej generacji są stosunkowo łatwo rozpoznawalne, charakteryzują się one tzw. *warstwą pośrednią* (ang. *middleware*), w której zaszyte są wszystkie mechanizmy wspierające pracę w środowisku rozproszonym (nie tylko dla danych).

Podstawowym wyzwaniem dla tego typu gridów jest zapewnienie i utrzymanie komunikacji pomiędzy zasobami sieci i ich użytkownikami. Termin zasobu ma szerokie znaczenie w kontekście systemów rozproszonych. Zasobami mogą być węzły obliczeniowe (klastry, superkomputery, serwery) oraz urządzenia do przechowywania danych. Należy pamiętać, że zasób nie musi reprezentować sprzętu lub innego zasobu fizycznego. Zasób to bardzo często aplikacja, która jest odpowiedzialna za kontrolowanie i udostępnianie konkretnego sprzętu komputerowego, wtedy mówi się o niej jako o usłudze (w rozumieniu zasobu). Dobrym przykładem może być tutaj usługa, która umożliwia dostęp do danych z bazy danych lub po prostu aplikacja, która umożliwia dostęp do systemu plików. W takim wypadku mechanizm komunikacji w gridzie musi zapewnić maksymalne wykorzystanie takiego zasobu, co jest trudnym zadaniem biorąc pod uwagę różnorodność zasobów. Stworzenie warstwy pośredniej jest typowym podejściem do rozwiązania omawianego problemu. Warstwa pośrednia w większości przypadków realizuje protokoły komunikacyjne i odpowiada za interakcje między nimi oraz wyższymi warstwami realizującymi zadania gridowe. Jest to bardzo elastyczne i ogólne podejście, które także daje możliwości rozwiązywania innych wyzwań stojących przed systemami gridowymi. Warstwa pośrednia jest mechanizmem bardzo ogólnym w porównaniu do wyższych warstw, które jako specjalistyczne oprogramowanie odzwierciedlają potrzeby użytkowników - użytkownicy są zwykle zainteresowani konkretnymi rozwiązaniami, które dostarczane są przez poszczególne funkcjonalności gridu w sposób przyjazny dla użytkownika.

W kontekście baz danych systemy gridowe, aby w pełni mogły być uznane jako rozwiązania gridowe muszą ponadto realizować szereg funkcji, są to przede wszystkim:

- Optymalizacja zasobów - systemy gridowy może pomóc w wyborze optymalnego (w różnych aspektach) zasobu do pracy;
- Łatwość obsługi - warstwa klienta powinna ułatwiać zdalne wykonanie najprostszych zadań;
- Zmniejszenie kosztów administracyjnych - usługa monitorowania systemu, wyższe zabezpieczenia dostępu wraz z narzędziami do instalacji różnych komponentów gridu. Może to ułatwiać wdrażanie aplikacji rozproszonych, a także zarządzanie oraz kontrolę systemu;
- Skalowalność - systemy gridowe muszą udostępniać mechanizmy pozwalające na automatyczny, łatwy, proces rozszerzenia swojej struktury fizycznej poprzez np. dodatkowe zasoby.

Przyglądając się najczęściej powtarzającym się problemom, akcentowanym w prasie fachowej, reklamach i badaniach naukowych [6], wynika, że szybki wzrost ilości danych jest zjawiskiem powszechnym. W rezultacie, jest to silny bodziec do rozwoju systemów gridowych. Technologicznie ludzkość jest gotowa do przechowywania tak dużej, i wciąż rosnącej, ilości danych w postaci rozproszonych zasobów, jednak, to również zmusza do zarządzania tymi danymi, co w końcowym efekcie porusza kwestie takie jak; przeglądanie danych, wyszukiwanie, filtrowanie, transfer danych i transformacja danych. Prawdziwe problemy zaczynają jeżeli dane te tworzą złożone struktury w pojęciu modeli biznesowych i muszą być dostępne do przetwarzania w rozproszonym globalnym środowisku gridowym. Główne problemy to:

- Prezentacja danych, które pochodzą z różnych heterogenicznych zasobów posiadających różne struktury - w aspekcie przyjętych modeli biznesowych, integracji danych pochodzących z różnych zasobów i ich globalnego przetwarzania;
- Przejrzysty dostęp do danych;
- Wyszukiwanie danych;
- Filtrowanie danych;
- Wiele innych procesów, które w dużej skali są trudne do realizacji bądź w ogóle niemożliwe.

Jedną z cech współczesnych organizacji jest to, iż różne ich jednostki organizacyjne używają różnych systemów do tworzenia, przechowywania i przeszukiwania danych mających dla nich jakieś znaczenie. Różnorodność źródeł danych wiąże się z brakiem koordynacji, różnym tempem adopcji nowych technologii, geograficznym oddaleniem, jak i łączeniem się ze sobą różnych firm. Jedynie poprzez integrację tych wszystkich systemów organizacje mogą skorzystać z pełnej wartości należących do nich danych. Wraz z rozwojem systemów informatycznych oraz sieci Internet, coraz częściej mówi się również o

integracji aplikacji - zarówno należących do tego samego przedsiębiorstwa (*Enterprise Application Integration, EAI*), jak i różnych organizacji (*Business To Business, B2B*) [7], [8].

Integracja danych i aplikacji może być realizowana na poziomie fizycznym za pomocą wielu różnorodnych technik. Ze względu na dążenie do zmniejszenia kosztów wytwórczych oprogramowania, obecnie odchodzi się raczej od implementacji własnych, specyficznych dla konkretnych systemów protokołów, dążąc do wykorzystania istniejącego już, uniwersalnego oprogramowania ułatwiającego komunikację między aplikacjami - tzw. *middleware*. Na przestrzeni lat zaprojektowano wiele rodzajów warstw pośrednich (*middlewares*): rozproszone obiekty, serwery aplikacyjne, kolejki komunikatów, serwery integracyjne, monitory transakcyjne, i in. Różne rodzaje warstw pośrednich pozwalają wykorzystać różne strategie integracyjne (integracja zorientowana na dane, na usługi, na komunikaty, na procesy biznesowe, poprzez portale itd.), wykorzystując do tego celu różnego rodzaju metody łączenia integrowanych elementów (połączenia point-to-point, albo many-to-many) oraz scenariusze komunikacji (komunikacja synchroniczna i asynchroniczna).

Jednym z bardziej znanych rodzajów warstw pośrednich są rozproszone obiekty, których prawdopodobnie najbardziej znanym reprezentantem jest standard CORBA [9]. Rozproszone obiekty definiowane są w tym standardzie jako fragmenty większych aplikacji, zaprojektowanych do wzajemnej współpracy, jednak działającymi na zupełnie odrębnych maszynach. Podstawowymi zaletami rozproszonych obiektów są: obiektowy model danych, zgodność z logiką biznesu, stosunkowo wysoki poziom abstrakcji, szereg usług (np. transakcyjność) dostępnych dla programistów, standardowy protokół wymiany danych pomiędzy elementami systemu rozproszonego, niezależność od użytego w implementacji języka programowania. Programista tworzący aplikację opisuje jej interfejs dostępny dla zdalnych klientów za pomocą specjalnego, niezależnego od implementacji języka - IDL (*Interface Definition Language*). Niestety, z czasem okazało się iż ta unifikacja metod dostępu do danych oraz łatwość dostępu do danych spowodowała u programistów pokusę ignorowania ograniczeń nakładanych przez sieć komputerową (czas dostępu, awaryjność) i projektowanie aplikacji rozproszonych w taki sam sposób, jak gdyby były to aplikacje nierozproszone. Przykładowo, naiwna implementacja operacji przetworzenia kolekcji zdalnych obiektów w sposób proceduralny (za pomocą iteratorów) oznacza wielokrotną wymianę danych między serwerem i klientem, proporcjonalną do ilości obiektów.

Ta cecha, w połączeniu z niespójnością samego standardu, złożonym API, niekompatybilnością pomiędzy oprogramowaniem dostarczonym przez różnych dostawców i innymi poważnymi problemami stała się przyczyną zmniejszającej się popularności standardu CORBA.

Potrzeby niektórych organizacji (np. instytutów naukowo-badawczych) wymagają integracji dostępnych w nich zasobów na bardzo niskim poziomie, tzn. poziomie prostych danych składowanych w bazach danych. Dla organizacji takich integracja danych za pomocą usług, czy z wykorzystaniem rozproszonych obiektów może być nieakceptowalna ze względu na ich zbyt wysokopoziomowy charakter implikujący pewną "sztywność" w sposobie dostępu. Alternatywnym rozwiązaniem może być budowa scentralizowanej bazy danych podobnej do hurtowni danych, jednak rozwiązanie to może być nieakceptowalne ze względu na koszt, czas dostępu do najbardziej aktualnej wersji danych, i in. Integracja zasobów kilku baz danych za pomocą federacji może okazać się najbardziej bezinwazyjną metodą integracji systemów informatycznych (np. w przypadku przejęcia jednej firmy przez drugą).

Federacyjna baza danych [10, 11] jest logicznym powiązaniem niezależnych od siebie, rozproszonych baz danych, tworzącym pojedynczy, zintegrowany system bazodanowy. Integrowane w ten sposób źródła danych mogą być nie tylko typowymi bazami danych (np. obiektowymi, relacyjnymi, repozytoriami XML) różnorodnych producentów, ale również płaskimi plikami, dokumentami tekstowymi, plikami arkuszy kalkulacyjnych, oraz wieloma innymi rodzajami danych ustrukturalizowanych i nieustrukturalizowanych. Architektura federacyjna powoduje że wszystkie te dane widoczne są jako jedna, wirtualna całość (stąd czasem używana nazwa - wirtualna baza danych).

Integrowane bazy danych udostępniają szereg swoich zasobów całej federacji. Zasoby te mogą być metadanymi (schematy bazodanowe), zwykłymi danymi, czy interfejsami programistycznymi umożliwiającymi korzystanie z takiej bazy danych. Suma udostępnionych w ten sposób danych razem z centralną, integracyjną bazą danych tworzy całą infrastrukturę federacyjną. Zintegrowane bazy danych udostępniają część lub całość swojej zawartości innym członkom federacji, pozostając jednak autonomię w ich lokalnym zarządzaniu.

Nowe źródła danych mogą być dodawane do federacji poprzez utworzenie dla nich tzw. osłon (ang. wrappers). Osłony są relatywnie nieskomplikowanymi, ale niskopoziomowymi programami umożliwiającymi fizyczne połączenie z federacją różnorodnych, heterogenicznych źródeł danych. Przykładowo, zadaniem osłony dla plików programu Microsoft Excel powinno być zaimplementowanie pewnego API umożliwiającego odczytywanie tych plików i dynamicznym udostępnianiu jego zawartości dla oprogramowania sterującego funkcjonowaniem federacyjnej bazy danych (np. w formie sterownika JDBC) zgodnie z modelem danych przyjętym dla niej.

Architektura federacyjnej bazy danych często obejmuje także komponenty zwane mediatorami. Mediator jest specjalnym modułem oprogramowania umieszczanym po stronie integrowanego zasobu. Jego zadaniem jest takie

przekształcenie lokalnych danych, by mogły być one wykorzystane przez globalnego użytkownika zgodnie z pewnymi regułami przyjętymi dla całej federacji. Mediator tłumaczy zapytanie zgodne z globalnym schematem federacji na taką jego formę, by mogło być one wykonane na danych lokalnych. Oprócz przekształceń związanych z różnymi schematami danych, przekształcane mogą być również same dane. Przykładem takiego zastosowania mediatorów jest dynamiczna (wirtualna) konwersja pensji z waluty używanej lokalnie (np. PLN) do waluty używanej w całej federacji (np. USD). Nawet taki wydawałoby się banalny problem wymaga rozstrzygnięć i umów, np. ustalenia serwisu bankowego wg którego na bieżąco będzie przeliczana waluta, czy zamiana aktualizacji wyrażonej w walucie obowiązującej w federacji na walutę obowiązującą w lokalnej walucie.

Krytyczną cechą federacyjnych baz danych jest stopień w jaki system taki jest w stanie upodobnić się do scentralizowanej bazy danych, oraz ukryć złożoność mechanizmów związanych z integracją danych w heterogenicznym i rozproszonym środowisku. Najczęściej mówi się o następujących poziomach przezroczystości w federacyjnych baz danych:

- Przezroczystość dostępu, czyli dostarczenie jednorodnych metod operowania na danych lokalnych i odległych,
- Przezroczystość położenia, czyli uwolnienie użytkowników od konieczności posiadania wiedzy na temat fizycznej lokalizacji danych w systemie rozproszonym,
- Przezroczystość współbieżności, czyli umożliwienie wielu użytkownikom jednoczesnego dostępu do danych przy zachowaniu pełnej spójności danych, bez konieczności umawiania się, czy niskopoziomowego programowania mechanizmów synchronizacyjnych,
- Przezroczystość heterogeniczności, czyli umożliwienie jednolitego traktowania danych pochodzących z różnych źródeł, zapisanych tam za pomocą różnych modeli danych,
- Przezroczystość skalowania, czyli umożliwienie dodawania nowych elementów systemu rozproszonego bez wpływu na działanie starych aplikacji i pracę użytkowników,
- Przezroczystość fragmentacji, czyli automatyczne scalanie obiektów, tabel lub kolekcji, których fragmenty przechowywane są w różnych miejscach,
- Przezroczystość replikacji, czyli umożliwienie tworzenia i usuwania kopii danych w innych miejscach geograficznych z bezpośrednim skutkiem dla efektywności przetwarzania, ale bez skutków dla postaci programów użytkowych lub pracy użytkownika końcowego,
- Przezroczystość optymalizacji, czyli możliwość wykorzystania bez wiedzy użytkownika szeregu strategii optymalizacyjnych czasu kompilacji lub wykonania, umożliwiających przyspieszenie wykonywania zapytań realizowanych na rozproszonej bazie danych,

- Przezroczystość awarii, czyli umożliwienie nieprzerwanej pracy większości użytkowników rozproszonej bazy danych w sytuacji, gdy niektóre z jej węzłów lub linie komunikacyjne uległy awarii,
- Przezroczystość migracji, czyli umożliwienie przenoszenia zasobów danych do innych miejsc bez wpływu na pracę użytkowników.

Wirtualne repozytorium jest mechanizmem umożliwiającym wirtualną prezentację danych fizycznych w postaci jednego globalnego schematu danych. Głównym zadaniem wirtualnego repozytorium jest ukrycie złożoności mechanizmów związanych z dostępem do lokalnych źródeł danych. W wirtualnym repozytorium realizowany jest proces mapowania fizycznych źródeł danych poprzez aktualizowalne obiektowe perspektywy [1, 4]. Użytkownik widzi dane z repozytorium w postaci schematu obiektów jaki został zaimplementowany poprzez schemat globalny. Perspektywa przez którą realizowana jest wirtualizacja implementuje podstawowe operacje na obiektach - CRUD (*Create, Read, Update, Delete*), które mogą zostać dowolnie rozszerzone zgodnie z logiką biznesową udostępnianego schematu danych. Kod perspektywy realizowany jest za pomocą języka SBQL [2], a dzięki deklaratywnemu charakterowi języka SBQL, te złożone mechanizmy często mogą być wyrażone w jednej linii kodu. Ważną cechą wirtualnego repozytorium jest jego działanie na bardzo rozproszonej architekturze zasobów danych.

ODRA (*Object Database for Rapid Application development*) [3] jest prototypem obiektowego środowiska programistycznego tworzonego w Polsko-Japońskiej Wyższej Szkole Technik Komputerowych. Jest to narzędzie nowej generacji przeznaczone dla programistów baz danych. Narzędzie to jest oparte o język SBQL [12]. Środowisko uruchomieniowe języka SBQL w ODRZE składa się z maszyny wirtualnej, która jest systemem zarządzania bazą danych działającej w pamięci komputera (ang. *main memory DBMS*) z infrastrukturą wspierającą rozproszenie danych. Głównym celem projektu ODRA jest opracowanie nowego modelu rozwoju aplikacji bazodanowych, można to osiągnąć poprzez zwiększenie poziomu abstrakcji na którym pracuje programista przy zastosowaniu nowego, uniwersalnego, deklaratywnego języka programowania, wraz z rozproszonym środowiskiem, zorientowanym na obiektową bazę danych. Takie podejście zapewnia wspólne funkcjonalności dla wielu popularnych technologii (takich jak relacyjno-obiektowe bazy danych, różne typy warstw pośrednich, języki programowania ogólnego przeznaczenia wraz z ich środowiskami uruchomieniowymi) w jednym, uniwersalnym, łatwym do nauczenia, interoperacyjnym i wygodnym do użycia środowisku programistycznym. ODRA zawiera następujące, zasadnicze rozwiązania realizujące cel projektu:

1. *Architekturę obiektową.* Podejście do danych w ODRZE różni się od obecnego sposobu postrzegania obiektowych baz danych, repre-

zowanego głównie przez standard ODMG [13] oraz technologiami bazodanowymi związanymi z językiem Java (np. [13], [14]). System opiera się na metodologii SBA ([2], [12]). Pozwala to na wprowadzenie do programowania bazy danych wszystkich popularnych mechanizmów obiektowych (np. obiekty, klasy, dziedziczenie, polimorfizm, enkapsulacja), jak również pewne mechanizmy nieznanne wcześniej (jak dynamiczne role obiektów [15], [16] lub oparte o interfejsy perspektywy bazy danych [1], [17]).

2. *Język zapytań rozszerzony do języka programowania.* Najważniejszą cechą bazy ODRA jest SBQL - obiektowy język zapytań i programowania. SBQL różni się od języków programowania i od znanych języków zapytań, ponieważ jest to język zapytań z taką samą pełną mocą obliczeniową jak popularne języki programowania. SBQL sam w sobie umożliwia stworzenie samodzielnej aplikacji zorientowanej na działanie z bazą danych.
3. *Wirtualne repozytorium jako warstwę pośredniczącą.* W środowisku sieciowym, możliwe jest podłączenie wielu komputerów na których działa systemem ODRA. Wszystkie systemy związane w ten sposób mogą współdzielić zasoby heterogenicznych, dynamicznie się zmieniającym, ale niezawodnym i bezpiecznym środowisku. Takie podejście do przetwarzania rozproszonego jest oparte o obiektowe, wirtualne, aktualizowalne perspektywy bazy danych [4]. Technologia ta może być postrzegana jako kontrybucja i nawiązanie do rozproszonych baz danych, *Enterprise Application Integration (EAI)*, systemów gridowych i peer-to-peer.

3. METODOLOGIA AUTOMATYCZNEJ INTEGRACJI ROZPROSZONYCH DANYCH OBIEKTOWYCH

Ten rozdział składa się z trzech części, w pierwszej części omówiona jest ogólna koncepcja integracji danych między heterogenicznymi rozproszonymi bazami danych w architekturze data-grid. Druga część przedstawia propozycję realizacji metody automatycznej integracji heterogenicznych rozproszonych obiektowych baz danych. Trzecia część zawiera przykład perspektywy integrującej rozproszone obiekty.

3.1. Koncepcja trójwarstwowego modelu integracji rozproszonych danych

Niniejsze podejście do integracji heterogenicznych i autonomicznych zasobów, opiera się na podejściu zastosowanym w federacyjnych bazach danych - w którym schemat globalny jest reprezentacją danych dostępnych dla klientów systemu. Co więcej, proponowana koncepcja rozszerza rozwiązanie użyte w projekcie eGov-Bus [3] o dodatkową warstwę integrującą rozproszone obiekty, która z punktu widzenia rozproszenia odpowiedzialna jest za automatyczne "sklejanie" fizycznie rozproszonych obiektów i nieprzerwane działanie systemu w czasie przetwarzania danych.

Celem naszego rozwiązania integracji w architekturze *data-grid* jest stworzenie w pełni automatycznego procesu integracji rozproszonych i heterogenicznych obiektów, które również muszą spełniać warunek przezroczystości (w aspekcie dostępu) dla użytkowników, jednocześnie minimalizując aktywność użytkowników w procesie udostępniania/dostępu do obiektów. Minimum, które musi wykonać użytkownik to przygotowanie mapowania lokalnych obiektów swojej bazy danych do schematu obiektów w wirtualnym repozytorium według określonych reguł, a następnie przyłączyć się do wirtualnego repozytorium. pozostała część integracji danych zostanie wykonana automatycznie przez mechanizmy integracji umieszczone w odpowiednio przygotowanej warstwie pośredniej systemu gridowego.

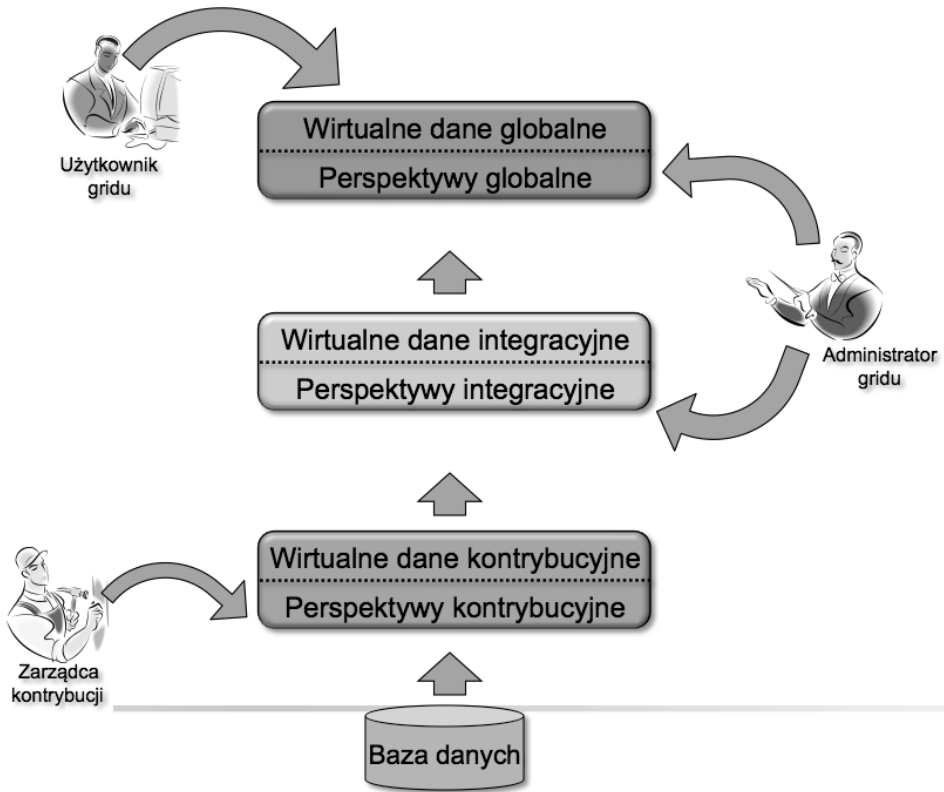
Odwzorowanie obiektów lokalnych do schematu wirtualnego repozytorium, musi być wykonane za pomocą aktualizowalnych obiektowych perspektyw. W procesie tym użytkownik musi utworzyć *perspektywę kontrybucyjną* w swoim lokalnym środowisku bazodanowym. Podczas tworzenia perspektywy użytkownik musi dokonać odwzorowania opierając się na informacji kontrybucyjnej pochodzącej ze środowiska gridowego. Taka informacja jest udostępniana przez konsorcjum gridowe np. formie opisu obiektu za pomocą dokumentu UML.

W bazach danych - w kontekście aplikacji rozproszonych (np. aplikacje WWW) [1], perspektywa oznacza dowolnie zdefiniowany obraz przechowywanych danych. Perspektywy mogą być użyte do zlikwidowania niezgodności pomiędzy heterogenicznymi źródłami danych poprzez ich wspólną integrację, co również zgodne jest z pojęciem mediacji [18]. Perspektywy w bazach danych są rozumiane jako dane zmaterializowane (co tworzy kopię wybranych danych) oraz dane wirtualne (dostępne poprzez definicję odwołującą się do danych fizycznych często o innej strukturze). Typowa definicja perspektywy jest procedurą, która może być wywołana poprzez zapytanie. Jedną

z najważniejszych cech perspektyw bazodanowych jest ich przezroczystość, co oznacza, że zapytanie ewaluujące obiekty z perspektywy niczym nie różni się od zapytania ewaluującego rzeczywiste obiekty - użytkownik tworząc zapytanie nie jest świadomy że do ewaluacji zapytania wykorzystuje perspektywę. Stąd, model danych oraz składnia języka zapytań dla perspektyw musi być zgodna z modelem danych oraz składnią języka dla danych fizycznych.

W prezentowanym podejściu zakładamy, że przy wykorzystaniu trzech oddzielnych warstw perspektyw można zrealizować kompletny i spójny mechanizm integracji dla wirtualnego repozytorium. Każda warstwa perspektywy będzie realizowała inny zakres zadań integracyjnych, a każda wyższa warstwa będzie zależna od warstwy niższej - działanie warstw będzie odpowiadało modelowi hierarchicznemu. Od strony klienta gridu poszczególne warstwy będą używane od najwyższej do najniższej. Każda warstwa może zawierać nieograniczoną liczbę niezależnych perspektyw. Opisywany model przedstawiony jest na rysunku 1.

Uproszczając podejście, założymy że użytkownik chce kontrybuować swoje lokalne obiekty do wirtualnego repozytorium. Lokalny schematu użytkownika zawiera tylko obiekty *Person* (obiekty te reprezentują pracowników firmy użytkownika). Obiekt *Person* zawiera podobiekty *FirstName* i *LastName* co przedstawiono na rysunku. 2. Od strony wirtualnego repozytorium, zgodnie ze schematem globalnym obiekty *Person* muszą być dostępne jako wirtualne obiekty *Employee*, które posiadają podobiekty *Name* oraz *Surname* - jest to pokazane na rysunku 3. Oznacza to, że użytkownik musi utworzyć perspektywę kontrybucyjną, która dokona odwzorowania pomiędzy rzeczywistymi i wirtualnymi obiektami zgodnie z odpowiednią definicją.



Rys. 1. Trójwarstwowy model integracji rozproszonych obiektów do gridu

Person
FirstName LastName

Rys. 2. Przykładowy obiekt Person

Employee
Name Surname

Rys. 3. Przykładowy obiekt Employee

Dla wyżej przedstawionego przykładu, zawartość perspektywy kontrybucyjnej dokonującej mapowania (w zależności od implementacji składni

aktualizowalnych obiektowych perspektyw w języku SBQL) powinna wyglądać tak jak jest to przedstawione na Listingu 1:

Listing 1. Przykładowa definicja perspektywy kontrybucyjnej dla mapowania obiektów Person i Employee

```
view EmployeeContribDef {  
  
  /* virtual object declaration */  
  virtual objects Employee: record {  
    Name: string;  
    Surname: string;  
  }[0..*] ;  
  
  /* definition of seeds for virtual objects */  
  seed: record {  
    p: Person; } [0..*] {  
    return (Person) as p;  
  }  
}  
  
/* definitions of CRUD procedures for virtual objects */  
on_retrieve {  
  return p.(  
    FirstName as Name,  
    LastName as Surname;  
  )  
}  
on_delete {  
  delete p;  
}  
on_update {  
  p := value.(  
    Name as FirstName,  
    Surname as LastName;  
  )  
}  
on_new {  
  create permanent Person(value.(  
    Name as FirstName,  
    Surname as LastName);  
  )  
}  
  
/* sub-view definitions */  
view NameDef {  
  virtual Name: string;  
  seed: record {  
    _name: Person.FirstName;  
  }  
  {  
    return p.FirstName as _name;  
  }  
  on_retrieve { return _name; }  
  on_update { _name := value; }  
}  
view SurnameDef {
```

```
virtual Surname: string;
seed: record {
  _surname: Person.LastName;
}
{
  return p.LastName as _surname;
}
on_retrieve { return _surname; }
on_update { _surname := value; }
}
```

Odwzorowanie obiektów kontrybucyjnych może być zrealizowane tylko wtedy kiedy odwzorowane obiekty wirtualne będą pasować do schematu, który będzie akceptowany przez perspektywę globalną. Głównym zadaniem *perspektywy kontrybucyjnej* jest przekształcanie dostępnych danych fizycznych w dane wirtualne zgodne ze schematem wymaganym i dozwolonym przez wirtualne repozytorium. Jak to jest przedstawione na Listingu 1 podstawowe procedury CRUD również wchodzą w skład definicji perspektywy. Implementacja tych procedur jest bardzo ważna dla dalszych działań, które będą wykonywane na wirtualnych obiektach kontrybucyjnych. Przy realizacji kontrybucji muszą być określone jakie operacje dozwolone są na obiektach kontrybucyjnych. Do warstwy kontrybucyjnej dostęp ma właściciel danych kontrybuowanych - tzw. zarządca kontrybucji, jest to osoba z uprawnieniami dostępu do bazy danych i składowanych obiektów w bazie danych. Osoba taka musi posiadać również uprawnienia do realizacji kontrybucji - jest to przedstawione na rysunku 1.

Perspektywa integracyjna jest umieszczona w środkowej warstwie omawianego modelu integracji (rysunek 1). Perspektywa ta musi zostać zaprogramowana przez administratora wirtualnego repozytorium, a schemat danych dostępny przez tą perspektywę musi być zgodny ze schematem wirtualnego repozytorium, przy czym użytkownik gridu nie ma dostępu do perspektywy integracyjnej, nie wie nawet o jej istnieniu. Zadaniem perspektywy integracyjnej jest przechowywanie informacji na temat rozproszonych zasobów kontrybucyjnych, które są częścią wirtualnego repozytorium. Definicja perspektywy integracyjnej zawiera specjalne struktury w których przechowywane są informacje o integracji zdalnych obiektów oraz ich fragmentacji, redundancji, itd. Struktury te są zarządzane automatycznie przez specjalny mechanizm zaszyty w warstwie pośredniej gridu. Na bazie wspomnianych struktur tworzone są kolekcje wirtualnych obiektów integracyjnych, których składowymi są zdalne wirtualne obiekty kontrybucyjne. Perspektywy integracyjne w czasie pracy gridu poddawane są wielokrotnej modyfikacji zawsze gdy zmienia się ilość kontrybuujących zasobów gridu. Tak

jak perspektywy kontrybucyjne, perspektywy integracyjne zawierają definicje operacji CRUD.

W najwyższej warstwie trójwarstwowego modelu integracji umieszczona jest *perspektywa globalna* (rysunek 1), która ostatecznie określa kształt globalnych wirtualnych obiektów dostępnych dla klientów w wirtualnego repozytorium. Perspektywa globalna jest statyczną definicją utworzoną i zarządzaną przez administratora gridu. Perspektywa globalna jest automatycznie propagowana w czasie podłączania się użytkownika do gridu, a jej schemat jest bezpośrednio dostępny dla użytkownika gridu. Definicja perspektywy globalnej powinna być zrealizowana zgodnie z umowami wirtualnego repozytorium umów, a więc bezpośrednio odzwierciedlać modelu biznesowy gridu. Perspektywa globalna zawiera również definicje operacji CRUD, aby określić zakres dozwolonych operacji na obiektach globalnych gridu.

Z punktu widzenia klienta gridu, model integracji działa od warstwy najwyższej do najniższej:

1. Użytkownik może użyć globalnych wirtualnych obiektów w zapytaniu w swoim środowisku lokalnym bazy danych;
2. Globalne wirtualne obiekty odwołują się do integracyjnych wirtualnych obiektów w lokalnej bazie danych użytkownika;
3. Integracyjne wirtualne obiekty wywołują połączenia do zdalnych rozproszonych zasobów wykorzystując specjalne obiekty kontrybucyjne, następnie w lokalnym środowisku bazy danych materializują zdalne obiekty jako kolekcje lokalne;
 - a. Lokalne obiekty kontrybucyjne traktowane są jak obiekty zdalne;
 - b. Zdalne obiekty dostępne ze zdalnych zasobów są przetwarzane przez perspektywy kontrybucyjne zdefiniowane w zdalnych zasobach;
4. Wyniki ewaluacji zapytań na obiektach zdalnych prezentowane są w postaci globalnych wirtualnych obiektów użytkownikowi, który uruchomił ewaluację zapytania.

Cała procedura ewaluacji rozproszonych obiektów opiera się na przetwarzaniu aktualizowalnych obiektowych perspektyw. Oryginalne obiekty fizyczne mogą być tworzone lub aktualizowane zgodnie z zaimplementowanymi procedurami CRUD w poszczególnych warstwach modelu integracyjnego.

3.2. Automatyczna integracja rozproszonych obiektów

Osiągnięcie w pełni zautomatyzowanego rozwiązania podłączania i odłączania rozproszonych obiektów w wirtualnym repozytorium jest głównym celem prezentowanej pracy doktorskiej. Ogólna koncepcja integracji stanowi, iż

zasoby muszą być w łatwy sposób podłączane do systemu rozproszonego bez jego modyfikacji, tak samo jak użytkownicy systemu mogą łączyć się i odłączać z wirtualnego repozytorium w dowolnym momencie jego pracy. Użytkownik wirtualnego repozytorium może wykorzystywać dostępne zasoby zgodnie ze swoimi potrzebami, jednak przy ograniczeniach jakie nakładają na niego jego uprawnienia zgodne z typem użytkownika. Takie rozwiązanie może być osiągnięte przez wprowadzenie dodatkowej warstwy pośredniej do systemu, która dostarczy mechanizmów pozwalających na osiągnięcie pełnej przezroczystości zasobów, dostawców danych oraz klientów [19]. Celem omawianego podejścia jest zaprojektowanie takiej platformy, w ramach której wszyscy klienci i dostawcy danych mają możliwość dostępu do wielu rozproszonych zasobów jednocześnie, bez ograniczeń obsługi i zarządzania procesem integracyjnym. Zakładamy, że powinien to być kompletny mechanizm realizujący przezroczystą integrację zdalnych obiektów ponad silnikiem bazy danych, wraz z odpowiednio przygotowanym silnikiem bazy danych oraz platformą komunikacyjną [19], [20], [21]. Aby to osiągnąć należy zaimplementować odpowiednią warstwę pośrednią (*middleware*) - można to osiągnąć łącząc ze sobą trójwarstwowy model integracji, wirtualne repozytorium [3] oraz sieć peer-to-peer jako platformę transportową.

Kluczowym elementem wymaganym do realizacji automatycznego procesu integracji jest *perspektywa integracyjna*. Perspektywa ta przechowuje dwie najważniejsze informacje związane z rzeczywistym procesem integracji w dynamicznym, rozproszonym środowisku:

1. Przechowuje typy obiektów wirtualnych w definicjach tzw. ziaren (*seeds*) perspektywy - (definicje są zaprezentowane na Listingu 2). Oznacza to, że obiekty zwracane w definicji klauzuli *return* będą miały taki typ jaki został określony w klauzuli *seed*;
2. Zawiera listę aktualnych kontrybucji automatycznie zarządzaną, przez zewnętrzny mechanizm, przechowywanych w klauzuli *return* definicji ziarna perspektywy. Lista musi być przechowywana w konkretnej konwencji zależnej od typu fragmentacji integrowanych rozproszonych obiektów [21].

Oznacza to, że każdy nowo zainicjowany wirtualny obiekt powinien posiadać ziarno, które jednoznacznie go identyfikuje i gdy wirtualny obiekt jest wołany, dane są pobierane z obiektu oryginalnego na który wskazuje ziarno. Kiedy jako ziarno użyty jest obiekt złożony, każdy jego podobiekt (może to być również zagnieżdżenie rekurencyjne) powinien posiadać swoje ziarno. Gdy perspektywa przykrywa bezpośrednio obiekty z lokalnej bazy danych sytuacja jest oczywista - obiekty zaszyte w ziarnach są znane i dostępne lokalnie. Sytuacja się komplikuje, kiedy w ziarnach zaszyte są inne wirtualne obiekty pochodzące z zasobów zdalnych, wtedy obiekty określone w ziarnach są lokalnie

niedostępne. Problem jest związany z kontrolą typologiczną takich lokalnie nieistniejących obiektów. Co więcej, może zdarzyć się, że zasób opisany w perspektywie odłączy się z wirtualnego repozytorium, wtedy nie ma w ogóle możliwości kontroli integrowanych obiektów. W takiej sytuacji deklaracja typów obiektów w ziarnach tak samo jak sama definicja ziarna musi zostać dynamicznie zmodyfikowana - najczęściej na żądanie, gdy np. pojawi się nowy zasób w systemie, lub istniejący się odłączy. Opisany problem został rozwiązany w implementacji mechanizmu warstwy pośredniej dla integracji.

3.3. Przykład definicji perspektywy integracyjnej

Rozpatrując uproszczony przykład definicji perspektyw omawiany w Listingu 1, oraz rozszerzając go dokonując rozproszenia na cztery niezależne zasoby kontrybuujące obiekty `Person` do obiektów `Employee` gdzie wirtualne obiekty kontrybucyjne będą nazywane w zależności od lokalizacji: `EmployeeContrib`, `EmployeeContrib1`, `EmployeeContrib2`, `EmployeeContrib3`, a `EmployeeContrib` będzie kolekcją obiektów bieżącego użytkownika bazy danych na której pokazujemy przykład integracji, definicja perspektywy integracyjnej będzie następująca - jak na Listingu 2:

Listing 2. Przykład definicji perspektywy integracyjnej dla mapowania obiektów `EmployeeContrib`

```
view EmployeeAutoIntegrationDef {  
  
  /* virtual object declaration */  
  virtual objects EmployeeIntegr: record {  
    NameIntegr: string;  
    SurnameIntegr: string;  
  }[0..*] ;  
  
  /* definition of seeds for virtual objects */  
  seed: record {  
    p: EmployeeContrib; } [0..*] {  
    return (  
      EmployeeContrib union  
      EmployeeContrib1 union  
      EmployeeContrib2 union  
      EmployeeContrib3  
    ) as p;  
  }  
}  
  
/* definitions of CRUD procedures for virtual objects */  
on_retrieve {  
  return p.(  
    Name as NameIntegr,
```

```

    Surname as SurnameIntegr;
}
on_delete {
    delete p;
}
on_update {
    p := value.(
        NameIntegr as Name,
        SurnameIntegr as Surname;
    )
}
on_new {
    create permanent EmployeeContrib(value.(
        NameIntegr as Name,
        SurnameIntegr as Surname);
    )
}

/* sub-view definitions */
view NameAutoIntegrationDef {
    virtual NameIntegr: string;
    seed: record {
        _name: EmployeeContrib.Name;
    }
    {
        return p.Name as _name;
    }
    on_retrieve { return _name; }
    on_update { _name := value; }
}
view SurnameAutoIntegrationDef {
    virtual Surnamentegr: string;
    seed: record {
        _surname: EmployeeContrib.Surname;
    }
    {
        return p.Surname as _surname;
    }
    on_retrieve { return _surname; }
    on_update { _surname := value; }
}
}
}

```

Ponieważ integracja realizowana jest na wirtualnych obiektach kontrybucyjnych, ziarno dla każdej perspektywy definiuje jako typ zwracany wirtualny obiekt kontrybucyjny (np.: *EmployeeContrib* dla *EmployeeIntegr* i *EmployeeContrib.Name* dla perspektywy *NameIntegr*). W perspektywie *EmployeeIntegr* jako typ obiektu który jest zwracany przez ziarno jest zadeklarowany pierwszy dostępny obiekt kontrybucyjny, w rzeczywistości jest obiekt pierwszej zgłoszonej do wirtualnego repozytorium kontrybucji. Gdy obiekt ten stanie się niedostępny (np. ze względu na to że zasób został odłączony) w jego miejsce zostanie przesunięty kolejny z dostępnych obiektów

kontrybucyjnych występujący za operatorem union. W przykładzie obiektem zwracany przez perspektywę jest kolekcja utworzona za pomocą operatora union. Łączenie zdalnych obiektów w kolekcje operatorem union może być realizowane dla rozproszenia obiektów o fragmentacji poziomej. W przypadku fragmentacji poziomej tworzenie kolekcji jest znacznie bardziej skomplikowane i wykracza poza ramy tej pracy doktorskiej. Zostały jednak wykonane badania [21] wskazujące drogę realizacji integracji dla obiektów o fragmentacji poziomej.

4. KONCEPCJA WARSTWY POŚREDNIEJ REALIZUJĄCEJ AUTOMATYCZNĄ INTEGRACJĘ ROZPROSZONYCH OBIEKTÓW

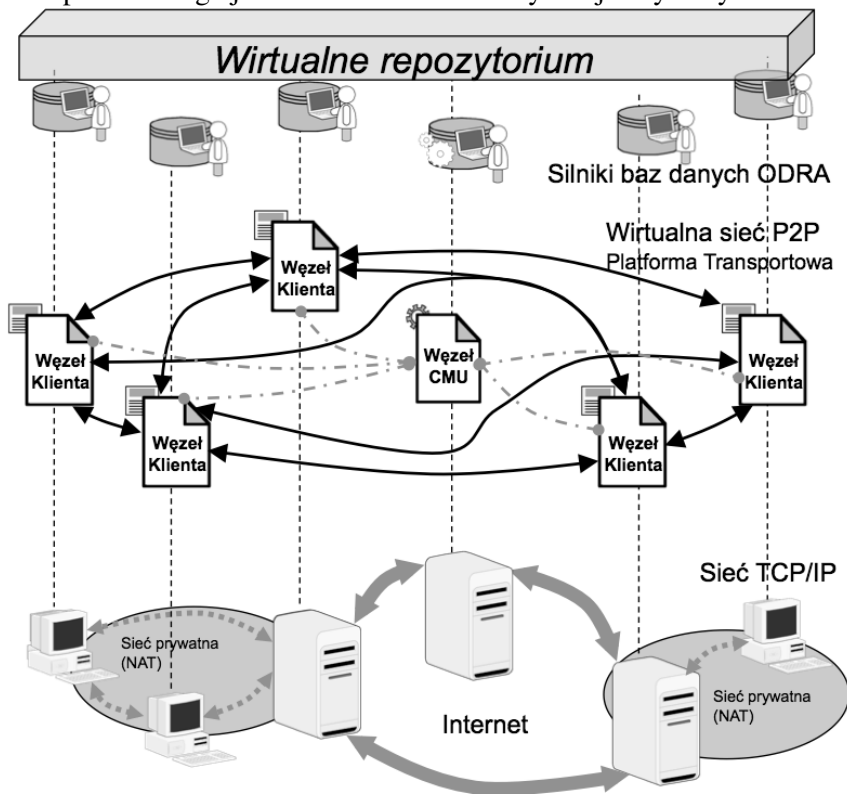
Prawdziwym wyzwaniem przy realizacji integracji danych w architekturze data grid dla wirtualnego repozytorium jest opracowanie mechanizmu łączącego w sobie funkcjonalności swobodnego dwukierunkowego przetwarzania danych pomiędzy klientami oraz dostawcami tych danych współdziałających w ramach jednego globalnego, wirtualnego składu danych. Idea komunikacji baz danych w architekturze gridu opiera się na przetwarzaniu danych między wszystkimi, niepowiązаныmi ze sobą silnikami baz danych podłączonych do wirtualnego repozytorium. Podejście to głównie skupia się na mechanizmach zapewniających w łatwy sposób:

- Zarządzanie zasobami;
- Przezroczystą integrację zasobów
- Przyłączanie użytkowników;
- Komunikację sieciową.

Powyższe zagadnienia zaszyte są wewnątrz ogólnej architektury proponowanej koncepcji sieci wirtualnej z warstwą pośredniczącą stworzoną aby umożliwić łatwą i skalowalną integrację dla społeczności użytkowników baz danych. Implementacja takiej platformy utworzy abstrakcyjną sieć komunikacyjną dla wirtualnego repozytorium. Przy takich założeniach zostanie stworzony prosty grid baz danych, który będzie pracował przy wykorzystaniu sieci o architekturze peer-to-peer.

Komunikacja sieciowa ukryta jest wewnątrz platformy transportowej zrealizowanej w architekturze peer-to-peer. Nasze badania dotyczące systemów przetwarzania równoległego rozproszonych danych takich jak Edutella [22], OGSA [23] prowadzą do wniosku, że grid baz danych w warstwie dostępnej dla użytkownika być niezależny od stosu TCP/IP oraz jego ograniczeń takich jak np. zapory ogniowe, systemy NAT czy wirtualne sieci prywatne. Procesy sieciowe

takiej jak dostęp do zasobów, włączenie i odłączenie od gridu powinny być przezroczyste dla jego uczestników, dlatego nasze rozwiązanie zakłada istnienie samowystarczalnej sieci wirtualnej ze spłaszczonym do minimum pojęciem węzła rozpoznawanego jako konkretna nazwa fizycznej bazy danych.



Rys. 4. Warstwa pośrednia dla realizacji integracji w architekturze data grid

Warstwa pośrednia składa się z dwóch warstw aplikacji (rysunek 4) - obiektowych silników baz danych będących jednocześnie warstwą najwyższego poziomu bezpośrednio dostępną dla klientów oraz w niższej warstwie aplikacji sieci peer-to-peer, które tworzą wirtualną sieć. Sieć lokalna oraz Internet są najniższą warstwą. Dla użytkowników bazy danych pracują jako heterogenicznych składy danych, jednak w rzeczywistości mogą być przezroczystie integrowane w wirtualnym repozytorium, wtedy użytkownicy mogą przetwarzać dane globalnie, poprzez schemat globalny wirtualnego repozytorium. W takiej architekturze, bazy danych podłączone do sieci wirtualnej współpracują równolegle i mogą bez ograniczeń przetwarzać rozproszone dane. Aby użytkownik mógł być częścią wirtualnego repozytorium musi spełnić dwa warunki:

1. Zgodnie z wytycznymi konsorcjum gridu przygotować perspektywę kontrybucyjną w swoim lokalnym środowisku bazy danych oraz skompilować ją;
2. Dołączyć swoją bazę danych do gridu używając lokalnie w swojej bazie danych odpowiedniej komendy.

Sieć wirtualna pracuje w architekturze rozproszonej, scentralizowanej sieci i dostarcza moduł zarządzania procesami samej sieci oraz gridu. Platforma transportowa implementuje Projekt JXTA [5] który jest głównym elementem sieci wirtualnej.

Sieć wirtualna składa się z dwóch rodzajów węzłów: wielu aplikacji klienta CU (*Client Unit*) oraz dokładnie jednego węzła zarządczego CMU (nazwanego *Central Management Unit*) przeznaczonego do zarządzania siecią wirtualną. CMU jest kluczowym elementem sieci wirtualnej (rysunek 4). Jego podstawową funkcją jest utrzymanie sieci przy życiu (co również ma bezpośredni wpływ na wirtualne repozytorium), poza tym zarządza integralnością wirtualnego repozytorium i dostępnością zasobów. Wewnątrz sieci peer-to-peer CMU odpowiedzialne jest za uruchomienie i zarządzanie gridem.

CU jest interfejsem wirtualnego repozytorium dla użytkowników baz danych. Każda baza danych musi mieć unikatową nazwę w lokalnych i globalnych schematach baz danych. Nazwa ta jest związana z nazwą węzła w sieci wirtualnej. Klient może podłączyć się do zdalnej bazy danych używając jej nazwy, która ponadto jest przechowywana w CMU.

5. PODSUMOWANIE

Przedstawione rozwiązanie przezroczystej integracji w obiektowych bazach danych ODRA oparte o trójwarstwową architekturę aktualizowalnych obiektowych perspektyw zakłada, że możliwe jest osiągnięcie równoległego i rozproszonego przetwarzania heterogenicznych zasobów danych jako jeden wirtualny zbiór obiektów. Poprzez wykonany prototyp udowodniono, że zaprojektowane rozwiązanie działa i jest spójne. Prototyp wykazał, że możliwe jest włączenie każdego obiektowego schematu danych do schematu globalnego gridu i wirtualnego repozytorium jeżeli zostaną spełnione wytyczne i wymagania konsorcjum które uruchomiło grid. Taka realizacja procesu integracji wymaga narzędzia - warstwy pośredniej, która ukryje wszystkie techniczne aspekty integracji. Warstwę pośrednią wykorzystano również do rozszerzenia podejścia integracji i jej automatyzacji. Celem było zredukowanie do minimum ingerencji użytkownika w proces tworzenia gridu, który od strony technicznej wymaga ogromnej wiedzy, a której użytkownicy nie posiadają. Kolejnym krokiem było dołączenie do warstwy pośredniej sieci peer-to-peer i w ten sposób

utworzenie wirtualnej sieci transportowej dla gridu. Takie posunięcie naturalnie zapewniło pełną przezroczystość procesu integracji. W prototypie natomiast osiągnięto następujące typy przezroczystości:

- dostępu,
- lokalizacji,
- zbieżności,
- heterogeniczności,
- skalowalności,
- fragmentacji,
- migracji.

Dalsze prace w tej tematyce powinny być ukierunkowane na pełne uwzględnienie pionowej i mieszanej fragmentacji danych przy integracji. Opracowanie integracji z globalnym modelem typologicznym, oraz budowa sieciowej warstwy transportowej dedykowanej transportowi obiektów baz danych ODRA.

LITERATURA

- [1] **Kozankiewicz H.:** Updateable Object Views. PhD Thesis (2005) <http://www.sbql.pl/phds/>.
- [2] **Subieta K.:** Theory and Construction of Object-Oriented Query Languages. PJIIT - Publishing House, Warsaw (2004) ISBN 83-89244-28-4.
- [3] **eGov-Bus** - <http://www.egov-bus.org/web/guest/home>.
- [4] **Kozankiewicz H., Stencel K., Subieta K.:** Integration of Heterogeneous Resources through Updatable Views. In : ETNGRID-2004, Proc. published by IEEE (2004).
- [5] **JXTA**. - <https://jxta.dev.java.net>.
- [6] **Gentzsch W.:** A Look into the Future of Grid Research. GRID today 4(48) (2005) <http://www.gridtoday.com/grid/517545.html>.
- [7] **Roth M., Wolfson D.:** From Data Management to Information Integration: A Natural Evolution.
- [8] **Lithicum D.S.:** Next Generation Application Integration: From Simple Information to Web Services. Addison Wesley (2003).
- [9] **CORBA**. - <http://www.omg.org/gettingstarted/corbafaq.htm>.
- [10] **McLeod D., Heimbigne D.:** A Federated architecture for information management. ACM Transactions on Information Systems 3(3), 253-278 (1985).
- [11] **Sheth A., Larson J.:** Federated Database Systems for Managing Distributed, Heterogenous, and Autonomous Databases. ACM Computing Surveys 22(3), 183-236 (1990).
- [12] **Subieta K.:** Stack-Based Approach (SBA) and Stack-Based Query Language (SBQL). Available at: <http://www.sbql.pl/>.
- [13] **Cook W.R., Rosenberger C.:** Native Queries for Persistent Objects: A Design White Paper

-
- [14] **Hibernate - Relational Persistence for Java and.NET.** - <http://www.hibernate.org>.
- [15] **Albano A., Bergamini R., Ghelli G., Orsini R.:** An Object Data Model with Roles. In : Proc. VLDB Conf., pp.39-51 (1993).
- [16] **Jodlowski A., Plodzien J.:** Objects and Roles in the Stack-Based Approach. In : Proc. DEXA Conf., Springer LNCS 2453 (2002).
- [17] **Kozankiewicz H., Leszczyłowski J., Subieta K.:** Updateable XML Views. In : Proc. of ADBIS'03, Springer LNCS 2798, pp.385-399 (2003).
- [18] **Kozankiewicz H., Leszczyłowski J., Subieta K.:** Implementing Mediators through Virtual Updateable Views. In : Engineering Federated Information Systems, Proceedings of the 5th Workshop EFIS 2003, Coventry, UK, pp.52-62 (July 17-18 2003).
- [19] **Kuliberda K., Blaszczyk P., Balcerzak G., Kaczmariski K., Adamus R., Subieta K.:** Virtual Repository Supporting Integration of Pluginable Resources. In : 17th DEXA 2006 and 2nd International Workshop on Data Management in Global Data Repositories (GRep 2006), Proceedings in IEEE Computer Society, Kraków, Polska, pp.657-661 (September 4 - 8, 2006).
- [20] **Kuliberda K., Adamus R., Wislicki J., Kaczmariski K., Kowalski T., Subieta K.:** Autonomous Layer for Data Integration in a Virtual Repository. In : OTM 2006, International Conference on Grid computing, high-performAnce and and Distributed Applications (GADA'06), Springer 2006 LNCS 4276, Montpellier, France, pp.1290-1304 (October 29 - November 3, 2006).
- [21] **Kuliberda K., Wislicki J., Kowalski T., Adamus R., Kaczmariski K.:** Procedures of Integration of Fragmented Data in a P2P Data Grid Virtual Repository. In : 4th International Conference on Service-Oriented Computing (ICSOC'06), Springer 2006 LNCS 4294, Chicago, USA, pp.557-568 (December 4 - 7, 2006).
- [22] **Nejdl W., Wolf B., Qu C., Decker S., Sintek M., Naeve A., Nilsson M., Palmer M., Tore R.:** EDUTELLA: A P2P Networking Infrastructure Based on RDF. In: 11th World Wide Web Conference - WWW2002, Honolulu, Hawaii, USA (2002).
- [23] **OGSA-DAI Project.** - <http://www.ogsadai.org.uk>.

ARCHITEKTURA GRIDU BAZODANOWEGO OPARTA O PODEJŚCIE PEER-TO-PEER

Abstract

In the article authors present an approach for realisation of object-oriented database grid using virtual peer-to-peer networking. The approach has been verified by implementation of fully functional prototype. The article shows in details a process for integration of distributed objects which is provided by the core mechanism of the prototype. Authors also described three-layer integration model based on object-oriented updateable views and middleware prototype containing mentioned peer-to-peer solution. Moreover the article contains analysis of solutions being the fundamental knowledge about integration of data.

Politechnika Łódzka
Katedra Informatyki Stosowanej