AUTOMATYKA • 2009 • Tom 13 • Zeszyt 3

Kamil Kuliberda*, Jacek Wiślicki*, ***, Tomasz M. Kowalski*,
Radosław Adamus*, Michał Meina**

# Integracja spadkowych danych relacyjnych do gridu obiektowego typu „data grid”

## 1. Wprowadzenie

Formalnie termin ‘grid’ znany jest jako termin określający rozproszone sieci obliczeniowe, jednak szybka ewolucja Internetu, globalny wzrost wymiany informacji przez Internet wytworzyły potrzebę przetwarzania danych w architekturze rozproszonej i tym samym rozwój systemów gridowych. Takie rozwiązania jak sieci P2P (*peer-to-peer*), w których możliwe jest przetwarzanie dużych ilości danych w postaci mediów, czy plików nie nadają się do zastosowań biznesowych i zarządzania danymi strukturalnymi. Taki typ danych przechowywany jest w bazach danych, a więc wymóg przetwarzania rozproszonych danych strukturalnych – opisanych modelem biznesowym był celem badań naukowych związanych 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. Technologia ta odnosi się do systemów tzw. ‘data-intensive’, którą nazywamy po prostu ‘data grid’. Taka globalna, wirtualna reprezentacja odseparowuje jej użytkowników od procesów zachodzących wewnątrz gridu, przede wszystkim integracji heterogenicznych danych (także spadkowych z innych systemów) w aspektach lokalizacji danych, ich fragmentacji, replikacji, redundancji – co jest nazwane przezroczystym przetwarzaniem danych. Bardzo skomplikowanym procesem jest przysłanianie (wrapping) zasobów gridu pochodzących z innych źródeł danych przez dedykowane struktury programistyczne tzw. osłony (*wrappers*).

Obecnie, istnieje wiele rozwiązań, gdzie różne formy danych pochodzące z rozproszonych zasobów zostają ujednolicone w jednym repozytorium ze wspólnym schematem da-

* Katedra Informatyki Stosowanej, Politechnika Łódzka
** Wydział Matematyki i Informatyki Uniwersytetu Mikołaja Kopernika w Toruniu
*** Stypendysta projektu „Innowacyjna dydaktyka bez ograniczeń – zintegrowany rozwój Politechniki Łódzkiej – zarządzanie uczelnią, nowoczesna oferta edukacyjna i wzmacnianie zdolności do zatrudniania, także osób niepełnosprawnych” współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego

nych. Takie podejście jest popularne szczególnie w środowiskach baz danych. Pozwala ono użytkownikom na osiągnięcie wielu form przezroczystego dostępu do różnych zasobów. Przy takich rozwiązaniach użytkownik nie jest świadomy, jaka jest rzeczywista forma danych których używa, za to jako osoba dostaje jedynie potrzebne informacje w najlepszym kształcie już do konkretnego zastosowania. Wśród wielu nowych koncepcji w zakresie nowoczesnych baz danych, omawiana dziedzina ewoluuje i rozwija się bardzo szybko, jako jednoznaczną odpowiedź świata nauki na potrzeby biznesu. Znamy już wiele potencjalnych zastosowań technologii dynamicznej integracji danych w nowoczesnym społeczeństwie, gdzie dane muszą być dostępne z każdego miejsca, w dowolnym czasie – na przykład e-administracja, e-nauka, e-lekarz. Istnieje wiele podejść do realizacji tej wyidealizowanej wizji. Jedne wykorzystują semantyczny opis danych oraz ontologię rozszerzoną o logikę, przy takim podejściu oprogramowanie stara się zrozumieć potrzeby użytkowników, wtedy gromadzi dane i przekształca je w żądaną formę (RDF RDFQL, OWL). Inne systemy, takie jak Oracle-10G oferują elastyczny mechanizm wywołań rozproszonych zapytań, jednak nadal są ograniczone przez model danych, a sam język nie wystarczy do realizacji rozproszonych zapytań – w takim wypadku korzystanie z niego jest nieelastyczne, złożone oraz pełne nieprzewidywalnych komplikacji. Istnieje także kilka otwartych technologii stworzonych do współdzielenia oraz wymiany danych, takich jak Edutella [9], OGSA-DAI [10] czy Via [16, 17]. Prezentowane w artykule nowe i unikalne rozwiązanie oferuje wszystkie niezbędne funkcje do rozproszonego przetwarzania danych, jak również wspiera wykorzystanie języka programistycznego jako bardzo prostego narzędzia zarządzania danymi w wyniku czego użytkownik otrzymuje nową jakość jaką jest przezroczysta integracja oraz wymiana danych spadkowych.

W pierwszej części artykułu prezentujemy podstawowe funkcjonalne elementy WR i DG oraz rozproszoną prezentację danych w WR, dalej platformę transportową (PT) oraz jej komponenty. Dalsza część skupia się na procesie integracji danych oraz użyciu osłon obiektowo-relacyjnych wraz z przykładem.

## 2. Obiekty rozproszone w data grid

Prezentowana propozycja przetwarzania rozproszonych zasobów i integracji heterogenicznych źródeł danych łączy różne podejścia prezentowane w podobnych rozwiązaniach [9, 10, 15, 16]. Nasze podejście bazuje na architekturze data grid, szerzej opisanej w pracy [7]. Wszystkie zasoby i usługi są zarządzane autonomicznie przez poszczególnych dostawców i nie zakłada się, że są one replikowane na centralnym serwerze. Zewnętrzne zasoby powinny być łatwo włączane i integrowane z systemem bez straty ciągłości jego działania.

Użytkownik (także jako dostawca danych – rys. 1) może podłączyć się do WR i używać (bądź udostępniać) zasoby zależnie od swoich wymagań, ich osiągalności i przydzielonych uprawnień. WR zapewnia warstwę pośrednią udostępniającą w pełni przezroczysty dostęp do rozproszonych zasobów (pofragmentowanych, replikowanych oraz heteroge-

nicznych ) dla wszystkich węzłów [1, 5, 6]. Ze strony użytkownika (klienta i dostawcy danych) wyróżnia się dwa typy schematów danych. Pierwszy z nich to *schemat kontrybucyjny* – jest to opis lokalnych zasobów akceptowalny przez WR, definiujący prawa i metody dostępu do lokalnych zasobów. Każdy dostawca na podstawie schematu kanonicznego definiuje w jaki sposób chce eksportować swoje dane w obrębie systemu, dzięki czemu może ograniczyć dostęp, bądź zdefiniować bardzie wyrafinowane metody integracji z globalnym systemem, wynikające z kontraktu biznesowego w obrębie całej sieci grid. Drugim schematem jest *schemat globalny* opisujący dostępne zasoby w ramach WR, na które składają się zasoby kontrybuowane przez wszystkich użytkowników. Podejście to z jednej strony umożliwia lokalną autonomię, a z drugiej pozwala na transparentną pracę z globalnie dostępnymi danymi.

Podstawowym problemem postawionym przed WR jest transformacja lokalnych klienckich danych dostępnych poprzez schemat kontrybucyjny na schemat globalny. Transformacja może polegać na bardzo skomplikowanej dekompozycji danych oraz integracji pofragmentowanych lub replikowanych części kolekcji. W naszej propozycji transformacja ta jest definiowana poprzez *aktualizowalną perspektywę* [2, 4, 18], która jest zdolna do zadania dowolnej operacji na danych i nie narzuca żadnych ograniczeń, które bardzo często występują w podobnych rozwiązaniach. Perspektywa w ODRA posiada pełne możliwości języka programowania, przez co jest o wiele bardziej elastyczna niż np. perspektywa SQL.

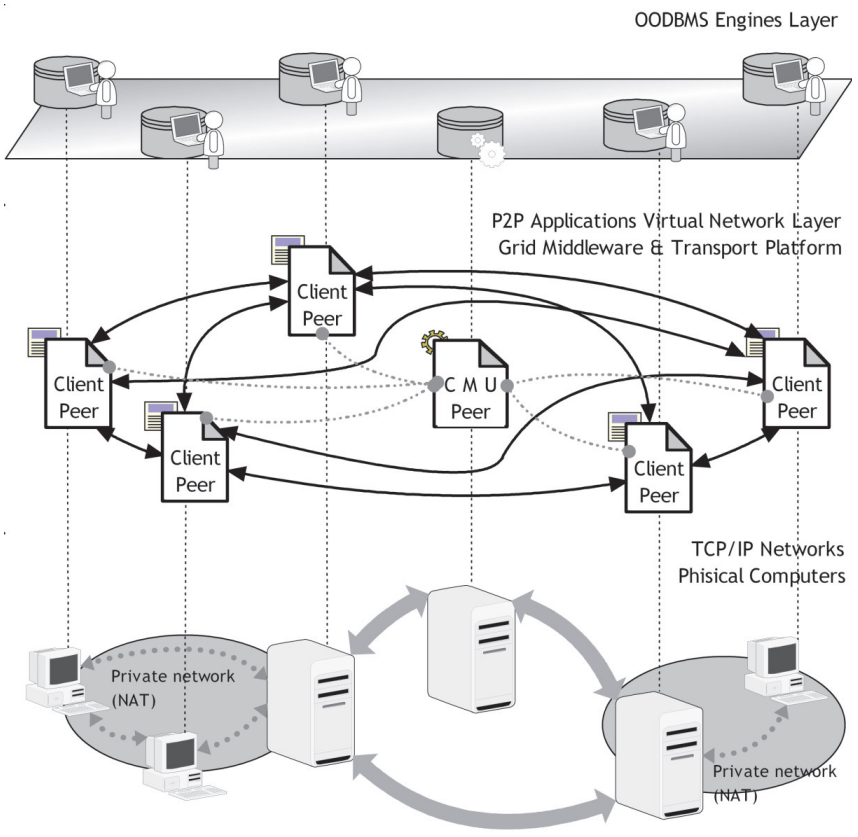
Proponowana architektura data grid zakłada istnienie wielu perspektyw udostępnionych dla każdego uczestnika WR. Ponadto każdy dostawca danych posiada *perspektywę kontrybucyjną*, która przekształca jego lokalny udział w akceptowalną kontrybucję. Dostawcy mogą również używać rozszerzenia – osłon – do istniejących SZBD (patrz [6, 7] oraz rozdz. 5). W podobny sposób klient używa perspektywy kontrybucyjnej do konsumpcji potrzebnych zasobów. To właśnie ta perspektywa wykonuje główne zadanie transformacji danych i w tym kontekście jej projektant musi być świadomy fragmentacji danych, replikacji, redundancji itp. [3, 5]. Ciało perspektywy może być opisane przez administratora (eksperta) lub poprzez automatyczną analizę schematów kontrybucyjnych.

## 3. Platforma transportowa i opis dostępu do danych

Komunikacja w systemie gridowym wymaga pewnych dodatkowych rozwiązań dla elastycznego transportu danych pomiędzy użytkownikami WR, a w szczególności należy zapewnić następujące funkcjonalności:

1) przezroczyste przetwarzanie danych (zapytań i ich wyników),
2) przezroczystą integrację zasobów,
3) dynamiczne przyłączanie i odłączanie się użytkowników bez straty ciągłości funkcjonowania całego systemu,
4) automatyczną indeksację kontrybuowanych zasobów,
5) infrastrukturę bezpieczeństwa oraz zaufania dla użytkowników.

Prezentowana poniżej koncepcja PT rozwiązuje wskazane wyżej problemy poprzez zastosowanie warstwy pośredniej, która wprowadza wspólną warstwę abstrakcji dla wszystkich użytkowników (również dostawców danych). Ujednolicony zostaje sposób transferu i komunikacji, logowania i komunikowania zdarzeń sieci formułując w ten sposób prosty grid bazodanowy o architekturze komunikacyjnej peer-to-peer [6] przy wykorzystaniu biblioteki JXTA [12]. Proponowana realizacja PT pozwala na uniezależnienie się od stosu protokołów TCP/IP i jego ograniczeń jakie powstają przy używaniu np. Firewalli, NAT, jak również dodatkowych lokalnych ograniczeń oraz zabezpieczeń. Dzięki temu nasze rozwiązanie jest o wiele bardziej elastyczne niż np. OGSA [10] czy Piazza [15, 18], gdzie komunikacja zależy od standardowych protokołów TCP/IP. Osiągnęliśmy to poprzez wprowadzenie pewnych zdarzeń i własności sieci tj. własnego systemu adresowania, procesów dostępu do danych, podłączania i rozłączania się użytkownika itp., które są przezroczyste dla użytkownika. W naszym rozwiązaniu interfejsem dostępu do gridu dla użytkowników WR jest aplikacja obiektowej bazy danych, która działa ponad warstwą PT.




**Rys. 1.** Warstwy komunikacyjne w architekturze data grid wraz z ich zależnościami

SZBD pełni rolę heterogenicznego źródła danych, który faktycznie jest zintegrowany na poziomie składu danych z interfejsem do sieci peer-to-peer. Użytkownik może dowolnie pracować z lokalnymi danymi, a dodatkowo kontrybuować je w obrębie WR; może również tymi samymi narzędziami przetwarzać dane dostępne w gridzie widziane na podstawie globalnego schematu i modelu biznesowego. Ta część systemu gridowego została zaimplementowana jako aplikacja bazodanowa dostępna przez silnik bazodanowy i język zapytań SBQL [13, 14] (”OODBMS Engine Layer” na rys. 1). W naszej architekturze poszczególne węzły mogą nawiązywać stałe bądź dynamiczne połączenia aby uzyskać współbieżne przetwarzanie danych. W obrębie wirtualnej sieci wyróżnia się jedną scentralizowaną jednostkę – Centralną Jednostkę Zarządzającą (patrz „CMU Peer” na rysunku 1). Do zadań CJZ na poziomie sieci p2p należą m.in.:

1) tworzenie sieci grid i jej konfiguracja,
2) zarządzenie siecią,
3) obsługa zdarzeń sieci tj. zmiany statusu, podłączenia węzłów, obsługa błędów sieci.

Warto zaznaczyć, iż sama sieć jest w stanie funkcjonować niezależnie od CJZ, niemniej jednak będzie ona miała wtedy ograniczone możliwości zarówno w warstwie transportu jak i warstwie aplikacji. Na potrzebę prototypowej implementacji założyliśmy, że istnieje tylko jeden CJZ.

Dla realizacji gridu w ramach wirtualnego repozytorium zostały zaimplementowane dwie oddzielne biblioteki – dla użytkowników korzystających z WR (tj. klientów i dostawców danych), która udostępnia interfejs do sieci p2p. W ten sposób formowane są węzły klienckie (patrz „Client Peer” na rys. 1) oraz biblioteka stanowiąca CJZ z mechanizmem indeksowania zasobów gridu. Biblioteka klientów zintegrowana jest z obiektowym silnikiem bazodanowym. W takiej architekturze jak przedstawionej na rysunku 1 każda baza danych posiada swoją unikalną nazwę w schemacie lokalnym oraz schemacie globalnym. Nazwa jest związana z unikalną nazwą w warstwie transportowej. Użytkownik może nawiązywać połączenia w ramach mapy węzłów propagowanej przez CJZ i których kopia jest przechowywana przez węzły klienckie. Status poszczególnych węzłów w gridzie jest kontrolowany przez CJZ, a status sieciowy przez warstwę komunikacyjną osobnym protokołem. Identyfikacja węzłów jest częścią implementacji narzędzia JXTA [12, 17].

## 4. Zdalne obiekty w rozproszonych źródłach

W rezultacie przetwarzania danych w ramach rozproszonych węzłów bazodanowych, mogą zostać przesłane obiekty pomiędzy klientami i dostawcami danych. Platforma transportowa nie powinna w żaden sposób warunkować bądź ograniczać takiej wymiany o ile nie wynika to z założeń kontraktu biznesowego. Zakładamy zatem, że każdy rodzaj obiektów (proste i złożone lub wyniki procedur bądź usług) mogą zostać przesyłane w ramach platformy transportowej. Obiekt, którego logiczna reprezentacja istnieje w innym miejscu niż jego fizyczna implementacja, nazywany jest *zdalnym obiektem*. Nasz system nie powinien również ograniczać operacji jakie można wykonywać na takim zdalnym obiekcie. Jedyne ograniczenia mogą wynikać z modelu biznesowego. W naszym rozwiązaniu każdy

obiekt otrzymuje swój identyfikator w czasie jego utworzenia, który razem z identyfikatorem macierzystego węzła służy do stworzenia globalnego identyfikatora obiektu. Wymiana obiektów między węzłami powoduje dalsze rozszerzenie identyfikatora, w związku z czym w czasie przesyłania obiektu identyfikator jest rozszerzany o kolejne identyfikatory węzłów biorących udział w komunikacji. Proces ten nazywa się *owijaniem identyfikatora*. W przypadku aktualizacji obiektu znana jest ścieżka wszystkich węzłów biorących udział w komunikacji, dzięki czemu mogą one również uczestniczyć w tej operacji.

System gridowy musi zapewniać cechę trwałości wszystkim współdzielonym zasobom. Wszystkie zmiany dokonane w ramach WR powinny być propagowane zatem na źródła danych w niższych warstwach. Proces ten jest najistotniejszym aspektem naszej implementacji. Dla łatwego zarządzania zawartością repozytorium stworzyliśmy mechanizm *globalnego indeksowania*, który pokrywa wszystkie techniczne szczegóły związane z adresacją zasobów oraz ich zarządzanie, przy czym jest jednocześnie narzędziem do zarządzania dynamicznie zmieniającej się konfiguracji sieci. *Indeks globalny* nie koniecznie musi być przechowywany na centralnym serwerze. Zakłada się jedynie, że system ten spełnia następujące właściwości:

1) zbiera i utrzymuje informacje o klientach i dostawcach danych,
2) utrzymuje informacje o stanie poszczególnych węzłów,
3) indeksuje lokalizacje obiektów,
4) utrzymuje informacje o typie fragmentacji rozproszonych zasobów,
5) rejestruje i utrzymuje informacje o udostępnionych zasobach,
6) zbiera statystyki sieci.

Indeks globalny jest obiektem o złożonej strukturze, który odzwierciedla obiektową strukturę schematu globalnego. Jest on dostępny z poziomu składni języka SBQL (jako typowy obiekt bazodanowy) w każdej bazie danych kooperującej w ramach WR. Oznacza to, że możemy ewaluować zapytania w ramach zawartości tej struktury. Niemniej jednak w odróżnieniu od typowej ewaluacji danych, wynik zapytania jest zmienny i zależy nie od składu danych tylko od aktualnej konfiguracji sieci (np. listy aktualnie podłączonych węzłów). Indeks jest również podstawowym źródłem wiedzy o zawartości WR. Bazując na zaindeksowanych danych odnoszących się do perspektyw systemowych możemy bez problemu integrować wszystkie zdalne dane, przy czym jeśli dane są już zintegrowane możemy odnieść się do nich przezroczyście – bez użycia indeksu. Dodatkowo indeks globalny zawiera informacje na temat typu fragmentacji rozproszonych danych. Wszystkie obiekty są zarządzane przez system perspektyw, który zostaje utworzony bądź zaktualizowany w ramach dynamicznie zmieniającej się konfiguracji sieci (np. podczas podłączania bądź rozłączania zasobów). Zawiera on również informacje o zależnościach między obiektami (złożoności obiektów itp.) pod kątem dostarczenia globalnego schematu.

Każdy zindeksowany obiekt w indeksie globalnym zawiera specjalny podobiekt nazwany *HFrag* (dla horyzontalnej fragmentacji) i *VFrag* (dla fragmentacji wertykalnej). Każdy z tych podobiektów zawiera również atrybut *ServerName*, którego zawartością jest zdalny obiekt – identyfikator zdalnego źródła danych (patrz rys. 4). Jeżeli zostanie zgło-

szona chęć współdzielenia jakiegokolwiek zdalnego zasobu odpowiednie obiekty (m.in. *ServerName*) zostaną zaktualizowane odpowiednimi informacjami.

Dostęp do zdalnych zasobów może być osiągniety przez ewaluację indeksu globalnego zapytaniem:

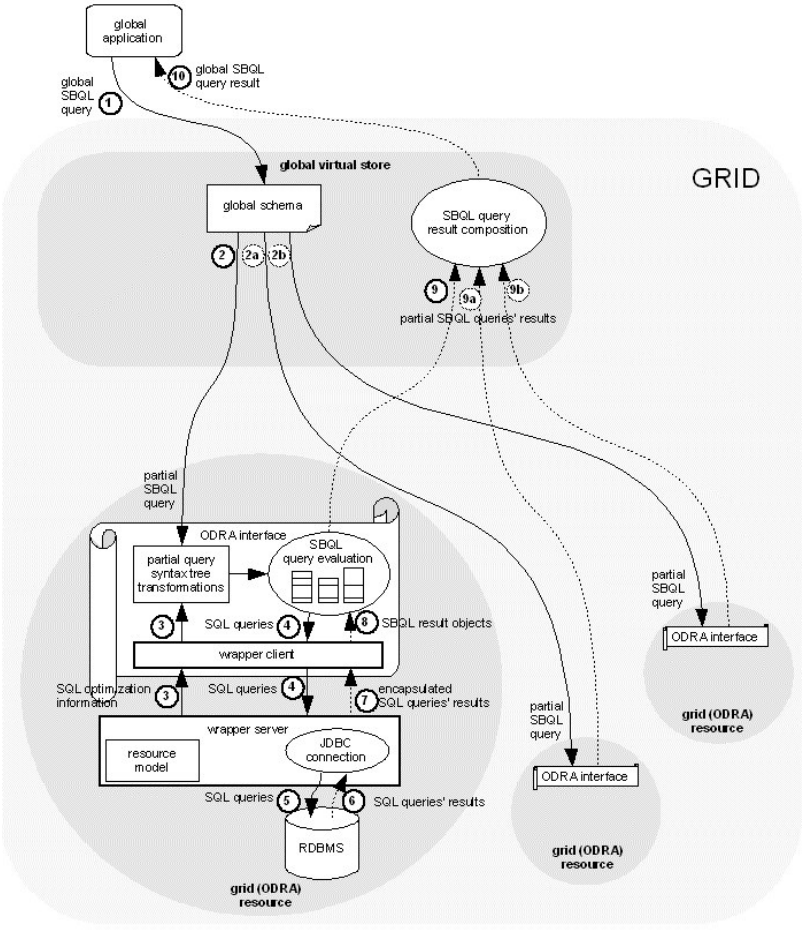
*GlobalIndex.Name_of_object_from_global_scheme.(Name_of_subobject)*
*.HFrag_or_VFrag_or_VFrag_object.ServerName_object;*

## 5. Przysłanianie zewnętrznych danych w wirtualne repozytorium

Prezentacja technik i problematyki integracji zasobów (osłaniania) oraz optymalizacji została szeroko opisana w pracach [7, 8, 11]. Przedstawiamy teraz aspekty implementacyjne modułu osłonowego. Zasób gridowy (silnik ODRA) oznacza jakikolwiek zasób danych, nad którym możliwe jest ewaluowanie zapytania SBQL i zwrócenie obiektów SBQL jako ich wyników (lokalna optymalizacja zapytań powinna być również zastosowana, o ile to możliwe). Jako, że wymagana jest tylko możliwość wykonania zapytań, natura zasobów jest nieistotna. W najprostszym przypadku gdzie zasobem jest baza danych ODRA – jej interfejs ma bezpośredni dostęp do składu danych. Niemniej jednak system gidowy ma na celu integracje zasobów, wyrażonych najczęściej w modelu relacyjnym. Interfejs w takim przypadku jest bardziej złożony, jako że nie ma bezpośredniego dostępu do składu danych – obiekty SBQL musza zostać stworzone dynamicznie na podstawie wyników zapytań SQL wykonanym na lokalnym relacyjnym SZBD. Taki przypadek (najczęściej spotykany w zastosowaniu systemów gridowych) wymaga wprowadzenia dodatkowej warstwy pośredniej – osłony (patrz rys. 2) w modelu klient/serwer. Taki rodzaj architektury został wprowadzony z kilku przyczyn. Jednym z nich jest łatwość implementacji oraz łatwość wdrożenia. Baza danych ODRA może zostać rozszerzona o wiele osłon w zależności od wymagań (np. dla relacyjnych bądź półstrukturalnych źródeł danych) i włączona w ramach modelu integracji bez istotnej straty pierwotnej efektywności. Ponadto serwer osłonowy może być stworzony niezależnie udostępniając protokół komunikacyjny dla swoich klientów. Oczywiście baza danych ODRA i osłaniany klient może działać na oddzielnych maszynach.

Proces ewaluacji zapytania w naszym środowisku gridowym jest pokazany na rysunku 2. Jedna z globalnych aplikacji gridowych wysyła zapytanie (strzałka 1), które wyrażone w języku SBQL i odnosi się do globalnego schematu i modelu biznesowego dostępnego przez użytkownikom grida. Na podstawie globalnego schematu i informacji w nim zawartych o fragmentacji, replikacji i fizycznym położeniu (utworzonej na podstawie scenariusza integracji), zapytanie jest wysyłane do odpowiednich dostawców. Na rysunku 2 ta faza jest zrealizowana strzałkami 2, 2a i 2b. Pojęcie *częściowe zapytanie* oznacza zapytanie odnoszące się do części zasobów kontrybuowanych w ramach poszczególnych węzłów (w przypadku horyzontalnej fragmentacji może to być to same zapytanie) i w większości przypadków częściowe zapytanie różni się od zapytania wyjściowego. Opis ewaluacji zapytania uwidoczniony przez strzałki 2a i 2b wykracza poza ramy tej pracy i jest traktowana tutaj jako ewaluacja przez „czarne skrzynki” (wszystkie zasoby spełniające wymagania systemu gridowego, włączając w to zasoby relacyjne tutaj opisane).

Częściowe zapytanie odnoszące się do relacyjnych zasobów jest przetwarzane przez osłonę w postaci interfejsu ODRA. Jak zostało uwidocznione wyżej lokalny interfejs nie posiada swojego fizycznego składu – może jedynie odzyskać wymagane dane z relacyjnego SZBD w locie w ramach następującego scenariusza: w pierwszej kolejności interfejs dokonuje optymalizacji zapytania (częściowego zapytania), włączając w to optymalizacje samego zapytania SBQL regułami wywodzącymi się bezpośrednio z tego języka i dodatkowo dokonanie odpowiedniej transformacji aby umożliwić wykonanie optymalizacji optymalizatorowi SQL, tak, aby ostatecznie ilość otrzymanych danych z relacyjnego SZBD była akceptowalnie minimalna. Informacje potrzebne do optymalizacji pod kątem relacyjnej bazy danych (indeksy, kardynalność, klucze główne/obce, relacje itp.) są przechowywane przez mechanizm osłony (strzałka 3) i właśnie na tej podstawie dokonywana jest optymalizacja na zasadzie transformacji drzewa składniowego zapytania SBQL.




**Rys. 2.** Ewaluacja zapytania przez osłonę

Ta optymalizacja polega w ogólności na wyszukiwaniu w drzewie składniowym wzorców przypominających optymalizowalne zapytania SQL. Odpowiednie poddrzewa są zamieniane na wywołanie procedury *execute_immediately* z parametrem zoptymalizowanego zapytania SQL (stworzonego na podstawie podstawianego poddrzewa).

Po fazie transformacji zapytania interfejs zaczyna normalną ewaluację zapytania SBQL. Gdy ewaluator napotka wywołanie procedury *execute_immediately* wysyła zapytanie na serwer przez klienta (strzałka 4: klient przekazuje zapytanie SQL bez żadnej modyfikacji). Serwer wykonuje zapytanie SQL poprzez JDBC (strzałka 5) i otrzymane rezultaty (strzałka 6) opakowane wysyła do klienta (strzałka 7). Następnie klient wytwarza rezultat wyników SBQL na podstawie rezultatu otrzymanego z serwera (ta operacja nie może zostać wykonana po stronie serwera, co ostatecznie wymusza konieczność zastosowania modelu klient/serwer) i dołącza go do oryginalnego stosu ewaluatora zapytania SBQL w celu dalszego jego przetwarzania (strzałka 8). W pożądanym przypadku (który jest nie zawsze możliwy do spełnienia) otrzymane krotki z serwera są udekorowane przez unikalne identyfikatory, które umożliwiają parametryzowanie zapytań SQL w obrębie drzewa składniowego zapytania SBQL i mogą być one wykorzystane w ewaluacji innych poddrzew tego drzewa. Po zakończonej ewaluacji interfejs przesyła wynik częściowego zapytania dalej (strzałka 9), gdzie jest on połączony z wynikami zwróconymi z innych źródeł (strzałka 9a i 9b) i gdzie, w reszcie powstaje wynik globalnego zapytania (uwzględniając fragmentację, replikację i redundancję). Ten wynik jest ostatecznie zwracany do globalnej aplikacji (strzałka 10).

## 6. Przykład integracji danych z uwzględnieniem osłony

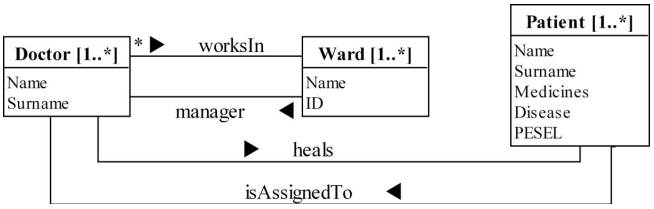
W tej sekcji przedstawiamy złożony przypadek pracy z danymi za pośrednictwem proponowanego przez nas rozwiązania gridowego (WR) włączając użycie dostawców w postaci relacyjnego SZBD dostępnego przez mechanizm osłonowy. Na początku opiszemy w jaki sposób stworzyć i wykorzystać scenariusz integracji, a następnie przedstawimy przykład dostępu do danych za pomocą zewnętrznego SZBD przez osłonę.

### 6.1. Integracja

Uwzględniając nasze poprzednie rozwiązanie definiujące scenariusz integracji dla poziomej i pionowej fragmentacji danych (szczegóły znajdują się w [5]), rozszerzymy obecny przykład o integrację różnych źródeł danych, gdzie niektóre ze źródeł przechowują obiekty wraz z dodatkowymi podobiektami, a ponadto obiekty mogą przechowywać jednocześnie te same informacje u różnych dostawców definując w ten sposób replikację. W integracji tak skomplikowanej struktury użyliśmy indeksu globalnego, który w tym przypadku jest mechanizmem zapewniającym podstawowe informacje o rozlokowaniu i strukturze obiektów. W przykładzie znajduje się obiekt nazwany *GlobalIndex*, który posiada podobiekty *HFrag* oraz *VFrag* odnoszące się do pofragmentowanych struktur danych.

Sytuacja może wyglądać dramatycznie, jeżeli założymy, że dane fizyczne są przechowywane w bazach danych typu relacyjnego. Ten fakt zmusza nas do wykorzystania osłony obiektowo-relacyjnej, aby dostać się do danych przechowywanych w takim systemie i włączenie ich do WR. Poniższy przykład zaprezentuje pełną moc naszego rozwiązania gridowego.

Na początku pokażemy procedury integracji rozproszonych danych, następnie przykład dostępu do danych przez osłonę ulokowanych w zewnętrznych źródłach jakimi są SZBD. Wszystkie przykłady będą odnosić się do schematu danych, który jest pokazany na rysunku 3.




**Rys. 3.** Schemat globalny danych dostępnych w wirtualnym repozytorium

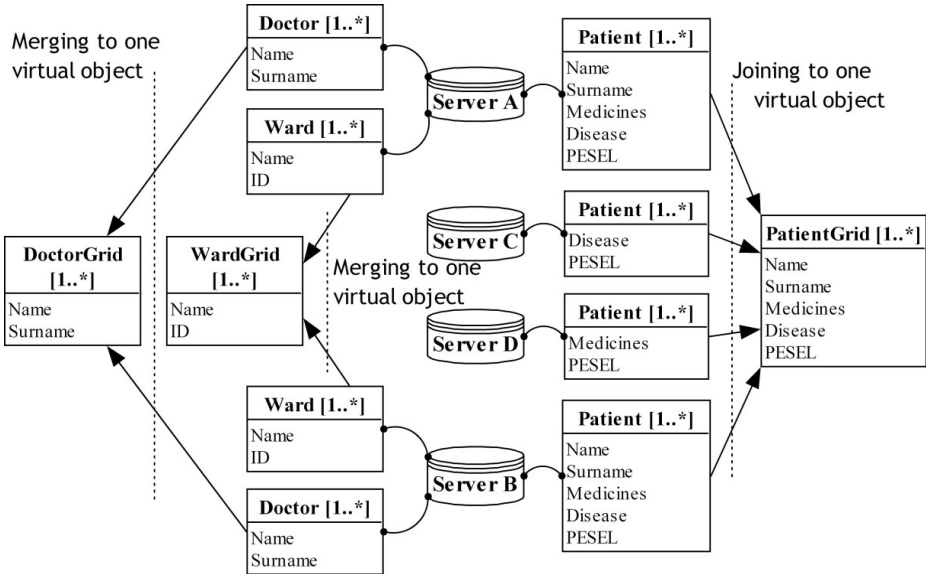
W przykładzie pokażemy metodologię tworzenia wirtualnych obiektów z obiektów o fragmentacji zmiksowanej, tj. zakładamy, że część obiektów jest pofragmentowana poziomo, a część pionowo, ponadto uwzględnimy dodatkowe zależności pomiędzy obiektami – zakładamy, że źródła obiektów posiadają podobną strukturę jednak o różnej zawartości. Przykład opiera się o dość prawdopodobny scenariusz gdzie dane udostępniane oraz przetwarzane są przez różne ośrodki zdrowia – schemat obiektów został pokazany na rysunkach 4 i 5. Dla naszego przykładu zakładamy, że każdy ośrodek zdrowia uczestniczący w gridzie znajduje się w odrębnej fizycznej lokalizacji, niektóre z agencji posiadają dane o doktorach oraz działach szpitali – w tym przypadku będzie użyta fragmentacja pozioma wirtualnych obiektów reprezentujących te byty. Każda agencja posiada również dane dotyczące pacjentów, jednak każda agencja będzie posiadała inne informacje o pacjentach – to wymusza utworzenie wirtualnych obiektów złożonych o fragmentacji pionowej. Pewne jest, że każdy ośrodek zdrowia będzie posiadał te same dane personalne pacjentów i w zależności od rodzaju ośrodka (szpital, klinika, poradnia lokalna) różne choroby, oraz ich opisy. Aby móc operować na obiektach o fragmentacji pionowej należy je złożyć wykorzystując jakąś informację wspólną opisującą takie obiekty. W naszym przykładzie będzie to atrybut „PESEL”, który jest unikatowy dla każdego obiektu pacjenta. Aby złożyć pofragmentowany obiekt pacjenta, należy w takim przypadku należy wykonać operację *join* na wszystkich obiektach pacjenta z tą samą wartością atrybutu PESEL. Proszę zauważyć, że w niniejszym przykładzie dokładne wiemy, które obiekty z których lokalizacji są pofragmentowane oraz jaki jest ich typ fragmentacji. Typ fragmentacji jest pokazany na rysunku 4.

Przykładowe zapytanie, jakie można zrealizować na opisywanym schemacie, to: wyświetl listę pacjentów chorych na ‘gruźlicę’, przypisanych do doktorów pracujących

w dziale ‘kardiologia’. Zapytanie SBQL z uwzględnieniem schematu globalnego wirtualnego repozytorium (rys. 4) będzie miało następującą postać:

PatientGrid **where** (Disease = „gruźlica” **and**

isAssignedTo.DoctorGrid.worksIn.WardGrid.Name = „kardiologia”);




**Rys. 4.** Integracja rozproszonych baz danych (fragmentacja pozioma oraz pionowa) w jedną wspólną strukturę wirtualnego repozytorium

Bazując na rozwiązaniu zaprezentowanym w [5], zdefiniowane mechanizmu integracji dla obiektów pofragmentowanych poziomo oraz pionowo jak i stworzenie definicji odpowiednich perspektyw kontrybucyjnych jest stosunkowo proste. Najpierw należy zdefiniować procedury złączenia (join) dla obiektów o fragmentacji pionowej. Następnie tak wytworzone wirtualne obiekty muszą być zcalone z obiektami o fragmentacji poziomej przy użyciu operatora unii (union). To wszystko musi być realizowane w odpowiednich perspektywach. W rezultacie otrzymamy pełne wirtualne obiekty złożone z obiektów pochodzących z różnych fizycznych źrodeł. W naszym przykładzie wirtualne obiekty *DoctorGrid* oraz *WardGrid* będą zbudowane przy wykorzystaniu informacji o fragmentacji poziomej. Obiekty *PatientGrid* muszą być zbudowane inaczej, ponieważ zależności pomiędzy obiektami, a ich zawartością pochodzącą z różnych źródeł – to musi być wzięte pod uwagę, w tym wypadku występują repliki tych obiektów jednak z różną zawartością niektórych atrybutów:

**create view** DoctorGridDef {
**virtual_objects** DoctorGrid {

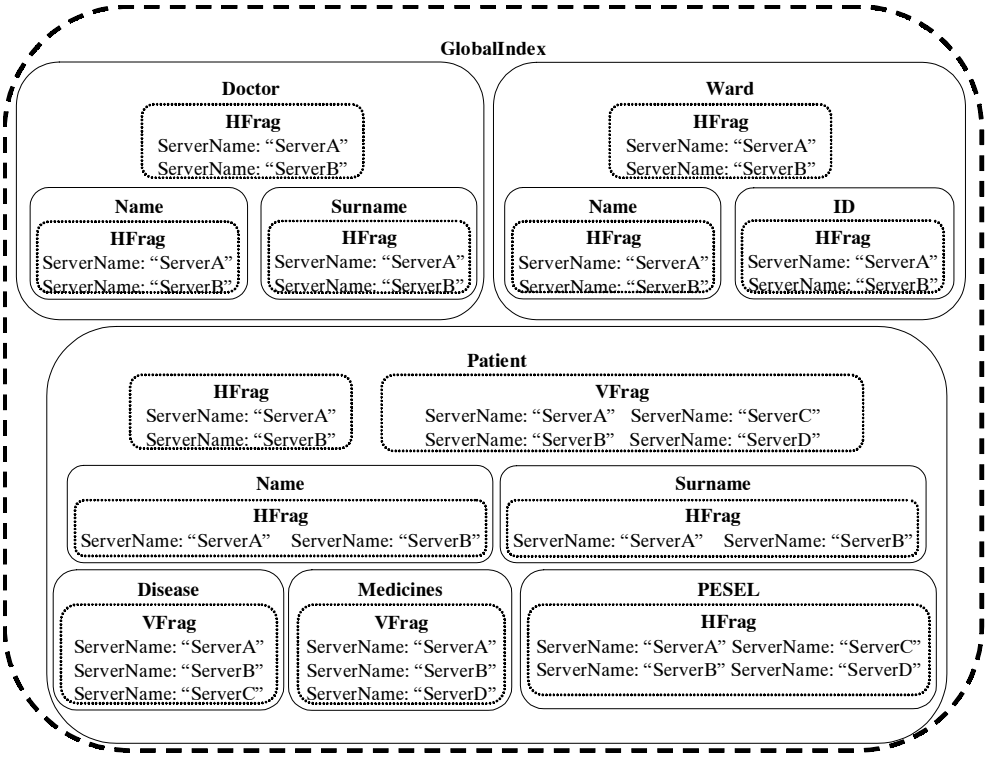
```
return (GlobalIndex.Doctor.HFrag.ServerName).Doctor as doc};
//zwraca zdalne niepofragmentowane obiekty doktorów
on_retrieve do {return deref(doc)}; //rezultat wyszukania wirtualnych obiektów
        create view NameDef {
        virtual_objects Name {return doc.Name as dn}; //tworzenie wirtualnego podobiektu
//Name dla obiektu DoctorGrid
        on_retrieve do {return deref(dn)}; //rezultat wyszukania wirtualnych obiektów
        };
        create view SurnameDef {//…};

        create view worksInDef {
        virtual_pointers worksIn {return doc.WorksIn as wi};
        on_retrieve do {return deref(wi)};
        };
};
create view WardGridDef {
virtual_objects WardGrid {
return (GlobalIndex.Ward.HFrag.ServerName).Ward as war};
on_retrieve do {return deref(war)};
        create view NameDef {
        virtual_objects Name {return war.Name as wn};
        on_retrieve do {return deref(wn)};
        };

        create view IDDef { //…};
};

create view PatientGridDef {
//poniższa procedura integruje wszystkie fizyczne obiekty "Pacjent" w kompletne wirtualne obiekty
"PatientGrid"niezależnie od rodzaju fragmentacji, proszę zwrócić uwagę, że niektóre obiekty re-
zutatu mogą posiadać niejawne repliki w WR (np. dane personalne)
virtual_objects PatientGrid {
return {
//tworzy bag indentyfikatorów dla wszystkich rozproszonych obiektów Pacjenta
bag ((GlobalIndex.Patient.HFrag.ServerName).Patient as patH),
        (GlobalIndex.Patient.VFrag.ServerName).Patient as patV)).
//tworzy listę wszystkich PESELI ze wszystkich serwerów wyłączając powtórki w obiekt uniquePesel
((distinct(patH.PESEL) as uniquePesel).
//bazując na liście unikatowych peseli, dla każdego unikatowego pesela tworzony jest bag
zawierający referencje do obiektów Pacjenta dla każdego źródła obiektów Pacjenta
bag((ref PatH where PESEL = uniquePesel),
(ref PatV where PESEL = uniquePesel))) as Patients };
//jako rezultat otrzymujemy tyle obiektów Pacjent ile jest instancji unikatowych obiektów PESEL
dostępnych ze wszystkich serwerów, proszę zwrócić uwagę, że niektóre obiekty PatientGrid mogą
posiadać repliki w postaci podobiektów takich jak Name, Surname, Address, PESEL, ponadto aby
pozbyć się sytuacji zapytanie o Name zwróci ilość tych obiektów dla jednego PESEL proponujemy
dodatkową procedurę wyszukiwania tych obiektów poprzez ich oddzielne wywoływanie.
};
//zwraca kompletną informację o każdym obiekcie Patient
on_retrieve do { return deref(Patients) };
```

```
//definiuje procedure dostępu do podobiektów Surname
        create view DiseaseDef {
        virtual_objects Disease {return distinct(Patients.Disease as PatDis};
        //zwraca zdalne niepofragmentowane obiekty Disease bez powtórzeń
        on_retrieve do {return deref(distinct(PatDis))};
        };
//poniżej powinny być definicje dostępu do takich obiektów jak Name, Surname, Medicines oraz
PESEL – ich składnie jest podobna do procedury dostępu do obiektu Disease
        create view isAssignedToDef {
        virtual_pointers isAssignedTo {return Patients.isAssignedTo as iat};
        on_retrieve do {return deref(iat)};
        };
    };
```



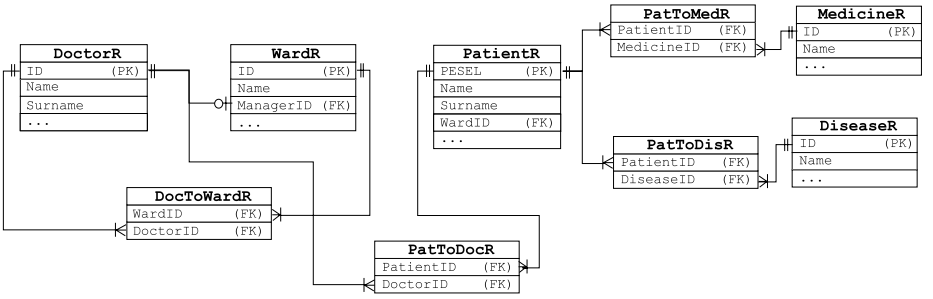

**Rys. 5.** Zawartość indeksu globalnego CJZ dla przykładu fragmentacji miksowanej

Powyższy przykład skupia się na przetwarzaniu danych w schemacie przedstawionym na rysunku 3. W przykładzie zostały zademonstrowane jedynie niezbędne procedury integracji dla obiektów o danej fragmentacji w relacji do przykładowego zapytania. Przedsta-

wione rozwiązanie nie ma limitu związanego z tworzeniem w pełni automatycznego procesu integracji poprzez rozszerzenie definicji perspektyw o dodatkowe procedury integracyjne względem indeksu globalnego.

### 6.2 Przesłanianie

Na rysunku 6 przedstawiony jest schemat relacyjny danych (po stronie back-end osłony) odpowiadający schematowi obiektowemu z rysunku 3. Dla realizacji przykładu zostały wykorzystane tylko najważniejsze jego części. Ponadto, gdybyśmy utworzyli definicję perspektyw dla osłony dla wszystkich tabel i ich zależności przykład byłby nieczytelny i nieakceptowalny dla użycia w niniejszym artykule. Wszystkie nazwy tabel z modelu relacyjnego przy mapowaniu na model obiektowy w definicji perspektywy są oznaczone literą „R" – co oznacza relacyjny, po to aby zwiększyć czytelność przykładu.




**Rys. 6.** Przykładowy schemat relacyjny dla osłony w odniesieniu do modelu obiektowego z rysunku 3

Proszę zwrócić uwagę, że wszystkie relacje pomiędzy tabelami są w schemacie obiektowym wyrażone poprzez wirtualne pointery; np. *worksIn*, *isAssignedTo*, itd.

```
create view DoctorDef {
virtual_objects Doctor {return DoctorR as doc };
virtual_objects Doctor(docID) {return (DoctorR where ID == docID) as doc };
        create view NameDef {
        virtual_objects Name {return doc.Name as dn};
        on_retrieve do {return dn};
        };
        create view worksInDef {
        virtual_pointers worksIn {return DocToWardR.WardID as w};
        on_retrieve do {return Ward(w) as Ward};
        };
};
create view WardDef {
virtual_objects Ward {return WardR as war };
```

```
virtual_objects Ward(warID) {return (WardR where ID == warID) as war };
        create view NameDef {
        virtual_objects Name {return war.Name as wn};
        on_retrieve do {return wn};
        };
};
create view PatientDef {
virtual_objects Patient {return PatientR as pat };
virtual_objects Patient(patID) {return (PatientR where PESEL == patID) as pat };
    on_retrieve do {return pat};
        create view DiseasePatDef {
        virtual_objects Disease {return DiseaseR as di};
        virtual_objects Disease(disID) {return (DiseaseR where ID == disID) as di};
        on_retrieve do {return (Disease(PatToDisR.DiseaseID)).Name};
        };
        create view isAssignedToDef {
        virtual_pointers isAssignedTo {return PatToDocR.DoctorID as d};
        on_retrieve do {return Doctor(d) as Doctor};
        };
};
```

Poniżej pokazane są poszczególne kroki ewalucacji zapytania z podrozdziału 6.1 realizowane przez osłonę przy uwzględnieniu dodatkowych informacji o unikatowych indeksach w tabelach relacyjnych (związki klucza głównego oraz obcego). Proces stosuje także optymalizację zapytania po stronie fornt-end osłony.

1. Przeprowadza funkcje jawnej dereferencji *deref* (dereference)
   *Patient* ***where*** (***deref***(Disease) = *“cancer”* ***and***
   *isAssignedTo.Doctor.worksIn.Ward.****deref****(Name) = “cardiology”);*
2. Podmienia *deref* na wywołania funkcji *on_retrieve* dla wirtualnych obiektów oraz *on_navigate* dla wirtualnych wskaźników
   *Patient* ***where*** *((Disease(PatToDisR.DiseaseID)).Name) = “cancer”* ***and***
   *isAssignedTo.(Doctor(d)* ***as*** *Doctor).Doctor.worksIn.Ward(w)* ***as***
   *Ward.Ward.(Name.wn) = “cardiology”);*
3. Zamienia wszystkie wywołania perspektyw na podzapytania z definicji perspektywy
   *PatientR* ***where*** *((DiseaseR* ***where*** *ID == PatToDisR.DiseaseID* ***as*** *di).Name) = “cancer”* ***and*** *(PatToDocR.DoctorID* ***as*** *d).(((DoctorR* ***where*** *ID == d)* ***as*** *doc)* ***as*** *Doctor).Doctor.(DocToWardR.WardID as w).((WardR where ID == w)* ***as*** *war)* ***as*** *Ward).Ward.((war.Name* ***as*** *wn).wn) = “cardiology”);*
4. Usuwa pomocnicze nazwy *w* oraz *d*
   *PatientR* ***where*** *((DiseaseR* ***where*** *ID == PatToDisR.DiseaseID).Name) = “cancer”* ***and*** *(((DoctorR* ***where*** *ID == PatToDocR.DoctorID)* ***as*** *doc)* ***as*** *Doctor).Doctor.(((WardR* ***where*** *ID == DocToWardR.WardID)* ***as*** *war)* ***as*** *Ward).Ward.(war.Name) = “cardiology”);*

5. Usuwa pomocnicze nazwy *doc* oraz *war*
   *PatientR* ***where*** *((DiseaseR* ***where*** *ID == PatToDisR.DiseaseID).Name) = "cancer"* ***and*** *((DoctorR* ***where*** *ID == PatToDocR.DoctorID)* ***as*** *Doctor).Doctor.((WardR* ***where*** *ID == DocToWardR.WardID* ***as*** *Ward).Ward.Name = "cardiology");*
6. Dekomponuje zapytanie na mniejsze podzapytania:
   *PatientR* ***where*** *((DiseaseR* ***where*** *ID == PatToDisR.DiseaseID).Name) = "cancer"* ***and*** *(DoctorR* ***where*** *ID == PatToDocR.DoctorID)* ***and*** *((WardR* ***where*** *ID == DocToWardR.WardID)* ***and*** *(WardR.Name = "cardiology")));*
7. Tutaj unikatowo indeksowane podzapytanie zostaje podmienione na klauzulę SQL *exec_ immediately*:
   *PatientR* ***where*** *((DiseaseR* ***where*** *ID == PatToDisR.DiseaseID* ***and*** *Name) = "cancer"* ***and*** *(DoctorR* ***where*** *ID == PatToDocR.DoctorID)* ***and***
   (**exec_immediately** (SELECT * FROM WardR WHERE ID == DocToWardR.WardID AND WardR.Name = "cardiology"))*);*
8. Ponieważ inne ograniczenie integralności danych jest dostępne dla osłony, odpowiedni wzorzec jest rozpoznawany i dodatkowe wywołanie klauzuli *exec_immediately* jest realizowane:
   *PatientR* ***where*** *((***exec_immediately** (SELECT * FROM DiseaseR WHERE ID == PatToDisR.DiseaseID AND DiseaseR.Name = „cancer") ***and*** *(DoctorR* ***where*** *ID == PatToDocR.DoctorID)* ***and*** (**exec_immediately** (SELECT * FROM WardR WHERE ID == DocToWardR.WardID AND WardR.Name = „cardiology"))*);*

W powyższym przykładzie którekolwiek z wywołń klauzuli *exec_immediately* w lokalnym środowisku SQL uruchamia natywną procedurę optymalizacji (z odpowiednim zastosowaniem indeksów, szybkich złączeń itd.).

## 7. Wnioski

W tym artykule autorzy zaprezentowali kompletne rozwiązanie przezroczystej integracji rozproszonych danych w wirtualnym repozytorium o architekturze data grid. Rozwiązanie wykorzystuje konsekwentnie kombinację różnych technologii, takich jak: implementację SBA dla obiektowych baz danych jako fundamentalne rozwiązanie do utworzenia sytemu opartego o architekturę data grid, język zapytań SBQL dla SBA, który stanowi narzędzia do dla użytkowników to przetwarzania rozproszonych danych, mechanizm aktualizowalnych perspektyw jako narzędzie do wirtualizacji rozproszonych danych w schematy biznesowe, sieć p2p bazującą na projekcie JXTA jako warstwę dla komunikacji wewnątrz gridu oraz osłonę obiektowo-relacyjną umożliwiającą integrację danych relacyjnych do gridu obiektowego. Prezentowane rozwiązanie działa jako kompletny prototyp systemu.

## Literatura


[1] 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.

[2] Habela P., Kaczmarski K., Kozankiewicz H., Lentner M., Stencel K., Subieta K., *Data-Intensive Grid Computing Based on Updateable Views*. ICS PAS Report 974, June 2004.

[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*. ETNGRID2004 WETICE2004, Proceedings published by IEEE.

[5] Kuliberda K., Adamus R., Wislicki J., Kaczmarski K., Kowalski T., Subieta K., *Autonomous Layer for Data Integration in a Virtual Repository*. International Conference on Grid computing, high-performAnce and Distributed Applications (GADA'06), Springer 2006 LNCS 4276, 1290–1304.

[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, Proc. in IEEE Computer Society.

[7] Kuliberda K., Wislicki J., Adamus R., Subieta K., *Object-Oriented Wrapper for Relational Databases in the Data Grid Architecture*. OTM Workshops 2005, Springer LNCS 3762, 2005, 367–376.

[8] Kuliberda K., Wislicki J., Adamus R., Subieta K., *Object-Oriented Wrapper for Semistructured Data in a Data Grid Architecture*. 9th International Conference on Business Information Systems 2006, LNI vol. P-85, GI-Edition 2006, 528–542.

[9] Nejdl W., Wolf B., Qu C., Decker S., Sintek M., Naeve A., Nilsson M., Palmer M., Risch T., *EDUTELLA, a P2P networking infrastructure based on RDF*. Proc. Intl. World Wide Web Conference, 2002.

[10] Open Grid Services Architecture, *Data Access and Integration Documentation*. http://www.ogsadai.org.uk.

[11] Plodzien J., *Optimization Methods in Object Query Languages*. PhD Thesis. IPIPAN, Warszawa, 2000.

[12] Project JXTA Community: http://www.jxta.org (*home-page*).

[13] Subieta K., *Stack-Based Approach (SBA) and Stack-Based Query Language (SBQL)*. http://www.ipipan.waw.pl/~subieta, Description of SBA and SBQL, 2006.

[14] Subieta K., *Theory and Construction of Object-Oriented Query Languages*. Editors of the Polish-Japanese Institute of Information Technology, 2004 (in Polish).

[15] Tatarinov I., Ives Z., Madhavan J., Halevy A., Suciu D., Dalvi N., Dong X., Kadiyska Y., Miklau G., Mork P., *The Piazza Peer Data Management Project*. ACM SIGMOD Record, 32(3), 2003.

[16] Tatarinov I., Halevy A., *Efficient Query Reformulation in Peer Data Management System*. SIGMOD Conference 2004, 539–550.

[17] Wilson B., *JXTA Book*. http://www.brendonwilson.com/projects/jxta/ (*home-page*).

[18] Kozankiewicz H., *Updateable Object Views*. PhD Thesis, http://www.ipipan.waw.pl/~subieta/ -> Finished PhD-s -> Hanna Kozankiewicz, 2005.