

Prace Naukowe

Wydziałowego Zakładu Informatyki

Biblioteka Główna i OINT
Politechniki Wrocławskiej



100100078385

Politechniki Wrocławskiej



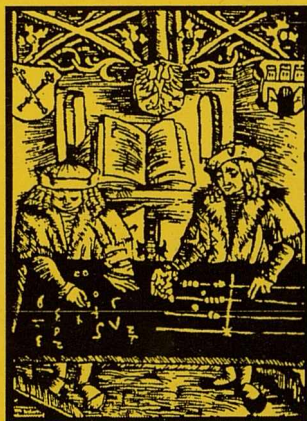
2

BAZY DANYCH

Redakcja naukowa

Zygmunt Mazur

Oficyna Wydawnicza
Politechniki Wrocławskiej





Prace Naukowe
Wydziałowego Zakładu Informatyki
Politechniki Wrocławskiej

2 BAZY DANYCH



Redakcja naukowa

Zygmunt Mazur



Oficyna Wydawnicza Politechniki Wrocławskiej
Wrocław 2001



WYDZIAŁOWY ZAKŁAD INFORMATYKI

**WYDZIAŁ INFORMATYKI I ZARZĄDZANIA
POLITECHNIKI WROCŁAWSKIEJ**

Wybrzeże Wyspiańskiego 27, 50-370 Wrocław

Recenzenci

Tadeusz MORZY

Witold REKUĆ

Opracowanie redakcyjne materiałów

Hanna MAZUR

Projekt okładki

Zygmunt MAZUR

Biblioteka Główna i OINT
Politechniki Wrocławskiej



001735204



30 7303 /5

© Copyright by Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław 2001

OFICyna WYDAWNICZA POLITECHNIKI WROCŁAWSKIEJ

Wybrzeże Wyspiańskiego 27, 50-370 Wrocław

ISBN 83-7085-554-7

Drukarnia Oficyny Wydawniczej Politechniki Wrocławskiej. Zam. nr 327/2001.

SPIS TREŚCI

Wstęp	5
Mazur Z., Piechota K., Architektury rozproszonych systemów baz danych	7
Mazur Z., Piechota K., Własności systemów zarządzania rozproszonymi bazami danych	23
Majkowski A., Mazur Z., Problemy integracji heterogenicznych baz danych	43
Gruber J., Mazur Z., Bazodanowe systemy klient-serwer i architektury wielowarstwowe	53
Gruber J., Mazur Z., Technologie projektowe i programistyczne dla aplikacji wielowarstwowych i zintegrowanych sieciowo systemów bazodanowych	77
Majkowski A., Integracja schematów heterogenicznych baz danych w modelu obiekowym	105

CONTENTS

Introduction	5
Mazur Z., Piechota K., Architectures of Distributed Databases	7
Mazur Z., Piechota K., Properties Distributed Database Management Systems	23
Majkowski A. Mazur Z., On Problems with Integration of Heterogeneous Databases	43
Gruber J., Mazur Z., Client/Serwer Database Systems and Multi-Layer Architectures	53
Gruber J., Mazur Z., Design Technologies and Software Tools for Building Distributed Multi-Layered Applications Integrated in the Network Environment	77
Majkowski A., Heterogeneous Databases Schema Integration Using Object Model	105

*Wyobrażenia wiecznie sprzeciwia się
tradycji, taka jest jej rola*

John Pleiffer

WSTĘP

Tematyka drugiego Zeszytu Naukowego „Bazy Danych” wiąże się z zagadnieniami rozproszonych baz danych. Rozproszony system baz danych to system baz danych, w którym występuje rozproszenie danych. Rozproszenie polega najczęściej na *replikacji* danych (kopia całości lub części danych) lub na *fragmentaryzacji* danych (*poziomej* – wyborze podzbioru danych, lub *pionowej* – wyborze podzbioru atrybutów). Rozproszone bazy danych są bardziej skomplikowane niż scentralizowane bazy danych z powodu dodatkowego czynnika komunikacji sieciowej. Dlaczego zatem wielu użytkowników podejmuje się tworzenia rozproszonych baz danych? Najważniejsze powody, to:

- potrzeba odwzorowania w systemie bazy danych geograficznego odwzorowania firmy użytkownika (oddziały, filie itp.),
- większa kontrola nad danymi w tych miejscach, gdzie są one potrzebne,
- utrzymywanie replikacji danych zwiększa niezawodność systemu (istotne np. w systemach bankowych),
- sprawność rozproszonego systemu może się istotnie poprawić, jeśli dokonamy prawidłowego rozproszenia danych, np. jeśli większość zapytań będzie kierowana do małej lokalnej bazy danych, to jest bardzo duże prawdopodobieństwo skrócenia czasu operacji wyszukiwania i aktualizacji danych.

Zasadniczą cechą rozproszonej bazy danych powinno być ukrycie przed użytkownikiem faktu rozproszenia. Dla użytkownika baza rozproszona powinna wyglądać jak jedna baza scentralizowana.

Zaprezentowane artykuły są wynikiem prac prowadzonych w Wydziałowym Zakładzie Informatyki na Wydziale Informatyki i Zarządzania Politechniki Wrocławskiej.

Przedstawione artykuły dotyczą:

- architektury rozproszonych systemów baz danych
- cechy systemów zarządzania rozproszonymi bazami danych oraz mechanizmów umożliwiające ich realizację
- czynników różnicujących systemy baz danych, składające się na system heterogenicznych baz danych
- analizy ewolucji systemów i środowisk bazodanowych do systemów rozproszonych z aplikacjami wielowarstwowymi oraz porównania możliwości technologii aplikacji wielowarstwowch z klasyczną technologią klient-serwer
- analizy metodologii i metodyk projektowania intranetowych oraz internetowych aplikacji bazodanowych w tych technologiach.
- metody integracji schematów wyrażonych w obiektowym modelu danych.

Autorzy składają podziękowanie Recenzentom za uwagi przedstawione w recenzjach, które zostały uwzględnione w trakcie przygotowywania ostatecznych wersji artykułów.

Zygmunt Mazur

*architektura systemu bazy danych,
baza danych, rozproszone bazy danych*

Zygmunt MAZUR*
Kamila PIECHOTA*

ARCHITEKTURY ROZPROSZONYCH SYSTEMÓW BAZ DANYCH

Szeroki dostęp do baz danych oraz sieci komputerowych, jaki osiągnięto w ostatnich latach, przyczynił się do wzrostu zainteresowania rozproszonymi bazami danych (DDB). Popularność tej technologii wynika z rosnącego przekonania, iż systemy scentralizowane i jednoprocessorowe powoli zbliżają się do teoretycznego progu szybkości przetwarzania. Jak w każdej wyłaniającej się technologii, tak i w systemach zarządzania rozproszonymi bazami danych (DDBMS) można wyliczyć spełnione i niespełnione obietnice. W artykule są omówione architektury rozproszonych systemów baz danych.

1. WPROWADZENIE

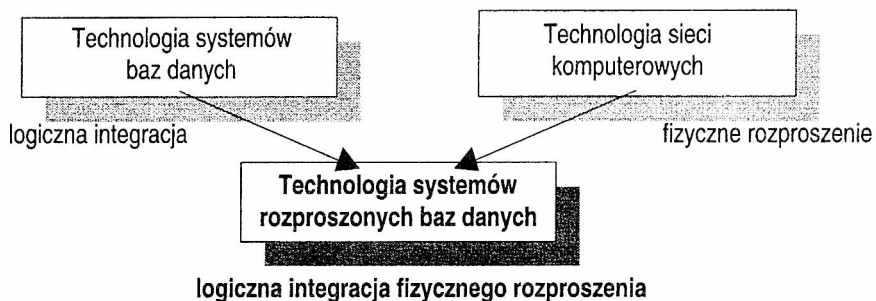
Istnieje wiele systemów zapewniających dostęp lokalny, jak i zdalny wielu użytkowników do dużych centralnych baz danych. Rosną wymagania użytkowników, którzy chcą dzielić dane w zespołach i działach przedsiębiorstwa. Klienci wymagają, aby w całym przedsiębiorstwie dane były błyskawicznie dostępne. Jednak szybkość dostępu do danych i poziom bezpieczeństwa jakie oferują systemy scentralizowane okazują się zbyt niskie. Dzięki rozpowszechnieniu systemów otwartych i architektury klient/serwer maleją rozmiary sprzętu i koszty obsługi aplikacji przetwarzania transakcji, przy jednoczesnym zachowaniu wydajności i niezawodności spotykanej dotychczas w komputerach klasy mainframe.

Tendencje panujące obecnie w architekturach dużych systemów baz danych w większości dotyczą architektur równoległych i rozproszonych. Popularność tego typu architektur wynika z rosnącego przekonania, iż systemy scentralizowane i jednoprocessorowe powoli zbliżają się do teoretycznego progu szybkości przetwarzania.

Intensywny rozwój technologii sieci komputerowych, wzrost niezawodności i wydajności, oraz zaawansowanie DBMS, stały się głównym motorem dla rozwoju technologii rozproszonych baz danych. Technologia ta łączy w sobie dwie cechy: rozproszenie i integrację. Rozproszenie dotyczy fizycznego rozproszenia danych po

* Wydziałowy Zakład Informatyki Politechniki Wrocławskiej, Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, mazur@ci-1.ci.pwr.wroc.pl

węzłach sieci komputerowej, zaś integracja wiąże się z logiczną integracją rozproszonych danych, dzięki której dla użytkownika rozproszona baza danych wydaje się być pojedynczą, jednolitą bazą danych. W przeciwieństwie do rozproszonych baz danych, bazy scentralizowane charakteryzują się zarówno integracją fizyczną, jak i logiczną.



Rys. 1. Czynniki rozwoju technologii rozproszonych baz danych
Fig. 1. Development agents of distributed databases technologies

Rozproszoną bazę danych (ang. *Distributed Database – DDB*) można zdefiniować jako logicznie zintegrowane kolekcje współdzielonych danych, które są fizycznie rozproszone w węzłach sieci komunikacyjnej. Wyróżnia się heterogeniczne i homogeniczne rozproszone bazy danych.

System zarządzania rozproszoną bazą danych (ang. *Distributed Database Management System – DDBMS*) jest zatem oprogramowaniem służącym do zarządzania rozproszoną bazą danych w taki sposób, aby fakt rozproszenia danych był transparentny (przeźroczysty) dla użytkownika.

Scentralizowany system zarządzania bazą danych (ang. *Database Management System – DBMS*) jest systemem zarządzającym pojedynczą bazą danych, zaś DDBMS jest pojedynczym DBMS zarządzającym wieloma bazami danych.

Rozproszony system baz danych (ang. *Distributed Database System – DDBS*) jest rozproszoną bazą danych wraz z systemem zarządzania rozproszoną bazą danych. Założenia będące podstawą tych definicji zostały wypunktowane poniżej:

1. Dane są przechowywane w wielu węzłach sieci komputerowej. Każdy z węzłów przyporządkowany jest pojedynczemu procesorowi. Z punktu widzenia DDBMS fakt, że jeden z węzłów jest maszyną wieloprocesorową, w sposób równoległy zarządzający danymi, jest nieistotny.
2. Procesory w poszczególnych węzłach sieci są połączone siecią komputerową. Należy podkreślić, że połączenie to jest dość swobodne, gdyż poszczególne procesory posiadają własny system operacyjny i pracują niezależnie (w odróżnieniu od systemów wieloprocesorowych).

3. Rozproszona baza danych jest bazą danych, a nie dowolnym zbiorem plików rozmieszczonych w węzłach sieci komputerowej. Istnieje znacząca różnica pomiędzy rozproszoną bazą danych a zbiorem plików zarządzanych przez rozproszony system plików. W rozproszonych bazach danych dane są w sposób logiczny powiązane (powiązanie to definiowane jest przy pomocy strukturalnego formalizmu, np. relacyjnego), zaś dostęp do nich powinien być zapewniony poprzez interfejs, czyniący transparentnym fakt rozproszenia.
4. System zarządzania rozproszoną bazą danych posiada pełną funkcjonalność DBMS. Powinien posiadać system przetwarzania rozproszonych transakcji, umożliwiać przetwarzanie zapytań, czy strukturalną organizację danych.

Z całą pewnością można powiedzieć, że technologia rozproszonych baz danych znajduje się obecnie w przełomowej fazie poszukiwania odpowiedzi na pytania, jak powinny wyglądać komercyjne systemy. Mimo, że sama koncepcja nie jest nowa (pierwsze publikacje ukazały się pod koniec lat 70-tych), na rynku nie istnieją obecnie produkty, które w sposób idealny realizowałyby założenia transparentności systemu zarządzania rozproszoną bazą danych. Jednak ambicją producentów jest udostępnianie w swoich produktach pewnych mechanizmów umożliwiających rozpraszanie danych i zarządzanie nimi. Większość dużych systemów zarządzania rozproszonymi bazami danych posiada możliwość replikowania danych do oddalonych geograficznie serwerów. Najczęściej jednak z użytkowaniem replik związane są pewne ograniczenia lub zachowanie spójności rozproszonych danych wymaga interwencji użytkownika.

Obecnie nie istnieją zaakceptowane standardy definiujące architekturę, czy funkcję rozproszonych systemów bazodanowych. Większość prototypów powstawało w sposób niezależny i są to systemy reprezentujące bardzo często różną filozofię rozpraszania danych, zarówno w przypadku homogenicznych (jednorodnych) rozproszonych baz danych jak i multibaz danych.

Celem artykułu jest przedstawienie problematyki związanej z projektowaniem rozproszonych baz danych. Projektowanie rozproszonych baz danych jest zagadnieniem znacznie bardziej skomplikowanym od projektowania scentralizowanych baz danych. Wymaga ono uwzględnienia dodatkowego czynnika - kosztu komunikacji sieciowej, który nie jest łatwy do oszacowania, a ma znaczący wpływ na efektywność i poprawność zarządzania rozproszonymi danymi.

Metodologia projektowania rozproszonych baz danych zależy od architektury systemu. W systemach ściśle zintegrowanych proces projektowania przedstawia strategię „z góry na dół” (*ang. top-down*), rozpoczynając od fazy analizy wymagań i projektowania globalnego schematu, kończąc na projektowaniu fizycznych schematów każdej z lokalnych baz danych. W przypadku systemów multibaz danych, proces projektowania jest przykładem strategii „z dołu do góry” (*ang. bottom-up*), której celem jest integracja istniejących już baz danych.

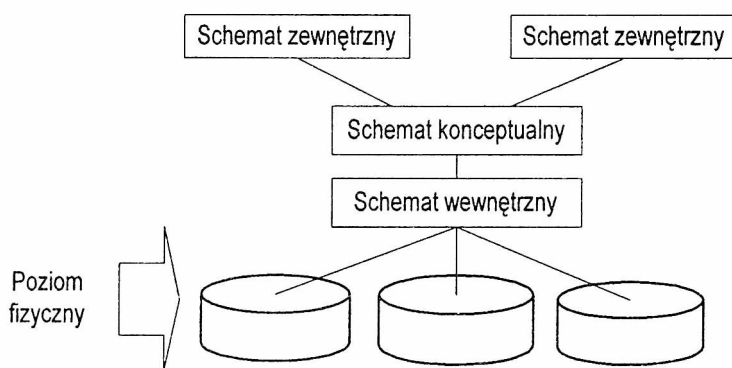
2. TYPOLOGIA ROZPROSZONYCH SYSTEMÓW ZARZĄDZANIA BAZAMI DANYCH

Systemy zarządzania rozproszonymi bazami danych stanowią szeroką gamę systemów, różniących się w sposób zasadniczy. W początkowej fazie technologia ta rozwijała się w sposób niezależny w wielu ośrodkach naukowych. Systemy te reprezentują zupełnie różne filozofie i były tworzone w celu realizacji różnych celów. Wybór metodologii jaką należy przyjąć przy projektowaniu rozproszonej bazy danych jest uzależniony od systemu, który tą bazą będzie zarządzać. W niniejszym rozdziale zostanie przedstawiony usystematyzowany podział typów systemów zarządzania rozproszonymi bazami danych.

W celu rozróżnienia problematyki dotyczącej pojedynczych węzłów rozproszonej bazy danych oraz całego systemu rozproszonej bazy danych używa się sformułowań odpowiednio lokalny węzeł i globalny węzeł. Zatem określenie lokalna baza danych dotyczy bazy danych znajdującej się w pojedynczym węźle sieci komputerowej, zaś globalna baza danych dotyczy logicznej integracji wszystkich lokalnych baz danych. Należy jednak pamiętać, że globalna baza danych nie istnieje fizycznie. Można powiedzieć, że jest rodzajem „wirtualnego” obiektu, którego komponenty są fizycznie przechowywane w rozproszonych, rzeczywistych bazach danych.

2.1. PODZIAŁ SYSTEMÓW ZE WZGLĘDU NA ARCHITEKTURĘ SYSTEMU

Większość komercyjnych scentralizowanych DBMS oparta jest na standardzie trzystopniowej architektury zgodnej z propozycją Zespołu Badawczego Systemów Zarządzania Bazami Danych organizacji ANSI-SPARC (Rys. 2). Standard ten definiuje różne typy opisu danych. Wyróżnia on trzy poziomy: wewnętrzny, konceptualny i zewnętrzny. Poziom konceptualny definiuje logiczny model wszystkich danych przechowywanych w bazie. Na tym poziomie nie jest brany pod uwagę sposób przechowywania danych czy wygląd poszczególnych widoków użytkowników. Definicja schematu konceptualnego jest produktem pierwszej fazy projektowania scentralizowanej bazy danych. Użytkownicy bazy danych, jak i programy, uzyskują dostęp do określonego wycinka danych zawartych w bazie. Wycinki te zdefiniowane są przez schematy zewnętrzne. Schemat konceptualny przekładany jest na schemat wewnętrzny. Jest on opisem najniższego poziomu danych zawartych w bazie. Tworzy on interfejs między bazą danych a systemem plików w systemie operacyjnym. Interfejs ten ma za zadanie umożliwienie dostępu do danych.



Rys. 2. Architektura systemu bazy danych

Fig. 2. Database system architecture

Trzy stopniowa architektura ANSI-SPARC dla scentralizowanych DBMS została szeroko zaakceptowana i znajduje odzwierciedlenie w większości komercyjnych produktów. Niestety nie istnieje ekwiwalentny standard architektury dla SZRBD. Technologia i pierwsze prototypy, jak już to było wspomniane, rozproszonych DBMS powstawały w sposób w dużej mierze niezależny. Stąd też każdy z systemów zaadoptował swoją własną specyficzną architekturę. Poniżej została przedstawiona klasyfikacja rozproszonych systemów bazodanowych, dla której punktem odniesienia jest ich architektura.

Każdy z rozproszonych systemów bazodanowych można zakwalifikować do jako jeden z dwóch typów:

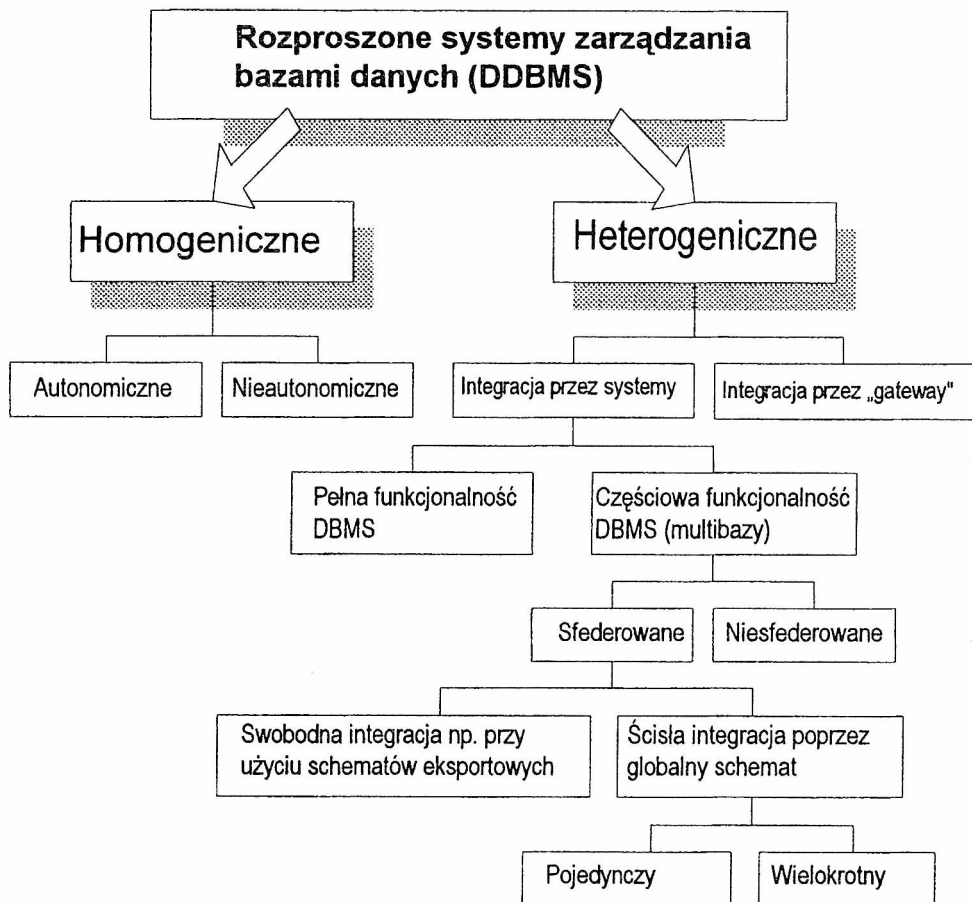
- Homogeniczny system zarządzania rozproszoną bazą danych (ang. *homogeneous DDBMS*)
- Heterogeniczny system zarządzania rozproszoną bazą danych (ang. *heterogeneous DDBMS*)

Typy te są oparte na całkowicie różnych filozofiach oraz tworzone są w celu realizacji różnych potrzeb. DDBMS należące do pojedynczego typu różnią się w sposób znaczący. Pełna klasyfikacja rozproszonych DBMS została przedstawiona na rys. 3.

Homogeniczne DDB najbardziej przypominają bazy scentralizowane, jednak dane nie są przechowywane w jednym miejscu, ale są fizycznie rozproszone w różnych punktach sieci komputerowej. Rysunek 4 przedstawia architekturę homogenicznego DDBMS. Kluczowy dla tej architektury jest fakt, że użytkownicy uzyskują dostęp do bazy danych poprzez globalny interfejs. Nie ma użytkowników globalnych.

Schemat globalny jest unią wszystkich lokalnych opisów danych. Widoki użytkowników są definiowane w oparciu o schemat globalny. Schemat architektury przedstawiony na rysunku 5 jest uszczegółowieniem rysunku 4.

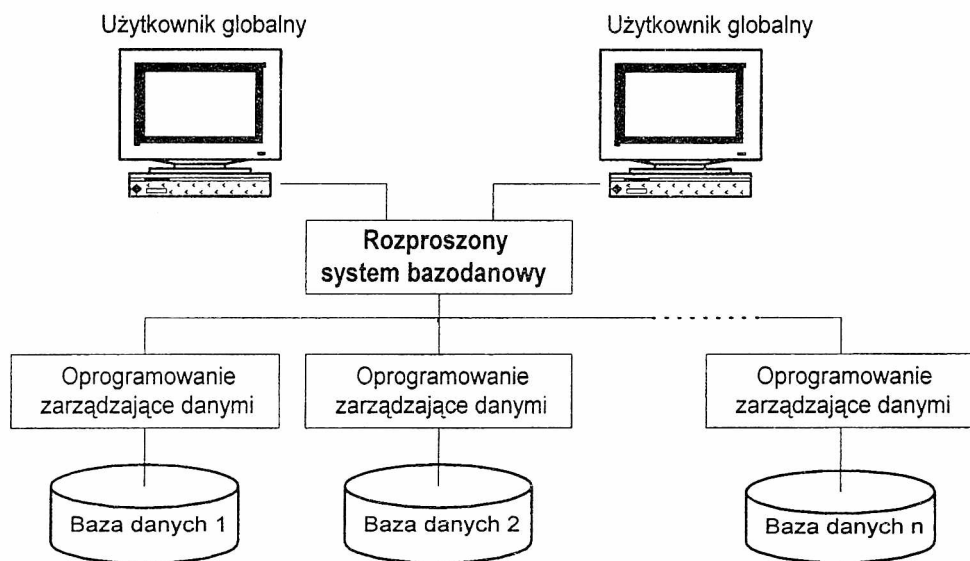
Porównując tę architekturę ze standardem ANSI-SPARC należy zauważyć, że nie są brane pod uwagę lokalne schematy dla lokalnych baz danych. Tak też się dzieje w praktyce.



Rys. 3. Klasyfikacja DDBMS

Fig. 3. Classification of DDBMS

Większość implementacji homogenicznych systemów nie posiada lokalnych schematów a oprogramowanie zarządzające lokalnymi danymi jest ograniczone do absolutnego minimum. Oprogramowanie to nie może być nazywane DBMS, ponieważ jego funkcje są bardzo ubogie. Część funkcji, takich jak np. kontrola dostępu, zarządzanie integralnością i bezpieczeństwem danych została przejęta przez system globalny.

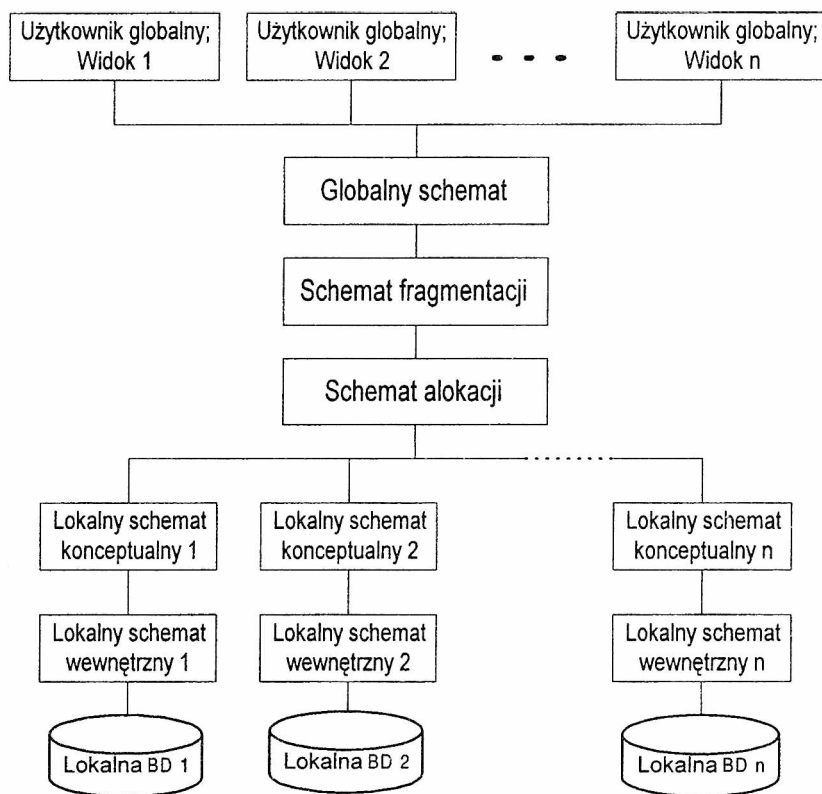


Rys. 4. Architektura homogenicznego DDBMS

Fig. 4. Architecture of homogeneous DDBMS

Większość implementacji homogenicznych systemów nie posiada lokalnych schematów a oprogramowanie zarządzające lokalnymi danymi jest ograniczone do absolutnego minimum. Oprogramowanie to nie może być nazywane DBMS, ponieważ jego funkcje są bardzo ubogie. Część funkcji, takich jak np. kontrola dostępu, zarządzanie integralnością i bezpieczeństwem danych została przejęta przez system globalny. Oprogramowanie to nie może być nazywane DBMS, ponieważ jego funkcje są bardzo ubogie. Część funkcji, takich jak np. kontrola dostępu, zarządzanie integralnością i bezpieczeństwem danych została przejęta przez system globalny.

Представленный схемат, в отношении до стандарду трьохповерхової архітектури ANSI-SPARC, został rozszerzony o dwa dodatkowe poziomy: schemat fragmentacji i schemat alokacji. Schemat fragmentacji określa w jaki sposób globalne relacje są dzielone pomiędzy poszczególne lokalne bazy danych. Schemat alokacji określa, w którym miejscu będą przechowywane poszczególne fragmenty. Poszczególne fragmenty mogą zostać zaalokowane w więcej niż jednym miejscu sieci komputerowej tzn. mogą być replikowane. Ma to na celu uzyskanie optymalnie maksymalnej szybkości przy rekonstruowaniu globalnej relacji. Problematyka fragmentacji, alokacji oraz replikacji zostanie szczegółowo omówiona w dalszej części pracy.



Rys. 5. Architektury homogenicznego DDBMS

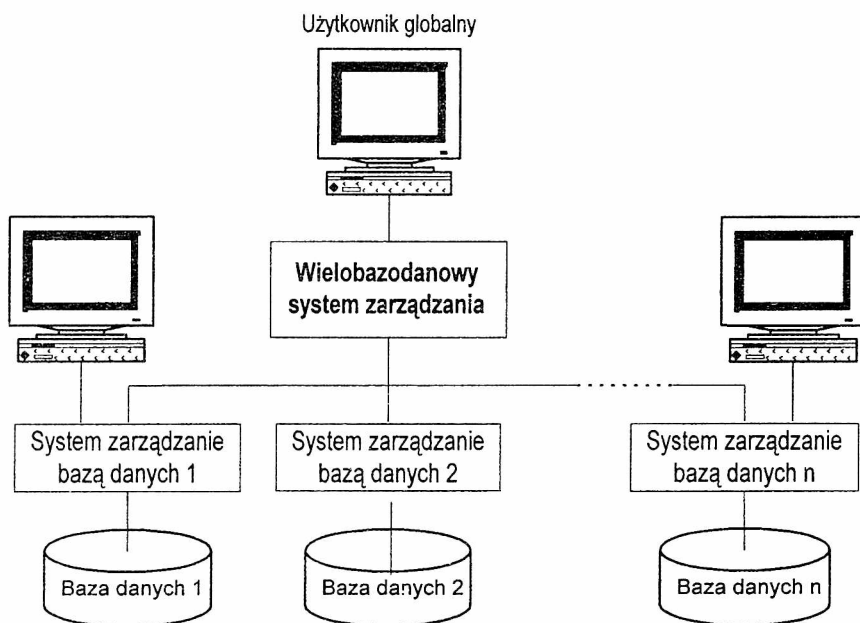
Fig. 5. Architecture of homogeneous DDBMS

Na rysunku 6 została wyróżniona inna główna klasa systemów współdzielących dane nazywanych systemami **heterogenicznymi**. Charakterystyczne dla tej klasy systemów jest wykorzystywanie różnych DBMS w poszczególnych węzłach. Wyróżniamy dwie podklasy systemów heterogenicznych:

1. systemy, w których systemy lokalne integrowane są przez inne systemy,
2. systemy, w których systemy lokalne komunikują się przez specjalne łącza (gateway).

Podział ten określa, w jaki sposób realizowana jest integracja poszczególnych lokalnych DDBMS. Pierwszym ze sposobów integracji jest stworzenie „nadsystemu” synchronizującego pracę wszystkich lokalnych DBMS. Drugim, znacznie prostszym sposobem jest stworzenie oprogramowania nazwanego gateway, działającego jak swojego rodzaju „haczyki”, umożliwiające komunikację z systemami znajdującymi się w innych lokalizacjach. W dużym uproszczeniu można powiedzieć, że jest to aplikacja działająca pomiędzy dwoma heterogenicznymi DBMS, pełniąc funkcję „nakładki” nad jednym z nich. Jej zadaniem jest ukrycie różnic pomiędzy obydwoma systemami,

umożliwiając komunikację między nimi w taki sposób, jakby były one tego samego typu.



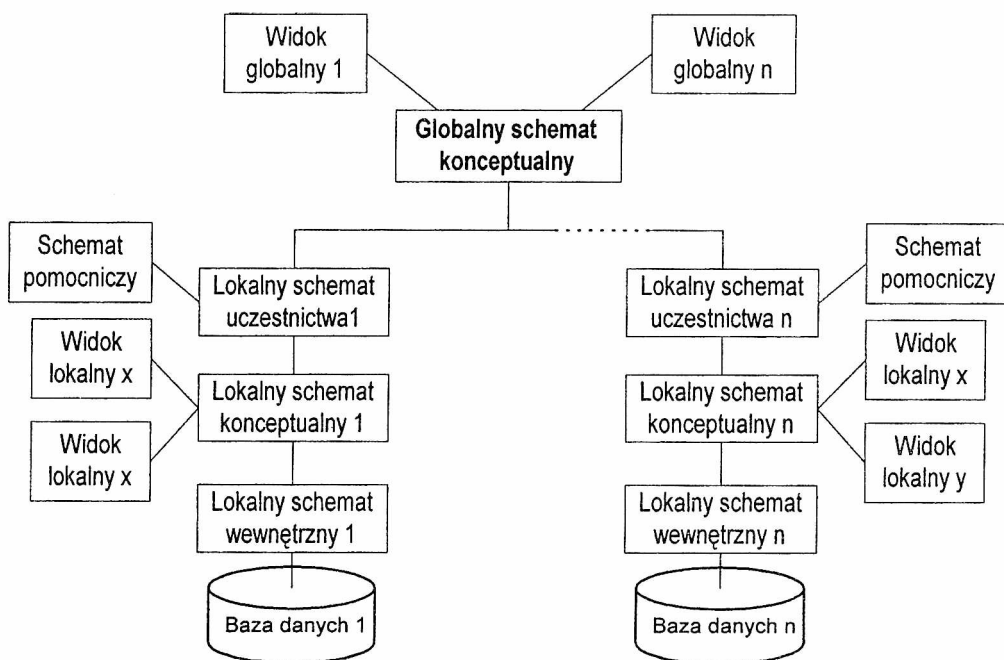
Rys. 6. Architektura systemu multibazodanowego
Rys. 6. Architecture of multidatabase system

Kolejny podział, w przedstawionej na rysunku 3 klasyfikacji, dotyczy systemów heterogenicznych, w których synchronizacja poszczególnych DBMS odbywa się poprzez dodatkowy system. Podział ten wyznacza nam, w jaki sposób „nadsystemy” realizują podstawowy zestaw funkcji, które powinien spełniać DBMS. Tak więc, możemy wyróżnić systemy posiadające pełną funkcjonalność DBMS, oraz systemy kładące nacisk tylko na niektóre aspekty (takie jak konwersja modeli danych pomiędzy poszczególnymi systemami), zapewniające tylko podstawowy zestaw funkcji. Systemy posiadające tylko częściową funkcjonalność DBMS nazywamy *systemami multibazodanowymi* (ang. *Multidatabase Management Systems - MDBMS*). System multibazodanowy nie musi zapewniać transparentności rozproszenia, transparentności replikacji oraz nie musi posiadać globalnego słownika danych [Rusink 1994].

W przeciwieństwie do systemów homogenicznych, w systemach multibazodanowych wyróżnia się dwa rodzaje użytkowników: lokalnych i globalnych. Głównym założeniem systemów multibazodanowych jest integracja istniejących już,

heterogenicznych źródeł danych. Ważną cechą tego rodzaju systemów jest zachowanie dostępu lokalnych użytkowników do lokalnych baz danych. Istnienie multibazy danych w żaden sposób nie powinno wpływać na działanie lokalnego DBMS.

Dalsza klasyfikacja systemów rozproszonych (dotycząca systemów multibazodanowych) przedstawiona została przez Shetha i Larsona [SL1990]. Wśród systemów multibazodanowych można wyróżnić systemy niesfederowane i sfederowane. Systemy sfederowane można podzielić na systemy posiadające schemat globalny (ściśła integracja) i nie posiadające schematu globalnego (ograniczona integracja). Schemat architektury typowego MDBMS ściśle zintegrowanego został przedstawiony na rysunku 7.



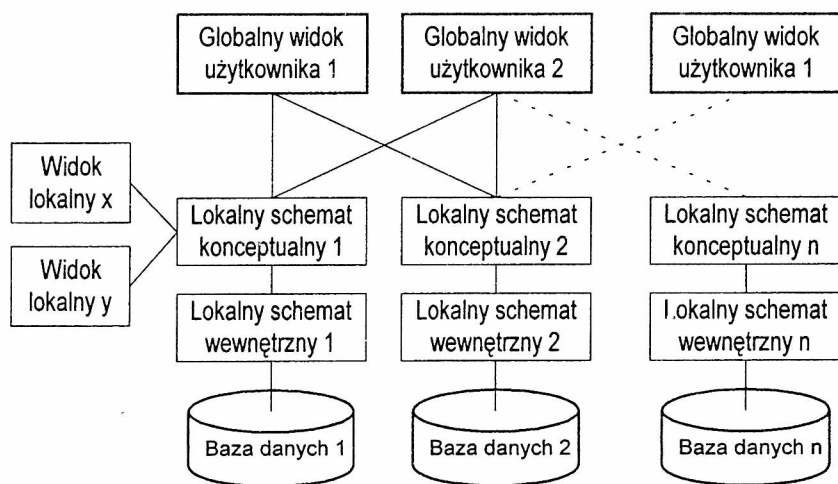
Rys. 7. Architektura ściśle zintegrowanego systemu multibazodanowego
Fig. 7. Architecture of completed integrated MDBS

Globalny schemat koncepcyjny jest logicznym widokiem na dane zawarte w multibazie danych. Jest to unia wszystkich lokalnych koncepcyjnych schematów. Nie jest wymagane, aby schematy lokalne były w całości odwzorowywane w schemat globalny. To na poziomie systemów lokalnych podejmowana jest decyzja, jaka część schematu lokalnego będzie partycypować w tworzeniu schematu globalnego. Ta dobrowolność nazywana jest lokalną autonomią. Lokalny węzeł uczestniczy w tworzeniu multibazy poprzez udostępnienie tzw. schematu uczestnictwa (*ang. participation schema*). Schemat ten jest widokiem lokalnego schematu koncepcyjnego.

Schemat pomocniczy (Rys. 7) opisuje reguły jakie stosowano w czasie przekładania schematu lokalnego na schemat globalny. Np. może to być reguła nakazująca przeliczenie jednostek miary z milimetrów na metry lub reguła, zezwalająca na przechowywanie wartości *null* w schemacie globalnym (może to nastąpić w przypadku, gdy w jednej z baz przechowywane informacje o pracowniku to adres i telefon, zaś w innej z baz tylko telefon). W niektórych z multibaz wyróżnia się także schemat fragmentacji, jako podział na części schematu globalnego. Jednak, w przeciwieństwie do homogenicznych baz danych, architektura ta nie wyróżnia schematu alokacji. Schemat taki nie jest projektowany, gdyż jest on niejako wpisany od początku w multibazę.

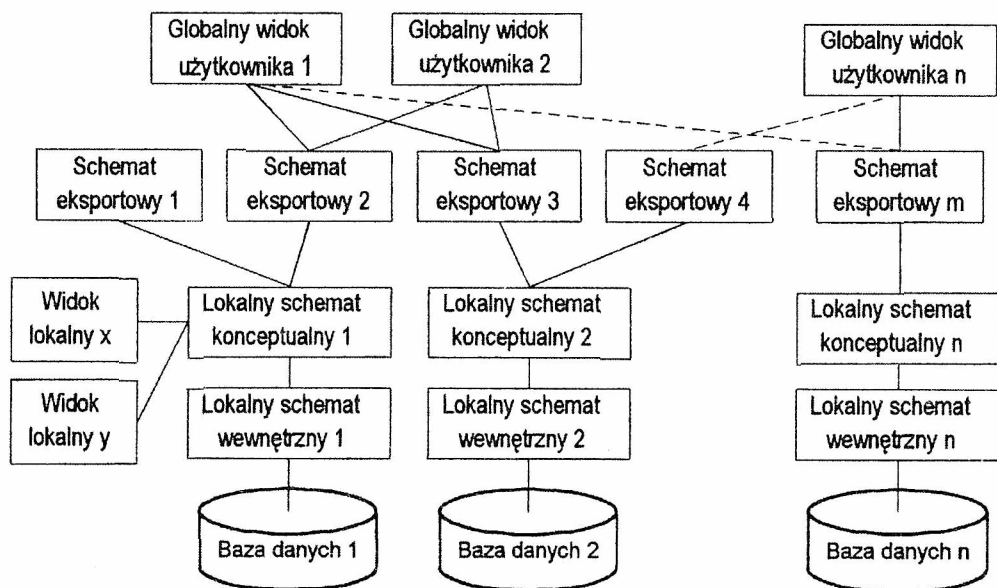
Charakterystyczną cechą systemów multibaz danych o swobodnej integracji jest to, iż nie posiadają one schematu globalnego. Konstrukcja globalnego schematu okazuje się często zbyt skomplikowana. Różnice semantyczne, jak i syntaktyczne pomiędzy poszczególnymi lokalnymi schematami okazują się często tak duże, że projektowanie globalnego schematu staje się zbyt kosztowne, szczególnie, gdy liczba globalnych zapytań jest niewielka.

Istnieją dwa różne podejścia do konstruowania systemów multibazodanowych o swobodnej integracji. W jednym z nich odpowiedzialność za stworzenie widoku opartego na lokalnych schematach powierza się właścicielowi globalnego widoku, dostarczając mu do tego celu odpowiedni język zapytań MSQ (ang. *Multi Structural Query Language*). Alternatywne podejście polega na zdefiniowaniu tzw. schematów eksportowych (analogicznych do schematów uczestnictwa w multibazach ściśle powiązanych), opartych na poszczególnych schematach lokalnych. Idea architektury omówionych systemów została przedstawiona na Rys. 8 i Rys. 9.



Rys. 8. System multibazodanowy o swobodnej integracji bez schematów eksportowych

Fig. 8. MDBS with free integration without export schemas



Rys. 9. System multibazowy o swobodnej integracji opartej o schematy eksportowe
 Fig. 9. MDBMS with free integration based on export schemas

Należy podkreślić, że podobnie jak nie istnieją standardy opisujące architekturę rozproszonych systemów baz danych, tak nie ma standardu dotyczącego terminologii dotyczącej tej dziedziny. Przedstawiona powyżej klasyfikacja jest jedynie próbą usystematyzowania rozproszonych systemów baz danych ze względu na architekturę. Przy opisie poszczególnych typów systemów podjęto próbę odniesienia się do standardu architektury ANSI-SPARC dla scentralizowanych systemów bazodanowych.

2.2.KLASYFIKACJA DDBMS WEDŁUG KRYTERIÓW NIE WYNIKAJĄCYCH Z ICH ARCHITEKTURY

Z uwagi na to, że systemy DDB są bardzo różnorodne, klasyfikacja przedstawiona powyżej nie systematyzuje wszystkich systemów w sposób jednoznaczny. Istnieje wiele kryteriów rozróżniających DDBMS. Do najważniejszych należą *heterogeniczność*, stopień *rozproszenia* oraz stopień *autonomiczności* systemów lokalnych.

Heterogeniczność w DDBMS może pojawiać się na wielu różnych poziomach systemu. Może ona dotyczyć różnic sprzętowych w poszczególnych węzłach, różnych systemów operacyjnych lub różnych protokołów sieciowych. Heterogeniczność może dotyczyć lokalnych DBMS. Mogą to być różne DBMS oparte o ten sam model danych

lub o różne modele (relacyjny, obiektowy, hierarchiczny, sieciowy). Nawet jeżeli lokalne DBMS są we wszystkich węzłach sieci takie same, heterogeniczność może wynikać z różnic semantycznych pomiędzy poszczególnymi modelami danych. Np. w jednym z węzłów atrybut *ulica* może zawierać nazwę ulicy oraz numer domu, natomiast w innym węźle, tylko nazwę ulicy.

Większość prac we wczesnym okresie rozwoju technologii rozproszonych baz danych dotyczy problematyki związanej z integracją heterogenicznych modeli danych. Przyjęto dwa podejścia: stworzenie globalnego modelu, na który przekładane są wszystkie lokalne bazy lub stworzenie translatora między każdą parą modeli. Niewątpliwą zaletą drugiej opcji jest to, iż lokalny użytkownik nie musi znać innych modeli i języków niż te, które dotychczas użytkował. Z drugiej strony takie podejście wymaga stworzenia $n*(n - 1)$ translatorów, gdzie n jest liczbą różnych modeli. Alternatywną strategią, znacznie częściej stosowaną, jest zdefiniowanie na poziomie globalnym tzw. *modelu kanonicznego* (ang. *canonical model*), na który są przekładane wszystkie modele lokalne. Podejście to wymaga zdefiniowania tylko $2n$ translatorów. Z czasem ogólnie akceptowanym i stosowanym modelem danych, stał się model relacyjny tak, że proces translacji modeli lokalnych dotyczył przede wszystkim semantyki, a nie syntaktyki.

Istnieje wiele możliwych sposobów rozproszenia danych. Dane mogą zostać *całkowicie zreplikowane* (ang. *fully replicated*) tzn. cała globalna baza danych przechowywana jest we wszystkich węzłach sieci. Całkowita replikacja jest używana wówczas, gdy wymagana jest wysoka tolerancja uszkodzeń oraz krótki czas dostępu (wszystkie dane rozproszonej bazy są dostępne pomimo uszkodzenia jednego z węzłów, nie ma opóźnień związanych z komunikacją sieciową). Projektowanie takiej bazy niczym nie różni się od projektowania bazy scentralizowanej. Rozproszenie polega tylko na wielokrotnym powieleniu całej bazy w każdym z węzłów sieci. Największym problemem całkowicie zreplikowanej bazy danych stanowi zachowanie jej integralności. Propagacja zmian i odtwarzanie stabilności bazy po awariach wymagają użycia skomplikowanych i czasochłonnych rozwiązań.

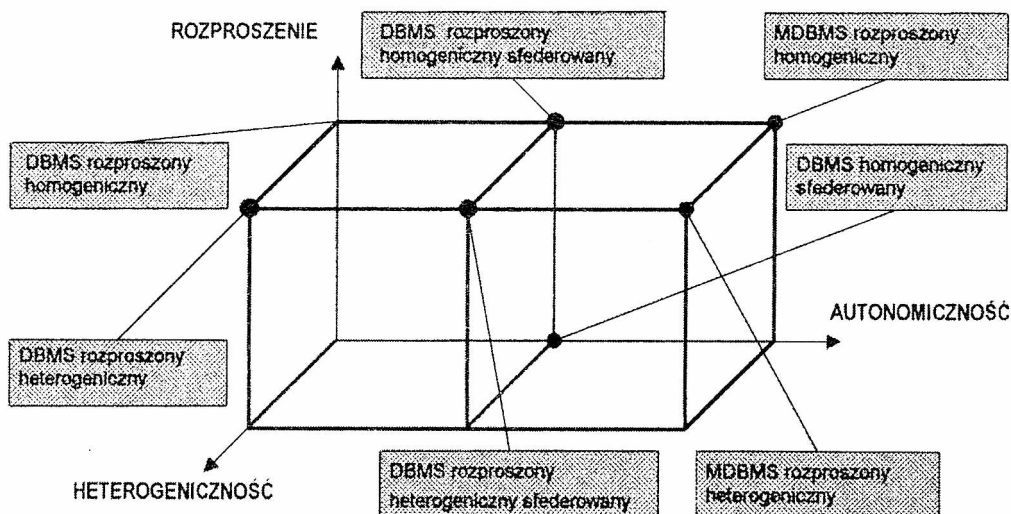
Ekstremalnie różnym sposobem rozproszenia jest *pełne współdzielenie* (ang. *fully partitioned*) zasobów danych tzn., że nie ma danych zreplikowanych. Podejście takie najczęściej spotykane jest w multibazach danych, gdzie już istniejące, niezależne, lokalne bazy danych integrowane są w jedną. Nie istnieje tutaj problem propagacji zmian, gdyż każda „jednostka” danych posiada tylko jedną lokalizację. Jednak wadą takiego podejścia jest niska tolerancja uszkodzeń oraz wysokie koszty dostępu do danych, związane z koniecznością sprowadzania ich z oddalonych lokalizacji.

Rozwiązaniem wypośrodkowanym jest *częściowa replikacja* danych (ang. *partial replication*). Stosowanie częściowej replikacji ma swoje uzasadnienie wówczas, gdy do niektórych obszarów globalnej bazy danych wymagany jest częstszy dostęp niż do pozostałych. Problematyka replikacji zostanie szczegółowo omówiona w pracy

[Mazur Pi2001], gdyż jest ona jednym z podstawowych zagadnień projektowania rozproszonych baz danych.

Kolejnym ważnym czynnikiem decydującym o sposobie rozproszenia bazy jest stopień *lokalnej autonomii* bazy. W ekstremalnym przypadku, całkowity brak autonomii lokalnej implikuje globalną, całkowitą kontrolę systemu, tak jak to się dzieje w homogenicznych rozproszonych bazach danych. W systemach tych nie ma użytkowników lokalnych, zaś lokalne oprogramowanie zarządzające bazami danych nie jest niezależne. W przypadku multibaz danych, lokalni użytkownicy posiadają niezależny dostęp do lokalnych baz danych, niezakłócony istnieniem multibazy oraz globalnych jej użytkowników.

W literaturze często rozróżniane są następujące typy autonomii: projektowania, uczestnictwa, wykonywania. *Autonomia projektowania* rozumiana jest jako dobrowolność, jaką posiadają lokalne systemy, w podejmowaniu decyzji dotyczących rodzaju danych jakie zawiera baza, wyboru modelu danych, wyboru struktur przechowujących dane. Ten rodzaj autonomii jest niejako wpisany w multibazy, gdyż lokalne bazy danych są projektowane i implementowane niezależnie od powstającej w późniejszym czasie globalnej bazy. *Autonomia uczestnictwa* daje prawo lokalnym systemom do decydowania o zakresie uczestnictwa lokalnych danych w tworzeniu globalnej bazy danych. *Autonomia wykonywania* określa dobrowolność w podejmowaniu decyzji przez lokalny system zarządzania, w celu odrzucenia lub zaakceptowania transakcji wstawienia, uaktualnienia lub usunięcia lokalnych danych przeprowadzanej przez system globalny.



Rys. 10. Sposoby implementacji DBMS
Fig. 10. Methods of implementations of DBMS

Podsumowując, architekturę rozproszonych systemów bazodanowych można rozpatrzeć jako funkcję trzech składowych: rozproszenia, autonomiczności, heterogeniczności (Rys. 10). Poszczególne składowe wykresu należy interpretować w sposób następujący:

- *Rozproszenie* – zmienna ta określa w jaki sposób realizowana jest obsługa rozproszonych danych. Wartość 0 odpowiada scentralizowanemu sterowaniu danymi, zaś wartość maksymalna całkowitemu rozproszeniu.
- *Heterogeniczność* – dotyczy systemów lokalnych. W celu określenia, czy poszczególne DBMS są heterogeniczne, bierze się pod uwagę szeroką gamę kryteriów, np.: sprzęty, protokoły sieciowe, menadżerów danych itd.
- *Autonomiczność* – wartość 0 interpretowana jest jako silna integracja poszczególnych baz danych, zaś wartość maksymalna jako całkowita ich izolacja.

W sposób zwarty charakteryzuje ona cechy podstawowych rodzajów DDBMS. Funkcję tę można zinterpretować jako przegląd możliwych sposobów zestawienia wielu baz danych tak, aby mogło je wykorzystywać wiele DBMS.

LITERATURA

- [BellGr1994] Bell D., Grimson J., *Distributed Database Systems*, Addison-Wesley Publishing Company, 1994.
- [Blink1996] Blinkowski G., *Systemy klient-server*, Helion, 1996.
- [CerPel1984] Ceri S., Pelagatti G., *Distributed Database – Principles and Systems*, McGraw-Hill Inc, 1984.
- [Cormic1972] McCormic W.T., Schweitzer P., White T.W., *Problem Decomposition and Data Reorganization by Clustering Technique*, 1972.
- [DaHw1984] Dayal U, Hwang H., *View Integration and Generalisation for Database Integration in a Multidatabase*, IEEE Trans.on Software Engineering, 10(6), 1984..
- [Date2000] Date C. J., *Wprowadzenie do systemów baz danych*, WNT, Warszawa 2000.
- [Diek1997] Diekhoff G., Nichols S., Setters T., *Distributed Database Management Systems and Their Use*, 1996.
- [ElmaPu1991] Elmagarmid A.K., Pu C., *Special Issue on Heterogeneous Databases*, ACM Comp. Surv., 1991.
- [Jasz1997] Jaskiewicz A., *Inżynieria oprogramowania*, Helion, 1997.
- [Kim1995] Kim W., *Modern Database Systems*, ACM Press, 1995.
- [MazurPi2001] Mazur Z., Piechota K., *Projektowanie rozproszonych baz danych*, Zeszyty Naukowe Wydziałowego Zakładu Informatyki Politechniki Wrocławskiej, Zeszyt 3 (w przygotowaniu), Oficyna Wydawnicza Politechniki Wrocławskiej, 2001.
- [Micha1997] Michałkiewicz P., *Replikacja danych*, Software 8, 1997.
- [Niezg1997] Niezgódka M., *Strategiczne kierunki rozwoju serwisów informatycznych*, INFOBAZY'97 - Konferencja pod patronatem Komitetu Badań Naukowych., 1997.
- [Özs1994] Özs M. T., *Changing Infrastructure – New Challenges in Distributed Database Management*, Laboratory of Database Systems Research, Department of Computing Sciences University of Alberta, 1994.
- [ÖzsVa1991] Özs M. T. i Valduriez P., *Principles of Distributed Database Systems*, Prentice-Hall, 1991.

- [ÖzsuVa1994] Özsu M. T. i Valduriez P., *Distributed Database Systems: Where are we now?*-IEEE Computer, 1994.
- [ÖzsuVa1994a] Özsu M. T., Valduriez P., *Distributed Database Management*, wykłady w postaci slajdów udostępnione przez Özsu M. T., 1994.
- [ÖzsuVa1996] Özsu M. T., Valduriez P., *Distributed and Parallel Database Systems – Technology and Current State-of-the-Art*, ACM Computing Surveys, 1996.
- [ÖzsuVa1997] Özsu M. T., Valduriez P., *Distributed and Parallel Database Systems, Handbook of Computer Science and Engineering*, CRC Press, 1997.
- [Papa1994] Papadopoulos C.V., *On the Heterogeneity of Distributed Database – Integrating Commit Protocols*, The 14th International Conference on Distributed Computing Systems, 1994.
- [red.Özsu1994] Özsu M. T., Dayal U., Valduriez P., *An Introduction to Distributed Object Management*, Distributed Object Management, 1994.
- [Rusink1994] Rusinkiewicz Marek, *Multidatabase systems*, Houston, Texas, June 1994.
- [SL1990] Sheth A., Larson J., *Federated Database Systems*, ACM Computing, Surveys, Vol. 22, No 3, Sept. 1990.

ARCHITECTURES OF DISTRIBUTED DATABASES

In recent years, the availability of databases and computer networks have given rise to a new field, distributed databases (DDB). The popularity of DDB technology is a result of rising conviction of that centralised and single processor systems approach slowly to the theoretical deadline of processing speed. As with any emerging technology, distributed database management systems (DDBMS) have their share of fulfilled and unfulfilled promises. The design of DDB still seems to be one of unsolved issue.

*baza danych, rozproszona baza danych,
system zarządzania rozproszoną bazą danych*

Zygmunt MAZUR*
Kamila PIECHOTA*

WŁASNOŚCI SYSTEMÓW ZARZĄDZANIA ROZPROSZONYMI BAZAMI DANYCH

Projektant rozproszonej bazy danych powinien posiadać podstawową wiedzę z zakresu mechanizmów funkcjonowania rozproszonych systemów zarządzania bazami danych (DDBMS), ponieważ sposób w jaki rozproszona baza danych zostanie zaprojektowana ma duży wpływ na optymalne wykonywanie transakcji i zapytań. W artykule zostaną omówione cechy systemów zarządzania rozproszonymi bazami danych i mechanizmy umożliwiające ich realizację.

1. WPROWADZENIE

Dla zrozumienia strategii projektowania, niezbędne jest zapoznanie się z podstawowymi mechanizmami zapewniającymi funkcjonowanie systemu zarządzania rozproszoną bazą danych. Sposób, w jaki zostanie zaprojektowana rozproszona baza danych, ma znaczący wpływ na możliwości optymalnie skutecznego przetwarzania zapytań czy transakcji. Dlatego projektant rozproszonej bazy danych powinien posiadać podstawową wiedzę z zakresu mechanizmów funkcjonowania rozproszonych systemów zarządzania bazami danych (DDBMS).

Jak każda rozwijająca się technologia, tak i technologia DDBMS posiada swoje wady i zalety. Praca ta jest próbą przedstawienia obecnego stanu zaawansowania technologii rozproszonych baz danych.

2. ZALETY ROZPROSZONYCH BAZ DANYCH

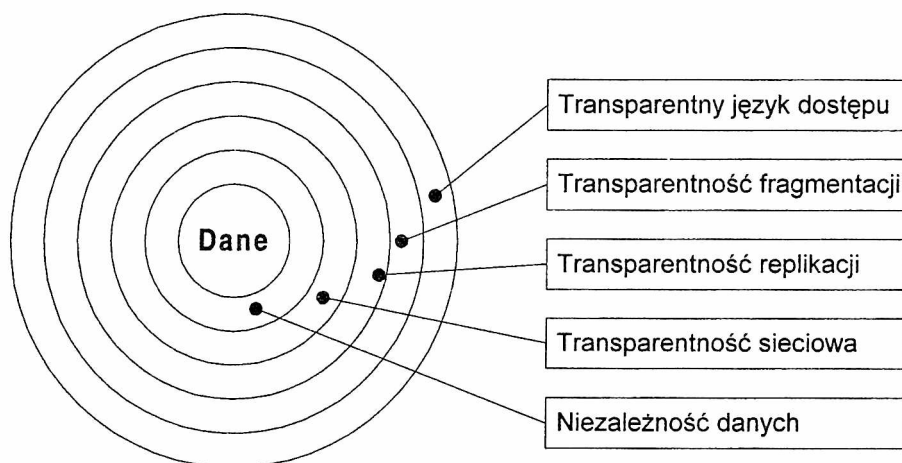
W niniejszym podrozdziale zostały zestawione najczęściej cytowane w literaturze zalety DDBMS i ogólnie zarysowane mechanizmy, dzięki którym są one realizowane. Zalety te należy jednak w wielu przypadkach traktować bardziej jako niespełnione obietnice tej technologii, niż cechy komercyjnych systemów.

* Wydziałowy Zakład Informatyki Politechniki Wrocławskiej, Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, mazur@ci-1.ci.pwr.wroc.pl

2.1. TRANSPARENTNE ZARZĄDZANIE ROZPROSZENIEM I REPLIKACJĄ DANYCH

Scentralizowane bazy danych zrewolucjonizowały podejście do przetwarzania danych, przenosząc odpowiedzialność za definiowanie struktur danych oraz ich zarządzanie, z aplikacji na wyspecjalizowane systemy zarządzania. Tak więc, programy aplikacyjne stały się niezależne od logicznej i fizycznej organizacji danych i na odwrót. Niezależność ta nazywana jest niezależnością danych (ang. *data independence*). Technologia rozproszonych baz danych (ang. *Distributed Database – DDB*) rozszerza znaczenie koncepcji niezależności danych na środowiska, gdzie dane są dodatkowo rozproszone i replikowane w węzłach sieci komputerowej. Transparentny dostęp do rozproszonych danych jest zapewniany dzięki kilku formom transparentności: transparentności sieciowej (rozproszenia), transparentności replikacji i fragmentacji. Transparentność (przezroczystość) należy rozumieć jako oddzielenie wyższych poziomów semantycznych systemu od zagadnień implementacyjnych niższych poziomów. Inaczej mówiąc, jest to ukrycie detali dotyczących implementacji przed wyższymi warstwami systemu i przed jego użytkownikami.

Transparentność sieciowa występuje w systemach gdzie istnienie sieci komputerowej jest niewidoczne dla aplikacji użytkowników. Wymagane jest, aby taki rozproszony system wykonywał poprawnie polecenia bez względu na fizyczną lokalizację danych i miejsce wydania polecenia (*transparentność lokalizacji*), oraz aby zapewniał unikalność nazewnictwa wszystkich obiektów systemu (*transparentność nazewnictwa*). Użytkownik nie musi wiedzieć, gdzie dane są zlokalizowane fizycznie. Z logicznego punktu widzenia ma zapewniony dostęp do danych w taki sam sposób, jakby były one w jednym miejscu.



Rys. 1. Warstwy transparentności DDBMS
Fig. 1. Layers of transparencies DDBMS

Transparentność replikacji jest to cecha systemu rozproszonego oznaczająca, że jeżeli istnieją repliki (kopie) obiektów bazy danych, to za zarządzanie nimi odpowiada system, a nie użytkownik. Powinien on mieć pewność, że dane, do których uzyskuje dostęp, są aktualne.

DDBMS powinien umożliwiać przeprowadzenie fragmentacji danych. Dane mogą być umieszczane tam gdzie są najczęściej wykorzystywane, co pozwala zredukować koszty komunikacji sieciowej. Jednak fakt fragmentacji bazy powinien być niewidoczny dla użytkownika (transparentność fragmentacji). Aby to zapewnić, rozproszony system musi posiadać mechanizmy pozwalające na konwersję zapytań użytkownika zdefiniowanych na globalnych (wirtualnych) relacjach, w zapytania zdefiniowane na fragmentach.

W swojej idealnej formie, pełna transparentność rozproszonego systemu bazodanowego mogłaby być zapewniona przez interfejs, w postaci języka zapytań, będącego elementem rozproszonego DBMS, który to język nie różniłby się od języka zapytań używanego w scentralizowanych bazach danych.

2.2. WYSOKA NIEZAWODNOŚĆ ZAPEWNIONA PRZEZ ROZPROSZONE TRANSAKCJE

Wysoka niezawodność systemów rozproszonych, jest uzyskiwana dzięki stosowaniu replikacji. Uszkodzenie jednego z węzłów lub linii komunikacyjnej nie jest jednoznaczne z uszkodzeniem całego systemu. DDBMS powinien dostarczać taki mechanizm, który w momencie uszkodzenia jednego z elementów DDB zapewni dostęp do części bazy będącej repliką uszkodzonej. Takim mechanizmem są rozproszone transakcje.

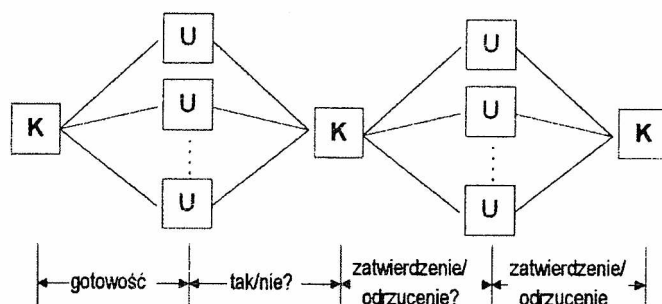
Transakcja składa się z sekwencji operacji wykonywanych na bazie danych. Jest wykonywana jako atomowa operacja (może się zakończyć na jeden z dwóch sposobów: zatwierdzenie lub odrzucenie) przeprowadzająca bazę danych z jednego stanu w inny. DBMS powinien gwarantować, że konkurencyjne wykonywanie transakcji, uszkodzenia systemu w czasie trwania transakcji, niepoprawne transakcje itd. nie spowodują zachwiania integralności bazy danych.

Rozproszona transakcja jest wykonywana we wszystkich miejscach, w których udostępniane są dane potrzebne do jej przeprowadzenia. W DDB aplikacja użytkownika formułuje transakcję na globalnym, logicznym widoku bazy danych, obarczając odpowiedzialnością za jej poprawne wykonanie DDBMS. System zarządzania rozproszona bazą danych odpowiada za dostęp do poszczególnych lokalnych baz danych, koordynując problemy związane z uszkodzeniami węzłów i łączy komunikacyjnych podczas wykonywania rozproszonej transakcji. Przetwarzanie transakcji i transparentność przetwarzania SZRBD są ściśle związane. Oba zagadnienia wymagają rozwiązania problemów związanych z nazewnictwem

rozproszonych obiektów oraz zarządzaniem rozproszonymi katalogami, jednak wymagania te wynikają z realizacji innych potrzeb.

Realizacja przetwarzania rozproszonych transakcji wymaga implementacji protokołów zarządzających kontrolą dostępu i niezawodnością transakcji rozproszonego systemu bazodanowego, które są znacząco bardziej skomplikowane niż ich odpowiedniki dla scentralizowanych DBMS. Typowy algorytm zarządzania kontrolą dostępu jest rozszerzeniem popularnego dwufazowego algorytmu *blokowania* (ang. *two-phase locking - 2PL*) o zarządzanie blokadami obiektów DDB znajdujących się w różnych lokalizacjach. Na rozproszone protokoły niezawodności (ang. *distributed reliability protocols*) składają się rozproszone protokoły wykonawcze (ang. *distributed commit protocols*) i procedury powrotu do stanu sprzed uszkodzenia (ang. *recovery procedures*).

Protokół wykonawczy, aby zapewnić atomowość rozproszonej transakcji, musi upewnić się, że dana transakcja spowoduje taki sam efekt (zatwierdzenia lub odrzucenia) we wszystkich miejscach, których dotyczy. Najczęściej stosowanym w komercyjnych produktach jest protokół 2PC (ang. *two-phase commit protocol*). Można w nim wyróżnić dwie fazy. W pierwszej fazie koordynator (K) zbiera potwierdzenia gotowości do wykonania transakcji od uczestników (U). W drugiej fazie następuje wykonanie transakcji przez wszystkich uczestników. Koordynator odrzuca transakcję, gdy nie uzyska potwierdzenia wszystkich uczestników w określonym czasie (Rys. 2). Odrzucenie transakcji odbywa się zgodnie z protokołami zakończenia (przeterminowania), które nie mogą być blokujące. Jeżeli zebrano wszystkie potwierdzenia, koordynator zatwierdza transakcję. W przypadku uszkodzenia jednego z miejsc, stosowane są protokoły odtwarzania.



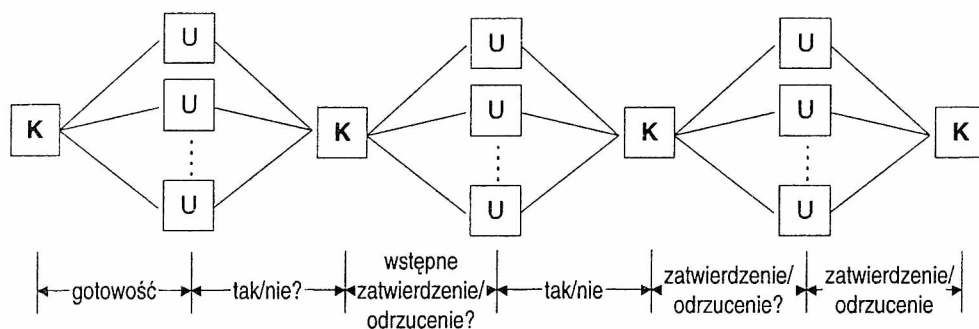
Rys. 2. Protokół 2PC

Fig. 2. Protocol 2PC

W rozproszonych bazach danych występują uszkodzenia zarówno miejsc, jak i łączy komunikacyjnych, dlatego powinno się stosować protokoły nie blokujące zakończenia, oraz niezależne odtwarzanie (na wypadek gdyby rozproszona baza na skutek awarii została rozdzielona). W przypadku uszkodzenia koordynatora, uczestnik może zostać zablokowany. Dla protokołu 2PC nie jest możliwa realizacja nie

blokującego protokołu zakończenia i nie gwarantuje on też niezależnego odtworzenia rozproszonej bazy danych do postaci spójnej.

Protokół 3PC jest algorytmem nie blokującym. Dla protokołu zatwierdzania blokowanie nie zachodzi, ponieważ w przeciwieństwie do 2PC, przeterminowanie z powodu braku koordynatora powoduje przejście do wyboru nowego koordynatora, który kończy transakcję (Rys. 3). Protokół zakończenia obsługuje przeterminowanie oczekiwania na potwierdzenie od miejsc biorących udział w transakcji. Protokoły odtwarzania są podobne do odpowiedników w 2PC, wymagają jednak wymiany większej liczby komunikatów, aby uniknąć blokowania. Protokół 3PC cechuje się większą niezawodnością, jednak komercyjne systemy go nie realizują.



Rys. 3. Protokół 3PC

Fig. 3. Protocol 3PC

Przy aktualizacji miejsca, wartość poprzednia jest tracona. Niezbędne jest zachowanie informacji o zmianie stanu bazy danych, która umożliwi przywrócenie spójności po wystąpieniu uszkodzenia (dla potrzeb procedur odtwarzania). Do tego celu stosuje się tzw. **log** czyli **dziennik** [Date2000], który dla pojedynczej transakcji zapamiętuje: identyfikator transakcji, typ akcji, pozycję wykonanej akcji, wartość pozycji przed modyfikacją, rekord zakończenia transakcji z warunkiem zakończenia. Log bazy danych może być zapisywany synchronicznie lub asynchronicznie. Do tego celu służy protokół utrzymywania logu z zapisem w przód (ang. *write-ahead logging* – WAL), aby w przypadku awarii dane były już w logu. Przy aktualizacji poza miejscem, w którym rozpoczęła się transakcja, stosuje się technikę cieni lub tworzenia plików różnicowych. W technice cieni tworzony jest obraz pozycji z nowymi wartościami. Poprzednia wartość zachowywana jest na wypadek konieczności jej odtworzenia. Technika plików różnicowych polega na tworzeniu pliku zawierającego wartość różnicową poprzedniej i nowej wartości pozycji. Zatwierdzenie polega na wpisaniu nowych wartości na podstawie pliku różnicowego.

Replikacja danych zwiększa poziom niezawodność, jednak z drugiej strony zawsze istnieje ryzyko, że w systemie istnieje kopia danych, która nie mogła być uaktualniana ze względu na jej chwilowe uszkodzenie lub niedostępność spowodowaną

uszkodzeniem łączy komunikacyjnego. Utrzymywanie zgodności replik wymaga implementacji protokołów kontroli replik (ang. *replica control protocols*). Najprostszy algorytm kontroli replik jest realizowany przez protokół ROWA (ang. *read one write all*). Zakłada on, że jedna operacja „logicznego odczytania” zreplikowanych danych jest realizowana przez pojedynczą operację fizycznego odczytania z dowolnej repliki, zaś operacja „logicznego zapisu” wymaga wykonania operacji fizycznego zapisu na wszystkich kopiach. W publikacjach można spotkać rozważania nad bardziej skomplikowanymi protokołami kontroli replik, opierającymi się na podejściu mniej restrykcyjnym przy wykonywaniu operacji zapisu.

Protokoły zarządzające kontrolą dostępu i niezawodnością transakcji są jednym z najintensywniej badanych problemów w rozproszonych bazach danych. Jednak ich implementacja w komercyjnych produktach jest często bardzo uproszczona. Być może wynika to z braku konkretnych testów porównawczych, które w sposób nie budzący wątpliwości wykazałyby wzrost skuteczności przetwarzania transakcji przy ich zastosowaniu w środowiskach rozproszonych. Komercyjne systemy zapewniają tylko pewien ograniczony poziom transparentnego przetwarzania. Niektóre z nich wymagają, aby wszystkie bazy podczas wykonywania rozproszonej transakcji były otwarte (np. Oracle), inne udostępniają tylko podstawowe mechanizmy, wymagając od użytkownika aplikacji kontrolowania operacji wykonania transakcji. Nie zapewniają one atomowości przetwarzania transakcji.

Niezawodność systemów DDB wynika również z braku punktów będących centrum zarządzania. W rozproszonym systemie bazodanowym nie powinno być węzłów, które odpowiadają za zarządzanie całym systemem. Wykluczona jest centralizacja którejkolwiek z funkcji DDBMS, jak np.: przetwarzania zapytań czy zarządzania transakcjami. Miejsca takie mogłoby się okazać wąskim gardłem przetwarzania, obniżającym niezawodność systemu. Awaria takiego punktu spowodowałaby niepoprawne działanie całego systemu.

2.3. MOŻLIWOŚĆ LEPSZEJ OPTIMALIZACJI DOSTĘPU DO DANYCH

Możliwość usprawniania dostępu do danych w DDBMS wynika z dwóch podstawowych czynników. Pierwszy z nich wiąże się z możliwością zarządzania bazą danych rozłożoną na fragmenty, co umożliwia lokalizację danych w miejscach, gdzie najczęściej zgłaszany jest dostęp do nich. Dzięki temu w poszczególnych węzłach DDB można umieścić mniejsze porcje danych, co zmniejsza współzawodnictwo o czas procesora oraz liczbę wykonywanych operacji dostępu do pamięci zewnętrznej, jaka występuje w alternatywnej scentralizowanej bazie danych. Lokalizacja danych w miejscach gdzie najczęściej są wykorzystywane, pozwala zredukować opóźnienia związane ze zdalnym dostępem do danych.

Drugi czynnik związany jest z wbudowaną możliwością DDBMS przetwarzania równoległego. DDBMS może przetwarzać równocześnie wiele zapytań. Jak już wcześniej stwierdzono, w rozproszonym DBMS nie powinno być węzła, który by odpowiadał za koordynację którejkolwiek z jego funkcji. Dotyczy to również przetwarzania zapytań. Każdy z węzłów DDB może być punktem zadania i wykonania zapytania zdefiniowanego na globalnym widoku bazy. Z drugiej strony, każde zapytanie może zostać przetworzone w sposób równoległy, rozbijając globalne zapytanie na podzapytania wykonywane w różnych węzłach, uzyskujące dostęp do różnych części DDB.

Z punktu widzenia przetwarzania zapytań (gdy dostęp do danych jest typu „tylko do odczytu”) korzystne jest replikowanie danych. Jednak należy pamiętać, że wykonanie operacji aktualizujących na zreplikowanej bazie, wymaga stosowania wysoce skomplikowanych protokołów kontroli dostępu i wykonywania transakcji.

Komercyjne systemy w celu zoptymalizowania dostępu do danych wykorzystują modele transakcyjne, będące uproszczeniem modelu idealnego. Jedną z alternatyw jest bieżące wykonywanie tylko transakcji typu „tylko do odczytu”, zaś wykonanie transakcji uaktualniania jest odkładane w czasie. W określonych odstępach czasu nagromadzone transakcje są wykonywane. Wówczas nie wykonuje się transakcji odczytu danych. Inna alternatywa polega na zarządzaniu dwiema kopiami tych samych danych w każdym z węzłów. Jedna z kopii udostępniana jest dla transakcji odczytu, na drugiej zaś dokonywane są na bieżąco operacje modyfikacji. Co określony odcinek czasu kopia używana do odczytu jest uaktualniana. Obie alternatywy nie eliminują jednak konieczności synchronizacji operacji modyfikacji na wszystkich kopiach. Jednak są jakimś rozwiązaniem stosowanym dla polepszenia możliwości szybkiego i skutecznego dostępu do danych, uzyskiwanego przez uproszczenie działania protokołów transakcyjnych.

Charakterystyka optymalności dostępu, czy zarządzania danymi dla systemów DDB nie jest dobrze zdefiniowana. Istnieje zbyt mało DDBMS zbliżonych do ideału, aby można było sformułować obiektywny osąd. Zaś dla istniejących systemów nie ma publikacji, na temat przeprowadzonych pomiarów tej cechy.

2.4. MOŻLIWOŚĆ ŁATWEGO I TANIEGO ROZBUDOWYWANIA SYSTEMÓW ROZPROSZONYCH

Rozproszone środowiska przetwarzania jest łatwiej przystosować do rosnących rozmiarów baz danych. Dodanie mocy przetwarzania, czy przechowywania danych, odbywa się w sposób bardziej naturalny, niż w systemach scentralizowanych. Cecha ta nazywana jest skalowalnością baz danych.

Rozwój technologii sprzętu komputerowego miał niewątpliwie wpływ na polepszenie atrakcyjności technologii DDB pod względem ekonomicznym. Zmiany cenowej i wydajnościowej charakterystyki sprzętu komputerowego spowodowały, że bardziej

opłacalne jest tworzenie systemów składających się z wielu słabszych maszyn, niż jednej o ekwiwalentnej mocy.

Inna ekonomiczna przesłanka wynika z różnicy w kosztach komunikacyjnych i telekomunikacyjnych. W poprzednim punkcie lokalizacja danych została użyta jako argument za DDB, gdyż pozwala na redukcję opóźnień związanych z komunikacją. Mniejszy koszt stosowania DDB jest również związany z ograniczeniem komunikacji sieciowej. Inaczej mówiąc, koszt rozproszenia danych wraz z koniecznością ich przepływu w sieci wykonywanym od czasu do czasu (gdy zachodzi konieczność wykonania rozproszonego zapytania), może być niższy niż koszt telekomunikacyjny częstych zdalnych dostępów do centralnej bazy danych. Należy jednak powiedzieć, że argument ten jest dość kontrowersyjny, gdyż większość komercyjnych DDBMS korzysta z szybkich sieci lokalnych. W takim przypadku koszt komunikacji jest minimalny, nie warty rozpatrywania. Wykorzystanie rozległej sieci komputerowej w technologii DDB jest ciągle zagadnieniem dyskutowanym i dość kontrowersyjnym.

3. NIEROZWIĄZANE PROBLEMY

Powyżej zostały ogólnie zarysowane najczęściej wymieniane zalety DDBMS. Jednak można powiedzieć, że w wielu przypadkach są to jedynie niespełnione obietnice. Istnieją problemy technologiczne, jak i związane z samym użytkowaniem rozproszonych baz danych, które powstrzymują rozwój tej technologii. Niektóre z nich zostały już zasygnalizowane. Niniejszy rozdział jest zestawieniem trudności, jakie spotyka się w technologii DDB. Starano się wykazać źródła i perspektywy rozwiązania tych trudności.

3.1. PROBLEMY ZWIĄZANE Z BADAANIAMI NAD POPRAWĄ WYDAJNOŚCI ROZPROSZONYCH BAZ DANYCH

Jak już wcześniej wspomniano, do tej pory nie zaproponowano kryteriów, które by w sposób obiektywny umożliwiły wykazanie wzrostu skuteczności przetwarzania danych, odnośnie do zaproponowanych projektowych rozwiązań w DDBMS. Również nie ma nie budzącej wątpliwości odpowiedzi na pytania, o skalowalność tradycyjnych protokołów czy algorytmów odnośnie geograficznego rozproszenia systemu lub zwiększania liczby komponentów systemu. Szczególne znaczenie ma zastosowanie odpowiednich mechanizmów przetwarzania transakcji w DDBMS, działających w sieci rozległej, której szybkość przesyłania danych pozostawia wiele do życzenia.

Wymienione problemy są tylko częścią bardziej ogólnego, dotyczącego możliwości poprawnego zobrazowania roli architektury sieci i protokołów sieciowych w badaniach nad poprawą wydajności systemów uzyskiwaną przez ich rozproszenie.

Większość badań nad wydajnością systemów rozproszonych opiera się na bardzo prostym, często nierealistycznym modelu kosztów komunikacji sieciowej, nie biorącym pod uwagę takich czynników jak obciążenia, wielkości komunikatów czy rozmiar sieci, oraz wzajemnej korelacji tych wartości. Dla zachęcenia producentów do zainteresowania się projektami algorytmów i protokołów mogących znaleźć zastosowanie w rozproszonych bazach danych, konieczne jest wykazanie opłacalności ich użycia. Symulacje dokonywane w tym celu, powinny być wiarygodne. Muszą one odzwierciedlać rzeczywiste warunki funkcjonowania rozproszonego.

Projektowanie DDB wymaga uwzględnienia większej liczby czynników, niż projektowanie scentralizowanych baz danych. Problemy wynikają głównie z trudności związanych ze zdefiniowaniem miar poszczególnych czynników oraz wagi wpływu każdego z nich na optymalność projektowanego systemu. Zaproponowane metodologie są raczej heurystycznymi zaleceniami. W wielu wypadkach projektant musi bazować na swoim doświadczeniu oraz intuicji.

Przyjęta metodologia zależy przede wszystkim od architektury DDBMS, dla którego projektowana jest baza danych. Różnice w przyjętych strategiach projektowania homogenicznych rozproszonych baz danych i multibaz danych wynikają z odmienności celów jakie mają być zrealizowane.

Problemy związane z projektowaniem zostały już zasygnalizowane przy okazji omawiania typologii DDBMS. Szczegółowe ich rozpatrzenie jest tematem dalszej części pracy.

3.2. PRZETWARZANIE ROZPROSZONYCH ZAPYTAŃ

Zapytanie (*ang. query*) jest to żądanie informacji z bazy danych. Przetwarzanie i optymalizacja zapytań jest jednym z podstawowych problemów w DBMS. W rozproszonych bazach danych, w porównaniu z bazami scentralizowanymi, jest to trudniejsze ze względu na większą liczbę parametrów wpływających na wydajność zapytań rozproszonych.

Języki umożliwiające manipulację danymi, są zazwyczaj językami nieproceduralnymi. Oznacza to, że użytkownik formułując zapytanie do bazy danych określa tylko jakie informacje chce uzyskać, nie definiując w jaki sposób ma to być zrobione. Ciężar poszukiwania sposobu odpowiedzi na zapytanie spoczywa na procesorze zapytań (*ang. query processor*), który pośredniczy między zapytaniem użytkownika a operacjami manipulacji danymi na poziomie fizycznym.

Procesor rozproszonych zapytań przekłada zapytanie wysokiego poziomu skierowane do rozproszonej bazy danych (jednak widzianej przez użytkownika jako scentralizowana baza danych) na możliwe najefektywniejszy plan wykonania zapytań niskiego poziomu skierowanych do lokalnych baz danych. W pracy procesora zapytań należy wyróżnić dwa ważne aspekty. Po pierwsze, transformacja zapytania musi być poprawna, tzn. plan wykonania zapytań lokalnych musi dawać spodziewane efekty.

W realizacji tego zadania okazuje się przydatna algebra relacyjna wraz z regułami przekształceń. Po drugie, plan wykonania powinien być optymalny, tzn. powinien minimalizować funkcję kosztu uzyskania dostępu do danych. Wymaga to rozpatrzenia przez procesor zapytania alternatywnych planów wykonania zapytania, w celu dokonania wyboru najlepszego z nich.

Ze względu na trudności związane z realizacją obu zadań równocześnie, są one najczęściej realizowane w dwóch oddzielnych etapach, nazywanych lokalizacją danych oraz globalną optymalizacją. Zazwyczaj oba etapy poprzedzone są etapem dekompozycji zapytania, której celem jest maksymalne uproszczenie zapytania oraz zapisania go w języku algebry relacyjnej. Zadaniem etapu lokalizacji danych jest przełożenie wejściowego algebraicznego zapytania zdefiniowanego na globalnej bazie danych na ekwiwalentny zbiór zapytań zdefiniowanych na fragmentach (tzn. zapytania zdefiniowane na fragmentach rozproszonej bazy danych przechowywanych w różnych węzłach sieci). Następnie cząstkowe zapytania mogą być upraszczane przez użyciu reguł algebry relacyjnej. Etap globalnej optymalizacji generuje optymalny plan wykonania zapytań dotyczących poszczególnych fragmentów, podejmując decyzje dotyczące kolejności wykonania zapytań, przesyłania danych pomiędzy węzłami, wyboru algorytmów operacji bazodanowych dotyczących zarówno baz lokalnych, jak i bazy rozproszonej. Z etapem tym związane jest wiele problemów, wynikających z różnorodności czynników jakie należy w nim uwzględnić.

Za kryterium oceny optymalności planu wykonania zapytania przyjmuje się całkowity koszt modelu przetwarzania zapytania. Uwzględnia on metody dostępu, koszt manipulacji danymi na poziomie fizycznym oraz statystyki zbierane do celów optymalizacji. Optymalizacja wymaga także uwzględnienia restrykcji nałożonych na zapytanie, takich jak np. limity czasowe.

Mimo, że współczesne języki zapytań są wysoko rozwinięte, optymalizacja dotyczy zapytań opartych na operacjach selekcji, projekcji i złączeń, oraz na operatorach logicznych. Dla tych rodzajów zapytań zostały opracowane strategie dające dobre wyniki optymalizacyjne. Szczególnie dotyczy to strategii dla operacji złączeń i półzłączeń. Istnieje jednak wiele innych ważnych rodzajów zapytań, np. opartych o operacje unii, produktu kartezjańskiego, zbioru różnicowego, generalizacji itd. Niektóre z tych operacji okazują się szczególnie przydatne w kontekście rozproszonych baz danych (jak np. operacja unii).

Jak to zostało pokazane, optymalizacja zapytań w rozproszonych bazach danych jest znacznie bardziej złożona, a zatem bardziej kosztowna, niż w przypadku scentralizowanych baz danych. Głównie wynika to z konieczności przetestowania znacznie większej liczby możliwych strategii wykonania zapytania. Konieczne jest przeanalizowanie kosztu optymalizacji, w kontekście zysku jaki jest otrzymywany dzięki zastosowaniu wygenerowanego planu wykonania zapytania. Wysoki koszt optymalizacji jest do zaakceptowania w przypadku poszukiwania optymalnego planu wykonania często zgłaszanych zapytań, gdyż z czasem zostanie on zamortyzowany.

W przypadku zapytań zgłaszanych jednokrotnie, optymalizacja jest operacją zbyt kosztowną.

Globalna optymalizacja zapytania jest zazwyczaj przeprowadzana przed wykonaniem zapytania (optymalizacja statyczna). Głównym problemem wynikający z takiego podejścia jest fakt, że model kosztów dla optymalizacji zapytań może być nieaktualny. Może to być spowodowane reorganizacją rozproszonej bazy danych lub zmianą wielkości fragmentów. Wymagane jest, co pewien czas, uaktualnienie zebranych danych o kosztach planów wykonania poszczególnych zapytań.

3.3. PRZETWARZANIE ROZPROSZONYCH TRANSAKCJI

Trudno jest uwierzyć, że pomimo intensywnych i prowadzonych przez wiele instytucji badań, nad przetwarzaniem rozproszonych transakcji, wciąż istnieją problemy kontrowersyjne i nie posiadające rozwiązań.

W zreplikowanych rozproszonych bazach danych operacje manipulacji danymi są zdefiniowane na logicznych obiektach danych. Protokoły kontroli replik odpowiedzialne są za odwzorowanie operacji zdefiniowanej na logicznych obiektach danych, na operacje zdefiniowane na wszystkich fizycznych kopiach tych obiektów. Ich zadaniem jest czuwanie nad zgodnością wszystkich kopii. Wcześniej omówiony algorytm ROWA jest najprostszą z możliwych metod. Jednak wymaga on, aby była zachowana zgodność replik, wszystkie z kopii muszą mieć identyczną wartość. Wykorzystanie takiego algorytmu, ze względu choćby na możliwość wystąpienia czasowych uszkodzeń niektórych z kopii, jest praktycznie niemożliwe.

Zostały opracowane liczne metody kontroli replik. Istnieje jednak potrzeba eksperymentalnego ich przetestowania. Pozwoliłoby to na porównanie konkurencyjnych technik. Jedną z przyczyn powolnego rozwoju technik replikacji jest brak powszechnie zaakceptowanego modelu tolerancji uszkodzeń. Na przykład jednym z modeli używanym do testowania zmian wydajności systemów dzięki zastosowaniu replikacji jest model Markowa. Zakłada on statystyczną niezależność poszczególnych uszkodzeń oraz rzadkość występowania podziału sieci spowodowanego uszkodzeniem jednego z węzłów. Dokładnie nie wiadomo, czy założenia te nie są zbyt dużym uproszczeniem (czy dla potrzeb badania replikacji można je zaakceptować), jak też nie wiemy jak wrażliwy jest model Markowa na te założenia. Bez empirycznych pomiarów, używanie modelu Markowa dla potrzeb symulacji działania protokołu replikacji jest zbyt ryzykowne (wyniki symulacji mogą być zbyt dalekie od rzeczywistości). Implementacja nowego protokołu kontroli replikacji bez wcześniejszego zbadania jego możliwości usprawnienia działania systemu, jest zbyt kosztowna. Tak więc, aby możliwe były symulacje dające rzeczywiste wyniki, istnieje potrzeba przeprowadzenia empirycznych badań mających na celu zebranie statystyk uszkodzeń w rzeczywistych systemach, które to badania

pozwołyby na konstrukcje prostego modelu typowych obciążeń związanych z uszkodzeniami.

Dla dalszego skutecznego rozwoju technologii rozproszonych baz danych konieczne jest staranne opracowanie modelu przetwarzania transakcji. Aby osiągnąć większą wydajność konieczne jest stworzenie bardziej złożonych modeli, opartych o skuteczniejsze mechanizmy zapewniania poprawności, niż np. prosta szeregowałość. Istniejące prace w tym kierunku można sklasyfikować w dwóch wymiarach: poszukiwania lepszych modeli transakcyjnych oraz struktur obiektów podlegających operacjom.

Z wielu poszukiwanie bardziej złożonych modeli transakcji jest szczególnie ważne dla technologii rozproszonych baz danych. Po pierwsze, przetwarzanie transakcji w rozproszonych systemach multibazodanowych pozwoli na odciążenie semantycznego skomplikowania modelu tych systemów. Po drugie, nowe typy aplikacji używane w najbliższej przyszłości będą wspomagane przez systemy rozproszonych baz danych (jak np.: projektowanie inżynierskie, systemy informacyjne w biurach, wspomaganie współpracy wielu pracowników itd.). Wymagają one użycia skomplikowanych modeli transakcji, wykorzystujących bardziej złożone operacje i wykonywanych na bardziej złożonych typach danych. Dodatkowo, aplikacje te wymagają opracowania nowych strategii współdzielenia danych, znacząco różniących się od mechanizmów, do których przywykliśmy. Tak np., aplikacje wspomagające zarządzanie współpracą nad realizacją projektów wymagają wydajniejszych metod pozwalających na współdzielenie zasobów, niż metody stosowane w standardowych aplikacjach uzyskujących dostęp do baz danych. Zmieniające się wymagania i nowe zastosowania powodują konieczność rozwoju nowych modeli transakcji oraz towarzyszących im kryteriów poprawności.

3.4. INTEGRACJA Z ROZPROSZONYM SYSTEMEM OPERACYJNYM

Przez długi okres czasu uważano, że budowanie DBMS zarówno scentralizowanych jak i rozproszonych z wykorzystaniem funkcji systemu operacyjnego jest niewskazane. Opinie te wynikały z rozbieżności wymagań stawianych DBMS i funkcjonalności istniejących systemów operacyjnych. Podobnie wymagania stawiane DDBMS (jak np.: przetwarzanie rozproszonych transakcji, włączając zarządzanie kontrolą dostępu i mechanizmy odtwarzania, wydajne zarządzanie obiektami trwałymi, bardziej skomplikowane metody dostępu) były rozbieżne z funkcjonalnością istniejących rozproszonych systemów operacyjnych. Dodatkowo, wykorzystanie rozproszonych systemów operacyjnych w DDBMS wymagało modyfikacji sposobu realizacji jego tradycyjnych funkcji (jak np.: zarządzanie zadaniami, nazewnictwo, zarządzanie buforami). W punkcie tym zostaną zasygnalizowane zagadnienia integracji

rozproszonych DBMS i rozproszonych systemów operacyjnych. Dotyczą one architektury systemów, transparentnego nazewnictwa zasobów, zarządzania trwałymi obiektami, zdalnej komunikacji i zarządzania transakcjami. Bardziej szczegółowe rozważania można znaleźć w pozycji bibliograficznej [ÖzsüVa1991].

Istotnym czynnikiem rozważań nad aspektami architektury jest fakt, że rozproszone bazy danych i systemy operacyjne nie są jedynymi argumentami integracji. Należy wziąć również pod uwagę protokoły komunikacji sieciowej, co dodatkowo komplikuje problem. Proponowana architektura powinna być na tyle elastyczna, aby móc dostosować funkcje rozproszonego DBMS, usługi rozproszonego systemu operacyjnego oraz dodatkowo standardy protokołów komunikacyjnych (jak np. ISO/OSI). Wykazano, że nie powinno się wbudowywać zbyt wiele funkcji DBMS w jądro systemu operacyjnego. Uważa się, że w systemie operacyjnym powinny być zaimplementowane tylko podstawowe usługi, w oparciu o które mogłyby być zaimplementowane funkcje DBMS. Modelem, który wydaje się najlepiej pasować do tak sprecyzowanych wymagań, jest architektura typu klient-serwer, z nie rozbudowanym jądrem, posiadającym tylko niektóre funkcjonalności, nie stojące w sprzeczności z usługami DBMS. W ten sposób pozostałe usługi mogłyby być bez przeszkód zaimplementowane na poziomie użytkownika. Szczególnie podkreślana jest użyteczność podejścia obiektowo zorientowanego, jako sposób strukturalizacji systemu pozwalającego na przeprowadzenie integracji w łatwiejszy sposób.

System nazewnictwa jest wzorowany na terminologii dostarczanej przez systemy operacyjne. Widzi się również potrzebę poszukiwania rozwiązań zagadnień i nazewnictwa w kontekście związku pomiędzy rozproszonymi katalogami i serwerami baz danych a problematyką i terminologią stosowaną w systemach operacyjnych.

Składowanie i zarządzanie trwałymi obiektami jest podstawową funkcją DBMS. Systemy operacyjne w sposób tradycyjny wykorzystują system plików. Istnieją tendencje aby użytkowanie DBMS nie różniło się od sposobu użytkowania systemu plików w systemach operacyjnych.

W dzisiejszych DBMS zarządzanie transakcjami jest ich integralną częścią. Dyskusja nad tym, czy powinno się i czy istnieje możliwość implementacji mechanizmów transakcji jako części systemu operacyjnego, toczy się już od dawna. Jednak ostatecznie nie odpowiedziano na to pytanie.

3.5. SYSTEMY MULTIBAZ DANYCH

Systemy multibazodanowe, pod względem organizacji, są alternatywą dla logicznie zintegrowanych rozproszonych baz danych. Fundamentalna różnica pomiędzy tymi dwoma podejściami organizacji rozproszonych baz danych wynika ze stopnia autonomiczności jaką posiadają składowe komponenty zarządzania w każdym z węzłów. W przypadku zintegrowanych DBMS poszczególne komponenty zostały zaprojektowane

tak, aby umożliwiać wzajemną współpracę. W skład systemów zarządzania multibazą danych wchodzi elementy, które mogą nie posiadać żadnych wbudowanych mechanizmów umożliwiających współdziałanie. W szczególności, mogą to być niezależne DBMS, co oznacza, że na przykład, poszczególne DBMS przetwarzają transakcje w sposób nie biorący pod uwagę przetwarzania transakcji rozproszonych obejmujących wszystkie węzły. W punkcie tym zasygnalizowane zostaną problemy związane z przetwarzaniem zapytań i zarządzaniem transakcjami. Bardziej szczegółowe rozważania można znaleźć w pozycji bibliograficznej [ElmaPu1991].

Autonomiczność oraz potencjalna heterogeniczność komponentów systemu jest źródłem problemów związanych z przetwarzaniem zapytań, a szczególnie z ich optymalizacją. Główny problem związany z optymalizacją globalnych zapytań, wynika z braku możliwości ustalenia wartości lokalnych funkcji kosztów, gdyż nie ma mechanizmów pozwalających na przekazanie danych o lokalnej optymalizacji do MDBMS. Sugeruje się, że optymalizacja semantyczna, oparta tylko o informacje ilościowe może być najlepszą strategią. Możliwa jest budowa hierarchicznego optymalizatora przetwarzającego optymalizację w pewnej części zapytanie globalnie, a następnie przekazującego podzapytania do dalszej obróbki systemom lokalnym. Rozwiązanie takie na pewno nie da optymalnych wyników, ale jednak w pewnej części pozwala na zmniejszenie kosztów przetwarzania zapytań globalnych.

Przetwarzanie transakcji w systemach multibazodanowych jest szczególnie skomplikowane. Poszczególne węzły są autonomiczne. Posiadają zatem własne mechanizmy przetwarzania transakcji (jak moduły zarządzania transakcjami, programy szeregujące zadania, procedury odtwarzania baz danych w przypadku przerwanych transakcji), które przystosowane są do potrzeb lokalnych systemów. Warstwa MDBMS posiada własne mechanizmy odpowiedzialne za przyjmowanie globalnych transakcji, które uzyskują dostęp do różnych źródeł danych, oraz koordynację ich wykonania. Globalna transakcja zostaje podzielona na podtransakcje, których wykonanie jest następnie zlecane poszczególnym lokalnym DBMS. Problemem jest jednak fakt, że MDBMS nie posiada informacji o lokalnych transakcjach, przez co nie może kontrolować lokalnych konfliktów, jak i konfliktów wynikłych bezpośrednio w czasie wykonywania części globalnej transakcji, spowodowanych przez lokalne transakcje o wyższych priorytetach.

Zaproponowano wiele rozwiązań problemu realizacji transakcji w multibazach danych. Jednak można o nich powiedzieć, że są to jedynie wstępne próby wykonywane w celu lepszego zrozumienia i sformalizowania problemu. Wiele z prób prezentuje zupełnie nowe podejścia konstrukcji modelu transakcji. Jednym ze szczególnych rozwiązań jest model zagnieżdżonych transakcji, oparty na wiedzy o typowych sekwencjach transakcji występujących w MDBMS. Wymaga on jednak dalszych prac projektowych i eksperymentalnych.

Podobnie mechanizmy zapewniające niezawodność i procedury odtwarzania używane w zintegrowanych DDBMS dla potrzeb MDBMS wymagają znaczącego

skorygowania. Zastosowanie standardowego protokołu 2PC, ze względu na autonomiczność lokalnych DBMS, nie jest możliwe, gdyż niemożliwe jest zapewnienie atomowości transakcji.

Prawdopodobnie największą barierą dla rozwoju systemów multibazodanowych są niejasności, co do samej definicji autonomiczności poszczególnych lokalnych systemów, choć jest ona głównym czynnikiem złożoności tych systemów. Cecha autonomiczności powinna być jasno i precyzyjnie scharakteryzowana. Większość taksonomii systemów zarządzania rozproszonymi bazami danych rozróżnia tylko trzy poziomy autonomiczności. Wiadomo jednak, że między systemami autonomicznymi i nie autonomicznymi można wyróżnić wiele systemów o wartości pośredniej tej cechy. Konieczne jest precyzyjne zdefiniowanie systemu nazywanego autonomicznym i nie autonomicznym, wyróżnienie i określenie systemów o różnym poziomie autonomiczności oraz przyporządkowanie zestawu funkcji jakie system bazodanowy powinien spełniać, aby zachować odpowiedni poziom autonomiczności. Pozwoliłoby to na prowadzenie bardziej konkretnych rozważań nad różnymi modelami transakcji czy semantykami wykonania dla systemów będących na poszczególnych poziomach gradacji autonomiczności. Gradacja ta mogłaby umożliwić zidentyfikowanie struktury warstwowej, analogicznej do standardowego modelu ISO/OSI, definiującej poziomy wzajemnej współpracy systemów autonomicznych i heterogenicznych baz danych. Takie podejście mogłoby w konsekwencji pozwolić na zdefiniowanie standardów interfejsów pomiędzy poszczególnymi warstwami. Za początek tych prac można uznać opracowanie standardów dotyczących metod zdalnego dostępu do danych RDA (ang. *Remote Data Access*). Standardy te stanowią fundament dla zdefiniowania praktycznych rozwiązań dla problemów wzajemnego współdziałania systemów zarządzania bazami danych.

4. CHARAKTERYSTYKA WYBRANYCH DBMS UWZGLĘDNIAJĄCA MOŻLIWOŚĆ ROZPROSZENIA DANYCH

Prace nad systemami zarządzania rozproszonymi bazami danych rozpoczęły się na początku lat 80-tych. Równolegle prowadzone były prace nad prototypami systemów rozproszonych. Do trzech najpopularniejszych i najczęściej cytowanych w literaturze należą: SDD-1 (praca działu odkryć Computer Cooperation of America), R Star (produkt IBM Research rozproszona wersja prototypu systemu R), Distributed INGRES rozproszona wersja prototypu INGRES University of California at Berkeley.

Wymienione systemy są przede wszystkim wynikami uniwersyteckich prac badawczych dobrze udokumentowanych, które miały znaczący wpływ na

zainteresowanie technologią rozproszonych baz danych. Większość z dzisiejszych DBMS wspomaga w pewnym, najczęściej ograniczonym zakresie, zarządzanie rozproszonymi zasobami danych. Do najpopularniejszych serwerów baz danych należą:

- CA-OpenIngres 1.2 – produkt Computer Associates International Inc. (Islandia)
- IBM DB2 2.1.1 – produkt IBM Corp (Armonk)
- Oracle 7 ver.7.3 – produkt Oracle Corp. (Redwood Shores)
- SQL Server 6.6 – produkt Microsoft Corp. (Redmond)
- Sysbase SQL Server 11 – produkt Sysbase Inc. (Emeryville)

CA-OpenIngres

Wersje 1.1 i 1.2 CA-OpenIngres zawierają graficzne narzędzia zarządzania obiektowymi bazami danych: DBA-Visual, CA-Unicenter, Configuration-by Form, CA-OpenIngres Control Program, CA-OpenIngres/ Replicator, CA-OpenIngres/ Star i Interactive Performance Monitor. Dają one możliwość monitorowania i strojenia bazy danych oraz zarządzania procesami serwera bazy danych.

CA-Unicenter jest zintegrowanym narzędziem zawierającym między innymi Intelligent Database Agents, który pozwala wykorzystać zasoby z innych DBMS (DB2, Oracle, Informix, Sysbase, Oracle) oraz Change Management zarządzający i monitorujący dostępne zasoby, pozwalający zarządzać użytkownikami w środowisku rozproszonym. CA-OpenIngres/Replicator pozwala tworzyć przeźroczyste dla aplikacji użytkownika repliki danych przechowywanych w lokalnych węzłach lub w innych miejscach sieci. Współpracuje również z innymi DBMS, co pozwala łączyć różne zestawienia serwerów baz danych za pomocą tego narzędzia. Możliwe są trzy typy replikacji: równorzędne (ang. *peer-to-peer*), główny/ podrzędny (ang. *master/ slave*), centralny/ oddział (ang. *central/ branch*). Do zarządzania rozproszoną bazą danych służy CA-OpenIngres/Star, który uruchamia optymalizację zapytań rozproszonych i używa protokołu 2PC do aktualizacji tabel.

Narzędzie Configuration-by-Form pozwala, między innymi, na ustalanie parametrów rozproszonych baz danych.

IBM DB2

Do obsługi replikacji IBM oferuje Distributed Database Connection Services. Nad integralnością replik czuwa protokół 2PC. Dane w replikach są synchronicznie uzgadniane przez DataPropagator Relational, DataPropagator NonRelational i DataRefresher, które są częścią Data Replication. DataJoiner umożliwia połączenia z DBMS: DB2, Informix, Sysbase, Oracle, Microsoft SQL Server.

Microsoft SQL Server

Replikacja w Microsoft SQL Server opiera się na protokołach publikacji i subskrypcji części lub całości tabel z fragmentacją poziomą lub pionową. Rozproszone transakcje są obsługiwane przez Microsoft Distributed Coordinator, który opiera się na protokole 2PC.

Oracle 7.0

Oracle Transparent Gateways i Oracle Procedural Gateways integruje w pełni DBMS Oracle z DB2, Informix, CA-Ingres i Sysbase. Oracle7 Advanced Replication Option pomaga zarządzać replikacjami, które opierają się na publikacji i subskrypcji danych. Umożliwia on śledzenie zmian w bazie danych w kilku miejscach.

Sysbase SQL Server

Sysbase SQL Server posiada DirectConnect, który jest połączeniem serwera do Adabas, DB2, CA-OpenIngres, Informix, Oracle i VSAM. Jest on używany przez Sysbase Replication Server do asynchronicznej homo- i heterogenicznej replikacji danych. Możliwe jest monitorowanie informacji o replikach.

Wszystkie wymienione DBMS oferują podstawowe narzędzia zarządzania serwerem baz danych. Wspomagają głównie zadania technicznego zarządzania bazami danych: tworzenia, modyfikacji i usuwania obiektów DBMS, tworzenia kopii zapasowej i odtwarzania. Administrator ma do dyspozycji narzędzia zarządzania typu „explorer”, które ułatwiają manipulowanie bazami danych oraz obiektami zawartymi w tych bazach.

Znacznie rozbudowane są narzędzia i mechanizmy replikacji. W niektórych przypadkach, jak w Sysbase, występują osobne serwery replikacji, które zarządzają synchronizacją i monitorowaniem.

Większość DBMS posiada rozbudowane moduły zbierania statystyk serwera baz danych. Jednak zbierane statystyki nie są wykorzystywane. Żaden DBMS nie oferuje narzędzia wspomagającego analizowanie alokacji danych, oraz projektowanie alokacji na podstawie zebranych statystyk. Żaden z wymienionych DBMS nie proponuje fizycznej fragmentacji oraz alokacji, na podstawie zgromadzonych statystyk. Narzędzie takie mogłoby być bardzo pomocne dla optymalizacji działania DBMS. Jak to zostanie pokazane w następnym rozdziale, projektowanie fragmentacji i alokacji jest zadaniem wysoce skomplikowanym i nie zawsze daje efekt optymalny. Wnikliwa analiza i monitorowanie już funkcjonującego systemu bazodanowego mogłyby pozwolić na zoptymalizowanie pierwotnego projektu. DBMS mógłby sam analizować (przez zbieranie statystyk) możliwe strategie fragmentacji i alokacji, opierając się na metodologiach, które zostały przedstawione w pracy [MazurPi2001].

Warto zauważyć, że wszystkie systemy DBMS oferujące pewne formy rozpraszania danych, dotyczą relacyjnych modeli danych. W pewnym stopniu wynika to z faktu, że obiektowe bazy danych pojawiły się później, ale przede wszystkim relacyjny model danych jest najprostszy do rozproszenia (łatwość wykonania fragmentacji relacji oraz procesu odwrotnego, rekonstrukcji).

5. KIERUNKI DALESZEGO ROZWOJU TECHNOLOGII ROZPROSZONYCH BAZ DANYCH

Rozproszenie jest zwykle identyfikowane jako jedna z najwazniejszych cech systemow bazodanowych nastepnej generacji. Pozwola one na wykorzystanie technologii bazodanowej w nowych dziedzinach zastosowan, o znacząco różniących się wymaganiami od tradycyjnych aplikacji przetwarzających dane przedsiębiorstw. Jednym z często podkreślanych aspektów rozwoju technologii rozproszonych baz danych jest aspekt zamiany modelu danych na model danych o większych możliwościach niż tradycyjny model relacyjny. Szczególnie dotyczy to takich dziedzin zastosowań jak aplikacje typu CAD/CAM, wspomaganie projektowania oprogramowania, systemy informacji, czy systemy ekspertowe, gdzie model relacyjny jest zbyt ubogi, zarówno pod względem semantyki struktur danych, jak i możliwości ich przetwarzania.

W ostatnich latach dużą popularnością cieszą się obiektowe modele baz danych. Istnieje wiele publikacji dowodzących wyższość modelu obiektowego nad relacyjnym. Dotyczy to zarówno lepszych możliwości modelowania, jak i manipulowania obiektowymi strukturami danych. Na rynku pojawiło się wiele obiektowo zorientowanych DBMS (ang. *object-oriented database management system* – *OODBMS*). Nie można jednak powiedzieć, że prace nad rozproszonymi obiektowymi DBMS są zaawansowane. Dotyczy to zarówno teorii, jak i praktyki. Wynika to z wysokiego stopnia skomplikowania problemu. Obecnie badania skupiają się nad zagadnieniami zarządzania katalogami danych i obiektami rozproszonymi (jak np.: CORBA), które bezpośrednio wiążą się z rozproszonymi, obiektowymi DBMS. Stanowią one obiecującą podstawę dla dalszego rozwoju rozproszonych baz danych.

Rosnąca w ostatnich latach popularność maszyn o wieloprocesorowej architekturze przyczyniła się do intensyfikacji badań nad równoległymi systemami bazodanowymi [Özsul1995]. Technologia ta pozwala osiągnąć podwyższenie wydajności przetwarzania oraz odporności na uszkodzenia. W kontekście rozproszonych baz danych, obecne rozważania dotyczą głównie zagadnień przetwarzania zapytań. Oczywiście jest, że obie technologie mają wiele wspólnego. Integracja z systemem operacyjnym, alokacja danych, równoległe algorytmy czy równoległa kompilacja są zagadnieniami rozważanymi odnośnie obu wieloprocesorowych architektur. Zasadnicza różnica polega na konieczności uwzględnienia, w przypadku rozproszonych baz danych, kosztów komunikacji sieciowej. Jednak nie trudno jest sobie wyobrazić integrację obu technologii, gdzie danymi poszczególnych węzłów rozproszonej bazy danych będą zarządzać równoległe systemy wykorzystujące lokalną architekturę wieloprocesorową.

LITERATURA

- [BellGr1994] Bell D., Grimson J., *Distributed Database Systems*, Addison-Wesley Publishing Company, 1994.
- [Blink1996] Blinkowski G., *Systemy klient-server*, Helion, 1996.
- [CerPell1984] Ceri S., Pelagatti G., *Distributed Database – Principles and Systems*, McGraw-Hill Inc, 1984.
- [Cormic1972] McCormic W.T., Schweitzer P., White T.W., *Problem Decomposition and Data Reorganization by Clustering Technique*, 1972.
- [DaHw1984] Dayal U, Hwang H., *View Integration and Generalisation for Database Integration in a Multidatabase*, IEEE Trans.on Software Engineering, 10(6), 1984..
- [Date2000] Date C. J., *Wprowadzenie do systemów baz danych*, WNT, Warszawa, 2000.
- [Diek1997] Dickhoff G., Nichols S., Setters T., *Distributed Database Management Systems and Their Use*, 1996.
- [ElmaPu1991] Elmagarmid A.K., Pu C., *Special Issue on Heterogeneous Databases*, ACM Comp. Surv., 1991.
- [Jasz1997] Jaskiewicz A., *Inżynieria oprogramowania*, Helion, 1997.
- [Kim1995] Kim W., *Modern Database Systems*, ACM Press, 1995.
- [MazurPi2001] Mazur Z., Piechota K., *Projektowanie rozproszonych baz danych*, Zeszyty Naukowe Wydziałowego Zakładu Informatyki, Oficyna Wydawnicza Politechniki Wrocławskiej, 2001.
- [Micha1997] Michałkiewicz P., *Replikacja danych*, Software 8, 1997.
- [Nieg1997] Niezgódka M., *Strategiczne kierunki rozwoju serwisów informatycznych*, INFOBAZY'97 - Konferencja pod patronatem Komitetu Badań Naukowych., 1997.
- [Özsu1994] Özsu M. T., *Changing Infrastructure – New Challenges in Distributed Database Management*, Laboratory of Database Systems Research, Department of Computing Sciences University of Alberta, 1994.
- [ÖzsuVa1991] Özsu M. T. i Valduriez P., *Principles of Distributed Database Systems*, Prentice-Hall, 1991.
- [ÖzsuVa1994] Özsu M. T. i Valduriez P., *Distributed Database Systems: Where are we now?*-IEEE Computer, 1994.
- [ÖzsuVa1994a] Özsu M. T., Valduriez P., *Distributed Database Management*, wykłady w postaci slajdów udostępnione przez Özsu M. T., 1994.
- [ÖzsuVa1996] Özsu M. T., Valduriez P., *Distributed and Parallel Database Systems - Technology and Current State-of-the-Art*, ACM Computing Surveys, 1996.
- [ÖzsuVa1997] Özsu M. T., Valduriez P., *Distributed and Parallel Database Systems*, *Handbook of Computer Science and Engineering*, CRC Press, 1997.
- [Papa1994] Papadopoulos C.V., *On the Heterogeneity of Distributed Database – Integrating Commit Protocols*, The 14th International Conference on Distributed Computing Systems, 1994.
- [red.Özsu1994] Özsu M. T., Dayal U., Valduriez P., *An Introduction to Distributed Object Management*, Distributed Object Management, 1994.
- [Rusink1994] Rusinkiewicz Marek, *Multidatabase systems*, Houston, Texas, June 1994.
- [SL1990] Sheth A., Larson J., *Federated Database Systems*, ACM Computing, Surveys, Vol. 22, No 3, Sept. 1990.

PROPERTIES DISTRIBUTED DATABASES MANAGEMENT SYSTEMS

A designer of distributed database should have basic knowledge how do distributed database management systems (DDBMS) work, because the way the distributed databases (DDB) is design has a big influence on optimal serving transactions and queries. The paper presents features of DDBMS and mechanisms that make them possible to be realized.

*heterogeniczne bazy danych,
integracja, schemat bazy danych,
systemy baz danych*

Arkadiusz MAJKOWSKI*
Zygmunt MAZUR*

PROBLEMY INTEGRACJI HETEROGENICZNYCH BAZ DANYCH

W celu uzyskania dostępu do danych zgromadzonych w wielu bazach, tworzy się systemy umożliwiające spójny dostęp do niezależnie istniejących, rozproszonych systemów baz danych. Problemy związane z budową takiego systemu wynikają z dwu źródeł autonomii systemów składowych i związanej z tą autonomią różnorodności. W artykule omówione zostały czynniki różnicujące systemy baz danych, składające się na system heterogenicznych baz danych.

1. WSTĘP

Wraz z rozwojem techniki komputerowej nastąpił w ostatnich latach gwałtowny przyrost ilości dostępnych informacji. Wzrost ilości informacji spowodował, że wzrosła wartość jakości tej informacji. Bardziej poszukiwane stały się informacje przetworzone, pochodzące z interpretacji dużej ilości danych. Krokiem w kierunku uzyskania możliwości przetwarzania dużej ilości danych jest dostęp do danych zgromadzonych w wielu bazach danych. Problem integracji rozproszonych i heterogenicznych baz danych nabrał znaczenia praktycznego wraz z pojawieniem się sieci komputerowych [Maz85]. Główną przeszkodą w uzyskiwaniu danych z wielu baz jest ich różnorodność. Artykuł ten omawia problemy integracji heterogenicznych baz danych, związane z różnorodnością systemów składowych.

* Wydziałowy Zakład Informatyki, Wydział Informatyki i Zarządzania Politechniki Wrocławskiej,
50-370 Wrocław, Plac Grunwaldzki 9, a.majkowski@ci.pwr.wroc.pl, z.mazur@ci.pwr.wroc.pl

2. PRZYCZYNY ZRÓŻNICOWANIA

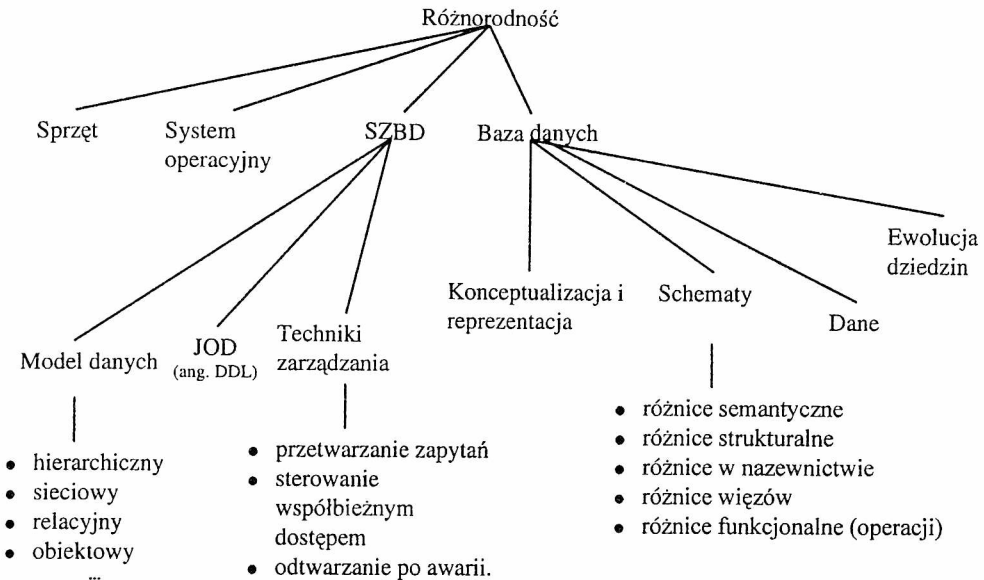
Cechą charakterystyczną systemów składowych tworzących system heterogenicznych baz danych jest ich autonomia. Systemy składowe istnieją niezależnie od siebie i funkcjonują w pełni autonomicznie. Bazy danych są obsługiwane przez działające niezależnie SZBD (*Systemy Zarządzania Bazami Danych*) i niezależnie administrowane.

Podstawowa przyczyna istnienia różnorodności baz danych wynika również z autonomii. Jest nią autonomia projektu, tworzenia i rozwoju składowych systemów baz danych. Na etapie projektu podejmowane są autonomiczne decyzje dotyczące tego co będzie modelowała baza danych, jaki będzie model konceptualny. Niezależnie dla każdego z systemów składowych ustalana jest reprezentacja danych, to znaczy wybierany jest model danych, język definicji i stosowane w schemacie nazewnictwo. Autonomicznie tworzony projekt bazy danych określa również jakie będą dostępne operacje na danych i jakie więzy integralności będą utrzymywane w bazie danych.

Fakt, że systemy te są niezależnie tworzone jest przyczyną różnic systemowych. Systemy mogą być implementowane na różnych platformach sprzętowych, z wykorzystaniem różnych systemów operacyjnych, mogą mieć możliwość komunikacji sieciowej przy użyciu różnych protokołów. Do obsługi bazy danych mogą zostać wybrane różne systemy zarządzania bazą danych. Systemy te mogą się różnić nie tylko udostępnianym modelem danych i językiem opisu danych (*JOD*, ang. *DDL*), ale również obsługiwanymi językami przetwarzania danych (*JMD*, ang. *DML*) i technikami zarządzania bazą danych. Wśród technik zarządzania bazą danych dla budowy systemu łączącego te bazy istotne są sposoby przetwarzania zapytań, przyjęte metody sterowania współbieżnym dostępem oraz algorytmy wycofywania transakcji i odtwarzania po awarii.

Wiele systemów baz danych pomimo, że są budowane według dobrze przyjętych wymagań i założeń, jest stale zmienianych i rozwijanych. Rozwój ten jest wynikiem zmian w potrzebach użytkowników systemu. Dwa homogeniczne systemy jeśli będą niezależnie rozwijane, to po pewnym czasie rozwój ten spowoduje powstanie rozbieżności pomiędzy nimi. W tym sensie autonomia ewolucji jest też źródłem różnic pomiędzy systemami.

Schematyczną klasyfikację różnic pomiędzy systemami ilustruje rysunek 1.



Rys. 1. Klasyfikacja czynników różnicujących systemy składowe baz danych

Fig. 1. Classification of differences between systems

3. RÓŻNICE POMIĘDZY BAZAMI DANYCH

Składowe systemu heterogenicznych baz danych budowane są przez różnych ludzi. Ludzie ci postrzegają ten sam wycinek rzeczywistości w różny sposób, w zależności od wymagań dotyczących budowanego systemu, własnego doświadczenia i możliwości technicznych. Skutkiem tego są różnice w procesie konceptualizacji oraz różne procesy abstrakcji przy tworzeniu schematu.

3.1. KONFLIKTY POMIĘDZY SCHEMATAMI BAZ DANYCH

Założmy, że dwóch projektantów bazy danych określa schematy dla nakładających się fragmentów rzeczywistości. Różnice w przyjętych rozwiązaniach na poszczególnych etapach projektowania schematu powodują powstawanie różnorodnych konfliktów. Dwóch projektantów może w różny sposób postrzegać te same byty świata rzeczywistego lub w różny sposób je klasyfikować. Na przykład, jeden z projektantów może zgrupować wszystkich studentów w klasę obiektów *Student*, natomiast drugi, będąc zainteresowanym jedynie studentami informatyki, utworzy klasę *Student Informatyki*. Te dwa zbiory obiektów będą powiązane relacją

zawierania się. Sytuację, gdy elementy dwu schematów reprezentujące zbiory bytów świata rzeczywistego są ze sobą powiązane ale nie są sobie równoważne, nazywa się **konfliktem semantycznym** [SP91]. Tego typu konflikty zwykle rozwiązuje się stosując mechanizm generalizacji. W powyższym przykładzie klasa *Student Informatyki* byłaby podklasą klasy *Student*.

Dwóch projektantów może uwzględnić w schemacie bazy różne atrybuty dla powiązanych bytów świata rzeczywistego. Na przykład założmy, że dwa schematy relacyjne S1 i S2 przechowują dane o samochodach:

S1: <nazwa, producent, cena >

S2: <nazwa modelu, zużycie paliwa, moc silnika>

Twórcy schematów S1 i S2 wybrali różne atrybuty dla przechowywanych obiektów ze względu na różne potrzeby w tym zakresie. Schemat S1 zawiera dane handlowe, a S2 techniczne. Taką sytuację, kiedy dwa elementy schematów reprezentujące powiązane byty świata rzeczywistego są opisane w bazie danych za pomocą różnych zbiorów atrybutów nazywamy **konfliktem opisu**. Do konfliktów opisu zaliczane bywają również konflikty:

- nazw,
- dziedzin,
- rodzaju powiązań,
- warunków integralności i operacji.

W ramach tego samego modelu danych dwóch projektantów może wybrać różne konstrukcje do reprezentacji tego samego fragmentu rzeczywistości, na przykład w obiektowym modelu danych obiekt książka może mieć atrybut znakowy *Autor* zawierający imię i nazwisko autora lub może zawierać referencję (dowiązanie) do obiektu *Autor*. Sytuację, w której dwa powiązane fragmenty rzeczywistości są modelowane za pomocą różnych struktur danych, nazywamy **konfliktem strukturalnym**.

Konflikty w schematach danych są podstawowym problemem podczas tworzenia schematu zintegrowanego dla opisującego dane z baz składowych. Gdyby schematy składowe nie były ze sobą powiązane, to schemat federacyjny byłby sumą schematów składowych. Źródłem konfliktów są różnice w reprezentacji powiązanych elementów z poszczególnych schematów.

Wybrane konflikty występujące w schematach, niezależnie od przyjętego modelu danych, przedstawia Tabela 1.

Rodzaj konfliktu	Opis	Przykład
Konflikt identyfikacji	to samo pojęcie reprezentowane przez różne obiekty	obiekt reprezentujący tą samą osobę może mieć różne identyfikatory w różnych bazach
Konflikt nazw – homonimy	różne pojęcia tak samo nazywane	staż raz może oznaczać odbycie stażu naukowego, a drugi raz staż pracy
Konflikt nazw – synonimy	te same pojęcia różnie nazywane	miejsce_pracy / jednostka
Konflikty strukturalne	te same pojęcie są reprezentowane przez różne konstrukcje modelu, lub te same konstrukcje ale o innej strukturze	miejsce_pracy raz jest atrybutem znakowym, a raz klasą
Konflikty semantyczne	te same pojęcia są różnie interpretowane	adres pracownika raz oznacza adres pocztowy miejsca pracy a drugi raz budynek i pokój pracownika
Konflikty danych	dane dotyczące tego samego pojęcia mają różne wartości	ten sam pracownik ma różne miejsca pracy w różnych bazach

Tabela 1. Wybrane konflikty w schematach baz danych

Table 1. Selected conflicts in schema of database

3.2. RÓŻNICE SPOWODOWANE EWOLUCJĄ DZIEDZIN

Różnice pomiędzy składowymi bazami danych, tworzącymi system heterogeniczny spowodowane są także zmianami w znaczeniu przypisywanemu wartościom przechowywanym w bazie danych. W istniejących obecnie systemach z bazami danych znaczenie przypisywane wartościom danych w większości nie jest wprost zapisane w bazie, lecz jest zawarte w oprogramowaniu aplikacyjnym i przyjętych na temat danych założeniach. Zmiana znaczenia danych w jednej z baz systemu heterogenicznego nie jest w takim przypadku widoczna dla zewnętrznych aplikacji co jest powodem potencjalnych nieprawidłowości w ich działaniu. W pracy [VH91] zmiany znaczenia przypisywanego wartościom danych określono jako **ewolucje dziedzin wartości**. Wymieniono tam następujące przykłady ewolucji dziedzin:

1. Zmiana znaczenia tej samej wartości z danej dziedziny.

Przykładem takiej zmiany jest rozbitcie działu o symbolu "A3" w pewnym przedsiębiorstwie na dwa działy "A3" i "A4". Po tej zmianie "A3" oznacza dział obejmujący zaledwie część tych pracowników i zasobów co poprzednio.

2. Zmiana rodzaju powiązań.

Przykładowo, na skutek zmian organizacyjnych powiązanie 1:n między działami i projektami zostanie zamienione na powiązanie m:n.

3. Zmiany ziarnistości.

Na przykład atrybut o nazwie ulica mógł pierwotnie zawierać tylko nazwę ulicy, a w wyniku zmian, również numer domu. W wyniku tej zmiany zapytania dotyczące adresów (np. średnia liczba mieszkańców na ulicy) będą źle działały.

4. Zmiany zakodowanych wartości. Wartości w bazie danych często zawierają zakodowane wartości, służące do przechowywania informacji nie uwzględnionej w systemie.

Przykładowo, aby zaznaczyć, że pracownik jest na urlopie, przypisuje się go do projektu o numerze 0. Zapytanie “nie poinformowane” o tym kodowaniu zapewne stwierdzi, że największym projektem w firmie był projekt nr 0 – przerwa urlopową.

5. Zmiany identyfikatorów.

Przykładem takiej zmiany jest np. przejście z indeksów towarowych na kody kreskowe. Umożliwienie czasowego funkcjonowania w systemie dwóch rodzajów identyfikatorów wymaga uwzględnienia tego w zapytaniach i programach.

6. Zmiany jednostek miary.

W wyniku zmiany jednostek miary w jednej z baz danych, odwołujące się do tych danych programy lub zapytania, będą dawały niepoprawne wyniki.

7. Ponowne wykorzystanie pól. W systemach, w których zmiana charakterystyk bazy danych jest niewskazana, praktykuje się ponowne wykorzystanie pól, co może prowadzić do błędnych wyników w programach tego nie uwzględniających.

3.3. RÓŻNICE REPREZENTACJI

Te same dane mogą być reprezentowane w różny sposób. Przykładem mogą być tu bazy danych reprezentujące informacje przestrzenne, np. geograficzne, w których stosuje się dwa alternatywne sposoby reprezentacji [WD91]. Jeden z nich oparty jest o model warstwowy, a drugi o obiektowy. W modelu warstwowym dane przechowywane są w postaci warstw. Warstwa traktowana jest jako funkcja przypisująca danym przestrzennym (np. punktom siatki) atrybuty (np. wysokość n.p.m. lub rodzaj upraw). W obiektowym modelu reprezentacji danych przestrzennych obiekty-dane dziedziczą atrybuty z podstawowych obiektów przestrzennych takich jak punkt, prosta, wielokąt. Jezioro będzie tu reprezentowane w postaci wielokąta wyznaczającego linię brzegową i dodatkowych atrybutów: nazwy, głębokości, pochodzenia, itp.

3.4. RÓŻNICE POMIĘDZY SCHEMATAMI I DANYMI W MODELU RELACYJNYM

W pracy [KS91] zaprezentowano klasyfikacje konfliktów pomiędzy schematami i konfliktów danych dla relacyjnych baz danych. Klasyfikacja ta dotyczy konfliktów,

które mogą wystąpić przy integracji heterogenicznych baz danych, gdy wszystkie składowe bazy danych są zbudowane w oparciu o relacyjny model danych. W ślad za autorami relację, nazwano tutaj tabelą relacyjną, lub krótko tabelą, a krotkę – rekordem.

3.4.1. KONFLIKTY MIĘDZY SCHEMATAMI

Do klasyfikacji konfliktów pomiędzy schematami relacyjnymi wykorzystano definicje danych w ANSI SQL, aby wymienić elementy schematów, które będąc różnie zdefiniowane mogą powodować powstawanie konfliktów.

I. Konflikty pomiędzy tabelami.

Konflikty te mają miejsce wtedy, gdy składowe bazy danych w różny sposób reprezentują dane w tabelach relacyjnych. Konflikty między tabelami można podzielić na konflikty pomiędzy pojedynczymi tabelami i pomiędzy wieloma tabelami. Konflikty pomiędzy jedną a wieloma tabelami traktowane są jako specjalny przypadek konfliktu pomiędzy wieloma tabelami.

A. Konflikty pomiędzy pojedynczymi tabelami. Mają one miejsce wtedy, gdy informacje są reprezentowane w pojedynczych tabelach z użyciem różnych nazw, struktur i więzów.

1. Konflikty nazw tabel zachodzą ponieważ w składowych bazach danych zastosowano różne nazewnictwo dla tabel. Można wyróżnić dwa typy tych konfliktów:

- **Różne nazwy dla równoważnych semantycznie tabel (synonimy).**
- **Te same nazwy dla różnych semantycznie tabel (homonimy).**

2. Konflikty struktur tabel dotyczą sytuacji gdy odpowiadające sobie tabele różnią się liczbą atrybutów, to znaczy, gdy w tabeli jednej z baz składowych brakuje atrybutu, który jest w równoważnej tabeli w drugiej bazie. Wyróżniono tutaj dwa przypadki:

- **Brak atrybutów.**
- **Brak atrybutów o domniemanych wartościach,** które to wartości mogą być ustalone na podstawie innych danych w bazie lub informacji dodatkowych.

3. Konflikty więzów dla tabel mają miejsce wtedy, gdy równoważne tabele w różnych bazach składowych zdefiniowano z różnymi więzami. Autorzy, za standardem SQL, wymieniają cztery typy więzów: definicję kluczy kandydujących, definicję kluczy głównych, definicję kluczy obcych i warunki kontrolne (*ang. check condition*). Konflikty więzów nie utrudniają definicji perspektyw (*ang. view*) i budowania zapytań odwołujących się do różnych baz, lecz powodują trudności w aktualizacji danych.

B. Konflikty pomiędzy wieloma tabelami występują wtedy, gdy składowe bazy danych reprezentują te same informacje w różnych liczbach tabel. Informacje zawarte w jednej z baz w pojedynczej tabeli, mogą być w innej bazie przechowywane w kilku tabelach w celu uniknięcia redundancji lub niespójności w wyniku modyfikacji. Pośród konfliktów pomiędzy wieloma tabelami można wyróżnić także klasy konfliktów takie, jak pomiędzy pojedynczymi tabelami, tj.: konflikty nazw, struktur i więzów. Konflikty te nie są osobno klasyfikowane, ponieważ można je zdekomponować na konflikty składowe i rozpatrywać oddzielnie.

II. Konflikty pomiędzy atrybutami.

Konflikty pomiędzy atrybutami wynikają z różnych definicji równoważnych semantycznie atrybutów. Podobnie jak w przypadku tabel konflikty między atrybutami podzielono na konflikty pojedynczych atrybutów i konflikty pomiędzy wieloma atrybutami.

A. Konflikty pomiędzy pojedynczymi atrybutami. Definicja atrybutu obejmuje jego nazwę, typ danych, więzy i wartość domyślną. Przyjmując, że typ danych jest szczególnym przypadkiem więzu, konflikty dla atrybutów sklasyfikowano następująco:

- 1. Konflikty nazw atrybutów**, podobnie jak dla tabel można tu wyróżnić synonimy i homonimy czyli:
 - **Różne nazwy dla równoważnych atrybutów.**
 - **Te same nazwy dla różnych atrybutów.**
- 2. Konflikty wartości domyślnych** objawiają się, podobnie jak konflikty więzów, przy aktualizacji danych. Jeśli atrybut ma różne wartości domyślne w różnych bazach, to wstawienie danych z wykorzystaniem domyślnych wartości atrybutu, spowoduje powstanie niezgodności danych.
- 3. Konflikty więzów dla atrybutów.**
 - **Konflikty typów danych** mają miejsce wtedy, gdy z semantycznie równoważnymi atrybutami związane są różne typy danych.
 - **Konflikty więzów integralności dla atrybutów** podobnie jak konflikty wartości domyślnych objawiają się podczas modyfikacji danych. Wartość atrybutu dozwolona w jednej z baz może nie być dozwolona w innej, z powodu różnicy więzów integralności.

B. Konflikty pomiędzy wieloma atrybutami dotyczą sytuacji, w których te same informacje są reprezentowane z użyciem różnych ilości atrybutów w różnych bazach.

III. Konflikty pomiędzy tabelami i atrybutami występują wtedy gdy jedno z baz reprezentują daną informację poprzez tabelę relacyjną, a inne poprzez atrybuty.

3.4.2. KONFLIKTY DANYCH

Konflikty danych mają miejsce wtedy, gdy w bazach składowych występują niezgodne dane.

A. Błędne dane są skutkiem niewłaściwego utrzymania bazy danych. Można wyróżnić dwie przyczyny występowania błędnych danych w bazie:

1. **Błędnie wprowadzone dane** powodują konflikt, gdy wartości danych reprezentujące tę samą informację różnią się pomiędzy składowymi bazami danych. Założono tutaj, że tożsamość informacji określana jest na podstawie zdefiniowanych w pewien sposób wartości kluczowych.
2. **Nieaktualne dane** powodują konflikt wtedy, gdy wartości danych reprezentujące tę samą informację zostały zaktualizowane nie we wszystkich składowych bazach danych.

B. Różne reprezentacje tych samych danych mogą wynikać z zastosowania różnych wyrażień, jednostek i dokładności.

1. **Różne wyrażenia** mogą zostać użyte do reprezentacji tych samych danych. Dotyczy to m.in. zastosowania różnych słów, różnych układów tekstu i różnych kodów dla tych samych wartości.
2. **Różne jednostki** zastosowane do danych liczbowych powodują, że ta sama wartość liczbową ma inne znaczenie.
3. **Różna dokładność** pamiętanych wartości ma miejsce wtedy, gdy do ich reprezentacji wykorzystywane są dziedziny o różnej liczebności. W przypadku, gdy dziedziny są równoliczne, to taki konflikt wartości autorzy klasyfikują jako konflikt różnych wyrażień.

Wymienione wyżej klasy konfliktów określone zostały przez autorów jako podstawowe. W praktyce najczęściej występują konflikty złożone. Konflikty złożone można zdekomponować na konflikty składowe, należące do wymienionych wyżej klas.

4. PODSUMOWANIE

W artykule omówione zostały czynniki różnicujące systemy baz danych, składające się na system heterogenicznych baz danych. Przedstawiono przyczyny powodujące istnienie różnic pomiędzy systemami składowymi. Zaprezentowana została ogólna klasyfikacja czynników różnicujących systemy składowe. W bardziej szczegółowy sposób omówiono wybrane problemy związane z niejednorodnością integrowanych systemów baz danych.

LITERATURA

- [Cha94] Chakravarthy Sh., *Distributed Data Management*, 14th Intl Conf. on Distributed Computer Systems, 1994.
- [KS91] Kim W., Seo J., *Classifying Schematic and Data Heterogeneity in Multidatabase Systems*, Computer, Dec. 1991.
- [Maz85] Mazur Z., *Bazy danych w sieciach komputerowych*, Konferencja Sieci Komputerowe – usługi, protokoły, modele., Wrocław 1985.
- [RPRG94] Reddy M.P., Prasad B.E., Reddy P.G., Gupta A., *A Methodology for Integration of Heterogeneous Databases*, IEEE Trans. on Knowledge and Data Eng. Vol. 6, No. 6, Dec. 1994.
- [Sal95] Saltor F., *Database Interoperability*, Międzynarodowe Seminarium w Sopocie 22 i 23 marca 1995.
- [SP91] Spaccapietra S., Parent Ch., *Conflicts and Correspondence Assertions in Interoperable Databases*, SIGMOD RECORD, Vol. 20, No. 4, Dec. 1991.
- [WD91] Worboys M.F., Deen S.M., *Semantic Heterogeneity in Distributed Geographic Database*, SIGMOD RECORD, Vol. 20, No. 4, Dec. 1991.
- [WK91] Weishar D., Kerschberg L., *Data/Knowledge Packets as a means of Supporting Semantic Heterogeneity in Multidatabase Systems*, SIGMOD RECORD, Vol. 20, No. 4, Dec. 1991.
- [VH91] Ventrone V., Heiler S., *Semantic Heterogeneity as Result of Domain Evolution*, SIGMOD RECORD, Vol. 20 No. 4, Dec. 1991.

ON PROBLEMS WITH INTEGRATION OF HETEROGENEOUS DATABASES

Heterogeneous database systems are build to provide uniform access to data collected in many database systems. The main problems in building such systems are autonomy of component database systems and their heterogeneity. This paper discusses issues concerning heterogeneity in components database systems. The differences are divided into two main categories system differences and semantic differences. System differences include discrepancies in hardware, operation systems, network protocols and differences in database managing systems (DBMS). DBMS can differ in data models, data definition languages (DDL) and DBMS managing techniques. Semantic differences concern conceptualisation of modelled world, conflicts between schemas and conflicts in data values.

*architektura klient-serwer,
architektura wielowarstwowa,
baza danych, Internet,
sieci intranetowe*

Jacek GRUBER*
Zygmunt MAZUR*

BAZODANOWE SYSTEMY KLIENT-SERWER I ARCHITEKTURY WIELOWARSTWOWE

Przedmiotem artykułu jest zwięzła analiza ewolucji systemów i środowisk bazodanowych do systemów rozproszonych z aplikacjami wielowarstwowymi oraz porównanie możliwości technologii aplikacji wielowarstwowych z klasyczną technologią klient-serwer. Technologie projektowe i programistyczne do integracji scentralizowanych i rozproszonych baz danych z sieciami intranetowymi i Internetem należą obecnie do najnowocześniejszych i najszybciej rozwijających się technologii informatycznych. Celem pracy jest uchwycenie najistotniejszych problemów projektowych i programistycznych występujących przy tworzeniu bazodanowych systemów klient-serwer i systemów bazodanowych z aplikacjami wielowarstwowymi.

1. WPROWADZENIE

Powszechne jest dążenie do integracji wielu sieci intranetowych w instytucjach w całościowe sieci korporacyjne [7, 10, 13, 22]. Systemy bazodanowe, biznesowe i systemy zarządzania – w skład w których wchodzi bazy danych, aplikacje, systemy zarządzania bazami danych DBMS (*Database Management Systems*) oraz, coraz częściej, systemy przepływu i archiwizacji dokumentów i przepływu pracy (*workflow*) – ewoluują w kierunku jednolitych korporacyjnych, rozproszonych środowisk systemowych i informacyjnych [5, 20]. Rozwiązania i technologie stosowane do budowy ta-

* Wydziałowy Zakład Informatyki Politechniki Wrocławskiej, Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, gruber@ci-1.ci.pwr.wroc.pl; mazur@ci-1.ci.pwr.wroc.pl

kich systemów ulegały stopniowej ewolucji i obecnie do najbardziej zaawansowanych należą architektury trójwarstwowe klient-serwer [13, 15] oraz technologie aplikacji wielowarstwowych [16, 25].

Aplikacje wielowarstwowe wykorzystuje się przede wszystkim do przetwarzania danych w dużych systemach informatycznych, w których z systemu korzysta jednocześnie 100 lub więcej użytkowników. Im bardziej złożona jest struktura przedsiębiorstwa, tym większe wymagania nakłada się na system informatyczny [26]. Zastosowanie architektury wielowarstwowej w systemie informatycznym pozwala odwzorować naturalny podział instytucji lub firmy na oddziały i filie. Takie systemy mogą też „rośnąć” wraz z rozwojem przedsiębiorstwa.

2. ROZPROSZONOŚĆ SYSTEMÓW BAZODANOWYCH

Systemy bazodanowe należą do najbardziej rozpowszechnionej, pod względem liczby i różnorodności zastosowań, klasy systemów oprogramowania. Prawie każdy użytkowy system informatyczny jest systemem bazy danych albo posiada niektóre funkcje systemu bazodanowego. W praktyce większość systemów baz danych to systemy relacyjne.

System bazodanowy składa się z dwóch komponentów [20]: (1) (a) bazy danych o określonej strukturze i organizacji przechowywanych tam danych z (b) aplikacją, która na tych danych operuje, oraz (2) systemu DBMS zarządzania bazą danych odpowiadającego za realizację dostępu do danych w bazie, przetwarzanie oraz wykonywanie operacji na danych bazy oraz za kontrolę stanu spójności bazy danych.

Każdy ze wspomnianych komponentów może pracować jako scentralizowana albo rozproszona. Bazę danych można poddać fragmentacji, replikacji i alokować do różnych (logicznie lub fizycznie) miejsc w rozproszonym środowisku pracy. Moduły oprogramowania użytkowników i administratorów systemów baz danych i systemów zarządzania bazami danych można również zreplikować i alokować w różnych miejscach środowiska pracy. System DBMS może być również rozproszony.

Dodatkową komplikację wnosi aspekt heterogeniczności komponentów systemu bazodanowego oraz aspekt stopnia powiązania jednej lub kilku baz danych, które są bazami „składowymi” pewnej „całościowej” bazy danych [20].

Dla systemów bazodanowych bardzo ważne jest zatem środowisko operacyjne oraz sieciowe, w których działają. Aby systemy bazodanowe mogły w tych środowiskach działać, same muszą posiadać dość skomplikowaną architekturę ze zbiorem różnorodnych, dość złożonych funkcji.

W najprostszym przypadku – jak to miało miejsce w początkowym okresie rozwoju systemów bazodanowych – systemy te posiadały architekturę host-terminal – i działały w środowiskach wieloprogramowych, wielodostępnych systemów operacyj-

nych z podziałem czasu. Dla sprawnego działania takich systemów bazodanowych najbardziej istotne były własności funkcjonalne i struktura części aplikacyjnej systemu bazodanowego.

Okazało się, że po długiej ewolucji systemów baz danych, osadzeniu ich w rozproszonych środowiskach operacyjnych i sieciowych, zastosowaniu do ich budowy coraz nowocześniejszych technologii programistycznych i komunikacyjnych oraz paradygmatów strukturalnych i funkcjonalnych – zagadnienia strukturalizacji i sprawności działania systemów bazodanowych wymagają jeszcze nadal doskonalenia [5, 22].

Nowe rozwiązania strukturalne oraz nowe rozwiązania modelowe i technologiczne, dotyczące w szczególności części aplikacyjnych systemów bazodanowych, zapewniają większą dostępność danych i polepszają wydajność przetwarzania danych ulokowanych w bazach danych [7, 8, 9].

W dalszej części pracy naszkicujemy zwięźle etapy rozwoju systemów bazodanowych starając się uchwycić te jego aspekty, które są związane zarówno z rozwojem jak i obecnym stanem technologii programistycznych i projektowych do budowy części aplikacyjnych tych systemów.

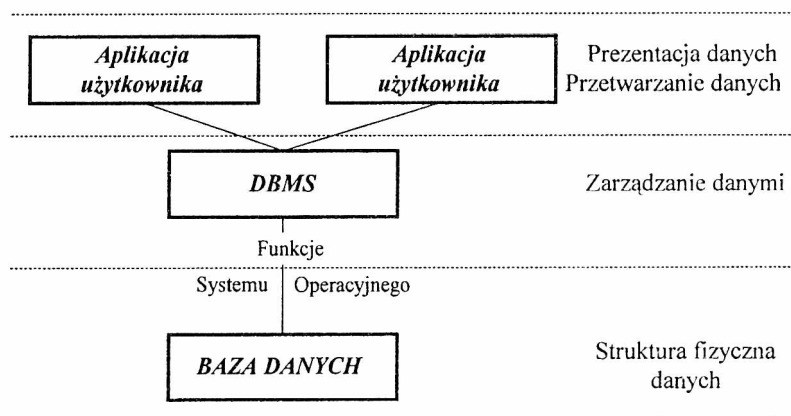
Rozwój systemów baz danych rozpoczął się w latach 60. od zastosowania architektury host-terminal. Ewolucja tych systemów oraz ich zastosowań doprowadziła do powstania trójwarstwowej architektury systemów bazodanowych, rozdzielającej wyraźnie elementy systemu bazy danych na trzy warstwy: prezentacji danych, przetwarzania danych oraz systemu zarządzania bazą danych. Współczesne zastosowania systemów bazodanowych powodują, że do ich działania w takiej architekturze niezbędne jest odpowiednie środowisko rozproszone, np.: CORBA [4, 22, 23], DCE [13], Internet.

3. STRUKTURA SYSTEMU BAZY DANYCH

W pierwszym okresie rozwoju systemów bazodanowych można było zaobserwować proces oddzielania danych od programów, tzn. dane przestawały być własnością wyłącznie jednego programu (użytkownika). Dzięki temu użytkownik nie musiał znać wszystkich szczegółów logicznej i fizycznej organizacji tych danych. Dane były traktowane jako wspólny zasób informacyjny pewnego systemu, z którego korzystać mogło wielu użytkowników.

Rozwój telekomunikacji oraz sieci komputerowych doprowadził do możliwości terytorialnego rozproszenia bazy danych na różne komputery, przy czym celowe stało się duplikowanie pewnych fragmentów bazy danych w kilku węzłach sieci komputerowej. Skutkiem tego utrzymanie bazy danych w stanie spójności z jednoczesnym zapewnieniem efektywnej obsługi wielu współbieżnych procesów, stało się zagadnie-

niem bardzo złożonym i doprowadziło – podobnie jak w systemach operacyjnych [14, 15, 24] – do warstwowej budowy systemów bazy danych. Strukturę tę naszkicowano na rysunku 1.



Rys. 1. Warstwowa struktura systemu bazodanowego

Fig. 1. The layered structure of database system

W poszczególnych warstwach logicznych mieszczą się następujące składowe [20]:

- Baza danych, czyli zbiór struktur danych zgromadzonych w pamięci zewnętrznej, np. w pamięci dyskowej.
- DBMS, czyli system zarządzania bazą danych.

Do podstawowych zadań DBMS można zaliczyć:

1. definiowanie danych,
2. gromadzenie danych w bazie danych,
3. operowanie na danych,
4. utrzymanie spójności danych,
5. umożliwienie współbieżnego dostępu do danych,
6. ochrona oraz poufność danych,
7. realizacja poleceń z aplikacji użytkownika.

W zależności od architektury systemu bazy danych, DBMS może być:

1. zbiorem podprogramów bibliotecznych, które w trakcie kompilacji są łączone z aplikacją użytkownika;
2. zbiorem podprogramów wykonujących funkcje aplikacji poprzez interpretacje;
3. niezależnym procesem, który wykonywany jest pod nadzorem systemu operacyjnego, komunikującym się z aplikacjami użytkowników jako oddzielnymi procesami.

- Aplikacje użytkowników, wykorzystując funkcje DBMS, wykonują następujące zadania:
 1. pobieranie danych bezpośrednio od użytkownika lub za pomocą urządzeń wejścia
 2. prezentowanie danych na ekranie komputera lub innego urządzenia wyjścia,
 3. przetwarzanie danych zgromadzonych w bazie oraz danych wprowadzanych do bazy.

W warstwie przetwarzania danych zaimplementowane są reguły danych nazywane regułami zarządzania lub regułami biznesowymi (*business rules*), które są charakterystyczne dla semantyki danej bazy.

4. EWOLUCJA ARCHITEKTURY SYSTEMÓW BAZ DANYCH

Proces rozwoju technologii informatycznych miał ścisły związek z aspektem rozproszonej architektury systemów użytkowych – w szczególności ze środowiskiem sprzętowym, operacyjnym i komunikacyjnym, w którym systemy te pracowały. Środowiska te ewoluowały od komputerów typu mainframe do środowisk w pełni rozproszonych [14, 15, 24].

Pierwsze systemy relacyjnych baz danych powstały w latach 70. wraz z pierwszymi dużymi komputerami typu mainframe oraz opublikowaniem w roku 1970 przez Codda pracy dającej podwaliny modelowi relacyjnego baz danych.

W latach 80., wraz z rozpowszechnieniem się komputerów klasy PC scentralizowane systemy DBMS i systemy baz danych rozpowszechniły się w postaci systemów jednoróżniowikowyc. Sieciowe systemy operacyjne pozwoliły z kolei na współużytkowania plików i drukarek w sieciach LAN. Dane przechowywane dotychczas w lokalnych pamięciach zewnętrznych zaczęto umieszczać na serwerach plików, skąd były bezpośrednio pobierane oraz przesyłane przez systemy DBMS pracujące na stacjach roboczych. Przetwarzanie realizowały w całości programy aplikacyjne alokowane na stacjach roboczych użytkowników.

Rozwój technologii sieciowych LAN i WAN doprowadził do powstania oraz rozpowszechnienia przetwarzania w trybie klient–serwer [13, 15]. Dane również były umieszczane na serwerze sieciowym, ale były udostępniane aplikacjom przez serwer bazy danych. Równocześnie, funkcje przetwarzania zaczął realizować serwer. Wiele wad prostej architektury klient–serwer oraz rozwój przetwarzania rozproszonego doprowadziły do powstania wielowarstwowej architektury klient–serwer, w której aplikacja jest podzielona na wiele części i umiejscowiona na wielu komputerach.

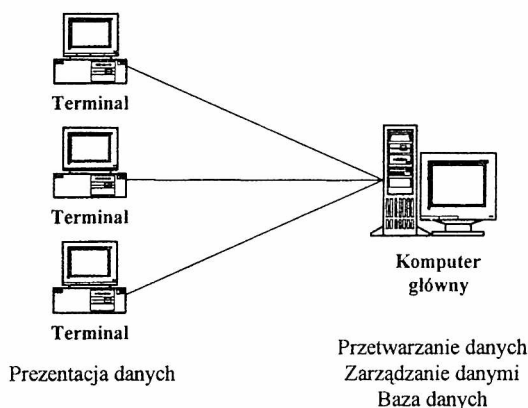
4.1. ARCHITEKTURA HOST-TERMINAL

Pierwsze systemy baz danych były budowane w oparciu o komputer główny (mainframe), na którym dane były gromadzone oraz przetwarzane. Taka architektura ma wiele zalet i z tego względu nadal jest stosowana w systemach, gdzie jest wymagana duża wydajność oraz bezpieczeństwo danych [1, 5].

W architekturze host-terminal można wyróżnić komputer główny oraz wiele terminali do niego przyłączonych. Komputer główny musi być wyposażony w następujące oprogramowanie:

- system operacyjny z wielodostępnością, wielozadaniowością i mechanizmami ochrony danych,
- system DBMS umożliwiający wielodostępny oraz współbieżny dostęp do danych,
- aplikacje komunikujące się z użytkownikami pracującymi w trybie dostępu terminalowego.

Terminale tekstowe składają się zwykle jedynie z monitora ekranowego oraz klawiatury i są przyłączone do komputera głównego za pomocą dedykowanych łączy fizycznych, na przykład łączy szeregowych RS-232. Terminale pełnią rolę prostych stacji końcowych do wyświetlania informacji i oraz wprowadzania danych. Wszystkie obliczenia odbywają się na komputerze głównym, gdzie dla każdego terminala zostają uruchomione odpowiednie procesy. Terminale oraz komputer główny mogą być również połączone w sieć komputerową. Można emulować działanie typowych terminali na różnych komputerach włączonych do sieci. Schemat strukturalno-funkcyjny architektury host-terminal przedstawiono na rysunku 2.



Rys. 2. Schemat strukturalno-funkcyjny architektury host-terminal

Fig. 2. Structure-functional schema of the host-terminal architecture

Architekturę host-terminal wyróżnia duża prostota funkcjonowania systemów bazodanowych w niej pracujących. Całość systemu bazy danych znajduje się na jednym

komputerze i jest systemem scentralizowanym. Systemem o takiej architekturze łatwo administrować i jest tani w utrzymaniu.

Ponieważ całość procesu przetwarzania danych jest realizowana na komputerze głównym, ruch przez łącza obejmuje jedynie dane wynikowe przeznaczone dla użytkownika, dane wprowadzane przez użytkownika oraz sygnały sterujące. Obciążenie łącza jest zatem bardzo małe. Dołączenie nowego użytkownika do systemu sprowadza się do przyłączenia nowego terminala oraz skonfigurowania środowiska użytkownika, dzięki czemu operacja ta jest prosta i tania. Bezpieczeństwo danych jest realizowane na kilku poziomach: (1) przy wejściu do systemu za pomocą identyfikacji i autoryzacji użytkowników, (2) poprzez ochronę plików, (3) poprzez mechanizm perspektyw na dane dla aplikacji i blokowanie dostępu do funkcji aplikacyjnych za pomocą odpowiednich środków projektowych i administracyjnych, (4) fizyczne oddzielenie użytkownika od dostępu do komputera głównego.

W przypadku spadku wydajności systemu spowodowanego przyrostem liczby nowych użytkowników można rozwijać infrastrukturę sprzętową komputera głównego. Przy dużej liczbie użytkowników można jednak osiągnąć barierę wydajności sprzętu komputera głównego. Trzeba go wtedy wymienić lub radykalnie zmienić architekturę systemu. Niestety, oba rozwiązania są stosunkowo bardzo kosztowne i z wielu powodów kłopotliwe.

Postępujący rozwój technologii informatycznych zmuszał do podejścia bardziej elastycznego przy projektowanych systemów bazodanowych [13]. Znaczenia nabierało uzyskanie możliwości integrowania systemów bazodanowych zarządzanych przez systemy DBMS różnych producentów i baz danych o różnej niskopoziomowej budowie strukturalnej oraz uzyskanie zgodności z różnym oprogramowaniem komunikacyjnym i narzędziowym, czego nie była w stanie zapewnić architektura host-terminal. Tabela 1 zawiera zestawienie najważniejszych zalet i wad tej architektury.

Tabela 1

Zestawienie głównych zalet i wad architektury host-terminal

Zalety	Wady
I. System scentralizowany: (1) prostota Architektoniczna, (2) niskie koszty administrowania, (3) scentralizowane zarządzanie. II. Łatwe i tanie dołączanie do systemu nowych użytkowników. III. Wysoka niezawodność systemu. IV. Wysokie bezpieczeństwo danych. V. Małe obciążenie łącza.	I. Ograniczona skalowalność. II. Wydajność ograniczona przez sprzęt. III. Duże obciążenie jednego komputera. IV. Uzależnienie od jednego producenta DBMS. V. Słaba integracja z innymi systemami. VI. Spadek wydajności przy dużej liczbie (>100) użytkowników. VII. Wysoka cena głównego komputera.

Mimo wielu wad architektura host-terminal jest nadal stosowana. Komputery główne wyposażone są zwykle systemy operacyjne będące odmianami systemu Unix – na przykład Solaris firmy Sun, AIX firmy IBM [14, 24], a do najczęściej stosowanych systemów DBMS należą: Oracle firmy Oracle [3] oraz DB/2 firmy IBM [7].

Po upowszechnieniu się komputerów klasy PC, powiększeniu ich możliwości funkcjonalnych i wydajności, bazodanowe systemy scentralizowane dostosowano do tej ciągle jeszcze prostej i stosunkowo mało wydajnej platformy sprzętowej i operacyjnej. Relatywnie duże systemy bazodanowe stały się dostępne dla pojedynczych użytkowników.

Komputery PC oraz minikomputery pozwalały wykorzystać wszystkie funkcje systemu bazy danych, ale tylko jednemu użytkownikowi. Na jednym minikomputerze znajdowały się wszystkie elementy systemu: aplikacja, prosty system DBMS oraz baza danych. Systemy te znalazły powszechne zastosowanie w małych firmach handlowych oraz usługowych i znajdują się do dziś w powszechnym użyciu.

Środowiskiem operacyjnym tych systemów jest najczęściej system operacyjny MS-DOS lub jego odpowiednik firmy IBM, albo systemy operacyjne klasy Windows. Jako systemy DBMS są stosowane najczęściej systemy baz danych standardu XBase, w szczególności Clipper, dBase, FoxPro. Koszty oprogramowania bazodanowego są tu znacznie niższe, niż dla architektury host-terminal [13].

4.2. ARCHITEKTURA Z SERWEREM PLIKÓW

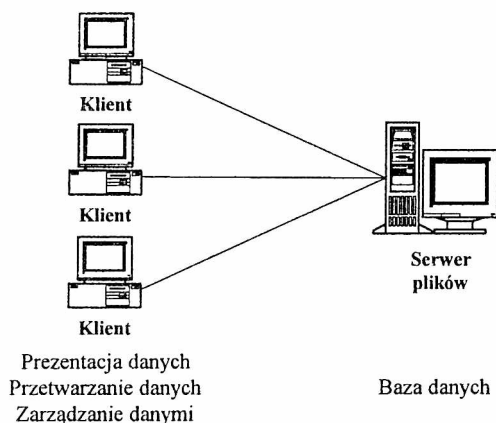
Połączenie wielu komputerów klasy PC w lokalną sieć komputerową umożliwiło budowę systemów wielodostępnych, podobnie jak to miało miejsce w architekturze host-terminal. Po wydzieleniu jednego komputera jako serwera plików można było otrzymać architekturę z serwerem plików. W architekturze tej mamy do czynienia z rozłożeniem elementów systemu bazy danych pomiędzy poszczególne komputery [10, 15].

Serwer plików świadczy usługi sieciowego systemu operacyjnego i udostępnia pliki znajdujące się na jednym komputerze (serwerze sieciowym) innym komputerom (klientom).

Baza danych jest przechowywana na serwerze sieciowym w postaci jednego lub wielu plików, które są dostępne współbieżnie dla wielu procesów i użytkowników. Sieciowy system operacyjny musi zatem posiadać takie cechy jak wielodostępność plików oraz mechanizmy ich ochrony. Komputery klientów zawierają natomiast wszystkie pozostałe elementy systemu bazy danych. Na każdym z nich musi być uruchomiona aplikacja użytkownika oraz musi się tam znajdować odpowiedni system DBMS. Ponadto, dla celów obsługi bazy danych znajdującej się na serwerze, komputer-klient musi zawierać usługę zapewniającą przezroczystą widoczność plików alokowanych na serwerze. Usługę tę może dostarczać np. oprogramowanie będące klientem systemu operacyjnego uruchomionego na serwerze.

Aplikacje użytkowników wykonują wszystkie swoje funkcje na współużytkowanej bazie danych znajdującej się na serwerze sieciowym. DBMS obsługuje bazę danych na polecenie aplikacji. W tym celu realizowane są bezpośrednio procesy odczytu i zapisu do plików poprzez funkcje sieciowego systemu operacyjnego. DBMS lub system operacyjny musi zapewniać mechanizm blokowania relacji bazy danych oraz pojedynczych rekordów w bazie danych. Schemat strukturalno-funkcjonalny architektury z serwerem plików przedstawiono na rysunku 3.

W architekturze z serwerem plików przetwarzanie oraz zarządzanie danymi jest rozłożone na wiele komputerów, dzięki czemu od serwera nie wymaga się bardzo dużej mocy obliczeniowej. System jest dobrze skalowalny, gdyż każdy nowy komputer oznacza nową moc obliczeniową przy nieznacznym wzroście obciążenia serwera. Jednak dodanie nowego użytkownika oznacza zwiększenie ruchu sieciowego i wzmożenie przepływu danych przez sieć. Ponieważ DBMS znajduje się na komputerze-kliencie, to wszystkie potrzebne do przetwarzania dane muszą być przesyłane przez sieć. Natężenie transmisji danych obniża w sposób wyraźny efektywność działania sieci. Jest to największa wada tej architektury.



Rys. 3. Schemat strukturalno-funkcjonalny architektury z serwerem plików

Fig. 3. Structure-functional schema of the architecture with a file server

Dodatkowo, serwer plików nie zapewnia należytego bezpieczeństwa danych. Ochrona odbywa się tylko na poziomie dostępu do całego systemu bazy danych, a nie jego elementów.

Wady i zalety architektury z serwerem plików zestawiono w tabeli 2.

Typowym przedstawicielem sieciowego systemu operacyjnego z usługą serwera plików jest system Novell NetWare firmy Novell. Można również stosować mniej wydajne systemy, np. Windows 95 firmy Microsoft albo oprogramowanie sieciowe Lantastic. Na komputerach klienckich używa się oprogramowania stacji roboczych systemu Novell Netware lub systemu Windows z oprogramowaniem klienta systemu

Novell Netware. Do najczęściej stosowanych tu systemów DBMS należą: Clipper, FoxPro, Progress. Często również systemy bazodanowe pracujące w środowisku systemu sieciowego Novell NetWare wykorzystują jako DBMS system Btrieve posiadający namiastkę cech architektury klient-serwer i opierający się o standard bazodanowy XBase.

Tabela 2

Wady i zalety architektury z serwerem plików

Zalety	Wady
I. Przetwarzanie rozłożone na wielu komputerach.	I. Niewielkie bezpieczeństwo danych.
II. Małe wymagania co do serwera sieciowego.	II. Komputery klientów muszą mieć znaczną moc obliczeniową.
III. Łatwe dołączanie do systemu nowych użytkowników.	III. Duże obciążenie sieci.
IV. Skalowalność.	IV. Uzależnienie od jednego producenta DBMS.
	V. Słaba integracja z innymi systemami.
	VI. Rozproszona administracja systemem, wyższe koszty administrowania.

4.3. ARCHITEKTURY KLIENT-SERWER

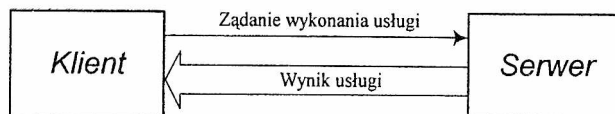
W tym rozdziale zostanie omówiona komunikacja w systemie klient-serwer a także dwu- oraz dwu-i-półwarstwowa architektura klient-serwer.

4.3.1. KOMUNIKACJA W SYSTEMIE KLIENT-SERWER

Architekturę klient-serwer tworzą serwer sieciowy i przyłączone do niego komputery-klienci. Zatem architektura infrastruktury sprzętowej jest identyczna jak w systemie sieciowym z serwerem plików [7, 9]. W przypadku systemów bazodanowych osadzonych w środowisku architektury klient-serwer, system DBMS funkcjonuje na serwerze. Dlatego serwer ten nazywamy serwerem bazy danych. Na komputerach-klientach znajdują się natomiast programy aplikacji oraz oprogramowanie pośredniczące w komunikacji między aplikacją a serwerem bazy danych.

Termin klient-serwer został zapożyczony z systemów operacyjnych [14, 24], w szczególności zaś z rozproszonych systemów operacyjnych. Jedyna różnica w zastosowaniu tego paradygmatu w rozproszonych systemach bazodanowych jest to, że procesy te funkcjonują na oddzielnych komputerach. Zwykle z jednym serwerem współdziała jeden lub wiele klientów.

Systemy bazodanowe dziedziczą zazwyczaj tryb pracy klient–serwer z systemów operacyjnych, w środowisku których pracują. Dlatego pod pojęciem architektury klient–serwer w systemach bazodanowych rozumie się zbiór połączonych w logiczną i funkcjonalną całość i powiązanych ze sobą par – usługodawca (serwer) oraz usługobiorca (klient). Podobnie jak w przypadku systemów operacyjnych zwykle jeden serwer współpracuje z wieloma klientami (Rys. 4). Serwer i klienci mogą, lecz nie muszą, pracować na jednym komputerze i w środowisku jednego systemu operacyjnego.



Rys. 4. Komunikacja procesów w architekturze klient–serwer
Fig. 4. Process communication in the client/server architecture

4.3.2. DWUWARSTWOWA ARCHITEKTURA KLIENT–SERWER

W architekturze klient–serwer w systemach bazodanowych klientami są aplikacje użytkownika, a serwerem jest serwer bazy danych. Aplikacje komunikują się z serwerem bazy danych, aby (1) pobrać dane z bazy danych albo je zmodyfikować. Serwer bazy danych wykonuje polecenia na rzecz aplikacji. Architektura klient–serwer, którą teraz opisujemy nosi nazwę architektury dwuwarstwowej. Mamy tu warstwę klientów oraz warstwę serwera [5, 7, 9].

Zadaniem warstwy klientów jest przetwarzanie danych oraz prezentacja danych odczytywanych lub zapisywanych do bazy danych.

Ponadto klient musi zawierać oprogramowanie zapewniające realizację funkcji komunikacji – najczęściej sieciowej – z serwerem. Funkcja ta jest dostarczana przez tzw. oprogramowanie middleware, czyli część aplikacji ulokowaną pomiędzy warstwą sprzętowo-operacyjną a oprogramowaniem aplikacji klienta.

Oprogramowanie middleware oferuje zestaw przezroczystych dla programisty usług, które umożliwiają połączenie różnych części aplikacji systemu bazodanowego w funkcjonalną całość [8, 16, 25, 29, 31]. Oprogramowanie to działa identycznie na różnych klienczkich platformach sprzętowych i operacyjnych. Jego działanie nie zależy od formatów danych używanych w różnych standardach i typach baz danych (systemów DBMS) oraz od języków, w których usługi te są definiowane oraz protokołów sieciowych.

Middleware to zbiór wielu różnych technologii, które powinny:

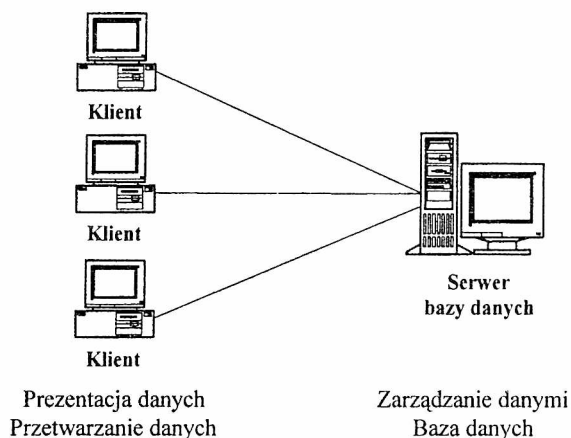
- oferować i świadczyć usługi komunikacyjne,
- współpracować z różnymi protokołami sieciowymi,
- dostarczać usług dostępu do danych i realizować połączenia pomiędzy klientami i serwerami w różnych standardach,
- realizować dostęp do różnych struktur danych.

Middleware dostarcza zatem zbiór międzywarstwowych usług komunikacyjnych w architekturze klient–serwer.

Wykonanie jednostkowego zadania przez serwer bazy danych przebiega według następującego scenariusza:

1. Za pomocą aplikacji w warstwie klienta użytkownik wprowadza wartość formułę z kryterium, według którego ma nastąpić szukanie rekordu.
2. Aplikacja przygotowuje odpowiednie zapytanie w języku zrozumiałym dla serwera.
3. Klient wysyła zapytanie do serwera.
4. Serwer realizuje zapytanie i formatuje wynik.
5. Serwer wysyła wynik zapytania do klienta.
6. Aplikacja odbiera dane i prezentuje je na ekranie klienta.

Schemat strukturalno-funkcjonalny dwuwarstwowej architektury klient–serwer przedstawiono na rysunku 5.



Rys. 5. Schemat strukturalno-funkcjonalny dwuwarstwowej architektury klient–serwer

Fig. 5. The structure-functional schema of two-layer architecture

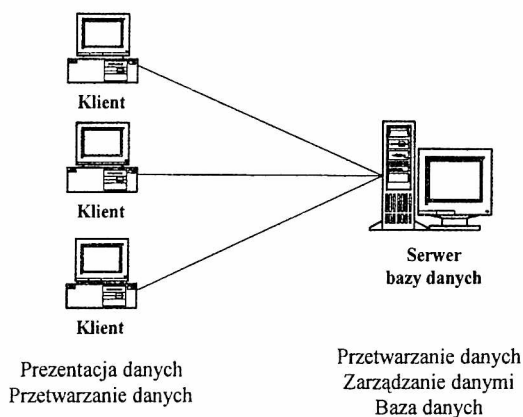
4.3.3. DWU-I-PÓŁWARSTWOWA ARCHITEKTURA KLIENT–SERWER

Etap przetwarzania danych realizowany przez aplikację składa się zwykle z sekwencji poleceń dla serwera bazy danych, zatem następuje wiele odwołań do serwera. W celu zmniejszenia ruchu sieciowego generowanego wielokrotnym łączeniem się klienta z serwerem, wiele serwerów baz udostępnia możliwość tworzenia procedur zdalnych (stored procedures) oraz wyzwalaczy (triggers) zapewniających wykonanie całości przetwarzania, jednorazowy zwrot wyników i tym samym zmniejszenie ilości odwołań do serwera [3, 9, 20, 38]. Wiele serwerów posiada takie możliwości funkcjonalne,

co prowadzi do dwu-i-półwarstwowej (niepełnej trójwarstwowej) architektury klient-serwer (Rys. 6). Znaczną część funkcji przetwarzania danych można wtedy przesunąć z aplikacji na serwer. Aplikacja jest wtedy „odchudzona”. Pozostała część przetwarzania danych musi jednak być realizowana przez aplikację, gdyż języki definiowania przetwarzania danych implementowane na serwerach nie zawsze są odpowiednio funkcjonalne i wygodne w stosowaniu. W pełni funkcjonalne są dopiero języki trzeciej generacji.

Wyzwalacze to specjalne procedury wywoływane automatycznie po zajściu zdarzenia aktualizacji danych (ich dodawaniu, usuwaniu lub modyfikacji) zmieniającego więzy w bazie.

Główną zaletą dwuwarstwowej i dwu-i-półwarstwowej architektury klient-serwer jest to, że zarówno baza danych jak i funkcje zarządzanie danymi są umiejscowione na tym samym komputerze. Zatem dostęp do instancji plikowych bazy danych jest realizowany lokalnie – tzn. na serwerze. Przez sieć transmitowane są jedynie wyniki przetwarzania, a nie wszystkie dane potrzebne do wykonania tej operacji. Transmituje się jedynie zapytania do bazy.



Rys. 6. Schemat strukturalno-funkcjonalny dwu-i-półwarstwowej architektury klient-serwer

Fig. 6. The structure-functional schema of two-and-half-a-layer client/server architecture

Serwer bazy zapewnia wysokie bezpieczeństwo danych. Ochrona odbywa się na kilku poziomach: (1) komunikacji użytkownika z serwerem poprzez mechanizmy identyfikacji i autoryzacji, (2) mechanizm widoków – czyli perspektyw użytkownika na dane, (3) fizyczną izolację użytkownika od serwera sieciowego.

Architektura klient-serwer jest dobrze skalowalna. Każdy nowy klient zwiększa moc systemu, natomiast nie zwiększa znacząco ruchu sieciowego – tak jak w przypadku architektury serwera plików. Skalowalność jest jednak ograniczona. System bazodanowy pracuje efektywnie, gdy aplikacje nie są skomplikowane funk-

cyjonalnie oraz gdy liczba użytkowników nie przekracza 100. Gdy aplikacje są bardzo skomplikowane i w celu przetwarzania danych żądają wielu dostępów do bazy, serwer musi mieć dużą moc obliczeniową lub w ogóle nie jest w stanie realizować usług. Serwer można wtedy rozbudować sprzętowo albo wymienić na bardziej wydajny. Pod tym względem postępowanie jest identyczne jak to ma miejsce przy stosowaniu architektury host-terminal.

Architektura klient-serwer jest elastyczna, gdyż aplikacja użytkownika może pracować na różnego rodzaju klientach: minikomputerach, stacjach roboczych o różnej wydajności, terminalach. Mając jedną bazę danych można tworzyć szereg różnych aplikacji klientów, które – mając wspólny dostęp do danych – operują na nich w zupełnie inny sposób. Administrator zarządzający serwerem nie jest uzależniony od konkretnych aplikacji oraz od problemów związanych ze wspólnym dostępem do bazy i może skoncentrować się na utrzymaniu bazy danych (zapewnieniu spójności danych i spójności informacyjnej, modyfikacjach struktury i zapewnieniu wydajnego operowania na bazie), a gdy jest bazą rozproszoną, to koncentruje się na poprawie wydajności i efektywności przetwarzania danych oraz problemach utrzymania wielodostępności do bazy.

Obsługa heterogenicznych systemów bazodanowych, w których stosuje się architekturę klient-serwer jest bardzo ułatwiona [9, 33]. Jeśli w przedsiębiorstwie istnieją bazy danych na różnych serwerach pracujących w różnych środowiskach operacyjnych, można tworzyć aplikacje, które udostępniają bazy danych zarówno rozproszone jak i bazy o różnym stopniu sfederowania oraz tzw. systemy wielobazodanowe, w których bazy są od siebie całkowicie niezależne.

W przypadku tworzenia systemu w architekturze klient-serwer od podstaw, istnieje możliwość doboru najlepszych dostępnych na rynku narzędzi do tworzenia aplikacji, oprogramowania komunikacyjnego i serwerów baz danych i rozwijania całości infrastruktury bazodanowej w przyszłości.

Niestety, stopień komplikacji systemu całościowego wykorzystującego architekturę klient-serwer jest dużo większy, niż systemów o innych architekturach, opartych na wykorzystaniu pojedynczych, spójnych pakietów oprogramowania. Zupełnie elementarnym wymogiem jest zgodność oprogramowania narzędziowego do projektowania i tworzenia struktury topologiczno-logicznej systemów bazodanowych klient-serwer ze standardami.

Duży stopień komplikacji systemu wiąże się z wyższymi kosztami administrowania systemem. Każda modyfikacja reguł biznesowych przetwarzania danych wymaga aktualizacji aplikacji zwykle różnorodnych klientów, które rozmieszczone są w różnych miejscach sieciowych oraz rekonfiguracji ich środowisk programowych.

Pewnym wyjściem może być zastosowanie dodatkowego serwera sieciowego, z którego każdy klient pobiera oprogramowanie aplikacji. Wystarczy wówczas wymienić aplikację w jednym miejscu sieciowym – na serwerze.

W systemach o architekturze dwu-i-półwarstwowej wzrastają koszty zapewnienia odpowiedniego poziomu bezpieczeństwa (integralności) danych oraz zagwarantowania ich spójności w bazie [11, 12].

Wady i zalety dwu-i-półwarstwowej architektury klient-serwer zostały podsumowane w tabeli 3.

Wymienione wyżej wady oraz zalety skłaniają do stwierdzenia, że architektura dwu-i-półwarstwowa nie najlepiej nadaje się do budowy takich systemów bazodanowych, w których ważna jest integralność oraz ochrona danych a liczba użytkowników jednocześnie korzystających z systemu wynosi 20 do 100 i zmienia się w czasie.

Systemy klient-serwer o architekturze dwuwarstwowej i dwu-i-półwarstwowej buduje się w oparciu systemy interfejsów graficznych (np. Windows 95, OS/2) w warstwie klientów oraz wysokowydajne systemy operacyjne (odmiany systemu Unix, Windows NT) w warstwie serwera. Aplikacje wytwarza się wykorzystując takie środowiska RAD (*Rapid Application Development*) jak: PowerBuilder [35], Delphi [16, 25], Visual Basic. Jako serwery bazy danych stosuje się: Sybase Server (prekursor architektury klient-serwer), Oracle [3, 33, 34], Informix [29]. Do przetwarzania danych na serwerze bazy danych służy standardowy język SQL (*Structured Query Language*). Komunikacja między aplikacją a klientem odbywa się poprzez sterowniki dedykowane, interfejs ODBC lub interfejs CLI (*Call Level Interface*) [35, 36].

Wiele wad architektury dwu-i-półwarstwowej zostało usuniętych dzięki zastosowaniu nowocześniejszych architektur klient-serwer.

Tabela 3

Wady i zalety dwu-i-półwarstwowej architektury klient-serwer

Zalety	Wady
I. Małe obciążenie sieci.	I. Wydajność ograniczona architekturą.
II. Przetwarzanie rozłożone na wielu komputerach.	II. Duży stopień złożoności systemu.
III. Bezpieczeństwo danych.	III. Komputery klientów muszą mieć znaczną moc obliczeniową.
V. Skalowalność.	IV. Kosztowne dołączanie do systemu nowych użytkowników.
VI. Elastyczność: różne rodzaje aplikacji użytkownika, różne technologicznie rodzaje klientów.	V. Rozproszona administracja systemem, wysokie koszty administrowania.
VII. Centralne administrowanie bazą danych.	VI. Drogie licencje na połączenia z serwerem bazy.
VIII. Obsługa środowisk heterogenicznych.	
IX. niezależnienie od jednego producenta narzędzi.	

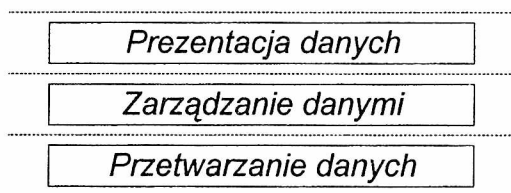
5. NOWOCZESNE ARCHITEKTURY KLIENT-SERWER

Od chwili powstania pierwszych systemów baz danych wyróżniano w nich pewne elementy. Byłoby bardzo korzystne, gdyby każdy z tych elementów stanowił odrębną część (warstwę) systemu całościowego, dobrze odizolowaną od innych i jedynie udostępniającą interfejsy z dostępem do jej usług. Jednakże dopiero od niedawna technologie informatyczne pozwalają na tworzenie systemów baz danych w trójwarstwowej architekturze klient-serwer. Dalsze zwiększanie liczby warstw prowadzi do najbardziej elastycznej spośród obecnie znanych architektur – architektury wielowarstwowej, nazywanej niekiedy architekturą klient-sieć [4, 22, 23, 28, 27, 31, 38].

5.1. TRÓJWARSTWOWA ARCHITEKTURA KLIENT-SERWER

Przy bardziej złożonych systemach bazodanowych przesunięcie części przetwarzania danych na serwer bazy może nadmiernie go obciążyć, przez co staje się niewydajny. Rozwiązaniem jest umieszczenie większości przetwarzania danych na dedykowanym komputerze w postaci serwera aplikacji, który stanowi środkową warstwę systemu.

Podział systemu bazy danych na trzy warstwy dotyczy bardziej podziału na trzy logiczne części (warstwy) – jak to pokazano na rysunku 7, niż umieszczenia systemu na trzech komputerach [3, 9, 20, 31, 38].



Rys. 7. Komponenty logiczne architektury trójwarstwowej
Fig. 7. Logical components of the three-layer architecture

Te trzy warstwy (wymienialiśmy je już wcześniej kilkakrotnie, jednak teraz spoglądamy na nie jeszcze z innej strony) to:

- Warstwa prezentacji danych – obejmuje interfejs użytkownika z systemem. Warstwa ta może zostać utworzona za pomocą tradycyjnych interfejsów graficznych (minikomputery, stacje robocze) lub tekstowych (terminale), stron Web (komputery sieciowe), tzw. kiosków, czy takich technologii przyszłości jak telewizja interaktywna, interaktywne sterowanie głosem (*interactive voice response*).
- Warstwa przetwarzania danych. W trójwarstwowej architekturze wszyscy użytkownicy mają dostęp do danych przez warstwę prezentacji, która komunikuje się z warstwą przetwarzania danych. Warstwa ta może zawierać, prócz typowych reguł danych, takie funkcje jak: szeregowanie wsadowe (*scheduled batch processes*), automatyczną obsługę zdarzeń.
- Warstwa zarządzania danymi. Dostarcza dane, na których operuje warstwa przetwarzania danych. Rolę warstwy pełni zwykle serwer bazy danych, ale mogą to być również dane pochodzące np. z arkusza kalkulacyjnego.

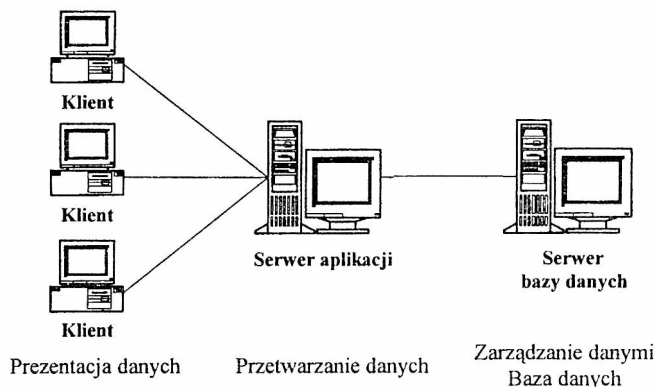
Trzy warstwy powinny być fizycznie rozdzielone pomiędzy dwa serwery programowe – każdy na odrębnym serwerze sieciowym – oraz wiele różnych klientów, np. z różnymi z interfejsami graficznymi aplikacji. Każdej warstwie logicznej odpowiada jedna warstwa fizyczna: (1) prezentacji danych – klienci, (2) przetwarzania danych – serwer aplikacji, (3) zarządzania danymi – serwer bazy danych. Nic nie stoi na przeszkodzie, aby serwer aplikacji umieścić na tym samym serwerze sieciowym, na którym znajduje się serwer bazy danych. Istotne jest to, że każda warstwa to oddzielny proces lub zbiór procesów.

Schemat strukturalno-funkcjonalny tej architektury trójwarstwowej przedstawiono na rysunku 8. Nowym elementem w tej architekturze jest serwer aplikacji. Jest to proces wykonujący usługi na zlecenie klientów. Dla klientów jest to serwer, ale dla serwera bazy danych proces ten jest klientem. Klienci nie komunikują się bezpośrednio z serwerem bazy danych, ale robią to poprzez serwer aplikacji.

Część przetwarzania danych może w dalszym ciągu może być realizowana na serwerze bazy danych – podobnie jak w architekturze dwu-i-półwarstwowej – jednakże należy tego unikać ze względu na brak przenośności oprogramowania pomiędzy serwerami różnych producentów.

Dużą korzyścią ze stosowania architektury trójwarstwowej jest łatwość implementacji zróżnicowanych technologicznie interfejsów użytkownika wykorzystujących tą samą warstwę przetwarzania danych. Utworzenie interfejsu nowego typu w istniejącym systemie bazodanowym sprowadza się do implementacji jedynie warstwy prezentacji. Przy wykorzystaniu technologii obiektowych zadanie zbudowania dwóch górnych warstw jest zadaniem stosunkowo łatwym.

Stosowanie architektury trójwarstwowej jest konieczne przy budowie aplikacji Web [34]. Rozmieszczanie aplikacji w trzech warstwach powoduje, że łatwiej jest stworzyć interfejs użytkownika z użyciem technologii stosowanych w Web.



Rys. 8. Schemat strukturalno-funkcjonalny architektury trójwarstwowej
 Fig. 8. Structure-functional schema of the three-layer architecture

Dla projektantów oraz programistów najważniejsze jest aby odseparować w systemie bazodanowym logikę prezentacji od logiki przetwarzania. W warstwie prezentacji aplikacja zawiera tylko interfejs użytkownika. Dodatkowym jej zadaniem może być sprawdzanie poprawności wprowadzanych przez użytkownika danych. Wykonywanie tej funkcji po stronie serwera mogłoby zbyt obciążać sieć. Aplikacje w warstwie prezentacji wywołują bezpośrednio funkcje lub metody (obiektywne) zdefiniowane w warstwie przetwarzania. Dlatego aplikacja warstwy prezentacji musi znać interfejs oferowany przez aplikację warstwy przetwarzania.

5.2. WIELOWARSTWOWA ARCHITEKTURA KLIENT-SERWER

Architektura wielowarstwowa pozostaje logiczną strukturą trójwarstwowa, ale warstwa środkowa jest w niej podzielona na wiele części (partitioning) i najlepiej jest, gdy są one rozmieszczone na różnych maszynach.

Duży serwer aplikacji może jednak znacznie obciążyć serwer sieciowy. Pewnym rozwiązaniem jest tutaj zwiększanie wydajności serwera, lecz lepszym sposobem może być zastosowanie kilku tańszych serwerów, na których umieszczamy elementy serwera aplikacji, czyli mniejsze serwery aplikacji. Dodatkowo, serwery sieciowe można fizycznie umieścić w odpowiednich oddziałach instytucji, dzięki czemu klienci korzystający z danego serwera są lokowani fizycznie blisko serwera aplikacji, z którego usług korzystają. Jeśli oddziały pozostają w pewnych zależnościach ujętych w więzach semantyki bazy danych, to jest to odwzorowywane przez wzajemne relacje klient-serwer pomiędzy serwerami aplikacji – jeden serwer korzysta z usług innego serwera.

Replikacja tego samego serwera aplikacji na dwóch lub więcej serwerach sieciowych zabezpiecza system całościowy przed awarią jednego z serwerów. W takim przypadku replika przejmuje klientów połączonych z serwerem aplikacji, który uległ awarii. Replikowanie serwera aplikacji może być również sposobem zwiększenia mocy obliczeniowej całego systemu.

5.3 ZADANIA I RODZAJE MIDDLEWARE

W architekturze trójwarstwowej oprogramowanie middleware [2, 8, 9, 16, 17, 30, 31] musi zapewniać dodatkowe usługi, które nie były konieczne w architekturze dwu-i-półwarstwowej. Usługi te powinien dostarczać serwer aplikacji. Są to:

- usługi połączeń – różne sposoby przekazywania komunikatów pomiędzy procesami,
- usługi zarządzania zasobami – zarządzanie wątkami, kolejkovanie zadań, usługi katalogowe nazw do określania położenia serwerów programowych w sieci,
- równoważenie obciążenia (*load balancing*) dostępnych serwerów – aplikacji i baz danych,
- usługi transakcyjne na poziomie warstwy środkowej – zarządzanie transakcjami z mechanizmem dwufazowego zatwierdzania – do obsługi różnych serwerów i źródeł danych,
- usługi bezpieczeństwa – ujednolicenie zarządzanie bezpieczeństwem,
- usługi nadzorujące i zarządzające middleware.

5.4. MONITOR TRANSAKCJI, SERWER TRANSAKCYJNY

Monitory transakcji (*Transaction Monitors – TM*) stosowano powszechnie w komputerach typu mainframe [20]. Głównym ich zadaniem było równoważenie obciążenia systemu przy bardzo dużej liczbie użytkowników. Monitor wykorzystuje własność polegającą na tym, że tylko niewielka ilość aplikacji wymaga wykonania danej usługi jednocześnie. Dlatego TM uruchamia znacznie mniejszą ilość połączeń do serwera bazy danych, niż jest aktywnych w danej chwili użytkowników.

W architekturze trójwarstwowej monitory transakcji egzystują w warstwie środkowej, obok serwerów aplikacji [8, 18, 19, 25, 30, 38]. Klient komunikuje się z serwerem bazy danych poprzez monitor transakcji. Gdy inicjuje on sesję, łączy się z procesem monitora transakcji, który to – w odpowiednim momencie – zażąda usługi od serwera bazy. Aby móc obsłużyć dużą liczbę transakcji jednocześnie, TM zbiera zlecenia a następnie kolejno je wykonuje. Komunikacja z klientem jest asynchroniczna. Po dostarczeniu zapytania bazodanowego do TM, klient może wykonywać inną pracę, a TM przejmuje odpowiedzialność za obsługę tego zapytania.

Drugim zadaniem TM jest zabezpieczanie przed awariami i zapewnienie niezawodności całościowego systemu bazodanowego. Jeśli TM stwierdzi, że jeden z serwerów przestał działać, wtedy stara się go uruchomić, a jeśli się to nie udaje, może skierować ruch sieciowy do innego, działającego serwera.

Przy użyciu TM można zarządzać danymi w bazodanowych systemach heterogenicznych (różne serwery baz danych, nietypowe źródła danych) stosując dwufazowe protokoły zatwierdzania transakcji.

5.5. PRZETWARZANIE BAZUJĄCE NA ROZPROSZONYCH KOMPONENTACH

Obecnie scharakteryzujemy krótko zaawansowane technologie bazodanowego przetwarzania rozproszonego. Nowym sposobem tworzenia aplikacji jest składanie ich z „klocków”, zwanych komponentami [6, 22, 25, 28, 32, 37]. Umożliwia to realizację przetwarzania bazującego na komponentach (*component-based computing*) z możliwością dowolnego rozmieszczania i rozpowszechniania komponentów w sieci.

Komponenty dobrze pasują do architektury klient-serwer, gdyż łączą dane i metody które mogą być umieszczane w dowolnym miejscu rozproszonej aplikacji.

Komponent jest rozwinięciem pojęcia obiektu, który:

- może komunikować się z innymi komponentami poprzez sieć zgodnie z jednym ze standardów przetwarzania rozproszonego (CORBA [22, 23, 25], DCOM [18]),
- jest napisany w jednym z języków programowania obiektowego,
- umożliwia przezroczysty dostęp z poziomu aplikacji do rozproszonych i lokalnych komponentów,
- może być łatwo wymieniony na komponent inaczej zaimplementowany.

Komponenty rozproszone mogą być umieszczane w dowolnym miejscu sieci. Można je łączyć w większą całość lub mogą być one wyspecjalizowane, przy czym nie są one związane z konkretnym językiem programowania.

Podział aplikacji na trzy warstwy nadal obowiązuje, ale w każdej części pojawiają się komponenty, zamiast procesów. Nie ma wyraźnego podziału na klienta i serwer – komponenty prezentacji przekazują swoje zapytania do pośrednika, który po znalezieniu odpowiedniego komponentu przetwarzającego przekazuje mu zapytanie.

Najważniejszą wadą paradygmatu komponentów rozproszonych są znaczne utrudnienia, a czasami zupełny brak możliwości równoważenia obciążenia.

W przypadku relacyjnych baz danych, między serwerem bazy a aplikacją zachodzi konflikt modeli – pomiędzy obiektowością w warstwach prezentacji i przetwarzania a strukturami typu tabele (relacjami bazodanowymi) na serwerze bazy danych.

Komponenty rozproszone muszą być budowane w pewnym standardzie, tak aby mogły się ze sobą komunikować niezależnie od rodzaju sieci, systemu komputerowego i języka, w którym zostały zaimplementowane. Ponadto potrzebne jest oprogramowanie pośredniczące do przekazywania komunikatów. CORBA i DCOM to dwa najbardziej rozpowszechnione modele (i standardy) komponentów rozproszonych. Istnieją też specjalizowane środowiska do tworzenia aplikacji rozproszonych oferujących własne modele rozproszoneści.

Model CORBA został zdefiniowany przez OMG (*Object Managment Group*) [23] dla potrzeb integracji aplikacji klient-serwer w heterogenicznym środowiskach rozproszonych. Komponenty można tworzyć w różnych językach. Użytkownik może zmieniać dowolnie alokację komponentów na konkretnych komputerach. Model CORBA jest oferowany na wiele platform sprzętowych, operacyjnych i dla wielu środowisk programistycznych stworzonych przez różnych producentów i wielu języków, np. C++, Java. Aby obiekty napisane w różnych językach mogły się ze sobą komunikować, do definiowania interfejsu klasy używa się języka IDL (*Interface Definition Language*).

Model DCOM wraz z ActiveX [18] jest odpowiedzią firmy Microsoft na zagrożenie dominacją standardu CORBA i jest do standardu CORBA bardzo podobny. Jest on dedykowany dla środowiska operacyjnego (i platformy) Windows NT, ale obecnie istnieją już jego implementacje na wszystkie popularne platformy systemowe. Komponenty ActiveX mogą być – podobnie jak komponenty CORBA – tworzone w różnych językach programowania [16, 17, 25] (np. C++ w Visual C++, Object Pascal z pakietu Delphi 3). Nazwa ActiveX ma znaczenie przede wszystkim marketingowe, ale z technicznego punktu widzenia znaczące są dwa standardy, na których ActiveX bazuje: interfejs DCOM (*Component Object Model*) oraz technologia OLE (*Object Linking and Embending*). Standard DCOM definiuje jednolity interfejs komunikowania się między sobą obiektów, który jest niezależny od języka programowania. Zatem DCOM standaryzuje sposób udostępniania metod obiektów.

5.6. WADY I ZALETY ARCHITEKTURY WIELOWARSTWOWEJ

Zastosowanie architektury wielowarstwowej zapewnia dość dobrą wydajność całłościowego systemu bazodanowego. Dość dobra jest też skalowalność (*scalability*) systemu pracującego w takim środowisku. Dzięki zastosowaniu monitora transakcji w warstwie środkowej system może obsługiwać bardzo dużo połączeń współbieżnie. Istnienie warstwy środkowej zapewnia też w pewnym stopniu równoważenie obciążenia serwerów baz danych. Dzięki elastyczności całego systemu można łatwo modyfikować reguły biznesowe przetwarzania

danych. Ponieważ znajdują się one jedynie w warstwie środkowej, łatwo jest je zmieniać. Wystarczy wymienić implementację komponentu. Interfejs tego komponentu pozostaje niezmienny.

W architekturze wielowarstwowej klientów zrealizowanych w oparciu o różne technologie, przy dołączeniu klienta nowego typu wystarczy zaimplementować na nowo jedynie interfejs użytkownika. Warstwa przetwarzania danych obsługuje dowolnych klientów i można ją pozostawić bez zmian [35, 36].

Stosując do budowy aplikacji komponenty można korzystać z różnych istniejących i sprawdzonych rozwiązań. Nie trzeba tworzyć własnych, bo zwykle jest to dość kosztowne.

Tabela 4

Wady i zalety architektury wielowarstwowej klient-serwer

Zalety	Wady
I. Małe obciążenie sieci.	I. Duży stopień złożoności systemu.
II. Przetwarzanie rozłożone na wielu serwerach.	II. Przyłączanie do systemu nowych użytkowników jest kosztowne.
III. Wysoki poziom bezpieczeństwa danych.	III. Rozproszone administrowanie systemem, wysokie koszty administrowania.
IV. Skalowalność.	IV. Konieczność dostrajania systemu.
V. Elastyczność aplikacji użytkowników, możliwość stosowania różnych technologii klientów i zmiennych reguł danych.	V. Niewielka ilość dostępnego oprogramowania narzędziowego wysokiej jakości.
VI. Centralne administrowanie bazą danych.	
VII. Obsługa środowisk heterogenicznych.	
VIII. Uniezależnienie od jednego producenta narzędzi programistycznych i produktów programowych.	
IX. Spójne przetwarzanie rozproszone w całym systemie.	
X. Nieograniczona wydajność.	

Jednocześnie architektura wielowarstwowa jest najbardziej skomplikowaną z dotychczas stosowanych. Składa się z wielu różnych elementów. Zawiera funkcje realizacji wielu różnorodnych połączeń. Powoduje to, że system osadzony w takim środowisku jest bardzo złożony, a nad stopniem jego komplikacji trudno zapanować. Wady i zalety architektury wielowarstwowej klient-serwer przedstawiono w tabeli 4.

6. PODSUMOWANIE

Integracja baz danych w sieciach intranetowych i w Internecie ma kluczowe znaczenie praktyczne w nowoczesnych systemach z zasobami informacyjnymi. Zagadnienia związane z integrowaniem baz danych w bazodanowe systemy rozproszone ciągle jeszcze należą do technologicznych nowości. Brak jest obecnie wystarczających doświadczeń praktycznych wymaganych przy tworzeniu zintegrowanych sieciowo baz danych oraz dostatecznej wiedzy o narzędziach do ich projektowania i wytwarzania. Dlatego konieczne było wykonanie badań i prac studialnych nad nowoczesnymi technologiami klient-serwer i technologią aplikacji wielowarstwowych. Wykonanie tych badań miało duży walor poznawczy i pozwoliło zebrać informacje o standardach integrowania baz danych z sieciami intranetowymi i Internetem, a w konsekwencji, dokonać analizy metodologii i metodyk projektowania intranetowych i internetowych aplikacji bazodanowych w tych technologiach.

Literatura

- [1] ADAMCZYK B., *Advantage Database Server w praktyce*. Software, nr 12/96, s. 52.
- [2] ASHTON G., *Tworzenie rozwiązań Internet/Intranet w GroupWise 5*. Software, nr 5/97, s. 103.
- [3] AUSTIN D., *Poznaj Oracle 8*. MIKOM, 1998.
- [4] BERG C., *Corba w Javie*. Software, nr 3/97, s. 40.
- [5] BERNARDYN J., GRUBER J., i in., *Integracja scentralizowanych i rozproszonych systemów DBMS i systemów baz danych z sieciami intranetowymi i Internetem*. Raporty Włdz. Zakł. Inform. PWr., Ser. SPR, nr 1, Wrocław, 1998.
- [6] BROWN M.H., NAJORK M.A., *Rozproszone obiekty aktywne*. Software, nr 5/97, s. 14.
- [7] BRUCE E., BOBBY M., *Client/Server computing architecture and distributed systems management*. Artech House, Boston, 1994.
- [8] CARONE T.C., *Oprogramowanie pośredniczące w środowisku klient-serwer*. Software, nr 1, 2/97, str. 8.
- [9] CHMIELEWSKI M., MAJ K., *Architektura dwu, dwu i pół, trójwarstwowa – klient-serwer*. Software, nr 6/96, s. 42.
- [10] COMER D.E., *Sieci komputerowe TCP/IP. T. 1–3, Projektowanie w trybie klient-serwer, wersja BSD*. WNT, Warszawa, 1997.
- [11] CZUJOWSKI W., GRUBER J., *Administrowanie dużymi systemami baz danych*. Informatyka, nr 9/98, s. 17.
- [12] CZUJOWSKI W., GRUBER J., *Problemy z utrzymaniem dużych baz danych*. Informatyka, nr 10/98, str. 10.
- [13] GORMAN M.M., *Enterprise databases in a client/server environment*. Wiley, New York, 1994.
- [14] GABASI M., DUPOUY B., *Przetwarzanie rozproszone w systemie Unix*. LUPUS, Warszawa, 1995.
- [15] HALL C.L., *Techniczne podstawy systemów klient-serwer*. WNT, Warszawa, 1996.
- [16] HENDERSON K., *Bazy danych w architekturze klient-serwer*. Robomatic, Wrocław, 1998.
- [17] HENDERSON K., REISDORPH K., *C++ Builder 3*. Helion, Gliwice, 1998.
- [18] JACOBS J., *Narzędzia Microsoftu do integracji baz danych z WWW*. Software, nr 9/96, s. 72.
- [19] KRAMER I.M., *Microsoft Transaction Server*. Software, nr 4/97, s. 46.

- [20] OZSU M.T., VALDURIEZ P., *Principles of distributed database systems*. Englewood Cliffs, New York, Prentice-Hall, 1991.
- [21] ROBERTS J., ZIGMOND D., *Wykorzystanie OODCE*. Software, nr 1, 2/97, s. 16.
- [22] SIEGEL J., *CORBA. Fundamentals and programming*. OMG Group, John Wiley & Sons, Inc., Wiley Computer. Publishing Group, New York, 1996.
- [23] STAWOWSKI M., *OMG CORBA*. Software, nr 8/96, s. 86.
- [24] STEVENS W.R., *Programowanie zastosowań sieciowych w systemie Unix*. WNT, Warszawa, 1995.
- [25] TEIXEIRA P., PACHECO X., *Delphi 4 – podręcznik profesjonalisty*. Helion. Gliwice, 1998.
- [26] VASKEVITCH D., *Strategie Klient-Serwer. Systemy informatyczne w procesie przekształcania przedsiębiorstw*. IDG, Warszawa, 1995.
- [27] *Client/Server Collapse*. 9612d07.html .
- [28] Components Architecture. 9709d13.html.
- [29] DataBlade Module Development Overview. Praca niesygnowana. INFORMIX Press, October, 1997.
- [30] *Microsoft Transaction Server: Infrastruktura aplikacji rozproszonych*. Praca niesygnowana.
- [31] *Middleware on the Web*. 9610i09.html.
- [32] *Next-Generation Middleware*. 9709d14.html.
- [33] *Oracle Server 8 – Administrative Guide*. Praca niesygnowana. Oracle Press, 1997.
- [34] *Oracle Web Application Server*. Praca niesygnowana. Oracle Press, 1995.
- [35] *Pliki pomocy oraz dokumentacja firmowa w postaci elektronicznej dołączona do następujących pakietów oprogramowania firmy Powersoft: InfoMaker for Windows, Power Designer, Power Builder Enterprise, Power++ Enterprise*.
- [36] *Pliki pomocników oraz dokumentacja firmowa w postaci elektronicznej dołączona do następujących pakietów oprogramowania firmy Sybase: Sybase SQL Anywhere v.5.5.04, Adaptive Server Enterprise for Windows NT with Sybase SQL Remote and Powersoft InfoMaker*.
- [37] *Reevaluating Distributed Objects*. 9701d13.html.
- [38] *Sybase SQL Server 11*. Praca niesygnowana. Software, nr 5/97, s. 46.

CLIENT/SERVER DATABASE SYSTEMS AND MULTI-LAYER ARCHITECTURES

Client/server database systems and multi-layer applications technologies have recently become the most vibrant and fastest growing areas in information technology, especially in database systems integration. This paper provides the comparative review of state-of-the-art in designing client/server system architectures and distributed database systems integrated in the company Intranet and in the Internet environment. These modern technologies have proven to increase flexibility, modularisation, code reuse, and improve maintenance of large distributed database systems. Features of particular interest include: distribution of database systems, database system structure, database system infrastructure evolution coverage of different architectures and their functionalities. Also, advantages and disadvantages of these architectures are enumerated for every described architecture.

*aplikacje wielowarstwowe,
baza danych,
Internet,
sieci intranetowe*

Jacek GRUBER*
Zygmunt MAZUR*

TECHNOLOGIE PROJEKTOWE I PROGRAMISTYCZNE DLA APLIKACJI WIELOWARSTWOWYCH I ZINTEGROWANYCH SIECIOWO SYSTEMÓW BAZODANOWYCH

Systemy bazodanowe integrowane w sieciach intranetowych i w Internecie będą miały kluczowe znaczenie praktyczne w nowoczesnych systemach z zasobami informacyjnymi. Jak na razie są one obecnie technologiczną nowością. Ciągłe jeszcze brak wystarczających doświadczeń praktycznych wymaganych przy ich tworzeniu i dostatecznej wiedzy o narzędziach do ich projektowania i wytwarzania. Dlatego konieczne było wykonanie badań i prac studialnych nad nowoczesnymi technologiami klient-serwer i technologią aplikacji wielowarstwowych. Ich wykonanie miało duży walor poznawczy i pozwoliło zebrać informacje o standardach integrowania baz danych z sieciami intranetowymi i Internetem a w konsekwencji dokonać analizy metodologii i metodyk projektowania intranetowych i internetowych aplikacji bazodanowych w tych technologiach.

1. WPROWADZENIE

Technologie projektowe i programistyczne do integracji scentralizowanych i rozproszonych baz danych z sieciami intranetowymi i Internetem należą obecnie do najnowocześniejszych i najszybciej rozwijających się technologii sieciowych. Integracja systemów bazodanowych dotychczas często pracujących jedynie w izolowanych sieciach lokalnych lub intranetowych stała się pilną koniecznością. Współczesne bazy danych muszą być sprawnymi systemami biznesowymi oraz powinny umożliwiać zarządzanie rozproszonymi terytorialnie instytucjami. Jednocześnie systemy bazodanowe powinny być elastyczne, ponieważ muszą nadążać za

* Wydziałowy Zakład Informatyki Politechniki Wrocławskiej, Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, j.gruber@ci.pwr.wroc.pl; z.mazur@ci.pwr.wroc.pl

rozwojem organizacyjnym, technologicznym i materialnym instytucji oraz przyrostem liczby użytkowników. Ważną cechą jakościową współczesnych korporacyjnych baz danych jest możliwość udostępniania części ich zasobów informacyjnych w globalnym środowisku sieciowym i operacyjnym jakim jest Internet. Sieci lokalne i Internet są naturalnymi środowiskami funkcjonowania zarówno aplikacji wielowarstwowych jak i systemów o architekturach klient–serwer.

W dziedzinie integracji sieciowej systemów bazodanowych dokonał się ostatnio znaczący postęp jakościowy, ale nadal odczuwalne są niedostatki odpowiednich narzędzi projektowych i programistycznych.

Celem praktycznym artykułu jest uchwycenie najistotniejszych problemów projektowych i programistycznych występujących przy tworzeniu aplikacji wielowarstwowych. Analizę przeprowadzimy na wybranych reprezentantach środowisk i narzędzi programistycznych do tworzenia rozproszonych systemów bazodanowych [16].

2. TECHNOLOGIE PROJEKTOWE I SYSTEMY CASE DO MODELOWANIA I PROJEKTOWANIA ZINTEGROWANYCH SYSTEMÓW BAZODANOWYCH I APLIKACJI WIELOWARSTWOWYCH

W niniejszym rozdziale przedstawimy charakterystykę własności funkcjonalnych i użytkowych wybranych systemów CASE dla bazodanowych środowisk rozproszonych i sieciowych

Istnieje co najmniej kilka wiodących narzędzi i środowisk CASE nowej generacji do modelowania i projektowania sieciowych systemów bazodanowych i aplikacji wielowarstwowych. Dokonamy analizy stanu rozwoju tych technologii projektowanych zwracając uwagę na ich możliwości automatyzacji w generowaniu prototypów lub szkieletów aplikacji intranetowych i internetowych systemów i środowisk bazodanowych oraz schematów scentralizowanych lub rozproszonych baz danych. Z konieczności będzie to analiza bardzo zwięzła.

Metodyki modelowania i projektowania aplikacji wielowarstwowych stosowane w kilku najnowszych, zaawansowanych technologicznie systemach projektowych i systemach CASE nowej generacji, dość istotnie różnią się od klasycznych metodyk projektowych i klasycznych systemów CASE – np. niektóre z nich posiadają możliwości modelowania przepływu pracy za pomocą metodologii Workflow.

Z przeprowadzonych badań studialnych wynika, że największe możliwości posiadają cztery następujące systemy projektowe: Power Designer 6.0 firmy SYBASE, Select Systems Engineer firmy SELECT (rozwiniecie produktu firmy Learmonth & Burchett Management Systems), Oracle Designer (Designer/2000 1.3) firmy ORACLE oraz Select Enterprise firmy SELECT.

W celu porównania ekspresji projektowej tych systemów, zbadano je [6] pod względem: rodzaju modelowania, metodologii modelowania (modeli) procesów biznesowych i funkcji, wykorzystywania paradygmatu obiektowego do modelowania, stosowania modelowania zachowania (dynamiki), metodologii modelowania danych, możliwości modelowania interfejsu użytkownika, ekspresji w normalizacji związków w relacjach bazodanowych, możliwości generowania szkieletowych i szczegółowych baz danych, możliwości generowania reguł na danych, generowania aplikacji łącznie z interfejsem użytkownika, wsparcia dla aplikacji wielowarstwowych i aplikacji klient-serwer, możliwości tworzenia repozytoriów projektów, automatyzacji dokumentowania i raportowania, możliwości symulowania zachowania projektowanych systemów, wymagań systemowych, wad i zalet, dostępności wersji testowych dla celów badawczych.

Przestudiowano też dość skape informacje nt. systemów: (1) System Architect firmy Popokins Systems & Software do modelowania danych, procesowego i obiektowego (system obsługuje różne metodologie i notacje), oraz (2) Visible Analyst Workbench firmy Visible Systems do modelowania procesowego i obiektowego. Bardziej szczegółowe omówienie wyników tych badań wykracza poza wyznaczone ramy niniejszej pracy.

2.1. POWER DESIGNER 6.0

Jest to oprogramowanie firmy SYBASE [21,23,47]. Modelowanie systemów bazodanowych, które zapewnia system Power Designer 6.0 można określić jako modelowanie bazodanowe. Obejmuje ono modelowanie procesów biznesowych i funkcji oraz tworzenie diagramów przepływu danych *Data Flow Diagram* (DFD) za pomocą metodologii Yourdona, SSADM oraz Gena-Sarsona. Można również tworzyć macierze procesów i danych CRUD Matrix. Za pomocą tych środków programowo-projektowych można definiować reguły biznesowe dla procesów, składnic danych, obiektów terminalnych oraz przepływów danych.

W zakresie modelowania obiektowego system Power Designer obiektowy odpowiednik DFD, zwany OFM (*Object Functional Model*). OFM pochodzi z metodologii obiektowej OMT, której jest składnikiem.

Power Designer 6.0 nie posiada wsparcia projektowego do modelowania zachowania systemów, czyli ich dynamiki.

Modelowanie danych oparte jest na tworzeniu modelu *Conceptual Data Model* (CDM) wg notacji opartej na metodologii Martina tworzenia modeli ERD (*Entity*

Relationship Diagram). Można też tworzyć model fizyczny bazy danych – *Physical Data Model*. Oba te modele są automatycznie synchronizowane. Istnieją duże możliwości tworzenia bardzo precyzyjnych reguł danych.

System Power Designer nie posiada wsparcia do projektowania i tworzenia interfejsu systemów bazodanowych.

Proces tworzenia modeli zbiorów encji pozwala na otrzymanie relacji znormalizowanych do trzeciej postaci normalnej (3NF).

Utworzony schemat bazy danych pozwala na generowanie bazy danych na poziomie kodu w języku SQL (*Structured Query Language*) na ponad 30 dostępnych platform systemowych i bazodanowych. Można zastosować podejście rewersowe do generowania schematu bazy. Fizyczną bazę danych można również wygenerować poprzez sterowniki ODBC (*Open Database Connectivity*) w trybie rewersowym.

Generowanie reguł danych można przeprowadzać w sposób rewersowy poprzez generowanie i zarządzanie wyzwalaczami oraz procedurami wbudowanymi w język SQL.

W Power Designer istnieją dość rozbudowane możliwości generowania aplikacji i ich interfejsu. System generuje oprogramowanie klienta w oparciu o fizyczny model danych oraz szablony obiektów aplikacyjnych za pomocą pakietów: PowerBuilder 4 i 5 [47], Visual Basic 3, 4 i 5 [9], Delphi 2, Optima++ 1.5, WWW dla NetImpact Dynamo. W trybie rewersowym jest to możliwe jedynie dla PowerBuilder. Tworzona aplikacja można posiadać tzw. atrybuty rozszerzone, ale dotyczy to jedynie aplikacji tworzonych dla PowerBuilder, Progress, Uniface, PowerHouse, Axiant, NSDK.

W zakresie tworzenia aplikacji wielowarstwowych system Power Designer oferuje wsparcie do tworzenia aplikacji dwuwarstwowych.

Baza diagramów, czyli tzw. repozytorium projektowo-implementacyjne, jest przechowywana na serwerze Sybase SQL Anywhere [49]. Istnieją możliwości uzyskiwania wielodostępu do tej bazy z określaniem praw do zapisów w bazie dla poszczególnych użytkowników uczestniczących w pracy grupowej nad projektem. Zarządzanie bazą diagramów jest realizowane za pomocą oprogramowania MetaWorks.

System posiada wbudowane funkcje automatycznego dokumentowania i raportowania o projekcie. Raporty są generowane w oparciu o własne szablony i można je eksportować do formatów plików edytorów MS Word, WordPerfect oraz PageMaker.

Power Designer nie pozwala na prowadzenie badań symulacyjnych zachowania tworzonego systemu. Brak jest też ścisłej integracji między elementami pakietu. Wymagania sprzętowe stacji roboczej, na której można realizować projekt nie są wygórowane – o ile nie wykorzystuje się składnika WarehouseArchitect projektowania hurtowni danych. Wystarczy komputer z systemem operacyjnym Windows 3.1/95/NT, 12 MB RAM oraz 55 MB pamięci dyskowej.

Zalety Power Designer to bardzo dobrej jakości modelowanie danych oraz powiązania i możliwości współdziałania z wieloma systemami DBMS.

Do nielicznych wad tego systemu projektowego można zaliczyć dość małą ekspresję modelowania procesów (możliwe jest tylko modelowanie DFD) oraz tworzenie aplikacji w trybie Reverse Engineering w trybie natywnym – jedynie na potrzeby systemu PowerBuilder.

W Internecie są dostępne wersje testowe pakietu zawierające następujące elementy: ProcessAnalyst, DataArchitect, AppModeler, WarehouseArchitect, streowniki ODBC. Oprogramowanie testowe posiada zablokowany zapis diagramów do pamięci dyskowej.

2.2. SELECT SYSTEMS ENGINEER (SELECT SE)

Producentem tego systemu jest firma SELECT [24]. System jest rozwinięciem produktu firmy Learmonth & Burchett Management Systems. Modelowanie jakie system ten oferuje można określić jako modelowanie danowo-procesowe i obiektowe.

System posiada spore możliwości modelowania procesów biznesowych i funkcji. Można posługiwać się diagramami DFD, diagramami hierarchii procesów i diagramami zależności procesów, a także wykorzystywać diagramy hierarchii modułów. Projektanci mają też do dyspozycji narzędzie do projektowania transakcji oraz szereg innych funkcji.

Funkcjonalności do projektowania obiektowego i modelowania zachowania systemu – czyli jego dynamiki – są bardzo duże i posiadają wielką siłę ekspresji. Z pewnością zasługują na osobne, obszernie omówienie. Można tu jedynie stwierdzić, że system Select Systems Engineer – podobnie jak szereg innych produktów firmy SELECT – jest nakierowany na projektowanie obiektowe.

W zakresie modelowania danych Select Systems Engineer oferuje tworzenie diagramów ERD, diagramów historii życia encji oraz tworzenie modułów z regułami danych.

Do modelowania interfejsu wykorzystuje się specjalizowaną metodologię obiektową StyleGuide.

Generowanie schematu bazy danych odbywa się za pomocą generowania kodu w języku SQL. Generator umożliwia przeprowadzenie tego procesu dla schematów przeznaczonych na ponad 10 platform systemowo-bazodanowych, m.in. na platformy: Sybase, Oracle, Informix, MS SQL Server. Proces generacji można też przeprowadzić w sposób rewersowy.

Za pomocą generowania wyzwalaczy oraz procedur wbudowanych w języku SQL można generować reguły danych. Możliwe jest to jednak tylko w trybie rewersowym.

Generowanie aplikacji i interfejsu ma na celu zbudowanie aplikacji klientów. Można generować aplikacje klienckie oraz tzw. aplikacje terminal/serwer. Aplikacje klientów można generować w sposób rewersowy na platformy projektowe PowerBuilder oraz Visual Basic.

Szczególnie rozbudowane i ekspresywne są funkcje wspierające projektowanie aplikacji wielowarstwowych klient-serwer. Pakiet Select Systems Engineer pozwala projektować i wytwarzać aplikacje dwuwarstwowe i trzywarstwowe. Istnieją mechanizmy generowania aplikacji serwerowych za pomocą narzędzi Entera oraz RPC Painter.

Repozytorium diagramów w postaci bazy danych może być rozproszone na kilku serwerach. Funkcje eksportowania i importowania diagramów są zrealizowane w specjalizowanym standardzie CDIF. Do bazy danych zawierającej repozytorium można uzyskiwać wielodostęp z określaniem praw użytkowników.

Funkcje automatycznej dokumentacji i generowania raportów są realizowane przez oddzielny moduł, który generuje również dokumentację projektową. Moduł jest zintegrowany z aplikacjami biurowymi Windows poprzez mechanizm OLE.

Select Systems Engineer posiada możliwości symulacji zachowania projektowanego systemu. Do realizacji tej funkcji wykorzystuje się animowane prototypy aplikacji OLTP.

Oprogramowanie projektowe zarządza bazą diagramów za pomocą modułu Object Manager i posiada bogate możliwości wspomagania testowania projektowanej aplikacji. Select Systems Engineer znacznie rozszerza metodyki projektowe stosując nowe obiekty projektowe.

Wymagania sprzętowe dla stacji roboczych nie są zbyt ostre. Jedynym wyjątkiem jest spore zapotrzebowanie na przestrzeń dyskową. System może pracować na stacji roboczej pod systemem operacyjnym Windows 3.1/95/NT, z pamięcią operacyjną 12 MB RAM i wymaga 300 MB przestrzeni na dysku twardym.

Reasumując, do głównych zalet systemu Select Systems Engineer należą: bogate modelowanie danych, skalowalność produkowanych modeli, wysokiej jakości modelowanie procesów z automatyczną synchronizacją diagramów, możliwość generowania aplikacji serwerowych, oraz – co zasługuje na szczególne podkreślenie – możliwość wyboru cyklu projektowego oraz zasad projektowania.

Za niewielką wadę systemu projektowego Select Systems Engineer można uznać wysoki stopień jego złożoności i komplikacji. Niedostępna jest również żadna jego wersja testowa.

2.3. OPROGRAMOWANIE ORACLE DESIGNER (DESIGNER/2000) FIRMY ORACLE

Scharakteryzujemy tu możliwości funkcjonalne wersji 2.1 [26, 39, 43, 44] tego oprogramowania. Jest ono bardzo solidne i wszechstronnie sprawdzone w praktyce projektowej. Można je określić jako narzędzie projektowania danowo-procesowego. System ten jest implementacją oraclowej metodologii projektowej Oracle CASE Method [2, 3].

W zakresie modelowania procesów biznesowych i funkcji system stosuje diagramy Data Flow Diagram, diagramy struktury modułów STC (*Structure Charts*) oraz diagramy hierarchii funkcji. Dodatkowo projektant ma do dyspozycji funkcję budowanie macierzy procesów i danych CRUD Matrix. Można również projektować transakcje. System posiada w tym zakresie szereg innych możliwości. Oprogramowanie Oracle Designer/2000 posiada też niewielkie możliwości modelowania zachowania (dynamiki) projektowanego systemu.

Modelowanie danych oferowane przez Oracle Designer/2000 opiera się na wykorzystywaniu diagramów ERD, tzw. diagramów z regułami danych oraz na modelowaniu rozproszenia danych.

Funkcję modelowania interfejsu aplikacji oparto zawarto w trzech podsystemach: modelowania formularzy – *Oracle Forms*, modelowania raportów – *Oracle Reports* oraz w module projektowania menu.

Projekt schematu bazy danych normalizowany jest do postaci 3NF. Generowanie schematu bazy danych odbywa się za pomocą generowania kodu w SQL dla wersji ANSI SQL 92 tego języka oraz – w trybie rewersowym – z baz Oracle 7. Istnieją możliwości wygenerowania schematu homogenicznej bazy rozproszonej Oracle.

W zakresie generowania reguł danych system projektowy Oracle Designer/2000 oferuje generowanie wyzwalaczy oraz procedur wbudowanych w języku SQL – lecz tylko w trybie Reverse Engineering.

Generowanie aplikacji klientów odbywa się w oparciu o tzw. szablony i wzorce. Można tego dokonać w trybie rewersowym za pomocą narzędzia Oracle Developer 2000 (2.0) [26], a także generując aplikacje dla Visual Basic 4, WWW lub dla Oracle Web Server.

Oracle Designer/2000 posiada wsparcie do projektowania i prototypowania aplikacji dwuwarstwowych klient-serwer.

Repozytorium diagramów zapewnia funkcje wielodostępu z określaniem praw dla użytkowników. Elementy bazy diagramów mogą być dzielone między wiele projektów. Import i eksport elementów diagramów wykonuje się za pomocą modułu Oracle CASE Exchange.

W pakiecie Oracle Designer/2000 zaimplementowano tworzenie wielu rodzajów standardowych raportów projektowych. Ich tworzenie zostało zintegrowane z aplikacjami biurowymi Windows poprzez mechanizm OLE.

Zapewniono ekspresywną symulację zachowania projektowanego systemu. Symulacja jest realizowana za pomocą animacji modeli procesów z wykorzystaniem multimedialnych.

Oprogramowanie systemu Oracle Designer/2000 zintegrowano z innymi narzędziami CASE oraz aplikacjami poprzez interfejs API.

Oracle Designer/2000 może z powodzeniem funkcjonować na stacjach roboczych Windows 95/NT oraz na serwerach platformy operacyjnej Windows NT Server.

System projektowy Oracle Designer/2000 posiada bogate modelowanie danych i bardzo dobre modelowanie procesów z automatyczną synchronizacją diagramów. Stanowi to jego główną zaletę. Do innych zalet należy zaliczyć ścisłą integrację oprogramowania Oracle Designer/2000 z pakietem Oracle Developer 2000 w zakresie generowania aplikacji oraz z oprogramowaniem serwerowym Oracle Server 7 i 8 [1].

Jednocześnie, jako wadę należy wskazać to, że Oracle Designer/2000 współpracuje dobrze jedynie z narzędziami firmy Oracle [22, 26, 45, 46].

W Internecie można znaleźć wersje próbne i Oracle Developer 2000 (2.0) [26].

2.4. SELECT ENTERPRISE

Jest to oprogramowanie CASE opracowane przez firmę SELECT [25, 49]. Modelowanie jakie mamy tu do dyspozycji można nazwać modelowaniem obiektowo-procesowym. Scharakteryzujemy krótko wersję 5.1 tego systemu projektowo-programistycznego.

Do modelowania procesów i funkcji system posiada możliwości budowania diagramów hierarchii procesów Process Hierarchy Diagrams, które pochodzą z metodologii obiektowej OMT [51]. Można również budować diagramy zależności procesów, tzw. Process Thread Diagrams. System Select Enterprise 5.1 posiada też możliwość modelowania procesów za pomocą diagramu przypadków użycia Use Case Diagram pochodzącego z metodologii Jacobsona [15, 51]. Do Select Enterprise wbudowano funkcjonalności tworzenia diagramów sekwencji funkcji – Sequence Diagram oraz ich odmiany – Collaboration Diagram. W Select Enterprise mamy zatem dużą różnorodność i wielką ekspresywność w tworzeniu modeli procesowych i funkcyjnych przyszłego systemu.

Modelowanie obiektowe oparto na wykorzystywaniu diagramu klas Class Diagram zaczerpniętego z metodologii OMT.

Zachowanie systemu modeluje się w Select Enterprise za pomocą diagramów stanu obiektów – tzw. State Diagram.

Do wspomaganie modelowania danych zastosowano diagramy stanu (State Diagram) – które są rozszerzeniem w stosunku do metodologii OMT. Rozszerzenie polega na tworzeniu diagramu w postaci relacji. Istnieje możliwość automatycznego generowania schematu bazy danych na podstawie diagramu klas. Brak jest mechanizmów do automatyzacji procesu normalizacji relacji bazodanowych.

Modelowanie interfejsu aplikacji realizowane jest za pomocą diagramów zaadoptowanych z metodologii obiektowej OMT.

Generowanie schematu bazy danych odbywa się za pomocą generowania kodu w języku SQL dla standardów ANSI SQL 92, MS SQL Server oraz Sybase. Select Enterprise umożliwia też rewersowe generowanie modelu danych dla platformy projektowej Select Enterprise. Można też kod ten generować w formacie ERX, co

możliwe jest dzięki rewersowej współpracy np. z pakietami Power Designer lub ERWin. Generowane przez Select Enterprise reguły danych są generowane podczas generowania aplikacji i w kodzie aplikacji są one również umieszczane.

Aplikację i interfejs generuje się w języku C++ w trybie Reverse Engineering, w języku Java, według specyfikacji IDL, w trybie rewersowym dla środowisk Power Builder oraz w Visual Basic. Możliwe jest rewersowe generowanie aplikacji dla Delphi oraz w języku SmallTalk.

Jako wsparcie do projektowania i tworzenia aplikacji wielowarstwowych klient-serwer w Select Enterprise projektant ma do dyspozycji rozszerzenie metodyki OMT w postaci metodyki SELECT. W systemie projektowym zawarto również narzędzie Prospective do projektowania aplikacji trzywarstwowych.

Repozytorium projektowe w postaci bazy diagramów ma charakter otwartej przechowalni diagramów. Istnieją możliwości organizowania wielodostępu z określaniem praw użytkowników. Zewnętrzny dostęp do repozytorium projektowego jest możliwy dzięki mechanizmowi OLE Automation. Można też wykonywać eksport i import modelu danych w formacie ERX.

System projektowy posiada możliwości automatycznego dokumentowania i raportowania w oparciu o wykorzystanie szablonów edytora MS Word, przy czym raporty mogą też być dowolnie projektowane.

Symulacja zachowania projektowanego systemu jest realizowana na podstawie diagramów *Sequence Diagrams*.

Wszelkie tworzone w projekcie diagramy są ze sobą ściśle synchronizowane a projektant ma dyspozycji pomoce metodyczne w trybie on-line.

Wymagania na zasoby stacji roboczych do projektowania są skromne. Wystarczające są stacje z systemem Windows 95/NT oraz wyposażeniem sprzętowym 16 MB pamięci RAM i przestrzenią dyskową około 40 MB.

Cenną zaletą systemu projektowego Select Enterprise jest możliwość obiektowego projektowania procesów i reguł biznesowych przystosowanych do architektury trzywarstwowej klient-serwer.

Wadą Select Enterprise jest stosunkowo mało ekspresywne modelowanie danych polegające na braku możliwości definiowania atrybutów dla powiązań oraz brak automatyzacji normalizacji.

W Internecie dostępne są próbne elementy pakietu Select Enterprise. Można zbadać możliwości pakietów Enterprise Modeller za pomocą Enterprise Modeller Demo, możliwości pakietów: generatora C++ Generator, C++ Reverse Engineering, Document Generator.

Ograniczenia wersji przeznaczonych do badań poznawczych polegają na ograniczeniu wykorzystywania ich możliwości do najwyżej dwóch projektów a ograniczenie dodatkowe polega na umożliwieniu utworzenia maksimum jednego diagramu każdego rodzaju w każdym z dwóch projektów.

3. STANDARDY KOMUNIKACYJNE I INTERFEJSY DOSTĘPU DO BAZ DANYCH W SIECIACH INTRANETOWYCH I W INTERNECIE

Ważnym aspektem technologii projektowych i programistycznych do projektowania aplikacji wielowarstwowych i zintegrowanych sieciowo systemów bazodanowych są standardy komunikacyjne, standardy interfejsów dostępu do baz danych w sieciach intranetowych i w Internecie, oraz narzędzia do wspomagania projektowania i implementacji systemów spełniających te standardy.

Obszerność tych standardów, wynikająca ze stosunkowo dużego stopnia komplikacji objętych nimi mechanizmów, a w ślad za tym duża różnorodność oprogramowania narzędziowego do projektowania i implementacji standardowych bazodanowych mechanizmów komunikacyjnych, uniemożliwia dokonanie w niniejszej pracy nawet pobieżnego ich przeglądu. Omówieniu tych standardów musiała by być osobna, obszerna praca.

W ramach wykonanych przez autorów badań przeanalizowano szczegółowo standardy wybranych mechanizmów komunikacyjnych stosowanych w stosie protokołowym TCP/IP [7]. Do badań wzięto również standardy dotyczące współpracy obiektów – COM i OLE. Dość szczegółowo zbadano standardy obiektów rozproszonych DCOM [13] oraz CORBA [5, 17, 19]. Zbadano też standard komunikacji obiektowej ActiveX [13]. Bardzo szczegółowo zbadano też mechanizmy wykorzystywane do prototypowania i implementacji mechanizmu zdalnego wywoływania procedur RPC [18].

Na potrzeby niniejszej pracy dokonano oceny, czy wymienione powyżej standardy komunikacyjne są – i w jakim stopniu – uwzględniane w zbadanych technologiach projektowych i systemach CASE oraz, czy są one uwzględnione w badanych technologiach, środowiskach programistycznych i oprogramowaniu narzędziowym do tworzenia systemów bazodanowych i aplikacji wielowarstwowych integrowanych w sieciach intranetowych i Internecie. Wybrane dane na ten temat zamieszczono w treści rozdziałów 3, 4 oraz 5 niniejszego artykułu. W rozdziale 5 szczególnie zaakcentowano zagadnienia zgodności ze standardami projektowanych i prototypowanych bazodanowych interfejsów aplikacyjnych.

4. TECHNOLOGIE I NARZĘDZIA PROGRAMISTYCZNE DO PROTOTYPOWANIA I BUDOWY APLIKACJI WIELOWARSTWOWYCH I SIECIOWO ZINTEGROWANYCH BAZ DANYCH

W praktyce projektowej i programistycznej i projektowej wykorzystuje się wiele pakietów i środowisk prototypowania i wytwarzania aplikacji wielowarstwowych i sieciowo zintegrowanych baz danych. Trudno jednoznacznie stwierdzić, które z nich są najbardziej przydatne, wygodne i wydajne. Dlatego poniżej prezentujemy krótko możliwości takiego oprogramowania na dwóch jego przykładowych reprezentantach – tzn. pakietu programistycznego Borland Delphi 3 Client/Server z technologią MIDAS, oraz pakietu MS Transaction Server.

4.1. BORLAND DELPHI 3 CLIENT/SERVER SUITE I TECHNOLOGIA MIDAS

Borland Delphi jest oprogramowaniem, które służy do tworzenia aplikacji uniwersalnych oraz bazodanowych pracujących w środowisku systemów Windows 95/NT. W skład pakietu wchodzi również narzędzia oraz technologie pozwalające tworzyć systemy relacyjnych baz danych we wszystkich architekturach, tzn. scentralizowanej, z serwerem plików, architekturach klient-serwer – dwuwarstwowej oraz wielowarstwowej. Nie można natomiast tworzyć systemów baz danych w architekturze host-terminal. Dzięki tym możliwościom Delphi zaliczana jest czasami do systemów zarządzania rozproszonymi relacyjnymi bazami danych.

Obecnie przedstawimy zwięźle budowę i własności funkcjonalne oprogramowania middleware MIDAS (Multi-tier Distributed Application Services Suite) zawartego w pakiecie Delphi. Za pomocą tego oprogramowania można tworzyć aplikacje wielowarstwowe. Technologia MIDAS jest oryginalną koncepcją działu Open Software firmy Borland zajmującego się technologiami systemów otwartych. MIDAS zawarto w pakiecie Delphi 3 Client/Server Suite [12] oraz w jego wersjach, które zawierają Client/Server Suite (np. Delphi 4 Client/Server [20]). Oprogramowanie to jest też dołączane do innych pakietów oprogramowania narzędziowego firmy Borland [38], np. do JBuilder 1, IntraBuilder 1.5, C++ Builder 2. Firma Borland stworzyła również technologię middleware Entera [50], której produkty wchodzi w skład m.in. pakietu Delphi 3 Enterprise. Entera ma więcej funkcjonalności typu middleware oraz wspiera kilka różnych systemów operacyjnych, tzn. Windows NT, Sun Solaris, HP-UX, IBM AIX.

4.1.1. PODSTAWOWE INFORMACJE O DELPHI

Pakiet Delphi można zaliczyć do oprogramowania narzędziowego typu RAD (*Rapid Application Development*), ponieważ posiada obiektową (wizualną) bibliotekę komponentów VCL (*Visual Component Library*). Komponenty reprezentują elementy sterujące aplikacji – przyciski, pola edycyjne do wprowadzania tekstu, ale również takie komponenty strukturalno-funkcjonalne systemu bazy danych jak tabela, zapytanie, procedura zdalna i wiele innych. Oprócz tego Delphi umożliwia tworzenie komponentów DCOM oraz ActiveX.

Własność wizualności komponentów pozwala tworzyć za pomocą myszki elementy aplikacji i automatycznie wbudowywać je w oknach aplikacji. Delphi wykorzystuje jako obiektowy język programowania język Object Pascal. W części proceduralnej jest on zgodny z językiem Turbo Pascal. W części obiektowej można definiować własne typy obiektowe, stosować hermetyzację, dziedziczenie, polimorfizm. Pewną nowością technologii obiektowej w Object Pascal w stosunku do innych języków obiektowych jest możliwość definiowania właściwości oraz obsługa zdarzeń.

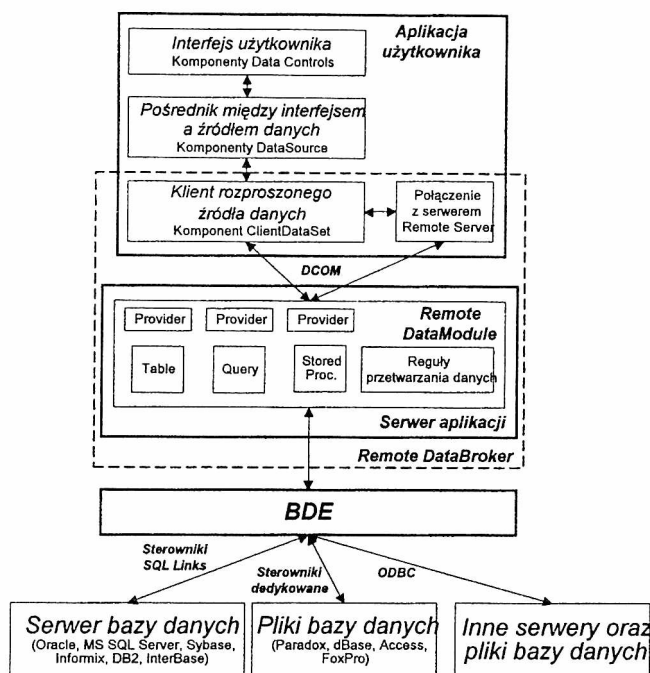
4.1.2. ARCHITEKTURY SYSTEMÓW BAZ DANYCH W DELPHI

W Delphi mamy wiele możliwości tworzenia aplikacji i systemów bazodanowych. Do tworzenia aplikacji używamy pewnych uniwersalnych elementów – komponentów do budowy aplikacji. Komponenty te możemy podzielić na trzy kategorie:

- **Data Controls** — elementy sterujące do tworzenia interfejsu użytkownika, wyświetlają dane pobierane z komponentów kategorii DataAccess, np. DBGrid (arkusz do przeglądania, edycji, dodawania oraz usuwania rekordów), DBEdit (edycja pojedynczego pola w bieżącym rekordzie tabeli).
- **DataSource** — komponent pośredniczący w wymianie danych między komponentami interfejsu użytkownika, a jednym ze źródeł danych.
- **DataSet** — źródła danych, czyli komponenty dostępu do danych, np. Table (dostęp do pojedynczej tabeli, operowanie na kursorze, wyszukiwanie danych, filtrowanie itd.), Query (zadawanie zapytań SQL), StoredProc (wywoływanie procedur zdalnych na serwerze bazy danych), ClientDataSet (klient rozproszonego źródła danych). Otwarta architektura komponentów DataSet pozwala używać identycznych komponentów Data Controls z różnymi źródłami danych.

Na rysunku 1 przedstawiono, możliwe do wytworzenia środkami pakietu Delphi, konfiguracje systemów bazodanowych. Aby otrzymać odpowiednią architekturę systemu, musimy go wytwarzać zgodnie z procedurą wyznaczającą pewną ścieżkę postępowania. Najpierw tworzymy interfejs użytkownika a następnie zajmujemy się komponentem DataSource, po czym następuje rozwidlenie na ścieżki realizacji następujących architektur:

- architektury scentralizowanej z serwerem plików tu używamy komponentów DataSet (Table, Query), BDE, plików bazy danych,
- architektury klient-serwer dwuwarstwowej oraz dwu-i-półwarstwowej przy użyciu komponentów DataSet (Query, StoredProc), BDE, serwer bazy danych,
- architektury wielowarstwowej – używamy tu komponentów ClientDataSet, serwerów aplikacji, serwera bazy danych.



Rys. 1. Warianty strukturalno-funkcyjne architektur systemów bazodanowych w środowisku Delphi

Fig. 1. Structure-functional variants of database systems' architectures in the Delphi environment

Borland Database Engine (BDE) jest modulem zarządzania relacyjną bazą danych z tzw. silnikiem bazy danych. BDE jest pośrednikiem między komponentami dostępu bazą danych. Baza danych może mieć postać: plików na serwerze plików, plików dysku na lokalnym, serwera bazy danych. BDE posiada następujące możliwości obsługi bazy danych:

- poprzez sterowniki dedykowane do obsługi plików bazy danych w różnych formatach,
- może współpracować ze sterownikami SQL Links do komunikacji z serwerami bazy danych,

- obsługuje dowolne bazy danych przez interfejs ODBC oraz odpowiedni sterownik ODBC.

Na rysunku 1 przedstawiono warianty strukturalno-funkcjonalne trójwarstwowych architektur systemów bazodanowych możliwe do wykorzystania w środowisku Delphi.

4.1.3. MIDDLEWARE MIDAS

MIDAS jest włączony do pakietu Delphi 3 oraz Delphi 4 i stanowi zestaw komponentów, serwerów i technologii do tworzenia serwerów aplikacji. Podstawą są możliwości funkcjonalne trzech brokerów:

- Remote DataBroker – realizujący rozproszony dostęp do danych baz danych,
- Business ObjectBroker – do zarządzania wieloma serwerami aplikacji,
- ConstraintBroker – realizujący automatyczną dystrybucję reguł integralności danych do aplikacji klientów.

Do utworzenia najprostszej aplikacji w architekturze trójwarstwowej wystarczy użyć jedynie Remote DataBroker. Składa się on z trzech oddzielnych części – klienta, środków komunikacji oraz serwera. Klient znajduje się w aplikacji klienta, a serwer w serwerze aplikacji.

Część stanowiąca klienta składa się z dwóch komponentów VCL: (1a) ClientDataSet oraz (1b) RemoteServer. RemoteServer realizuje nawiązanie i utrzymuje połączenia z serwerem aplikacji oraz pobiera listę komponentów dostarczających metody dla klientów.

Komponent ClientDataSet umożliwia zdalne wywoływanie metod w serwerze aplikacji, dzięki czemu komponent ten może pobierać lub wysyłać dane do serwera.

Wywoływanie metod (czyli komunikacja) odbywa się zgodnie ze standardem DCOM.

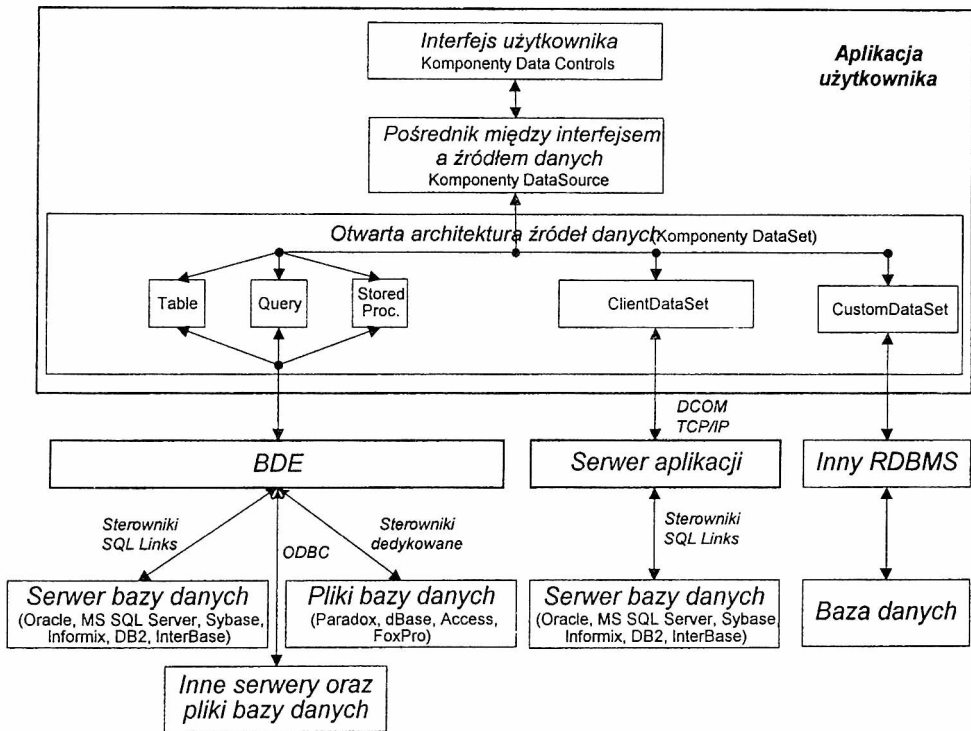
Część Remote DataBroker znajdująca się w serwerze aplikacji to *Remote DataModule*, czyli zdalny moduł danych, który zawiera: komponent Provider (dostawcy usługi), komponenty źródła danych (Query, StoredProc) oraz reguły przetwarzania danych. Moduł Remote DataModule jest serwerem DCOM udostępniającym interfejs dla klientów.

Metody serwera – operując na danych pochodzących z różnych źródeł danych – realizują przetwarzanie danych. Komponenty źródła danych poprzez BDE obsługują bazę danych na serwerze bazy lub bazy w postaci lokalnych plików (podobnie jak w przypadku architektury dwuwarstwowej i dwu-i-półwarstwowej).

Na rysunku 2 przedstawiono komponenty strukturalno-funkcjonalne wymagane do utworzenia w środowisku Delphi prostego systemu bazodanowego o architekturze trójwarstwowej. Dla uproszczenia przedstawiono tylko jeden serwer aplikacji, ale pojedyncza aplikacja klienta może również komunikować się z wieloma serwerami aplikacji.

Za pomocą Business ObjectBroker można zarządzać aplikacją wielowarstwową oraz równoważyć obciążenia serwerów. ObjectBroker zarządza globalnym rejestrem zawierającym spis serwerów aplikacji dostępnych w systemie. W rejestrze tym aplikacja klienta poszukuje serwera oferującego potrzebny komponent lub usługę.

ConstraintBroker umożliwia scentralizowanie zarządzania bazami danych. Umożliwia on utworzenie w serwerze aplikacji repozytorium reguł sprawdzania integralności i poprawności danych baz oraz przesyłanie ich do aplikacji klientów. Reguły integralności ładowane są z serwerów baz danych i przesyłane do klienta przy pierwszym połączeniu z serwerem aplikacji. Przy takim wykorzystaniu techniki MIDAS pierwsza weryfikacja wprowadzonych danych odbywa się w aplikacji klienta i dzięki temu nie jest potrzebne dwukrotne kodowanie tych samych reguł w Pascalu i SQL.



Rys. 2. Komponenty strukturalno-funkcjonalne prostego systemu bazodanowego o architekturze trójwarstwowej w środowisku Delphi

Fig. 2. Structure-functional components of simple three-layer database system architecture in the Delphi environment

4.2. PAKIET MS TRANSACTION SERVER

Przykładem oprogramowania do implementacji dobrej jakości bazodanowych aplikacji dwuwarstwowych i wielowarstwowych udostępniających bazy danych w intranetach i w Internecie jest pakiet programistyczny Transaction Server 3.0 firmy Microsoft [14]. Oprogramowanie to posiada stosunkowo duże możliwości funkcjonalne i jest bardzo przydatne do tworzenia infrastruktury systemów bazodanowych ogólnego przeznaczenia. Posiada rozbudowane możliwości implementacji i rozpraszania aplikacji pomiędzy różne platformy serwerów. Oferuje mechanizmy komunikacyjne w standardzie DCOM.

Transaction Server 3.0 dobrze nadaje się do tworzenia łatwo skalowalnych i modyfikowalnych aplikacji wielowarstwowych. Tworzone za pomocą tego pakietu warstwy wewnętrzne (pośrednie) aplikacji wielowarstwowych są mało wymagające pod względem zapotrzebowania na pamięć i moc obliczeniową wymaganą w warstwach zewnętrznych wielowarstwowych aplikacji bazodanowych. Pakiet stanowi wysokiej jakości środowisko programistyczne do implementacji większości infrastruktury niezbędnej do uruchamiania oprogramowania biznesowego oraz uwalnia programistów od rozwiązywania zagadnień i wytwarzania oprogramowania do zarządzania: komunikacją, bezpieczeństwem, katalogami, procesami, wątkami, przyłączaniem do baz danych oraz kontrolą ich spójności w systemach bazodanowych przeznaczonych do pracy w środowisku zintegrowanym w bazach intranetowych i w Internecie.

5. TECHNOLOGIE I ŚRODOWISKA PROGRAMISTYCZNE DO INTEGRACJI BAZ DANYCH Z INTERNETEM I ŚRODOWISKIEM WWW

Bardzo znaczący przyrost ilościowy i jakościowy komercyjnych zastosowań sieci Internet spowodował znaczący postęp w rozwoju technologii projektowych i technik programistycznych nakierowanych na zintegrowanie systemów bazodanowych ze środowiskiem WWW. Technologie te są stale rozwijane i już obecnie umożliwiają projektowanie i budowę skomplikowanych scentralizowanych i rozproszonych systemów bazodanowych – nie tylko relacyjnych, ale i obiektowych – wykorzystujących możliwości funkcjonalne interaktywnych serwerów Webu. Zarówno Internet jak i sieci intranetowe stały się dobrej jakości środowiskiem dla systemów bazodanowych. Sieciowe systemy bazodanowe w WWW pod wieloma względami dorównują już systemom baz danych i systemom zarządzania bazami danych działającym w spo-

sób klasyczny w środowiskach operacyjnych i sieciowych. Całkowicie zasadne jest więc stwierdzenie, że internetowe środowisko WWW zaczyna wchłaniać bazy danych i związane z nimi zastosowania systemów bazodanowych. Za pomocą WWW można już obecnie nie tylko uzyskać dostęp do części zasobów informacyjnych klasycznych baz danych ale i zasoby te powiększać. Powszechne jest już publikowanie niektórych danych z baz danych – np. dla celów reklamy lub innych celów komercyjnych. Istnieje ponadto możliwość wprowadzania danych do baz – np. w celu zdalnego dokonania zakupów i realizacji innych usług i działań komercyjnych.

Zwykle dane przechowywane są w relacyjnych bazach danych pracujących pod kontrolą takich powszechnie stosowanych systemów zarządzania bazami danych jak: Sybase, Informix, Oracle, Progress, dBase, MS Access.

Aplikacje systemów bazodanowych realizują dostęp do danych za pomocą technologii klient-serwer. Udostępnianie użytkownikowi danych z bazy umożliwia oprogramowanie klienta działające na stacjach roboczych.

Znaczny postęp technologiczny nastąpił w wyniku zastąpienia oprogramowania klientów w rozproszonych lub scentralizowanych aplikacjach bazodanowych przeglądarkami WWW. W celu realizacji tego rodzaju dostępu konieczne jest wykorzystanie możliwości funkcjonalnych odpowiednich serwerów oraz zastosowanie oprogramowania scalającego bazę danych z internetową siecią rozległą albo siecią intranetową. Takie rozwiązanie umożliwia uniknięcie rozwiązywania licznych problemów związanych z tworzeniem klasycznego oprogramowania klient-serwer oraz wieloplatformowy dostęp do bazy danych. Ponieważ wykorzystuje się przy tym standardowe rozwiązania sieciowe, nieistotne jest na jakiej platformie sprzętowej i w jakim środowisku operacyjnym działa oprogramowanie aplikacji systemu bazodanowego.

Już obecnie istnieje dość dużo produktów w postaci obszernych i dość kompletnych pakietów oprogramowania integrujących bazy danych z sieciami intranetowymi i Internetem (źródła literaturowe i internetowe szacują ich ilość na ok. 50).

Oprócz pakietów oprogramowania integrującego oferowanego przez duże firmy, np. Microsoft [10, 13], Borland [34, 38], Netscape [36] albo Oracle [22, 28, 30, 46], oprogramowanie to dostarczają również mniejsze firmy, np. NetDynamics [38, 40] (pakiet NetDynamics), Allaire [37] (oprogramowanie ColdFusion) oraz szereg firm małych i mniej znanych.

Daje się zauważyć spore różnice w budowie oferowanych produktów oprogramowania. Poniżej postaramy się je uchwycić.

Można zauważyć, że odchodzi się już od rozwiązań z zastosowaniem statycznych stron HTML w serwerach i przeglądarkach WWW. Pewna część a nawet całość prezentowanej strony jest generowana dynamicznie przez serwer w odpowiedzi na żądanie aktualizacji przedstawianych danych. Kiedy żądanie takie nadchodzi od przeglądarki, serwer posyła zapytanie do bazy danych, której dane są prezentowane i podczas otrzymywania odpowiedzi konstruuje stronę HTML. Następnie serwer przesyła wygenerowaną stronę do przeglądarki, która zgłosiła to żądanie.

Idea udostępniania baz danych alokowanych na serwerach WWW jest znana od dość dawna, ale dopiero nowe produkty oprogramowania znacznie to ułatwiają. Do niedawna serwery obsługi żądań dostępu do baz danych wykorzystywały standard CGI. Obecnie odchodzi się od tego standardu (są wyjątki w niektórych znanych produktach) ze względu na powolność tej metody dostępu według tego standardu. Metody dostępu realizowane według CGI powodują, że serwer musi wywoływać zewnętrzny program do obsługi żądania dostępu do bazy danych, a następnie uzyskaną informację program ten musi przekazać poprzez serwer do przeglądarki. Otwiera on oraz zamyka połączenie z bazą danych, przeszukuje bazę danych i formatuje wyniki zapytań w HTML. Wydajność takiego procesu jest bardzo mała.

W pewnym, lecz niewielkim stopniu, proces ten jest usprawniony w standardzie FastCGI, ale producenci oprogramowania nie sięgają po rozwiązania tego standardu.

Wiodące firmy starają się forsować własne rozwiązania integrujące bazy danych z Internetem. Szczególny wysiłek w tym kierunku jest widoczny w strategii rozwoju produktów firmy Microsoft oraz firmy Netscape.

Oprogramowanie LiveWire firmy Netscape wykorzystuje usługi Netscape Server API (NSAPI) i jest umieszczane na serwerze z oprogramowaniem sieciowym Netscape, co uniemożliwia zainstalowanie go wraz z serwerami sieciowymi nie wykorzystującymi i nie znającymi NSAPI.

Pakiety oprogramowania Internet Database Connector (IDC), Visual InterDev oraz SQL Server firmy Microsoft natomiast zostały zbudowane za pomocą Visual InterDev API (ISAPI) i zintegrowane z firmowym pakietem oprogramowania Internet Information Server (IIS). Podobnie jak w przypadku pakietu LiveWire nie jest możliwa integracja tego oprogramowania z dowolnym serwerem Webu.

Firma Oracle także lansuje swoje rozwiązanie integracji baz danych z Internetem. Jej oprogramowanie Web Request Broker (WRB) stosuje zamiast CGI własne rozwiązanie interfejsu i działa wydajnie tylko w połączeniu z WebSerwerem firmy Oracle, chociaż umożliwia też współpracę z produktami innych firm. Firma Oracle nie dołącza wspomnianych adapterów w standardowych pakietach oprogramowania do połączenia z serwerami innych producentów.

Oprogramowanie IntraBuilder firmy Borland to produkt otwarty na współpracę z oprogramowaniem innych firm. Pozwala na współpracę nie tylko z Borland Web Server (BWS), ale również z serwerami obsługującymi Interfejs NSAPI, ISAPI oraz CGI. Takie podejście firmy Borland uniezależnia użytkownika w pewnym stopniu od produktów jednego producenta i pozwala na wybór spośród szerszej gamy produktów.

Podobną strategię tworzenia oprogramowania integrującego bazy danych z Internetem wybrała firma NetDynamics. Jej pakiet oprogramowania o tej samej nazwie pozostawia możliwość wyboru rozwiązań. Można wykorzystywać WebSite API (WSAPI), ISAPI, NSAPI i CGI, choć firma ta odradza wykorzystywanie CGI.

Poza mechanizmami komunikacji, oprogramowanie integrujące bazy danych z Internetem poszczególnych producentów różni się złożonością i stopniem komplikacji.

cji. Poniżej omawiamy krótko wymienione powyżej pakiety oprogramowania w kolejności wzrostu tych czynników.

5.1. OPROGRAMOWANIE IDC

Oprogramowanie IDC firmy Microsoft [35] jest najprostszym ze wszystkich wspomnianych wyżej produktów jeśli chodzi sposób generowania stron HTML. Wykorzystuje skrypt (.IDC) zawierający m. in.: parametry dla sterownika ODBC, nazwę użytkownika, opcjonalnie hasło oraz zdanie w języku SQL ze definicją operacji bazodanowej, plik wzorcowy (.HTX) podobny do HTML i służący do generowania stron HTML. IDC potrafi łączyć się z dowolnymi źródłami danych w standardzie ODBC, ale nie obsługuje połączeń do innych baz danych, np. Oracle.

Program IDC działa na zasadzie odwoływania się do skryptów .IDC umieszczonych na serwerze. Plik skryptów jest udostępniany w odpowiedzi na żądanie URL nadchodzące z przeglądarki. Jeśli URL kończy się na .IDC to serwer wie, że chodzi o żądanie dotyczące .IDC i przekazuje je do realizacji. Program IDC czyta z kolei skrypt i komunikuje się z bazą danych. Gdy program IDC otrzyma potrzebne dane z bazy danych, to formatuje wynik zgodnie ze wzorcem zawartym w pliku .HTX.

Program IDC działa w sposób nieskomplikowany i nadaje się do prostych zastosowań i oferuje dość prymitywne rozwiązania. Szczególnie dotkliwy jest brak wsparcia IDC dla zastosowań, które wymagają odświeżania uzyskiwanych przez przeglądarkę danych za pomocą budowania dynamicznych zapytań SQL.

Głównym ograniczeniem jest to, że do każdego zapytania musi istnieć odpowiedni plik .HTX. Ponadto brak jest możliwości pisania skryptów po stronie serwera, co oznacza, że nie można w ten sposób pokonać wspomnianych już niedogodności związanych z dynamiką odpowiedzi za pomocą dynamicznego generowania plików HTML. Ponadto zastosowana tu metoda połączeń może (choć nie musi) doprowadzić do szybkiego osiągnięcia progu ilości możliwych do obsłużenia połączeń.

5.2. PAKIET INTRABUILDER

Oprogramowanie IntraBuilder [29, 34] firmy Borland zawiera komplet narzędzi do wytworzenia w pełni funkcjonalnego serwera dostępu do baz danych. Poza narzędziami do tworzenia tych połączeń w skład tego pakietu programistycznego wchodzi Borland WebServer lub Netscape FastTrack Server oraz przeglądarka Netscape Navigator. Można też korzystać z innych serwerów Web stosujących NSAPI, ISAPI lub CGI. Dostęp do licznych baz danych uzyskuje się za pośrednictwem stosowanego również w pakiecie Delphi silnika baz danych Borland Database Engine oraz oprogramowania Borland SQL Links. Programy te zapewniają realizację dostępu do baz

danych w standardzie XBase np. dBase, oraz pracujących pod kontrolą i w formatach innych systemów DBMS, np. Paradox, Access, MS SQL Server, Borland InterBase, a także w wersji klient-serwer zapewniają dostęp do baz danych takich systemów DBMS jak Oracle, Sybase, Informix i IBM DB/2. Pakiet IntraBuilder może korzystać ze sterowników ODBC ale sterowniki te nie są dostarczane wraz z pakietem.

Integralną częścią pakietu jest oprogramowanie do projektowania aplikacji IntraBuilder Explorer. Program realizuje zarządzanie wszystkimi składnikami projektu. Wykorzystuje technikę drag&drop, co bardzo ułatwia tworzenie wzorców stron. IntraBuilder umożliwia dwukierunkowe wpływanie na realizowany projekt. Można tworzyć i modyfikować projekt zarówno za pomocą narzędzi wizualnych jak i modyfikując bezpośrednio kod źródłowy. Uzgadnianie efektów obu tych podejść następuje automatycznie i natychmiastowo.

IntraBuilder realizuje technologię projektowania zorientowanego obiektowo. Można więc tworzyć biblioteki klas, używać tych wielokrotnie i wykorzystywać dziedziczenie właściwości. Wykorzystując kreatory (ekspertów) można szybko wytworzyć szkielet aplikacji nie używając wcale języków HTML lub JavaScript. Dodatkowo istnieje możliwość wykorzystania wzorców formatek, raportów i tabel.

Aby przyspieszyć działanie, kod aplikacji znajdujący się po stronie serwera jest prekompilowany i uruchamiany na wirtualnej maszynie JavaScript Virtual Machine firmy Borland. Ciekawym rozwiązaniem jest też dodawanie do kodu standardowego fragmentu skryptu, który pozwala rozpoznać z jakiej przeglądarki korzysta użytkownik co umożliwia poprawne wyświetlanie strony niezależnie od tego, czy używa się przeglądarki Netscape Navigator czy też Internet Explorer.

IntraBuilder może również pracować w środowisku rozproszonych baz danych. W tym celu zastosowano technologię Remote Agent i wsparcie programowe dla Business ObjectBroker oraz Entry. W przeciwieństwie do IDC firmy Microsoft istnieje możliwość tworzenia skryptów w JavaScript po stronie serwera a także możliwość korzystania z apletów Javy, ActiveX Controls, OLE Automation Servers i windowsowych bibliotek DLL.

IntraBuilder jest zatem produktem dobrze nadającym się do zastosowań ogólnych, niemniej jednak część rozwiązań wymaga jeszcze dopracowania.

5. 3. OPROGRAMOWANIE LIVEWIRE

LiveWire [36] jest środowiskiem do tworzenia aplikacji sieciowych. Oferuje ono duże możliwości funkcjonalne i jest proste w użyciu. Oprogramowanie jest przeznaczone dla administratorów sieci i programistów do tworzenia aplikacji z interfejsem realizowanym na stronach WWW. Zawiera ono komplet narzędzi do tworzenia aplikacji WWW, m.in. biblioteki do realizacji połączeń z bazami danych oraz kompilator języka JavaScript. Zaletą programu jest możliwość realizacji bezpośrednich połączeń

z bazami danych Informix, Oracle oraz Sybase. Można realizować też połączenia poprzez ODBC z innymi bazami danych. Zastosowanie JavaScript – podobnie jak w pakiecie IntraBuilder – ułatwia tworzenie aplikacji, ponieważ w środowisku zawarto obiekt typu „baza danych”, który pomaga otwierać i zamykać połączenia, wyszukiwać dane i aktualizować bazę danych. Środkami języka można też łatwo odbierać informacje o formularzu HTML.

LiveWire zawiera też edytor HTML, menadżera aplikacji oraz menadżera połączeń z serwerem sieciowym. Jest on zorientowany na projektowanie obiektowe, co pozwala tworzyć własne biblioteki klas do użycia w różnych projektach. Odwołania do bazy następują przez obiekt „baza danych”, którego metody definiują sposób dostępu do danych.

Jednym z ograniczeń LiveWire jest to, że utrzymywanie połączenia z bazą danych – w celu minimalizacji dołączeń i odłączeń od bazy – w czasie trwania sesji, połączenie to musi być realizowane bez zmiany praw dostępu. W przypadku zmiany użytkownika lub praw dostępu konieczne jest rozłączenie i ponowne połączenie.

Do zalet pakietu należy posiadanie menadżera aplikacji. Pozwala on uruchamiać, zatrzymywać i debugować aplikacje napisane w JavaScript. Można też dodawać, usuwać i zmieniać konfigurację aplikacji.

Producent połączył też pakiet LiveWire z pakietem Enterprise Server 3.0, co umożliwia tworzenie skryptów po stronie serwera i dynamiczne generowanie stron HTML.

Pakiet LiveWire jest oprogramowaniem wysokiej jakości i bardzo wydajnie integrującym WWW z bazami danych.

5.4. PAKIET NETDYNAMICS

Oprogramowanie NetDynamics [40,41] oferowany przez firmę NetDynamics jest również oprogramowaniem bardzo dobrej jakości, porównywalnym z IntraBuilder firmy Borland. Pozwala łatwo tworzyć szkielety całych aplikacji. Podobnie jak IntraBuilder i LiveWire jest zorientowany obiektowo. Posiada wsparcie do projektowania obiektowego w postaci przeglądarki hierarchii obiektów oraz tzw. „inspektora obiektów”. Inspektor umożliwia generowanie kodu w Javie posiada wsparcie dla ActiveX. Mamy również możliwość korzystania z ODBC i środków programistycznych do realizacji dostępu do baz Oracle, Informix, Sybase, DB/2, oraz – podobnie jak w pakiecie IntraBuilder – możliwości pracy w środowisku rozproszonych baz danych. Kod aplikacji po stronie serwera jest prekompilowany w celu przyspieszenia jego wykonywania. Kod można uruchamiać w środowisku pakietów oprogramowania Javasoft, Java Developers Kit, Symantec Cafe i innych. NetDynamics ma wbudowany debager Javy. Umożliwia też automatyczne generowanie kodu HTML i Java dla klienta, oraz kodu Java dla serwera.

Istnieją różne wersje oprogramowania NetDynamics – w tym dla Windows NT, a także na inne platformy sprzętowo-programowe – w tym unixowe.

5.5. PAKIET OPROGRAMOWANIA INTEGRACYJNEGO COLDFUSION

Oprogramowanie ColdFusion [37] firmy Allaire wyróżnia się tym, że wykorzystuje własny specyficzny język CMFL (Cold Fusion Markup Language). Pomimo prostoty język ten jest bardzo funkcjonalny. Jakość pakietu Cold Fusion jest porównywalna do jakości pakietu LiveWire firmy Netscape. Za pomocą języka o składni prostszej niż JavaScript i Perl – ale nieco bardziej złożonej niż IDC – można łatwo implementować zaawansowane funkcje systemowe aplikacji. Język ten posiada możliwość manipulacji zmiennymi na dynamicznych stronach Web. Posiada on ponad 150 funkcji, umożliwia tworzenie tablic wielowymiarowych a także oferuje dostęp do zmiennych środowiskowych HTTP. Do niewielkich wad tego języka należy brak własnych sterowników dostępu do baz danych. Wykorzystuje on jedynie sterowniki ODBC. Cold Fusion w wersji 3.0 posiada wsparcie dla VRML, co wydaje się własnością ciekawą ale mało użyteczną. Użyteczną funkcją jest natomiast weryfikacja użytkownika za pomocą istniejących już mechanizmów zabezpieczeń.

Cold Fusion pozwala też monitorować aktywność użytkownika, np. sprawdzać, jakie strony aktualnie on przegląda. Problemy bezpieczeństwa zostały w Cold Fusion potraktowane bardzo szczególnie. Jest tam wiele mechanizmów polepszających bezpieczeństwo – od autoryzacji użytkownika poczynając, a na mechanizmach FireWall kończąc.

Cold Fusion jest oprogramowaniem jakościowo zbliżonym do pakietu LiveWire oraz NetDynamics.

5.6. ORACLE WEBSERVER

Rozwiązania użyte w pakiecie Oracle WebServer są wystarczające tam, gdzie za-inwestowano już sporo środków w bazy danych Oracle a wymaga się, aby przenieść działające aplikacje klient-serwer do środowiska sieci rozległych – np. Internetu. Wadą tego oprogramowanie jest to, że musi ono działać w środowisku DBMS Oracle 7 i 8. Oprogramowanie WebServer nie obsługuje ODBC. Nie można też sięgać do baz danych zarządzanych przez systemy DBMS innych producentów i standardów – np. standardu XBase.

Do uruchamiania aplikacji sieciowych wykorzystuje się *Web Request Broker* (WRB). WRB zarządza wewnętrznie wieloma jednoczesnymi połączeniami do bazy danych Oracle, co zwiększa wydajność.

WRB wyposażono w interfejs programisty, który pozwala do programu WebServer dołączać zewnętrzne pakiety. Pakietami mogą być albo interfejsy programowania, albo gotowe pakiety aplikacji.

WebServer wyposażono standardowo w pakiety PL/SQL (proceduralne rozszerzenie języka SQL), języka Java, oraz LiveHTML będący firmową implementacją serwera WWW firmy Oracle. Pakiet Java pozwala pisać aplikacje po stronie serwera. Można też wywoływać procedury PL/SQL z apletów Javy aby obsługiwać dostęp do baz danych. Możliwe jest przyłączenie WRB do serwerów sieciowych, ale firma Oracle nie udostępnia odpowiednich adapterów.

Zastosowanie pakietu WebServer jest odpowiednie tam, gdzie od dawna stosuje się bazy danych Oracle. Pakiet WebServer jest mało elastyczny, a jego najistotniejszą wadą jest to, że nie posiada możliwości współpracy z bazami danych i systemami DBMS innych producentów.

5.7. DOSTĘP DO SQL-OWEJ BAZY DANYCH Z APLETU JAVY

Jednym ze sposobów tworzenia aplikacji wielowarstwowych systemów bazodanowych w sieciach intranetowych i w Internecie jest realizowanie dostępu do SQL-owej bazy danych z apletu Javy.

Powodem jest to, że WWW jest bardzo użytecznym środkiem dystrybucji informacji bazodanowych, np. różnych informacji biznesowych. Podejście tradycyjne do dystrybucji informacji bazodanowych opiera się na programowaniu tzw. aplikacji wertykalnych za pomocą wbudowanego SQL albo innej techniki klasycznej. Wadą tego podejścia jest konieczność redystrybucji aplikacji każdorazowo po wprowadzeniu do niej modyfikacji. Podejście oparte na WWW wykorzystuje prosty i bardzo użyteczny model publikacji z interfejsem w HTML używanym do realizacji interfejsu użytkownika oraz centralnie utrzymywanymi programami CGI, umożliwiającymi dostęp do bazy danych i udostępniającymi dane na stronach WWW. Jest to mniej kosztowne i bardzo elastyczne ze względu na często zachodzącą konieczność redystrybucji oprogramowania aplikacyjnego warstw wewnętrznych. Ponieważ jednak protokół CGI jest zorientowany transakcyjnie, to występują problemy związane z jego ograniczonymi możliwościami (np. debugowania wytwarzanego oprogramowania).

Narzędzia programowe – np. JavaScript, HotJava [4, 32, 33] – pozwalają dokonać lokalnego przetworzenia danych przed wysłaniem zapytania do bazy danych, ale mechanizm ten nie jest odpowiedni dla dużych aplikacji. Zwykle korzysta się z modelu programowania rozproszonego Javy. Mamy wtedy możliwość połączenia zalet obu wymienionych podejść, tzn.: interaktywnego (a nawet działającego w trybie rzeczywistym) interfejsu użytkownika i niskiego kosztu rozwiązania opartego na WWW. Dodatkowo, nie jest konieczne stosowanie CGI. W nowszych wersjach Javy istnieje pakiet JAVA.SQL umożliwiający dostęp do baz danych zgodny ze standardem ODBC

poprzez sterowniki JDBC. Aby skorzystać z bazy danych wystarczy zainstalować taki sterownik.

5.8. SYSTEM WEBMAGIC

Magic jest popularnym systemem do tworzenia aplikacji bazodanowych. Aplikacje mogą działać we własnym środowisku tego systemu albo w innych środowiskach i systemach o architekturze klient-serwer. Zastosowana w pakiecie Magic technologia tworzenia aplikacji umożliwia dużą wydajność tworzenia takich aplikacji.

MagicWeb [11] jest rozszerzeniem systemu Magic i posiada możliwości tworzenia aplikacji internetowych i intranetowych współpracujących z bazami danych w środowisku WWW. Aplikacje tworzone za pomocą pakietu WebMagic mogą przetwarzać w środowisku WWW złożone transakcje i prezentować dane na stronach WWW. WebMagic umożliwia więc tworzenie sieciowo otwartych aplikacji bazodanowych. Możliwe jest tworzenie aplikacji bazodanowych do pracy w środowisku WWW w sieciach intranetowych i w Internecie z uwzględnieniem: (1) braku komunikacyjnych połączeń ciągłych, (2) pośredniej kontroli komunikacji z użytkownikiem, (3) dużej dynamiki zmian aplikacji, (4) różnorodności środowiska sprzętowego, (5) wymaganej dużej skalowalności aplikacji.

Współpraca z użytkownikiem jest realizowana poprzez przeglądarkę WWW, np. Netscape Navigator firmy Netscape lub Internet Explorer firmy Microsoft. Przeglądarka interpretuje dokument w formacie HTML wysyłany z serwera WWW i wyświetla go realizując bezpośrednią interakcję z użytkownikiem dostarczając w tym celu interfejs użytkownika.

Aplikacje i całe systemy bazodanowe z systemem zarządzania bazą danych włącznie systemów tworzone za pomocą pakietu Magic oraz rozszerzenia technologicznego WebMagic, pracujące w środowisku WWW, są bardzo elastyczne pod względem ich rozbudowy i mechanizmów strojenia w celu dostosowywania tych aplikacji do rosnącej liczby obsługiwanych stacji klienckich i użytkowników

System WebMagic posiada moduły elastycznego projektowania aplikacji WWW. Zestaw wbudowanych modułów narzędziowych obejmuje: generator Magic (Magic Development System), MagicWeb Connector, MagicWeb Broker, serwer aplikacji Magica (Magic Application Server), generator dokumentów w formacie HTML (MagicWeb HTML Generator), narzędzia do obsługi wzorców dokumentów w formacie HTML (MagicWeb Tags and Templates), moduł Web-Magic Auditor.

6. PODSUMOWANIE

Systemy bazodanowe integrowane w sieciach intranetowych i w Internecie za pomocą technologii wielowarstwowych będą miały duże znaczenie praktyczne dla unowocześnienia zasobów informacyjnych sieci intranetowych wszystkich instytucji i firm. Systemy te ciągle jeszcze są technologiczną nowością a dodatkowo brak jest dostatecznej wiedzy o narzędziach do ich projektowania oraz wystarczających doświadczeń praktycznych wymaganych przy ich tworzeniu.

W niniejszej pracy poczyniono próbę w kierunku całościowej analizy standardów i technologii integrowania baz danych z sieciami intranetowymi i Internetem. Analiza materiałów źródłowych oraz literaturowych umożliwiła z kolei dokonanie pogłębionej analizy metodologii i metodyk projektowania intranetowych i internetowych aplikacji bazodanowych w technologii klient-serwer i technologiach aplikacji wielowarstwowych.

Jednym z dodatkowych rezultatów prac było szczegółowe zbadanie transakcyjnych metod i mechanizmów podwyższania dostępności i niezawodności w rozproszonych bazach danych. Zbadano te z mechanizmów tego rodzaju, które są wbudowywane i implementowane w drugiej i innych warstwach wewnętrznych wielowarstwowych aplikacji bazodanowych w sieciach intranetowych i Internecie. Dokonano też próby sformułowania metodyki utrzymywania spójności i dostępności rozproszonych baz danych środkami administrowania rozproszonych systemów zarządzania bazami danych i systemów bazodanowych.

LITERATURA

- [1] AUSTIN D., *Poznaj Oracle 8*. MIKOM, 1998.
- [2] BARKER R., *CASE Method – Modelowanie związków encji*. WNT, Warszawa, 1996.
- [3] BARKER R., *CASE Method – Modelowanie funkcji i procesów*. WNT, Warszawa, 1996.
- [4] BERG C., *Dostęp do SQLowej bazy z apletu Javy*. Software, nr 4/97, s. 32.
- [5] BERG C., *Corba w Javie*. Software, nr 3/97, s. 40.
- [6] BERNARDYN J., GRUBER J., JASIŃSKI M., MIROŃCZUK P., JÓŹWIĄK I., *Integracja scentralizowanych i rozproszonych systemów DBMS i systemów baz danych z sieciami intranetowymi i Internetem*. Raporty Wyd. Zakł. Inform. PWr., Ser. SPR, nr 1, Wrocław, 1998.
- [7] COMER D.E., *Sieci komputerowe TCP/IP, T. 1-3, Projektowanie w trybie klient-serwer, wersja BSD*. WNT, Warszawa, 1997.
- [8] DYBIEC S., ROUSELLE P., *Fast CGI: udoskonalenia i aplikacje WWW*. Software, nr 8/97, s. 28.
- [9] EIDHEL L., BRUST A., *Using Visual Basic 5*. Platinum Edition, 1998.
- [10] FREY A., *Web-To-Database Communication with API-Based Connectivity*. NetWorld-OnLine.
- [11] GORAWSKI M., KOZIATEK A., *MagicWeb*. Software, nr 4/97, s. 32.
- [12] HENDERSON K., *Bazy danych w architekturze klient-serwer*. Robomatic, Wrocław, 1998.
- [13] JACOBS J., *Narzędzia Microsoftu do integracji baz danych z WWW*. Software, nr 9/96, s. 72.
- [14] KRAMER I.M., *Microsoft Transaction Server*. Software, nr 4/97, s. 46.

- [15] MARCINKIEWICZ J., *Metody obiektowe w analizie i projektowaniu zastosowań informatycznych*. Informatyka, nr 10/96, s. 21.
- [16] OZSU M.T., VALDURIEZ P., *Principles of distributed database systems*. Englewood Cliffs, New York, Prentice-Hall, 1991.
- [17] SIEGEL J., *CORBA. Fundamentals and programming*. OMG Group, John Wiley & Sons, Inc., Wiley Computer Publishing Group, New York, 1996.
- [18] SIPE S., *RPC w Windows 95 i Windows NT*. Software, nr 5/97, s. 25.
- [19] STAWOWSKI M., *OMG CORBA*. Software, nr 8/96, s. 86.
- [20] TEIXEIRA P., PACHECO X., *Delphi 4 – podręcznik profesjonalisty*. Helion. Gliwice, 1998.
- [21] ZIMNIAK J., CHMIELEWSKI M., *Nowe cechy PowerBuilder Enterprise 5.0*. Software, nr 9/96, s. 68.
- [22] *Building Oracle Web Sites*. Praca niesygnowana. Prentice Hall PTR, 1998.
- [23] *Dostępna dokumentacja firmowa różnych pakietów oprogramowania firm: Microsoft, Inc., Oracle, Inc., Borland, Inc., NetDynamics, Inc., ColdFusion, Inc.*
- [24] *Dokumentacja firmowa, podręczniki, opisy, wersje próbne oprogramowania firmy SELECT SYSTEMS ENGINEER firmy SELECT Software Tools, Inc.* www.selectst.com/Products.
- [25] *Dokumentacja firmowa, podręczniki, opisy, wersje próbne oprogramowania firmy SELECT Enterprise V6.0e firmy SELECT Software Tools, Inc.* www.selectst.com/Products.
- [26] *Dokumentacja firmowa, oprogramowanie demonstracyjne, opisy, podręczniki do oprogramowania firmy Oracle.* www.oracle.com/products/trial/html/trial.html.
- [27] *Informacje o Oracle WebServer.* www.oracle.com/products/websystem/webserver.
- [28] *Informacje techniczne o Oracle WebServer.* www.us.oracle.com/products/webserver.
- [29] *IntraBuilder Professional 1.0*. Internet, nr 3/97, s. 24.
- [30] *Informacje o Oracle WebServer.* www.oracle.com/products/websystem/webserver.
- [31] *Informacje techniczne o Oracle WebServer.* www.us.oracle.com/products/webserver.
- [32] *Integrating DataBase with Java via JDBC*. Praca niesygnowana. Java World, maj, 1996.
- [33] *Java i zarządzanie bazami danych*. Software, nr 7/97, s. 32.
- [34] *Ogólne informacje o pakiecie IntraBuilder.* www.borland.com/intrabuilder/papers.
- [35] *Opis IDC.* www.windows.com/IDC/default.asp.
- [36] *Opis LiveWire 1.01.* www.netscape.com/comprod/server_central/product/livewire/index.html.
- [37] *Opis ogólny ColdFusion 3.0.* www.allaire.com/Products/ColdFusion/30/Overview/index.cfm.
- [38] *Opis oprogramowania narzędziowego firmy Borland.* www.borland.com/papers/interde.html.
- [39] *Opisy oprogramowania firmy Oracle.* www.com.pl/index_produkty.htm.
- [40] *Opis techniczny NetDynamics 4.0.* www.netdynamics.com/product/overview/nd400_overview.htm.
- [41] *Opis techniczny NetDynamics 3.1.* www.netdynamics.com/product/overview/nd310_overview.htm.
- [42] *Opis techniczny NetDynamics 4.0.* www.netdynamics.com/product/overview/nd400_overview.htm.
- [43] *Oracle CDM Advantage, Release 1.0 for Windows 3.1/Windows 95/Windows NT.* Oracle Corporation, November, 1996, Release 2.0.1.
- [44] *Oracle Method – Custom Development – Requirements Modelling using Designer/2000.* CDM Standards and Guidelines Library, Volume 1, Release 2.1.0, October, 1996.
- [45] *Oracle Server 8 – Administrative Guide.* Praca niesygnowana. Oracle Press, 1997.
- [46] *Oracle Web Application Server.* Praca niesygnowana. Oracle Press, 1995.
- [47] *Pliki pomocników oraz dokumentacja firmowa w postaci elektronicznej dołączona do następujących pakietów oprogramowania firmy Powersoft: InfoMaker for Windows, Power Designer, Power Builder Enterprise, Power++ Enterprise.*
- [48] *Pliki pomocników oraz dokumentacja firmowa w postaci elektronicznej dołączona do następujących pakietów oprogramowania firmy Sybase: Sybase SQL Anywhere v.5.5.04, Adaptive Server Enterprise for Windows NT with Sybase SQL Remote and Powersoft InfoMaker.*
- [49] *Pliki pomocników oraz dokumentacja firmowa w postaci elektronicznej dołączona do oprogramowania SELECT Enterprise V6.0e firmy SELECT Software Tools, Inc.* www.selectst.com/Products.

- [50] *Strategia firmy Borland – dostęp do baz danych w sieciach rozległych.*
www.borland.com/about/papers/bgg.
- [51] Unified Modelling Language, UML Notation Guide v.1.1, ad/97-08-05 – Rational Software, Corp., and other, Sept., 1997.

DESIGN TECHNOLOGIES AND SOFTWARE TOOLS FOR BUILDING DISTRIBUTED MULTI-LAYERED APPLICATIONS INTEGRATED IN THE NETWORK ENVIRONMENT

Diversity in hardware and software is a fact of life, and network environment is becoming more diverse, not less as computers evolve. In the information systems powerful servers working in teams serve massive databases and surprisingly capable desktop machines provide an increasingly complex array of services, and interact with each other and with bigger systems to compute information in complex information systems. Varied types of computer and network hardware and operating systems are needed to get work done, but the complexity is difficult to deal with. Developing information and database systems by the mean of multi-layer application technologies is making this possible. This paper covers methodologies, techniques and tools that will be fundamental to programmers developing significant database applications in distributed environments. Also, the paper presents some of the typical problems found in software engineering, that can be solved by an approach based on middleware technologies, giving an example of the technology.

Arkadiusz MAJKOWSKI*

INTEGRACJA SCHEMATÓW HETEROGENICZNYCH BAZ DANYCH W MODELU OBIEKTOWYM

Cechą charakterystyczną pewnej klasy systemów heterogenicznych baz danych, zwanych federacjami baz danych, jest udostępnianie globalnego schematu opisującego dane we wszystkich bazach składowych. Jednym z kroków w procesie tworzenia federacji baz danych jest integracja schematów wyrażonych we wspólnym modelu danych. Artykuł ten stanowi propozycję metody integracji schematów wyrażonych w obiektowym modelu danych. Charakterystyczną cechą tej metody jest zastosowanie tak zwanej ścisłej hermetyzacji w budowie schematów obiektowych, czyli udostępniania w interfejsie obiektu wyłącznie operacji. Prezentowana metoda integracji wymaga znajomości semantyki integrowanych schematów.

1. CEL INTEGRACJI SCHEMATÓW

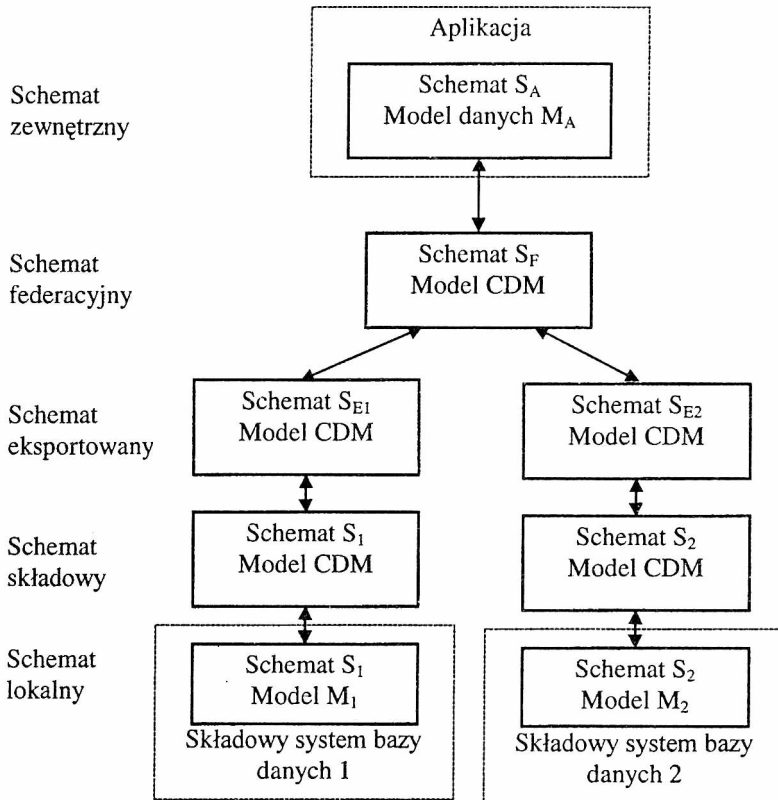
Integracja schematów jest procesem, który na wejściu otrzymuje zbiór schematów baz danych, a na wyjściu dostarcza nowy schemat. Heterogeniczne schematy wejściowe, wyrażone wszystkie w tym samym, wspólnym modelu danych opisują dane w bazach podlegających integracji. Schemat wyjściowy opisuje bez utraty informacji dane zgromadzone we wszystkich tych bazach łącznie.

Celem procesu integracji schematów jest utworzenie nowego schematu bazy danych, w oparciu o pewien zadany zbiór schematów. Nowopowstały schemat zintegrowany powinien opisywać dane zawarte w bazach, których schematy podlegają integracji, tak jak gdyby była to jedna baza danych. Schemat zintegrowany powinien więc stanowić pewnego rodzaju połączenie schematów wejściowych. Połączenie to odpowiada pojęciowo sumie zbiorów, wynik połączenia zawiera wszystkie elementy, ale elementy powtarzające się reprezentowane są tylko jednokrotnie.

Zastosowaniem integracji schematów jest głównie integracja heterogenicznych baz danych.

* Wydziałowy Zakład Informatyki, Wydział Informatyki i Zarządzania Politechniki Wrocławskiej, 50-370 Wrocław, Wybrzeże Wyspiańskiego 27, a.majkowski@ci.pwr.wroc.pl

Rysunek 1 przedstawia pięciopoziomową architekturę schematów w systemie heterogenicznych baz danych.



Rys. 1. Pięciopoziomowa architektura schematów w systemie heterogenicznych baz danych
Fig. 1. Five-layer architecture of schema in the system of heterogenic database

Schemat składowej bazy danych nazywany jest schematem lokalnym. Wyrażony jest w modelu właściwym dla danej składowej bazy danych. Ponieważ składowe bazy danych mogą mieć różne modele (np. relacyjny, sieciowy, obiektowy, związków encji), więc schematy lokalne mogą różnić się modelami danych. Aby ułatwić dostęp do systemu, dokonuje się translacji schematów lokalnych do wspólnego modelu danych (zwanego czasem również modelem kanonicznym – ang. *canonical data model CDM*) por. [PBE95],[KFA94]. Schemat uzyskany w wyniku tej translacji nazywany jest schematem składowym. Każda z baz danych uczestniczy w federacji eksportując część swojego schematu składowego zwaną schematem eksportowanym. Schemat federacyjny, zwany też globalnym, powstaje w wyniku integracji kilku schematów eksportowanych. Dostosowując do potrzeb użytkownika i nadając

odpowiednie prawa dostępu ze schematu federacyjnego (globalnego) tworzy się schemat zewnętrzny. Taka architektura schematów odpowiada klasie systemów heterogenicznych baz danych zwanych federacjami lub mocno powiązanymi systemami baz danych [SL90].

W literaturze przedmiotu można również natknąć się na integrację schematów, jako proces łączenia prac kilku zespołów niezależnie projektujących fragmenty schematu jednej, dużej bazy danych.

Przedstawiona tu metoda integracji została opracowana z ukierunkowaniem na integrację heterogenicznych baz danych.

2. OBIEKTOWY MODEL DANYCH

Obiektowy model danych w literaturze definiowany jest na wiele różnych sposobów. Próbą standaryzacji tych wielu podejść jest obiektowy model danych zaproponowany przez ODMG (*Object Database Management Group*). Model danych ODMG ma stanowić podstawę do budowy interfejsów do bazy danych niezależnych od implementacji SZBD (systemu zarządzania bazą danych) i poprzez to umożliwiać budowę przenośnych aplikacji pracujących z obiektowymi bazami danych. Pomimo swojej niedoskonałości (por. [Sub98]), model danych ODMG ze względu na swoją popularność oraz rozdzielenie specyfikacji (typów) od implementacji (klas), może stanowić dobrą podstawę do rozważań na temat integracji schematów. Pełną definicję modelu danych ODMG można znaleźć w [CB97]. Poniżej przedstawiono krótki zarys tego modelu.

Model danych ODMG

Podstawowymi strukturami danych w modelu ODMG są **obiekty** i **literały**. Literały nie mogą zmieniać wartości i są za pomocą tej wartości identyfikowane. Obiekty są identyfikowane za pomocą specjalnego **identyfikatora** (OID) i mogą zmieniać swoje wartości.

Struktura obiektu określona jest przez listę **atrybutów**, posiadających nazwy i literałowe dziedziny, oraz listę **związków** z innymi obiektami o zagwarantowanej spójności referencyjnej dla każdej liczebności związku (1:1, 1:n, n:m). **Stan obiektu** określają wartości przechowywanych przez obiekt atrybutów i utrzymywanych związków z innymi obiektami. Zachowanie obiektu określa lista sygnatur **operacji** (**metod**), które można na rzecz tego obiektu wykonać. Każda metoda może mieć parametry wejściowe i wyjściowe określonego typu oraz może zwrócić rezultat określonego typu.

Strukturę i zachowanie obiektu specyfikuje **typ**. Zbiór aktualnie istniejących obiektów danego typu nazywany jest **ekstensją** tego typu. Typy tworzą hierarchię.

Każdy typ dziedziczy z nadtypów, a jego ekstensja jest podzbiorem ekstensji nadtypu. Typ obiektu jest ustalany w trakcie tworzenia obiektu. Każdy obiekt jest bezpośrednią instancją jednego typu i pośrednią instancją wszystkich nadtypów. W typie może być zdefiniowany klucz określający unikatowość wartości grupy atrybutów w ekstensji.

W modelu ODMG wszystkie instancje danego typu mają ten sam interfejs, ale mogą mieć różne implementacje. **Klasa** jest implementacją typu, realizującą pewien interfejs. Klasa ma dokładnie jeden typ. Wiele klas może implementować ten sam typ.

Klasy zorganizowane są w postaci hierarchii zbudowanej w oparciu o relację generalizacji/specjalizacji. Klasy znajdujące się niżej w hierarchii stanowią **specjalizację** (podklasy) klas będących wyżej. O klasach położonych wyżej mówimy, że stanowią **generalizację** (nadklasy) klas będących niżej. Typ implementowany przez specjalizację danej klasy jest **podtypem** jej typu, a typ implementowany przez generalizację **nadtypem**. Obiekty podtypu posiadają wszystkie własności swojego nadtypu i dodatkowo mogą posiadać własności specyficzne dla podtypu. Własność ta określana jest mianem **dziedziczenia**. Obiekt podklasy może być użyty wszędzie tam, gdzie obiekt nadklasy, ale nie odwrotnie.

Model ODMG dopuszcza wielokrotne dziedziczenie. W przypadku konfliktu przyjmowana jest własność z nadklasy pierwszej w sekwencji dziedziczenia.

Baza danych opiera się na schemacie zdefiniowanym w języku ODL i zawiera zdefiniowane w tym schemacie obiekty (wystąpienia typów).

Przykład 1

Zdefiniowane poniżej dwa interfejsy typów obiektów mogą stanowić fragment schematu opisującego rachunki bankowe. Jeden z interfejsów opisuje obiekt *Rachunek*, a drugi, powiązany z nim związkiem 1:N, obiekt *Transakcja*. Obiekty typu *Rachunek* mają atrybuty *nrRachunku*, *dataOtwarcia*, *saldo* i *blokada*, połączone są związkiem o nazwie *transakcje* z wieloma obiektami typu *Transakcja* i udostępniają metodę *KsięgujTransakcje*. Obiekt typu *Transakcja* ma atrybuty *nrTran*, *data*, *rodzaj* i *wartość*, i jest połączony związkiem z dokładnie jednym obiektem typu *Rachunek*.

```
interface Rachunek
{
    attribute string nrRachunku;
    attribute Date dataOtwarcia;
    attribute double saldo;
    attribute boolean blokada;
    relationship Set<Transakcja> transakcje inverse Transakcja::doRachunku;
    void KsięgujTransakcje( in Transakcja tran );
}
interface Transakcja
{
    attribute integer nrTran;
    attribute Date data;
    attribute integer rodzaj;
    attribute double wartość;
    relationship Rachunek doRachunku inverse Rachunek::transakcje;
}
```

Ścisła hermetyzacja

Zauważmy, że w interfejsie typu każdy atrybut i związek możemy zastąpić dwiema operacjami, z których jedna będzie służyła do odczytu wartości atrybutu lub związku (DAJ_własność()) , a druga do jego zmiany (ZMIEN_własność()). Zdefiniowany w ten sposób interfejs odpowiada tak zwanej ścisłej hermetyzacji, czyli umożliwia dostęp do obiektu jedynie poprzez zdefiniowane operacje.

Podójście zakładające ścisłą hermetyzację ma zarówno zalety jak i wady. Z argumentami przeciwko ścisłej hermetyzacji w bazach danych o modelu obiektowym można zapoznać się w pracy [Sub98]. Zastosowanie ścisłej hermetyzacji ma jednak istotne zalety w kontekście integracji heterogenicznych baz danych.

- Rozdzielenie w schemacie operacji odczytujących i modyfikujących dane pozwala niezależnie rozpatrywać problemy odczytu własności obiektu oraz modyfikacji tejże własności.
- W naturalny sposób, poprzez udostępnienie tylko operacji DAJ_własność(), modelowane są te elementy integrowanych schematów, które nie mogą podlegać modyfikacji.
- Przyjęcie operacji jako jedynych udostępnianych własności obiektu jest naturalnym rozwiązaniem w przypadku tworzenia interfejsu do systemów nie będących bazami danych, por. [BNPS89] i [GCO91].

3. PRZYKŁADOWE SCHEMATY ILUSTRUJĄCE PROCES INTEGRACJI

Jako ilustracja do rozważań na temat integracji schematów posłużą dwa schematy baz danych, przechowujących informacje o rachunkach bankowych. Przyjęto, że schematy te pochodzą z dwóch relacyjnych baz danych.

Przykład 2

Schemat lokalny SL1; model relacyjny:

Schemat 1 opisuje dane hipotetycznego systemu bankowego. W systemie tym rejestrowane są dane o kontach osobistych (*KontaOsob*), operacjach księgowanych na tych kontach (*Operacje*), blokadach nałożonych na konta osobiste.

```
KontaOsob(NrKonta, DataOtw, Obciążenia, Uznania, Imie, Nazwisko, AdresL1, AdresL2)
Operacje(NrOper, NrKonta, Data, Opis, Typ, Kwota)
Blokady(NrBlokady, NrKonta, OdKiedy, DoKiedy)
```

Schemat składowy SS1:

Schemat ten powstał z translacji schematu lokalnego w modelu relacyjnym na model obiektowy. Z informacjami na temat translacji schematów relacyjnych na obiektowe można zapoznać się w artykule [KFA94].

```
interface KontoOsob
{
    attribute string nrKonta;
    attribute Date dataOtw;
    attribute double obciążenia;
    attribute double uznania;
    attribute string imie;
    attribute string nazwisko;
    attribute string adresL1;
    attribute string adresL2;
    relationship Set<Operacja> operacje inverse Operacja::konto;
    relationship Set<Blokada> blokady inverse Blokada::konto;
}

interface Operacja
{
    attribute Date data;
    attribute integer nrOper;
    attribute string opis;
    attribute double kwota;
    attribute integer typ;
    relationship KontoOsob konto inverse KontoOsob::operacje;
}

interface Blokada
{
    attribute Date odKiedy;
    attribute Date doKiedy;
    relationship KontoOsob konto inverse KontoOsob::blokady;
}
```

Schemat eksportowany SE1:

Schemat eksportowany powstaje ze schematu składowego w wyniku selekcji i zamiany atrybutów na operacje. Dla uproszczenia zapisu pominięto słowo kluczowe void języka ODL przed operacjami nie zwracającymi wyniku.

```
interface KontoOsob
{
    string DAJ_nrKonta();
    Date DAJ_dataOtw();
    double DAJ_obciążenia();
    double DAJ_uznania();
    string DAJ_imie();
    string DAJ_nazwisko();
    string DAJ_adresL1();
    string DAJ_adresL2();
    DAJ_operacje(out Set<Operacja>);
    DAJ_blokady(out Set<Blokada>);
    ZMIEN_nrKonta(in string);
    ZMIEN_dataOtw(in Date);
    ZMIEN_obciążenia(in double);
    ZMIEN_uznania(in double);
    ZMIEN_imie(in string);
    ZMIEN_nazwisko(in string);
    ZMIEN_adresL1(in string);
    ZMIEN_adresL2(in string);
    ZMIEN_operacje(in Set<Operacja>);
    ZMIEN_blokady(in Set<Blokada>);
}
```

```

interface Operacja
{
    Date    DAJ_data();          ZMIEN_data(in Date);
    integer DAJ_nrOper();        ZMIEN_nrOper(in integer);
    string  DAJ_opis();          ZMIEN_opis(in string);
    double  DAJ_kwota();         ZMIEN_kwota(in double);
    integer DAJ_typ();           ZMIEN_typ(in integer);
    DAJ_konto(out KontoOsob);    ZMIEN_konto(in KontoOsob);
}

interface Blokada
{
    Date DAJ_odKiedy();          ZMIEN_odKiedy(in Date);
    Date DAJ_doKiedy();          ZMIEN_doKiedy(in Date);
    DAJ_konto(out KontoOsob);    ZMIEN_konto(in KontoOsob);
}

```

Schemat lokalny SL2- relacyjny:

Schemat 2 opisuje dane dotyczące klientów (Klienci), prowadzonych na rzecz tych klientów rachunków (Rachunki) oraz księgowanych na rachunkach transakcjach (Transakcje).

Klienci(NrKl, Imię, Nazwisko, Miasto, KodP, Ulica, Nr)
 Rachunki(NrRach, NrKl, DataOtw, Saldo, Zablokowany)
 Transakcje(NrTran, NrRach, Data, Rodzaj, Wartość)

Schemat składowy SS2:

Schemat ten powstał z translacji schematu lokalnego w modelu relacyjnym na model obiektowy.

```

interface Klient
{
    attribute string nrKl;
    attribute string imie;
    attribute string nazwisko;
    attribute string miasto;
    attribute string kodP;
    attribute string ulica;
    attribute string nr;
    relationship Set<Rachunek> rachunki inverse Rachunek::wlaściciel;
}

interface Rachunek
{
    attribute string nrRachunku;
    attribute Date dataOtwarcia;
    attribute double saldo;
    attribute boolean zablokowany;
    relationship Klient wlaściciel inverse Klient::rachunki;
    relationship Set<Transakcja> transakcje inverse Transakcja::doRachunku;
}

```

```

interface Transakcja
{
    attribute integer nrTran;
    attribute Date data;
    attribute integer rodzaj;
    attribute double wartość;
    relationship Rachunek doRachunku inverse Rachunek::transakcje;
}

```

Schemat eksportowany SE2:

Schemat powstał ze schematu składowego w wyniku selekcji i zamiany atrybutów na operacje.

```

interface Klient
{
    string DAJ_nrKl();           ZMIEN_nrKl(in string);
    string DAJ_imie();           ZMIEN_imie(in string);
    string DAJ_nazwisko();       ZMIEN_nazwisko(in string);
    string DAJ_miasto();         ZMIEN_miasto(in string);
    string DAJ_kodP();           ZMIEN_kodP(in string);
    string DAJ_ulica();          ZMIEN_ulica(in string);
    string DAJ_nr();             ZMIEN_nr(in string);
    DAJ_rachunki(out Set<Rachunek>); ZMIEN_rachunki(in Set<Rachunek>);
}

interface Rachunek
{
    string DAJ_nrRachunku();      ZMIEN_nrRachunku(in string);
    Date DAJ_dataOtwarcia();      ZMIEN_dataOtwarcia(in Date);
    double DAJ_saldo();           ZMIEN_saldo(in double);
    boolean DAJ_zablokowany();    ZMIEN_zablokowany(in boolean);
    DAJ_właściciel(out Klient);   ZMIEN_właściciel(in Klient);
    DAJ_transakcje(out Set<Transakcja>); ZMIEN_transakcje(in Set<Transakcja>);
}

interface Transakcja
{
    integer DAJ_nrTran();         ZMIEN_nrTran(in integer);
    Date DAJ_data();              ZMIEN_data(in Date);
    integer DAJ_rodzaj();         ZMIEN_rodzaj(in integer);
    double DAJ_wartość();         ZMIEN_wartość(in double);
    DAJ_doRachunku(out Rachunek); ZMIEN_doRachunku(in Rachunek);
}

```

Zdefiniowane powyżej schematy eksportowane SE1 i SE2 będą podlegały integracji w celu ustalenia schematu federacyjnego.

4. USTALENIE KLAS RÓWNOWAŻNOŚCI W ZBIORZE TYPÓW OBIEKTÓW POCHODZĄCYCH ZE SCHEMATÓW EKSPORTOWANYCH

Każdy z podlegających integracji schematów eksportowanych SE_i zawiera pewien zbiór typów obiektów ZTO_{SE_i} . Oznaczmy przez SE zbiór schematów eksportowanych podlegających integracji. Zbiór wszystkich typów obiektów ZTO będzie określony następująco: $ZTO = \bigcup_{SE \in SE} ZTO_{SE_i}$

Przykład 3

$ZTO_{se_1} = \{\text{KontoOsob}, \text{Operacja}, \text{Blokada}\}$

$ZTO_{se_2} = \{\text{Klient}, \text{Rachunek}, \text{Transakcja}\}$

$ZTO = \{\text{KontoOsob}, \text{Operacja}, \text{Blokada}, \text{Klient}, \text{Rachunek}, \text{Transakcja}\}$

Ponieważ typy obiektów w zbiorze ZTO pochodzą ze schematów niezależnych baz danych, więc w zbiorze ZTO mogą występować powtarzające się typy. Schemat zintegrowany natomiast powinien modelować rzeczywistość tak jak gdyby był schematem jednej bazy danych, czyli nie powinien zawierać powtarzających się typów. Powoduje to konieczność identyfikacji powtórzeń typów w zbiorze ZTO .

Dla potrzeb identyfikacji powtórzeń nie można zastosować bezpośredniego porównania definicji typów. Typy pochodzące z poszczególnych schematów są zwykle niezależnie projektowane i przez to istnieje wśród nich duża różnorodność rozwiązań. Odpowiadające sobie typy mogą mieć różne nazwy i różne struktury. W celu ustalenia odpowiedniości typów nie można więc oprzeć się na porównaniu ani nazw ani struktur danych.

Do ustalenia równoważności typów wykorzystać można wiadomości na temat modelowanego w bazie danych wycinka rzeczywistości. Wiedza na temat modelowanej dziedziny jest zewnętrzną w stosunku do schematu bazy danych. Do znajdowania równoważnych typów w pracy [RPRG94] zaproponowano wykorzystanie pojęcia stanu świata rzeczywistego dla typu obiektu (ang. real world state RWS). Stan świata rzeczywistego ($S\acute{S}R$) dla typu obiektu to zbiór tych bytów w świecie rzeczywistym, które są modelowane w bazie danych przez obiekty tego typu. Gdy dwa typy A i B w różnych bazach danych modelują te same zbiory bytów w świecie rzeczywistym, to mówimy, że typy te mają ten sam stan świata rzeczywistego $S\acute{S}R(A) = S\acute{S}R(B)$. Nie oznacza to, że typy A i B reprezentują te same informacje o bytach w świecie rzeczywistym. Nawet jeśli informacje są te same, to typy mogą udostępniać różne interfejsy dostępu do danych.

Korzystając z pojęcia $S\acute{S}R$ do porównywania typów obiektów można w zbiorze typów obiektów ZTO zdefiniować relacje równoważności $\approx$. Dwa typy obiektów TO_i i TO_k ze zbioru ZTO są równoważne wtedy i tylko wtedy, gdy mają te same stany świata rzeczywistego:

$$TO_i \approx TO_k \Leftrightarrow S\acute{S}R(TO_i) = S\acute{S}R(TO_k)$$

Relacja $\approx$ posiada cechy relacji równoważności:

1) zwrotność:

$$TO_i \approx TO_i \text{ ponieważ: } S\acute{S}R(TO_i) = S\acute{S}R(TO_i)$$

2) symetryczność:

$$TO_i \approx TO_k \Rightarrow TO_k \approx TO_i \text{ ponieważ: } S\acute{S}R(TO_i) = S\acute{S}R(TO_k) \Rightarrow S\acute{S}R(TO_k) = S\acute{S}R(TO_i)$$

3) przechodniość:

$$(TO_i \approx TO_k \wedge TO_k \approx TO_m) \Rightarrow (TO_i \approx TO_m) \text{ ponieważ:}$$

$$(S\acute{S}R(TO_i) = S\acute{S}R(TO_k) \wedge S\acute{S}R(TO_k) = S\acute{S}R(TO_m)) \Rightarrow S\acute{S}R(TO_i) = S\acute{S}R(TO_m)$$

Należy zauważyć, że dwa typy obiektów mogą być w relacji równoważności pomimo, że udostępniają różne interfejsy do danych.

Korzystając z relacji równoważności można podzielić zbiór ZTO na rozłączne klasy równoważności. Zbiór klas równoważności zbudowany w oparciu o relację $\approx$ na podstawie zbioru ZTO oznaczmy przez $KZTO$. Klasę równoważności, do której należy typ $TO_i \in ZTO$, oznaczmy przez $||TO_i||$.

$$||TO_i|| = \{ TO_k \mid TO_k \in ZTO \wedge TO_k \approx TO_i \}$$

$$KZTO = \{ ||TO_i|| \mid TO_i \in ZTO \}$$

Przykład 4

Elementy wymienione w nawiasach kwadratowych [] tworzą klasę równoważności.

$$S\acute{S}R(\text{KontoOsob}) = S\acute{S}R(\text{Rachunek})$$

$$S\acute{S}R(\text{Operacja}) = S\acute{S}R(\text{Transakcja})$$

$$\begin{aligned} ||\text{Klient}|| &= [\text{Klient}] \\ ||\text{Blokada}|| &= [\text{Blokada}] \\ ||\text{Operacja}|| &= [\text{Operacja}, \text{Transakcja}] \\ ||\text{Rachunek}|| &= [\text{KontoOsob}, \text{Rachunek}] \end{aligned}$$

$$KZTO = \{ [\text{KontoOsob}, \text{Rachunek}], [\text{Operacja}, \text{Transakcja}], [\text{Klient}], [\text{Blokada}] \}$$

5. WYODRĘBNIE NIE RÓWNOWAŻNYCH OPERACJI

W danej klasie równoważności występują typy obiektów mające ten sam stan świata rzeczywistego, ale potencjalnie realizujące różne operacje na danych czyli udostępniające różne interfejsy.

Rozważmy zbiór typów składających się na jedną z klas równoważności $\|TO_i\|$. Oznaczmy przez ZOP_{TO_i} zbiór operacji udostępnianych przez typ TO_i . Interfejs każdego z typów $TO_i \in \|TO_i\|$ składa się z pewnej liczby operacji $OP_i \in ZOP_{TO_i}$. Zbiór $ZOP_{\|TO_i\|}$ taki że: $ZOP_{\|TO_i\|} = \bigcup_{TO_i \in \|TO_i\|} ZOP_{TO_i}$ będzie się składał ze wszystkich

operacji udostępnionych przez typy tworzące klasę równoważności $\|TO_i\|$.

W zbiorze tym występować będą operacje udostępniane przez typy obiektów modelujące te same byty świata rzeczywistego. Istnieje duże prawdopodobieństwo, że twórcy różnych baz danych definiując typy dla tych samych bytów świata rzeczywistego zaprojektowali dla tych typów podobne operacje. Wśród tych operacji występować mogą operacje równoważne. Ze względu na autonomię projektową powodującą zróżnicowanie rozwiązań, ustalenie równoważności operacji jest równie trudne, jeśli nie trudniejsze od ustalenia równoważności typów.

Podobnie jak w przypadku równoważności typów, ustalenia równoważności operacji można dokonać w oparciu o znajomość modelowanego przez schemat wycinka rzeczywistości. Rozważmy operacje ze zbioru $ZOP_{\|TO_i\|}$. Są to operacje dotyczące typów równoważnych w sensie relacji $\approx$. Operacje te można podzielić na dwie grupy: takie, które nie zmieniają stanu obiektu na rzecz którego są wykonywane oraz takie, które ten stan zmieniają. Operacje z tych dwóch grup nie mogą być równoważne, w związku z tym operacji równoważnych należy poszukiwać niezależnie w obu tych grupach. W obu grupach o zgodności operacji można mówić w przypadku zgodności sygnatur operacji, to jest zgodności ilości i typów parametrów oraz typu zwracanej wartości. W przypadku operacji nie zmieniających stanu obiektu podstawą do ustalenia ich równoważności powinna być zgodność zwracanych przez te operacje wartości. Operacje zmieniające stan obiektu należy porównywać uwzględniając nie tylko ich składnię i wartości zwracane w wyniku ich wykonania, ale również sposób w jaki zmieniają one stan obiektu. O równoważności takich operacji można mówić wtedy, gdy w wyniku ich wykonania następują zgodne zmiany stanów obiektu. Operacje powodujące efekty uboczne inne niż zmiana stanu obiektu (por. [Sub98]), będą w tej wersji metody integracji traktowane jako nie posiadające operacji równoważnych.

Rozważając równoważność operacji należy rozważyć cztery aspekty:

- 1) Zgodność sygnatur operacji, to jest zgodność liczby i typów parametrów oraz typu zwracanej wartości.

- 2) Zgodność zwracanych przez te operacje wartości. Operacje można uznać za zgodne, gdy w wyniku ich wykonania zwracane są wartości charakteryzujące te same cechy, reprezentowanych przez odpowiednie typy, bytów z modelowanej rzeczywistości. Dodatkowo zwracane wartości powinny być reprezentowane w ten sam sposób.
- 3) Zgodność zmiany stanów obiektu w wyniku wykonania operacji, rozumianą następująco:
Rozważmy dwa obiekty O_1 typu TO_1 i O_2 typu TO_2 , $TO_1 \approx TO_2$, modelujące ten sam byt świata rzeczywistego X , znajdujący się w stanie S_1 . Jeżeli po każdym wykonaniu operacji OP_i na obiekcie O_1 i operacji OP_j na obiekcie O_2 , obiekty zarówno O_1 jak i O_2 modelują byt X , znajdujący się w tym samym stanie S_2 , to operacje OP_1 i OP_2 powodują zgodne zmiany stanów obiektów.
- 4) Brak efektów ubocznych, innych niż zmiana stanu obiektu na rzecz, którego operacja jest wykonywana.

Operacje zgodne we wszystkich czterech wymienionych wyżej aspektach będziemy uznawać za **równoważne**. Relację równoważności operacji OP_1 i OP_2 oznaczymy $OP_1 \equiv OP_2$.

Powyższą definicję równoważności operacji nazwijmy integracyjną równoważnością operacji lub krócej równoważnością operacji.

W oparciu o powyższą definicję można w zbiorze $ZOP_{||TO||}$ zdefiniować relację równoważności operacji, dzielącą ten zbiór na rozłączne klasy równoważności. Zbiór klas równoważności zbudowany w oparciu o relację $\equiv$ na zbiorze $ZOP_{||TO||}$ oznaczymy przez $KZOP_{||TO||}$. Klasę równoważności do której należy operacja OP_i należąca do zbioru $ZOP_{||TO||}$ oznaczymy przez $||OP_i||$.

$$||OP_i|| = \{ OP_k \mid OP_k \in ZOP_{||TO||} \wedge OP_k \equiv OP_i \}$$

$$KZOP_{||TO||} = \{ ||OP_i|| \mid OP_i \in ZOP_{||TO||} \}$$

Każda z klas równoważności operacji w zbiorze $KZOP_{||TO||}$ jest zbiorem równoważnych składniowo i semantycznie operacji i stanowi podstawę do definicji jednej operacji w typie schematu zintegrowanego.

Przykład 5

Poniżej przedstawiono zbiory operacji dla równoważnych typów Rachunek oraz KontoOsob, klasę operacji równoważnych operacji `DAJ_NrRachunku()` i zbiór klas równoważności operacji dla typów równoważnych typowi Rachunek. Dla uproszczenia zapisu pominięto parametry operacji.

```

ZOPRachunek = {
  DAJ_nrRachunku(),    ZMIEN_nrRachunku(),
  DAJ_dataOtwarcia(),  ZMIEN_dataOtwarcia(),
  DAJ_saldo(),         ZMIEN_saldo(),
  DAJ_zablokowany(),   ZMIEN_zablokowany(),
  DAJ_właściciel(),    ZMIEN_właściciel(),
  DAJ_transakcje(),     ZMIEN_transakcje()
}

```

```

ZOPKontoOsob = {
  DAJ_nrKonta(),       ZMIEN_nrKonta(),
  DAJ_dataOtw(),       ZMIEN_dataOtw(),
  DAJ_obciążenia(),    ZMIEN_obciążenia(),
  DAJ_uznania(),       ZMIEN_uznania(),
  DAJ_imie(),          ZMIEN_imie(),
  DAJ_nazwisko(),      ZMIEN_nazwisko(),
  DAJ_adresL1(),       ZMIEN_adresL1(),
  DAJ_adresL2(),       ZMIEN_adresL2(),
  DAJ_operacje(),      ZMIEN_operacje(),
  DAJ_blokady(),       ZMIEN_blokady(),
}

```

Przynależność operacji O_a, O_b, O_c do jednej klasy równoważności oznaczono przez zapis ich w nawiasach kwadratowych $[O_a, O_b, O_c]$.

```

||DAJ_NrRachunku()|| = [DAJ_NrRachunku(), DAJ_NrKonta()]

```

```

KZOP||Rachunek|| = {
  [KontoOsob::DAJ_nrKonta(), Rachunek::DAJ_nrRachunku()],
    [KontoOsob::ZMIEN_nrKonta(), Rachunek::ZMIEN_nrRachunku()],
  [KontoOsob::DAJ_dataOtw(), Rachunek::DAJ_dataOtwarcia()],
    [KontoOsob::ZMIEN_dataOtw(), Rachunek::ZMIEN_dataOtwarcia()],
  [KontoOsob::DAJ_obciążenia()], [KontoOsob::ZMIEN_obciążenia()],
  [KontoOsob::DAJ_uznania()], [KontoOsob::ZMIEN_uznania()],
  [KontoOsob::DAJ_imie()], [KontoOsob::ZMIEN_imie()],
  [KontoOsob::DAJ_nazwisko()], [KontoOsob::ZMIEN_nazwisko()],
  [KontoOsob::DAJ_adresL1()], [KontoOsob::ZMIEN_adresL1()],
  [KontoOsob::DAJ_adresL2()], [KontoOsob::ZMIEN_adresL2()],
  [KontoOsob::DAJ_operacje()], [KontoOsob::ZMIEN_operacje()],
  [KontoOsob::DAJ_blokady()], [KontoOsob::ZMIEN_blokady()],
  [Rachunek::DAJ_saldo()], [Rachunek::ZMIEN_saldo()],
  [Rachunek::DAJ_zablokowany()], [Rachunek::ZMIEN_zablokowany()],
  [Rachunek::DAJ_właściciel()], [Rachunek::ZMIEN_właściciel()],
  [Rachunek::DAJ_transakcje()], [Rachunek::ZMIEN_transakcje()]
}

```

Gdyby każdy typ $TO_i \in ||TO_i||$ realizował operacje z każdej z klas równoważności w zbiorze $KZOP_{||TO_i||}$, to w schemacie zintegrowanym można by umieścić typ odpowiadający klasie równoważności $||TO_i||$ udostępniający tyle operacji, ile jest klas równoważnych operacji. W rzeczywistości każdy z typów $TO_i \in ||TO_i||$ może

udostępnić różną ilość operacji. Dodatkowo, jedynie część z operacji udostępnianych przez TO_i , może mieć równoważne odpowiedniki z innych typów. Na podstawie klasy równoważności $||TO_i||$ nie można więc utworzyć jednego typu globalnego udostępniającego wszystkie rodzaje operacji odpowiadających elementom zbioru $KZOP_{||TO_i||}$, ponieważ obiekty nie wszystkich baz składowych będą potrafiły zrealizować cały zbiór operacji. Tworząc jeden typ globalny dla $||TO_i||$ należałoby ustalić dlań taki interfejs, aby każdy z typów $TO_i \in ||TO_i||$ miał odpowiedniki dla każdej operacji z tego interfejsu. Interfejs ten składałby się ze zbioru operacji zbudowanego w oparciu o podzbiór zbioru $KZOP_{||TO_i||}$ określony następująco:

$$KZOP-MIN_{||TO_i||} = \{ KOP \mid KOP \in KZOP_{||TO_i||} \wedge (\forall TO_i \in ||TO_i||, \exists OP_i \in ZOP_{||TO_i||}, OP_i \in KOP) \}$$

W ten sposób otrzymalibyśmy typ schematu zintegrowanego, którego interfejs byłby realizowany w całości przez wszystkie obiekty, których typy należą do zbioru $||TO_i||$. Interfejs ten w pewnych przypadkach byłby dużym zawężeniem funkcjonalnym interfejsów udostępnianych przez schematy składowe. Przyczyną tego jest fakt, że zbiór $KZOP-MIN_{||TO_i||}$, stanowiący jak gdyby część wspólną interfejsów udostępnionych przez typy z $||TO_i||$, będzie pomijał te operacje, które nie występują w interfejsach wszystkich typów.

Przykład 6

Zbiór klas równoważnych operacji realizowanych przez wszystkie typy równoważne typowi rachunek.

```
KZOP-MIN||Rachunek|| = {
  [KontoOsob::DAJ_nrKonta(), Rachunek::DAJ_nrRachunku()],
  [KontoOsob::ZMIEN_nrKonta(), Rachunek::ZMIEN_nrRachunku()],
  [KontoOsob::DAJ_dataOtw(), Rachunek::DAJ_dataOtwarcia()],
  [KontoOsob::ZMIEN_dataOtw(), Rachunek::ZMIEN_dataOtwarcia()]
}
```

6. ROZSZERZENIE ZBIORU OPERACJI

Załóżmy że interfejs jednego z typów $TO_i \in ||TO_i||$ nie zawiera operacji równoważnej z operacją $OP_k \in ZOP_{TO_k}$, $TO_k \in ||TO_i||$, to znaczy $\neg \exists (OP_i \in ZOP_{TO_i}) OP_i \equiv OP_k$. Pojawia się wówczas pytanie, czy na obiektach typu TO_i nie można wykonać operacji równoważnej operacji OP_k udostępnionej przez typ TO_k . Zauważyć należy, że w pewnych przypadkach wykonanie operacji OP_k na obiekcie TO_k jest równoważne wykonaniu na obiekcie typu TO_i pewnej złożonej operacji, wykorzystującej kilka operacji udostępnionych przez typ TO_i . Przykładem takiej operacji złożonej jest wyliczenie salda konta osobistego jako różnicy kwot

uznaniowych i obciążających. Operacje złożone można zdefiniować np. w celu wyliczenia pewnej wartości niedostępnej bezpośrednio, konwersji typów lub konwersji jednostek miary, w których wartości są wyrażone. Poszukiwanie operacji złożonej polega na ustaleniu jak w kategoriach danego schematu wyrazić operację równoważną operacji zdefiniowanej w innym schemacie. W przypadku operacji złożonych zwracających i zmieniających wartości pewnej cechy, możliwość zdefiniowania jednej z nich, np. zwracającej, nie pociąga za sobą istnienia drugiej, zmieniającej. Ilustruje to przykład 7.

Szczególnymi przypadkami operacji złożonych mogą być “operacje-zaśleпки”, które zamiast nie modelowanej przez dany typ własności bytu rzeczywistego zwracają wartość stałą lub symulują zmianę stanu obiektu na stan nie dający się w danym typie zamodelować. Wprowadzenie tego typu operacji zaślepek musi być stosowane ze szczególną rozważą, ze względu na możliwe konsekwencje w postaci przekłamania informacji.

Oznaczmy przez $\Phi^k_{TO_i}$ k -tą operację złożoną na obiekcie typu TO_i . Znalezienie operacji złożonej $\Phi^k_{TO_i}$ równoważnej pewnej operacji OP_i , sprawia że na typie TO_i nie posiadającym operacji należącej do klasy $\|OP_i\|$ można wykonać taką operację. Powoduje to, że operacja klasy $\|OP_i\|$ przestaje być wykluczana ze względu na typ TO_i z interfejsu typu odpowiadającego typom klasie $\|TO_i\|$ w schemacie zintegrowanym. Zdefiniowanie złożonych operacji może więc dać możliwość powiększenia zbioru operacji w typie schematu zintegrowanego. Ze względu na poszerzenie zbioru operacji udostępnionych przez typ schematu zintegrowanego odpowiadający klasie $\|TO_i\|$ celowe jest poszukiwanie w każdym typie $TO_i \in \|TO_i\|$ operacji złożonych odpowiadających tym klasom równoważności operacji, w których brakuje operacji z typu TO_i .

Oznaczmy przez θ_{TO_i} podzbiór zbioru $KZOP_{\|TO_i\|}$, $TO_i \in \|TO_i\|$ składający się z tych klas równoważności operacji, które nie zawierają operacji z typu TO_i .

$$\theta_{TO_i} = \{ \|OP_k\| \mid \neg \exists (OP_i \in \|OP_k\|), OP_i \in ZOP_{TO_i} \}$$

Zbiór poszukiwanych operacji złożonych dla typu TO_i ($ZPOZ_{TO_i}$) będzie się składał z operacji równoważnych operacjom należącym do klas równoważności w zbiorze θ_{TO_i} .

$$ZPOZ_{TO_i} = \{ \Phi^k_{TO_i} \mid \forall x \in \theta_{TO_i}, \forall y \in x, y \equiv \Phi^k_{TO_i} \}$$

Ponieważ nie wszystkie poszukiwane operacje złożone będą możliwe do zdefiniowania, więc zbiór znalezionych operacji złożonych $ZZOPZ_{TO_i}$ będzie podzbiorem zbioru operacji poszukiwanych $ZPOPZ_{TO_i}$.

Przykład 7

Zbiór poszukiwanych operacji złożonych dla typu Rachunek. Poniżej wypisano jakim operacjom odpowiadają poszukiwane operacje złożone.

```
ZZPOPZ_Rachunek = { $\Phi^1$ _Rachunek,  $\Phi^2$ _Rachunek,  $\Phi^3$ _Rachunek,  $\Phi^4$ _Rachunek,  $\Phi^5$ _Rachunek,  $\Phi^6$ _Rachunek,  $\Phi^7$ _Rachunek,
 $\Phi^8$ _Rachunek,  $\Phi^9$ _Rachunek,  $\Phi^{10}$ _Rachunek,  $\Phi^{11}$ _Rachunek,  $\Phi^{12}$ _Rachunek,  $\Phi^{13}$ _Rachunek,  $\Phi^{14}$ _Rachunek,  $\Phi^{15}$ _Rachunek,  $\Phi^{16}$ _Rachunek
}
```

```
 $\Phi^1$ _Rachunek  $\equiv$  KontoOsob::DAJ_obciażenia()
 $\Phi^2$ _Rachunek  $\equiv$  KontoOsob::ZMIEN_obciażenia()
 $\Phi^3$ _Rachunek  $\equiv$  KontoOsob::DAJ_uznania()
 $\Phi^4$ _Rachunek  $\equiv$  KontoOsob::ZMIEN_uznania()
 $\Phi^5$ _Rachunek  $\equiv$  KontoOsob::DAJ_imie()
 $\Phi^6$ _Rachunek  $\equiv$  KontoOsob::ZMIEN_imie()
 $\Phi^7$ _Rachunek  $\equiv$  KontoOsob::DAJ_nazwisko()
 $\Phi^8$ _Rachunek  $\equiv$  KontoOsob::ZMIEN_nazwisko()
 $\Phi^9$ _Rachunek  $\equiv$  KontoOsob::DAJ_adresL1()
 $\Phi^{10}$ _Rachunek  $\equiv$  KontoOsob::ZMIEN_adresL1()
 $\Phi^{11}$ _Rachunek  $\equiv$  KontoOsob::DAJ_adresL2()
 $\Phi^{12}$ _Rachunek  $\equiv$  KontoOsob::ZMIEN_adresL2()
 $\Phi^{13}$ _Rachunek  $\equiv$  KontoOsob::DAJ_operacje()
 $\Phi^{14}$ _Rachunek  $\equiv$  KontoOsob::ZMIEN_operacje()
 $\Phi^{15}$ _Rachunek  $\equiv$  KontoOsob::DAJ_blokady()
 $\Phi^{16}$ _Rachunek  $\equiv$  KontoOsob::ZMIEN_blokady()
```

Pseudo-programowy zapis realizacji operacji złożonych:

```
 $\Phi^5$ _Rachunek -> Rachunek::DAJ_właściciel().DAJ_imie();
 $\Phi^6$ _Rachunek -> Rachunek::DAJ_właściciel().ZMIEN_imie();
 $\Phi^7$ _Rachunek -> Rachunek::DAJ_właściciel().DAJ_nazwisko();
 $\Phi^8$ _Rachunek -> Rachunek::DAJ_właściciel().ZMIEN_nazwisko();
 $\Phi^9$ _Rachunek -> Rachunek::DAJ_właściciel().Klient::DAJ_Ulica()+
" "+Rachunek::DAJ_właściciel().Klient::DAJ_Nr();

 $\Phi^{11}$ _Rachunek -> Rachunek::DAJ_właściciel().Klient::DAJ_KodP()+
" "+Rachunek::DAJ_właściciel().Klient::DAJ_Miasto();

 $\Phi^2$ _Rachunek(double noweObciążenia) ->
Rachunek::ZMIEN_Saldo(Rachunek::DAJ_Saldo()-noweObciążenia);

 $\Phi^4$ _Rachunek(double noweUznania) ->
Rachunek::ZMIEN_Saldo(Rachunek::DAJ_Saldo()+noweUznania);
```

Zbiór znalezionych operacji złożonych dla typu Rachunek.

```
ZZOPZ_Rachunek = { $\Phi^2$ _Rachunek,  $\Phi^4$ _Rachunek,  $\Phi^5$ _Rachunek,  $\Phi^6$ _Rachunek,  $\Phi^7$ _Rachunek,  $\Phi^8$ _Rachunek,
 $\Phi^9$ _Rachunek,  $\Phi^{11}$ _Rachunek}
```

Zbiór poszukiwanych operacji złożonych dla typu KontoOsob.

Poniżej wypisano jakim operacjom odpowiadają poszukiwane operacje złożone.

$ZPOPZ_{KontoOsob} = \{\Phi^1_{KontoOsob}, \Phi^2_{KontoOsob}, \Phi^3_{KontoOsob}, \Phi^4_{KontoOsob}, \Phi^5_{KontoOsob}, \Phi^6_{KontoOsob}, \Phi^7_{KontoOsob}, \Phi^8_{KontoOsob}\}$

$\Phi^1_{KontoOsob} \equiv \text{Rachunek}::\text{DAJ_Saldo}()$
 $\Phi^2_{KontoOsob} \equiv \text{Rachunek}::\text{ZMIEN_Saldo}()$
 $\Phi^3_{KontoOsob} \equiv \text{Rachunek}::\text{DAJ_Zablokowany}()$
 $\Phi^4_{KontoOsob} \equiv \text{Rachunek}::\text{ZMIEN_Zablokowany}()$
 $\Phi^5_{KontoOsob} \equiv \text{Rachunek}::\text{DAJ_Właściciel}()$
 $\Phi^6_{KontoOsob} \equiv \text{Rachunek}::\text{ZMIEN_Właściciel}()$
 $\Phi^7_{KontoOsob} \equiv \text{Rachunek}::\text{DAJ_Transakcje}()$
 $\Phi^8_{KontoOsob} \equiv \text{Rachunek}::\text{ZMIEN_Transakcje}()$

Pseudo-programowy zapis realizacji operacji złożonych:

$\Phi^1_{KontoOsob} \rightarrow \text{KontoOsob}::\text{DAJ_Uznania}() - \text{KontoOsob}::\text{DAJ_Obciążenia}();$

$\Phi^2_{KontoOsob}(\text{double noweSaldo}) \rightarrow$
 if noweSaldo > ($\text{KontoOsob}::\text{DAJ_Uznania}() - \text{KontoOsob}::\text{DAJ_Obciążenia}()$) then
 $\text{KontoOsob}::\text{ZMIEN_Uznania}(\text{noweSaldo} + \text{KontoOsob}::\text{DAJ_Obciążenia}())$
 else
 $\text{KontoOsob}::\text{ZMIEN_Obciążenia}(\text{KontoOsob}::\text{DAJ_Uznania}() - \text{noweSaldo});$

$\Phi^3_{KontoOsob} \rightarrow \text{SprawdźBlokade}(\text{KontoOsob}::\text{DajBlokady}(), \text{teraz});$

$\Phi^4_{KontoOsob}(\text{nowaBlokada}) \rightarrow$
 if nowaBlokada then
 $\text{KontoOsob}::\text{ZMIEN_Blokady}(\text{DodajBlokadę}(\text{DajBlokady}(), \text{teraz}))$
 else
 $\text{KontoOsob}::\text{ZMIEN_Blokady}(\text{ZakończBlokadę}(\text{DajBlokady}(), \text{teraz}));$

Zbiór znalezionych operacji złożonych dla typu Rachunek.

$ZZOPZ_{KontoOsob} = \{\Phi^1_{KontoOsob}, \Phi^2_{KontoOsob}, \Phi^3_{KontoOsob}, \Phi^4_{KontoOsob}\}$

Dzięki znalezieniu operacji złożonych wykonalnych na typie TO_i , rozszerzony został zbiór operacji udostępnionych przez ten typ. Zbiór operacji, składający się ze zbioru operacji udostępnionych przez typ TO_i tj. ZOP_{TO_i} oraz ze zbioru znalezionych operacji złożonych dla tego typu tj. $ZZOPZ_{TO_i}$ nazwijmy rozszerzonym zbiorem operacji dla typu TO_i i oznaczmy przez $RZOP_{TO_i}$.

Przykład 8

```

RZOPRachunek = {
  DAJ_nrRachunku(),   ZMIEN_nrRachunku(),
  DAJ_dataOtwarcia(), ZMIEN_dataOtwarcia(),
  DAJ_saldo(),        ZMIEN_saldo(),
  DAJ_zablokowany(),  ZMIEN_zablokowany(),
  DAJ_właściciel(),   ZMIEN_właściciel(),
  DAJ_transakcje(),   ZMIEN_transakcje(),
   $\Phi^2_{\text{Rachunek}}$ ,  $\Phi^4_{\text{Rachunek}}$ ,  $\Phi^5_{\text{Rachunek}}$ ,  $\Phi^6_{\text{Rachunek}}$ ,  $\Phi^7_{\text{Rachunek}}$ ,  $\Phi^8_{\text{Rachunek}}$ ,  $\Phi^9_{\text{Rachunek}}$ ,  $\Phi^{11}_{\text{Rachunek}}$ 
}

```

```

RZOPKontoOsob = {
  DAJ_nrKonta(),      ZMIEN_nrKonta(),
  DAJ_dataOtw(),      ZMIEN_dataOtw(),
  DAJ_obciążenia(),   ZMIEN_obciążenia(),
  DAJ_uznania(),      ZMIEN_uznania(),
  DAJ_imie(),         ZMIEN_imie(),
  DAJ_nazwisko(),     ZMIEN_nazwisko(),
  DAJ_adresL1(),       ZMIEN_adresL1(),
  DAJ_adresL2(),       ZMIEN_adresL2(),
  DAJ_operacje(),     ZMIEN_operacje(),
  DAJ_blokady(),      ZMIEN_blokady(),
   $\Phi^1_{\text{KontoOsob}}$ ,  $\Phi^2_{\text{KontoOsob}}$ ,  $\Phi^3_{\text{KontoOsob}}$ ,  $\Phi^4_{\text{KontoOsob}}$ 
}

```

W oparciu o znalezione operacje złożone, w każdym z typów $TO_i \in ||TO_i||$ otrzymujemy rozszerzony zbiór operacji. W oparciu o rozszerzone zbiory operacji każdego z typów możemy skonstruować rozszerzony zbiór operacji udostępnianych przez wszystkie typy wchodzące w skład klasy równoważności $||TO_i||$.

$$RZOP_{||TO_i||} = \bigcup_{TO_i \in ||TO_i||} RZOP_{TO_i}$$

7. REPREZENTACJA ZBIORU TYPÓW RÓWNOWAŻNYCH W SCHEMACIE ZINTEGROWANYM

W oparciu o definicję integracyjnej równoważności operacji, w zbiorze $RZOP_{||TO_i||}$ można wyodrębnić klasy równoważności operacji. Każda z tych klas będzie zawierała równoważne operacje będące bądź operacjami pochodzącymi z integrowanych schematów bądź operacjami złożonymi. Zbiór klas równoważności zbudowany w oparciu o relację $\equiv$ na zbiorze $RZOP_{||TO_i||}$ oznaczmy przez $KRZOP_{||TO_i||}$. Każda z klas równoważności operacji w zbiorze $KRZOP_{||TO_i||}$ jest zbiorem równoważnych składniowo i semantycznie operacji i stanowi podstawę do definicji jednej operacji

kandydującej do schematu zintegrowanego. Oznaczmy przez $ZOK_{||TO_i||}$ zbiór operacji kandydujących do schematu zintegrowanego jako reprezentacja operacji ze zbioru $RZOP_{||TO_i||}$.

$$ZOK_{||TO_i||} = \{OK \mid \exists ||OP_j|| \in KRZOP_{||TO_i||}, \forall OP_i \in ||OP_j||, OK \equiv OP_i\}$$

Nazwy dla tych operacji mogą pochodzić od nazw operacji udostępnianych w schematach składowych lub mogą zostać ustalone na nowo. Pozostała część sygnatury musi być zgodna z sygnaturami równoważnych operacji ze zbioru $RZOP_{||TO_i||}$.

Przykład 9

```

KRZOP||Rachunek|| = {
[KontoOsob::DAJ_nrKonta(), Rachunek::DAJ_nrRachunku()],
[KontoOsob::ZMIEN_nrKonta(), Rachunek::ZMIEN_nrRachunku()],
[KontoOsob::DAJ_dataOtw(), Rachunek::DAJ_dataOtwarcia()],
[KontoOsob::ZMIEN_dataOtw(), Rachunek::ZMIEN_dataOtwarcia()],
[KontoOsob::DAJ_obciążenia()],
[KontoOsob::ZMIEN_obciążenia(),  $\Phi^2_{Rachunek}$ ],
[KontoOsob::DAJ_uznania()],
[KontoOsob::ZMIEN_uznania(),  $\Phi^4_{Rachunek}$ ],
[KontoOsob::DAJ_imie(),  $\Phi^5_{Rachunek}$ ],
[KontoOsob::ZMIEN_imie(),  $\Phi^6_{Rachunek}$ ],
[KontoOsob::DAJ_nazwisko(),  $\Phi^7_{Rachunek}$ ],
[KontoOsob::ZMIEN_nazwisko(),  $\Phi^8_{Rachunek}$ ],
[KontoOsob::DAJ_adresL1(),  $\Phi^9_{Rachunek}$ ],
[KontoOsob::ZMIEN_adresL1()],
[KontoOsob::DAJ_adresL2(),  $\Phi^{11}_{Rachunek}$ ],
[KontoOsob::ZMIEN_adresL2()],
[KontoOsob::DAJ_operacje()],
[KontoOsob::ZMIEN_operacje()],
[KontoOsob::DAJ_blokady()],
[KontoOsob::ZMIEN_blokady()],
[Rachunek::DAJ_saldo(),  $\Phi^1_{KontoOsob}$ ],
[Rachunek::ZMIEN_saldo(),  $\Phi^2_{KontoOsob}$ ],
[Rachunek::DAJ_zablokowany(),  $\Phi^3_{KontoOsob}$ ],
[Rachunek::ZMIEN_zablokowany(),  $\Phi^4_{KontoOsob}$ ],
[Rachunek::DAJ_właściciel()],
[Rachunek::ZMIEN_właściciel()],
[Rachunek::DAJ_transakcje()],
[Rachunek::ZMIEN_transakcje()]
}

```

```

ZOK||Rachunek|| = {
DAJ_nrKonta(), ZMIEN_nrKonta(),
DAJ_dataOtwarcia(), ZMIEN_dataOtwarcia(),
DAJ_obciążenia(), ZMIEN_obciążenia(),
DAJ_uznania(), ZMIEN_uznania(),

```

```

DAJ_imie(),           ZMIEN_imie(),
DAJ_nazwisko(),       ZMIEN_nazwisko(),
DAJ_adresL1(), ZMIEN_adresL1(),
DAJ_adresL2(), ZMIEN_adresL2(),
DAJ_operacje(),       ZMIEN_operacje(),
DAJ_blokady(), ZMIEN_blokady(),
DAJ_saldo(),          ZMIEN_saldo(),
DAJ_zablokowany(),   ZMIEN_zablokowany(),
DAJ_właściciel(),    ZMIEN_właściciel(),
DAJ_transakcje(),    ZMIEN_transakcje()
}

```

Rozważając tak otrzymany zbiór operacji zauważmy, że każdy typ $TO_i \in ||TO_i||$ może realizować operacje równoważne wszystkim lub tylko części operacji ze zbioru $ZOK_{||TO_i||}$. Zbiór operacji realizowanych przez wszystkie typy będzie jak gdyby częścią wspólną zbiorów operacji udostępnianych przez każdy z typów TO_i . Oznaczmy przez $ZWO_{||TO_i||}$ zbiór wspólnych operacji typów z $||TO_i||$:

$$ZWO_{||TO_i||} = \{ OW \mid OW \in ZOK_{||TO_i||} \ \forall TO_i \in ||TO_i|| \ \exists OP_j \in ZOP_{TO_i}, \ OP_j \equiv OW \}$$

Przykład 10

```

ZWO_{||Rachunek||} = {
    DAJ_nrKonta(),           ZMIEN_nrKonta(),
    DAJ_dataOtwarcia(),      ZMIEN_dataOtwarcia(),
                                ZMIEN_obciążenia(),
                                ZMIEN_uznania(),
    DAJ_imie(),              ZMIEN_imie(),
    DAJ_nazwisko(),          ZMIEN_nazwisko(),
    DAJ_adresL1(),
    DAJ_adresL2(),
    DAJ_saldo(),             ZMIEN_saldo(),
    DAJ_zablokowany(),      ZMIEN_zablokowany()
}

```

W pewnych przypadkach zbiór ten może zawierać znacznie mniej operacji niż interfejs każdego z integrowanych typów. Dążenie do reprezentacji wszystkich typów należących do jednej klasy równoważności za pomocą jednego typu realizującego tylko operacje “wspólne” dla wszystkich typów równoważnych, może więc spowodować, że część z operacji udostępnianych przez typy obiektów ze schematów integrowanych będzie w schemacie zintegrowanym niedostępna. Spowodowałoby to znaczne ograniczenie możliwości dostępu do danych i skutkiem tego utratę informacji. W celu uzyskania pełnego dostępu do danych musi istnieć możliwość wykonania wszystkich operacji udostępnianych przez typy z $||TO_i||$.

Z drugiej strony reprezentacja zbioru typów równoważnych za pomocą jednego typu realizującego wszystkie operacje ze zbioru $ZOK_{||TO_i||}$ ma również wady. Żądania wykonania operacji na obiekcie typu globalnego muszą zostać przełożone na wykonania operacji na obiektach typów składowych. Jeśli dany obiekt z bazy składowej należy do typu, który nie realizuje pewnej operacji z interfejsu typu

globalnego, to operację taką można wykonać tylko wtedy jeśli dany obiekt ma swój odpowiednik należący do takiego typu, który realizuje dana operację. Gdy nie ma odpowiednika danego obiektu w żadnym typie składowym realizującym daną operację, to takiej operacji nie można wykonać na tym obiekcie. Wobec tego obiekt ten, jako nie realizujący w pełni interfejsu takiego typu globalnego, nie będzie miał swojego reprezentanta w ekstensji typu globalnego. Niedostępność takich obiektów z poziomu schematu zintegrowanego oznacza również utratę danych. Rozwiązanie z jednym typem udostępniającym wszystkie operacje ze zbioru $ZOK_{||TO_i||}$ jest więc również niedobre.

Budując reprezentację dla typów z $||TO_i||$ trzeba pogodzić wymaganie pełnego dostępu do danych z zapewnieniem wspólnej reprezentacji dla odpowiadających sobie części integrowanych schematów. Model obiektowy dostarcza mechanizmu, który to umożliwi - hierarchii typów. Typ realizujący pewien interfejs może mieć podtypy – specjalizacje, których interfejs jest rozszerzeniem interfejsu danego typu. Zbiór obiektów-instancji danego typu zawiera w sobie zbiór instancji, każdego z podtypów.

Hierarchia typów reprezentujących w schemacie zintegrowanym typu z $||TO_i||$ budowana będzie na podstawie udostępnianych przez typ operacji w oparciu o relację specjalizacji/generalizacji. Na szczycie tej hierarchii powinien się znaleźć typ udostępniający operacje ze zbioru $ZWO_{||TO_i||}$, oznaczmy go przez $T^0_{||TO_i||}$. Na najniższym poziomie będzie typ udostępniający wszystkie operacje ze zbioru $ZOK_{||TO_i||}$, oznaczmy ten typ przez $T^{max}_{||TO_i||}$. Na poziomach pośrednich będą typy odpowiadające tym typom z $||TO_i||$, które udostępniają operacje nie posiadające w $RZOP_{||TO_i||}$ równoważnych operacji z innych typów. W pośrednich węzłach hierarchii mogą się również znaleźć typy udostępniające operacje, równoważne operacjom udostępnianym przez pewien podzbiór typów z $||TO_i||$.

Typ $T^0_{||TO_i||}$ znajdujący się na szczycie hierarchii będzie udostępniał interfejs realizowany przez wszystkie równoważne typy ze schematów integrowanych. Ekstensja tego typu będzie zawierała obiekty-instancje pochodzące z wszystkich równoważnych typów z $||TO_i||$. Typ $T^0_{||TO_i||}$ będzie stanowił generalizację wszystkich typów równoważnych ze schematów integrowanych.

Typ $T^{max}_{||TO_i||}$ będzie udostępniał najbogatszy interfejs zawierający operacje równoważne dla wszystkich operacji w interfejsach typów składowych. Ekstensja tego typu będzie składała się z obiektów-instancji, reprezentujących zbiory odpowiadających sobie obiektów, z takiego podzbioru typów składowych, w którym realizowalne będą wszystkie wymagane operacje. Typ $T^{max}_{||TO_i||}$ będzie stanowił specjalizację wszystkich typów równoważnych ze schematów integrowanych.

Ekstensje typów pośrednich będą składały się z obiektów reprezentujących takie obiekty lub zbiory odpowiadających sobie obiektów z baz składowych, które będą mogły zrealizować wszystkie operacje danego typu pośredniego.

Za pośrednictwem interfejsów zdefiniowanych w wyspecjalizowanych podtypach typu $T^0_{||TO||}$ możliwy będzie dostęp do wszystkich danych reprezentowanych przez integrowane schematy.

Hierarchia będzie utworzona w oparciu o relację specjalizacji, porządkującą typy według wspólnych operacji. Mówimy że typ TO_b jest specjalizacją typu TO_a ($TO_b \geq TO_a$), a typ TO_a generalizacją typu TO_b gdy :

$$TO_b \geq TO_a \Leftrightarrow \forall OP_i \in ZOP_{TO_a}, OP_i \in ZOP_{TO_b}.$$

Mówimy, że typ TO_a jest nadtypem typu TO_b , a typ TO_b jest podtypem typu TO_a .

Tworzona hierarchia typów będzie zawierała typy, które w schemacie zintegrowanym będą reprezentowały zbiór równoważnych typów ze schematów integrowanych. Podstawą do określenia typów tworzących hierarchię reprezentującą typy z $||TO_i||$ będzie zbiór operacji $ZOK_{||TO_i||}$. W oparciu o ten zbiór wyznaczony zostanie zbiór wszystkich potencjalnie użytecznych interfejsów typów. Zbiór ten będzie się składał z:

- 1) interfejsów typów odpowiadających typom ze zbioru $||TO_i||$. Przez typ odpowiadający typowi $TO_i \in ||TO_i||$ należy rozumieć typ schematu zintegrowanego zawierający po jednej operacji równoważnej dla każdej operacji z interfejsu typu TO_i .
- 2) interfejsów typów będących maksymalnymi generalizacjami dwu lub więcej typów odpowiadających typom ze zbioru $||TO_i||$. Przez typ będący maksymalną generalizacją zbioru typów należy rozumieć taki typ, którego żaden podtyp nie jest generalizacją danego zbioru typów.
- 3) interfejsów typów będących maksymalnymi specjalizacjami dwu lub więcej typów odpowiadających typom ze zbioru $||TO_i||$. Przez typ będący maksymalną specjalizacją zbioru typów należy rozumieć taki typ, którego żaden nadtyp nie jest specjalizacją danego zbioru typów.

Tak utworzony zbiór interfejsów należy ułożyć w postaci hierarchii w oparciu o relację specjalizacji. Utworzona w ten sposób hierarchia będzie acyklicznym grafem skierowanym. Węzłami grafu będą utworzone typy, łuki natomiast będą łączyły nadtyp z jego podtypem. Powstały graf stanowi hierarchię typów, reprezentujących w możliwie pełny sposób zbiór równoważnych typów ze schematów integrowanych.

W szczególnym przypadku, gdy wszystkie typy będą realizowały ten sam rozszerzony zbiór operacji, hierarchia typów zredukuje się do jednego typu.

W przypadku gdy zbiór $||TO_i||$ jest dwuelementowy to hierarchia typów będzie składała się z czterech typów: typu $T^0_{||TO||}$, dwu typów reprezentujących typy z $||TO_i||$ oraz typu $T^{max}_{||TO||}$.

W praktyce, prawdopodobnie, możliwie pełna reprezentacja zbioru składającego się z więcej niż dwu typów równoważnych będzie zbyt rozbudowana. Część z typów, w węzłach pośrednich pomiędzy węzłem odpowiadającym typowi $T^0_{||TO||}$, a węzłem odpowiadającym typowi $T^{max}_{||TO||}$, może być mało użyteczna ze względu na brak

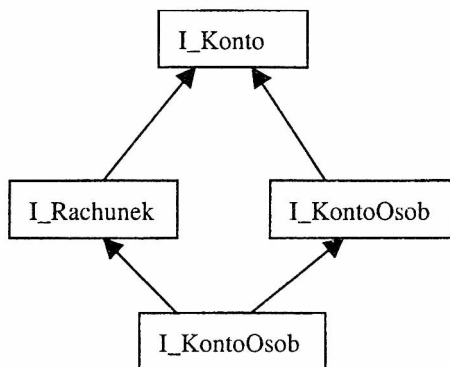
dobrego odniesienia do modelowanej rzeczywistości. Mając na uwadze ograniczenie nadmiernego rozrostu schematu zintegrowanego, celowe byłoby usunięcie z hierarchii nieużytecznych typów.

Nazwy dla typów odpowiadających typom ze zbioru $\|TO_i\|$, mogą pochodzić od ich odpowiedników w schematach integrowanych lub mogą zostać ustalone na nowo. Pozostałe typy muszą otrzymać nowe nazwy.

Przykład 11

Hierarchia typów odpowiadająca klasie $\|Rachunek\|$ będzie się składała z czterech typów:

- typu I_Konto , będącego na szczycie hierarchii, realizującego operacje wspólne dla obu podtypów,
- typu $I_Rachunek$, odpowiadającego typowi $Rachunek$,
- typu $I_KontoOsob$, odpowiadającego typowi $KontoOsob$ oraz
- typu $I_RachunekOsob$, realizującego wszystkie operacje ze zbioru $ZOK_{\|Rachunek\|}$.



Rys. 2. Hierarchia typów reprezentujących typy równoważne z klasy $\|Rachunek\|$

Fig. 2. The hierarchy of types representing equivalent types from class $\|Rachunek\|$

Dwukropek i identyfikator po nazwie typu w definicji interfejsu oznaczają w języku ODL, że dany typ jest specjalizacją typu o tym identyfikatorze. Dla skrócenia zapisu pominięto w zapisie operacji typy zwracanego wyniku i parametrów.

```

interface I_Konto
{
  DAJ_nrKonta(),           ZMIEN_nrKonta(),
  DAJ_dataOtwarcia(),      ZMIEN_dataOtwarcia(),
                           ZMIEN_obciążenia(),
                           ZMIEN_uznania(),
  DAJ_imie(),              ZMIEN_imie(),
  DAJ_nazwisko(),          ZMIEN_nazwisko(),
  DAJ_adresL1(),
  DAJ_adresL2(),
  DAJ_saldo(),             ZMIEN_saldo(),

```

```

DAJ_zablokowany(),          ZMIEN_zablokowany()
}

interface I_Rachunek: I_Konto
{
DAJ_właściciel();           ZMIEN_właściciel();
DAJ_transakcje();           ZMIEN_transakcje();
}
interface I_KontoOsob: I_Konto
{
DAJ_obciążenia();           ZMIEN_obciążenia();
DAJ_uznania();              ZMIEN_uznania();
                             ZMIEN_adresL1();
                             ZMIEN_adresL2();
DAJ_operacje();             ZMIEN_operacje();
DAJ_blokady();              ZMIEN_blokady();
}

interface I_RachunekOsob : I_Konto, I_Rachunek
{
}

```

8. PODSUMOWANIE

Przedstawiona w tym artykule metoda integracji schematów oparta jest o obiektowy model danych ODMG. W modelu tym rozdzielono specyfikację typu definiowaną przez interfejs od jego implementacji definiowanej przez klasę. To rozdzielenie specyfikacji od implementacji przystaje dobrze do procesu integracji heterogenicznych baz danych, gdzie reprezentowane przez schemat zintegrowany specyfikacje typów mają wiele implementacji odpowiadających składowym bazom danych. W metodzie tej przyjęto zastosowanie tak zwanej ścisłej hermetyzacji, czyli udostępniania przez obiekt w interfejsie jedynie operacji. Metoda ta bazuje na znajomości semantyki integrowanych schematów.

Integracja schematów przebiega w kilku krokach. Początkowo typu udostępniane przez integrowane schematy dzielone są na klasy równoważności. Podstawą do tego podziału jest zgodność zbiorów bytów świata rzeczywistego modelowanych przez typy. Dla każdej z tych klas równoważności jest następnie, niezależnie tworzona reprezentacja w schemacie zintegrowanym. Do zbioru udostępnianych przez typy operacji dodawane są operacje złożone, które uzupełniają operacje zdefiniowane w interfejsach typów. Operacje udostępniane przez wszystkie typy równoważne oraz zdefiniowane dla tych typów operacje złożone są dzielone na klasy operacji równoważnych syntaktycznie i semantycznie, tworząc rozszerzony zbiór operacji. W oparciu o klasy równoważnych operacji ustalane są operacje schematu zintegrowanego. W oparciu o te operacje i relację specjalizacji budowana jest hierarchia typów reprezentująca wszystkie równoważne typy ze schematów integrowanych.

W ramach dalszych prac planowane jest uwzględnienie możliwości wspólnej reprezentacji w schemacie zintegrowanym bytów, które są modelowane w schematach integrowanych przez różne konstrukcje modelu. Rozwinięciem opisanej tu metody integracji będzie także uwzględnienie połączenia, rozłącznych hierarchii reprezentujących poszczególne klasy typów równoważnych w oparciu o ich interfejsy i istniejące w schematach integrowanych relacje specjalizacji/generalizacji. Potencjalnym rozszerzeniem tej metody mogłoby być również rozważenie równoważności operacji z efektami ubocznymi odnoszącymi się do innych obiektów.

LITERATURA

- [BNPS89] Bertino E., Negri M., Pelagatti G., Sbatella L., *Integration of Heterogeneous Database Applications through an Object-Oriented Interface*, Information Systems, Vol. 14, No. 5, Sep. 1989.
- [CB97] Cattell R.G.G., *The Object Database Standard: ODMG 2.0*, Morgan Kaufmann Publishers, Inc., San Francisco 1997.
- [GCO91] Gagliardi R., Caneve M., Oldano G., *An Operational Approach to the Integration of Distributed Heterogeneous Environments*, Databases: Theory, Design and Applications, 4/1991.
- [KFA94] Klas W., Fischer G., Aberer K., *Integrating Relational and OO Database Systems using a Metaclass Concept*, Journal of Systems Integration, 4, s. 341–372, 1994.
- [Maz85] Mazur Z., *Bazy danych w sieciach komputerowych*, Materiały z konferencji Sieci komputerowe – protokoły, usługi modele, Wrocław, 1985.
- [PBE95] Pitoura E., Bukhres O., Elmagarmid A., *Object Orientation in Multidatabase Systems*, ACM Computing Surveys, Vol. 27, No. 2 Jun. 1995.
- [RPRG94] Reddy M.P., Prasad B.E., Reddy P.G., Gupta A., *A Methodology for Integration of Heterogeneous Databases*, IEEE Trans. on Knowledge and Data Eng. Vol. 6, No. 6, Dec. 1994.
- [SL90] Sheth A., Larson J., *Federated Database Systems*, ACM Computing Surveys, Vol. 22, No. 3, Sept. 1990.
- [Sub98] Subieta K., *Obiektość w projektowaniu i bazach danych*, Akademicka Oficyna Wydawnicza PLJ, Warszawa, 1998.

HETEROGENEOUS DATABASES SCHEMA INTEGRATION USING OBJECT MODEL

Federated database system provides global schema, defining an integrated view of the heterogeneous component databases. One of the required steps to build federated database system is integration of schemas translated to common data model. This paper presents a new method for integration of schemas that are modelled by means of ODMG object model. The advantage of this method is its ability to integrate objects which interfaces consist of only operations. The integration process requires semantic knowledge about integrated schemas.





Wydawnictwa Politechniki Wrocławskiej
są do nabycia w następujących księgarniach:
„Politechnika”
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław
budynek A-1 PWr., tel. (0-71) 320-25-34,
„Tech”
plac Grunwaldzki 13, 50-377 Wrocław
budynek D-1 PWr., tel. (0-71) 320-32-52
Prowadzimy sprzedaż wysyłkową

ISBN 83-7085-554-7