

**Prace Naukowe
Instytutu Informatyki Stosowanej
Politechniki Wrocławskiej**

Biblioteka Główna i OINT
Politechniki Wrocławskiej



7

BAZY DANYCH

Redakcja naukowa

Zygmunt Mazur



**Oficyna Wydawnicza
Politechniki Wrocławskiej**



Prace Naukowe
Instytutu Informatyki Stosowanej
Politechniki Wrocławskiej

7

BAZY DANYCH

Redakcja naukowa

Zygmunt Mazur



Oficyna Wydawnicza Politechniki Wrocławskiej
Wrocław 2006

NS 135 853



Politechnika Wrocławska

Institut Informatyki Stosowanej

Recenzent

Andrzej KWIECIEN

Opracowanie redakcyjne materiałów

Hanna MAZUR



Projekt okładki

Zygmunt MAZUR

329273 L/1

Wszelkie prawa zastrzeżone. Żadna część niniejszej książki, zarówno w całości, jak i we fragmentach, nie może być reprodukowana w sposób elektroniczny, fotograficzny i inny bez zgody wydawcy.

© Copyright by Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław 2006

OFICyna WYDAWNICZA POLITECHNIKI WROCLAWSKIEJ

Wybrzeże Wyspiańskiego 27, 50-370 Wrocław

<http://www.oficyna.pwr.wroc.pl>

oficwyd@pwr.wroc.pl

ISBN 83-7085-990-9

Drukarnia Oficyny Wydawniczej Politechniki Wrocławskiej. Zam. nr 961/2006.

Aluc 2007/0178/k

SPIS TREŚCI

<i>Wstęp</i>	5
D. Król, B. Knabel, <i>Wyszukiwanie danych w bazie danych na platformie J2EE</i>	7
K. Frączkowski, Z. Mazur, <i>SOA – architektura zorientowana na usługi</i>	21
A. Liber, J. Dybowski, <i>Konstrukcja niedostrzegalnych kanałów komunikacyjnych z wykorzystaniem protokołu IP i ich zastosowanie w podpisywaniu danych</i>	29
Ł. Pławiak, A. Wilczek, <i>Dane przestrzenne według specyfikacji OpenGIS</i>	71
Z. Fryźlewicz, M. Apostoluk, <i>Generyczna warstwa pośrednia dostępu do bazy danych</i>	79
Z. Mazur, <i>Zarządzanie długimi transakcjami</i>	97
Z. Fryźlewicz, K. Kunicki, <i>Wstęp do WS-Policy</i>	117
A. Piechowiak, A. Wilczek, <i>Standard SQLJ</i>	127
Z. Mazur, P. Mielcarek, M. Parczewski, <i>HSSE – Transparentna ewolucja schematu w relacyjnych bazach danych</i>	139

CONTENTS

<i>Introduction</i>	5
D. Król, B. Knabel, <i>Query Evaluation for Databases on the J2ee Platform</i>	7
K. Frączkowski, Z. Mazur, <i>SOA – Service Oriented Architecture</i>	21
A. Liber, J. Dybowski, <i>The Construction of Imperceptible Channels of Communication with the Utilization of the IP Protocol and Their Use In The Data Signing</i>	29
Ł. Pławiak, A. Wilczek, <i>Geospatial Data by OpenGIS</i>	71
Z. Fryżlewicz, M. Apostoluk, <i>Generic Database Middleware</i>	79
Z. Mazur, <i>A Long Live Transactions Management</i>	97
Z. Fryżlewicz, K. Kunicki, <i>An Introduction to WS-Policy</i>	117
A. Piechowiak, A. Wilczek, <i>SQLJ Standard</i>	127
Z. Mazur, P. Mielcarek, M. Parczewski, <i>HSSE – The Transparent Evolution of Schema in Relational Databases</i>	139

*Świat nie jest dla ludzi normalnych.
Jest dla znormalizowanych.*

Anonim

WSTĘP

Oddajemy w Państwa ręce kolejny, siódmy numer Zeszytu Naukowego serii „*Bazy danych*” *Prace Naukowe Instytutu Informatyki Stosowanej Politechniki Wrocławskiej*. Pierwszy zeszyt tej serii został wydany w 2000 r. pod szyldem *Wydziałowego Zakładu Informatyki*. Tematyka bieżącego Zeszytu Naukowego związana jest:

- z wyszukiwaniem danych w bazie danych na platformie J2EE,
- z nową, przyjazną architekturą zorientowaną na usługi czyli SOA,
- z wykorzystaniem protokołu IP do transmisji dodatkowych informacji np. podpisów cyfrowych,
- z danymi przestrzennymi i standardem OpenGis,
- z zagadnieniami związanymi z generyczną warstwą dostępu do baz danych,
- z zarządzaniem długimi transakcjami,
- ze standardem *Web Services Policy Framework* (WS-Policy), a w szczególności z implementacją tego standardu zawartą w *Web Services Enhancements 2.0*,
- ze standardem SQLJ (*SQL for Java*),
- z dostępnością schematu relacyjnej bazy danych podczas jego modyfikacji.

Coraz więcej projektów informatycznych stosuje obiektowe technologie, oparte na językach Java lub C# do budowy logiki biznesowej oraz relacyjne bazy danych do przechowywania obiektów. Między podejściem obiektowym i relacyjnym istnieje pewne niedopasowanie. Programiści radzą sobie z nim odwzorowując obiekty na wiersze tabeli w relacyjnej bazie danych. Istnieje wiele narzędzi pomocnych w tym procesie i pozwalających na utrwalanie obiektów bez utraty dziedziczenia, polimorfizmu oraz kolekcji. Pierwszy artykuł Zeszytu 7. prezentuje uzyskane wyniki oraz wnioski z przeprowadzonych badań dla trzech wybranych narzędzi: Castor, Hibernate oraz Jakarta OJB, dla których został zaimplementowany test OO7 (standardowy zestaw eksperymentów do badania wydajności obiektowych baz danych).

Wiele projektów informatycznych zakończyło się niepowodzeniem ze względu na zbyt długi czas realizacji przedsięwzięcia. W tym czasie często ulegały zmianom założenia biznesowe i wymagania funkcjonalne, których uwzględnienie w projekcie wymagało ponownej analizy i przeprojektowania modelu bazy danych i aplikacji bazodanowej. Dlatego też niezwykle istotne jest zorientowanie narzędzi na usługi co umożliwi elastyczne wprowadzanie zmian. Projektowanie i implementację takich systemów umożliwia wykorzystanie architektury zorientowanej na usługi SOA (ang. *Service Oriented Architecture*). Zagadnienie to omówione jest w artykule drugim.

Problematyka ukrywania dodatkowych informacji w protokołach komunikacyjnych jest przedmiotem intensywnych badań. Trzeci artykuł omawia zagadnienia związane z wykorzystaniem protokołu IP do transmisji dodatkowych informacji np. podpisów cyfrowych.

Autorzy czwartego artykułu przedstawiają specyfikację OpenGIS. Standard przechowywania danych przestrzennych w relacyjnych bazach danych według specyfikacji OpenGIS odnosi się przede wszystkim do wektorowego modelu danych. W standardzie tym istnieje model pojęciowy, w którym zdefiniowano typy danych geometrycznych i operacje sprawdzania ich zależności oraz dwie metody implementacji standardu w bazach danych. Pierwsza zakłada wykorzystanie wyłącznie podstawowych konstrukcji języka SQL, druga wprowadza definicje geometrycznych typów danych, które można implementować w bazach danych wspierających abstrakcyjne typy danych języka SQL.

W artykule piątym przedstawiony jest opis generycznej warstwy pośredniej dostępu do baz danych ułatwiającej programowanie logiki biznesowej, zapobiegającej dublowaniu kodu, oraz porządkującej aplikację.

Artykuł szósty definiuje pojęcie transakcji długich oraz omawia problemy związane z zarządzaniem transakcjami długimi w systemach baz danych.

Kolejne dwa artykuły dotyczą zagadnień standaryzacyjnych. Pierwszy z nich dotyczy standardu *Web Services Policy Framework (WS-Policy)*, który dostarcza model oraz syntaktykę umożliwiającą definiowanie polis dla usług webowych (*Web Services*) a drugi przedstawia standard *SQLJ (SQL for Java)*, który jest próbą ograniczenia negatywnych konsekwencji wynikających z osadzania instrukcji języka SQL w aplikacjach języka Java.

Modyfikując system informatyczny np. w celu dodania nowej funkcjonalności, często występuje problemem dostępności schematu w czasie jego aktualizacji. Ostatni artykuł przedstawia propozycję modelu HSSE (ang. *Hot Swap Schema Evolution*) do rozwiązania problemów dostępności schematu w czasie jego modyfikacji, wykorzystującego wersjonowanie schematu relacyjnego oraz opcjonalne rozszerzenie języka zapytań.

Autorzy składają podziękowanie Recenzentowi za uwagi zamieszczone w recenzji, które zostały wykorzystane w trakcie przygotowywania końcowych wersji artykułów.

Zygmunt Mazur

Dariusz KRÓL*
Bartłomiej KNABEL

WYSZUKIWANIE DANYCH W BAZIE DANYCH NA PLATFORMIE J2EE

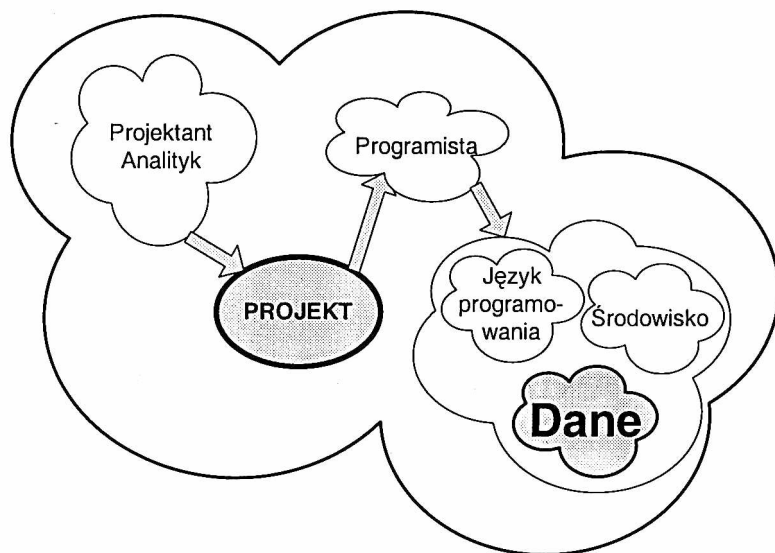
Coraz więcej projektów informatycznych stosuje obiektowe technologie, oparte na językach Java lub C# do budowy logiki biznesowej oraz relacyjne bazy danych do przechowywania obiektów. Między podejściem obiektowym i relacyjnym istnieje pewne niedopasowanie. Programiści radzą sobie z nim w taki sposób, że odwzorowują obiekty na wiersze tabeli w relacyjnej bazie danych. Istnieje wiele narzędzi pomocnych w tym procesie i pozwalających na utrwalanie obiektów bez utraty dziedziczenia, polimorfizmu oraz kolekcji. Dla trzech wybranych narzędzi (Castor, Hibernate, OJB) został zaimplementowany test OO7 – standardowy zestaw eksperymentów do badania wydajności obiektowych baz danych. Niniejsza praca prezentuje uzyskane wyniki oraz wnioski z przeprowadzonych badań.

1. WPROWADZENIE

Prostota i idąca za tym popularność relacyjnych baz danych sprawiły, że najprawdopodobniej będą one funkcjonowały jeszcze przez długi czas. Z drugiej jednak strony wydaje się, że programista używający obiektowego języka programowania chciałby korzystać z bazy danych również w sposób obiektowy. Jeżeli więc w zdecydowanej większości przypadków nie chce, nie umie, bądź nie może korzystać z bazy obiektowej, można pozwolić mu na posługiwanie się bazą relacyjną w sposób obiektowy. Okazuje się, że istnieją na rynku rozwiązania tego typu, gdzie można niewielkim nakładem sił zupełnie przeźroczyć korzystać z relacyjnej bazy danych za pomocą obiektowego języka programowania.

Celem pracy jest sprawdzenie działania opisywanego rozwiązania w praktyce, ze szczególnym nastawieniem na jego wydajność. Omawiany problem został przedstawiony na rysunku 1. Największa zaznaczona na nim „chmurka” symbolizuje wspólny język nie tylko modelowania danych, ale także programowania oraz bazy danych.

* Politechnika Wrocławska, Wydział Informatyki i Zarządzania, Instytut Informatyki Stosowanej, 50-370 Wrocław, Wybrzeże Wyspiańskiego 27, Dariusz.Krol@pwr.wroc.pl



Rys. 1. Wspólny model pojęciowy

Model obiektowy jest bardzo atrakcyjny do tworzenia systemów informacyjnych. W systemach takich przyjęło się stosować relacyjne bazy danych. Jednym z języków programowania, który może zostać użyty do zaimplementowania jest Java. Z językiem Java związana jest również technologia J2EE, która stanowi podstawową platformę systemów działających w Internecie. Ponieważ właśnie w Internecie istnieje najwięcej systemów informacyjnych, J2EE wydaje się być najlepszą technologią do stosowania narzędzi mapujących w praktyce.

Problem oceny wydajności narzędzi ORM (ang. *Object-Relational Mappers*) [SZPUNAR] nie został jeszcze rozwiązany. Nie ma żadnego uniwersalnego testu. Jedyne rozwiązanie zostało zaproponowane przez firmę Object Identity i stanowi ono implementację testu OO7 dla narzędzi zgodnych ze standardem JDO.

Test OO7 [KRYNSKI] został opracowany w roku 1993 z myślą o badaniu wydajności obiektowych baz danych. Stał się on niezwykle popularny, istnieje wiele opracowań związanych z OO7, który jest szeroko wykorzystywany również do badania wydajności dostępu do danych zamieszczonych w plikach XML [XOO7]. Nic nie stoi na przeszkodzie, aby ten test wykorzystać do zbadania wydajności narzędzi mapujących.

Każde z testowanych narzędzi cechuje się inną strukturą pliku konfiguracyjnego oraz innym dialektem używanym w kodzie Javy. Jednak podczas implementacji udało się przygotować testy w jednolity sposób dla każdego z wybranych systemów.

W pracy ze względu na ograniczoną objętość artykułu opisano jedynie eksperymenty dotyczące zapytań. Wyniki z eksperymentów dotyczących nawigacji oraz modyfikacji opisane zostały w [KNABEL].

2. OPIS WYBRANYCH NARZĘDZI ORM

Obecnie znaleźć można wiele, także komercyjnych, produktów implementujących standard JDO. Istnieje również kilka dobrych narzędzi, które umożliwiają operowanie na relacyjnych bazach danych w sposób obiektowy, nie będących implementacją JDO. Do testów wybrano dwa takie systemy (Castor oraz Hibernate). Trzeci z testowanych systemów – Jakarta OJB jest zgodny z JDO. Wszystkie z przedstawionych rozwiązań są bezpłatne.

2.1. CASTOR

System Castor nie implementuje standardu JDO, choć posiada interfejs JDO. Castor cieszy się szczególną popularnością w zastosowaniach związanych z plikami XML, kiedy to dane nie są przechowywane w RSZBD, ale w repozytoriach XML.

W systemie Castor wspierane są następujące RSZBD: DB2, Hypersonic SQL, Informix, InstantDB, Interbase, mySQL, Oracle, PostgreSQL, SAP DB, MS SQL Server, Sybase. Pełną listę możliwości Castora można zobaczyć na stronie <http://www.castor.org/status.html>.

2.2. HIBERNATE

Ten produkt wspierany jest przez JBoss Group i stanowi doskonały szkielet odwzorowań obiektowo-relacyjnych. Obsługuje on prawie wszystkie RSZBD (DB2, PostgreSQL, mySQL, Oracle, Sybase, MS SQL Server, SAP, Informix, Hypersonic SQL, Ingres, Progress, Interbase, Pointbase, FrontBase, Firebird). Hibernate jest dojrzałym systemem i cechuje go stabilność. Bardzo duża ilość dziennych informacji pojawiających się na domowym forum dyskusyjnym świadczy o zaangażowaniu wielu programistów w ten produkt. W porównaniu do pozostałych produktów Hibernate jest najczęściej cytowanym narzędziem.

System ten wyposażony został w interfejs ODMG. Wraz z biblioteką dostarczony jest także zestaw narzędzi pozwalających na generowanie kodu źródłowego klas aplikacji na podstawie schematu danych. Konfiguracja mapowań w Hibernate może być zapisana w pojedynczym pliku XML, bądź też z użyciem Xdoclet w formacie JavaDoc.

System Hibernate używa obiektowy język zapytań, wspiera dziedziczenie oraz pozwala na definiowanie kluczy złożonych, wszystkich możliwych rodzajów relacji (m.in. dwukierunkowych), a także na kaskadową aktualizację danych. Nowa wersja Hibernate 3 wspiera składowanie danych w plikach XML. W Hibernate zaimplementowanych jest również kilka rodzajów buforowania danych przyspieszających pracę w środowisku wielowątkowym. Kompletną listę możliwości Hibernate znaleźć można na stronie <http://www.hibernate.org/4.html>.

2.2. OBJECT RELATIONAL BRIDGE

System ten jako jedyny z testowanych implementuje standard JDO. Jest on częścią projektu firmy Apache o nazwie Jakarta. Posiada on najwięcej standardowych API – ODMG, JDO, PB (Persistence Broker), OTM (Object Transaction Manager). Używa standardowego języka zapytań do obiektowych baz danych – OQL. Biblioteka OJB, podobnie jak poprzednie, może być używana w środowisku J2EE. Możliwe jest korzystanie z dziedziczenia, a także używanie wszelkich relacji między obiektami.

Jak wynika z przeprowadzonych testów (patrz rozdział 3) OJB posiada najlepiej działający system buforowania. OJB posiada także system synchronizacji, który pozwala na korzystanie z tej samej bazy danych przez wiele aplikacji z użyciem wspólnego bufora.

Obsługiwane są następujące RSZBD: DB2, Firebird, Hsqldb, Informix, MaxDB, MsAccess, MsSQL, mySQL, Oracle, postgresQL, Sapdb, Sybase. Na stronie <http://db.apache.org/ojb/features.html> znajduje się więcej informacji na temat możliwości OJB.

Podsumowując ten podrozdział można stwierdzić, że najważniejsze różnice przyjętego modelu w stosunku do modelu zaproponowanego pierwotnie przez autorów testu OO7 są następujące:

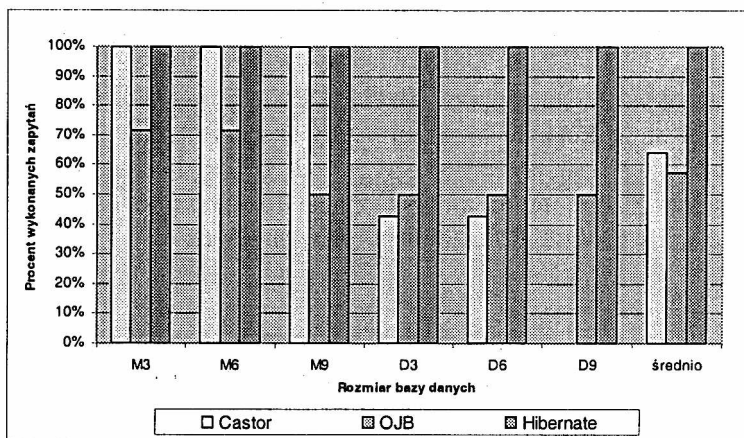
- implementacja nie zawiera abstrakcyjnej klasy Zestaw,
- data jest typu `java.util.Date` zamiast `Integer`,
- wszędzie, gdzie zaproponowane było użycie łańcucha jako tablicy znaków (`char[]`), używano typu `java.lang.String`,
- używano standardowych kolekcji Javy, np. `ArrayList`.

3. OPIS EKSPERYMENÓW

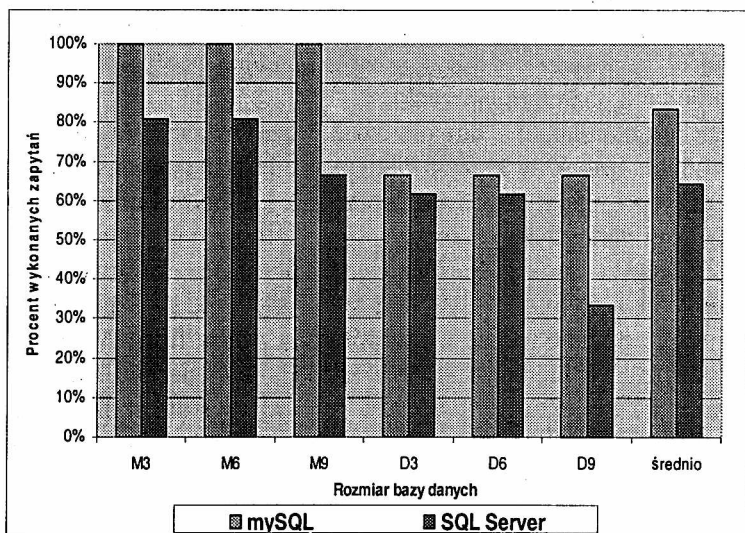
Do przeprowadzenia testów wybrano środowisko Eclipse 3.1M5 z JRE w wersji 1.5. Serwer aplikacji stanowił Apache Tomcat w wersji 5.0.27. Zarówno badania, jak i implementacja prowadzone były w systemie Windows XP Professional. Wykorzy-

stano MySQL 4.0.17, Microsoft SQL Server 2000, Castor 0.9.6, Hibernate 3.0 oraz OJB 1.0.3. Ponieważ każdy test uruchamiany był kilka razy, przy wyniku umieszczamy informację, czy wynik pochodzi ze startu zimnego (pusty bufor), czy gorącego (bufor pełny).

Rysunek 2 ilustruje procent pomyślnie przeprowadzonych zapytań z podziałem na narzędzia ORM.



Rys. 2. Procent pomyślnie wykonanych zapytań dla wybranych narzędzi ORM



Rys. 3. Procent pomyślnie wykonanych zapytań w zależności od RSZBD

Dla małych baz testowych zawierających 58 tys. obiektów, 88 tys. obiektów i 118 tys. obiektów (oznaczonych odpowiednio M3, M6 i M9) Castor i Hibernate pozwoliły przeprowadzić wszystkie zaplanowane eksperymenty. OJB natomiast pozwoliło na przeprowadzenie jedynie od 50 do 70% testów. W przypadku dużych baz zawierających 508 tys. obiektów, 808 tys. obiektów i 1 mln 108 tys. obiektów (oznaczonych odpowiednio D3, D6 i D9) jedynie z systemem Hibernate udało się przeprowadzić wszystkie zapytania. Castor oraz OJB poprawnie pozwoliły wykonać mniej niż połowę zadań. Ostatnia seria danych pokazuje średni wynik dla 6 baz.

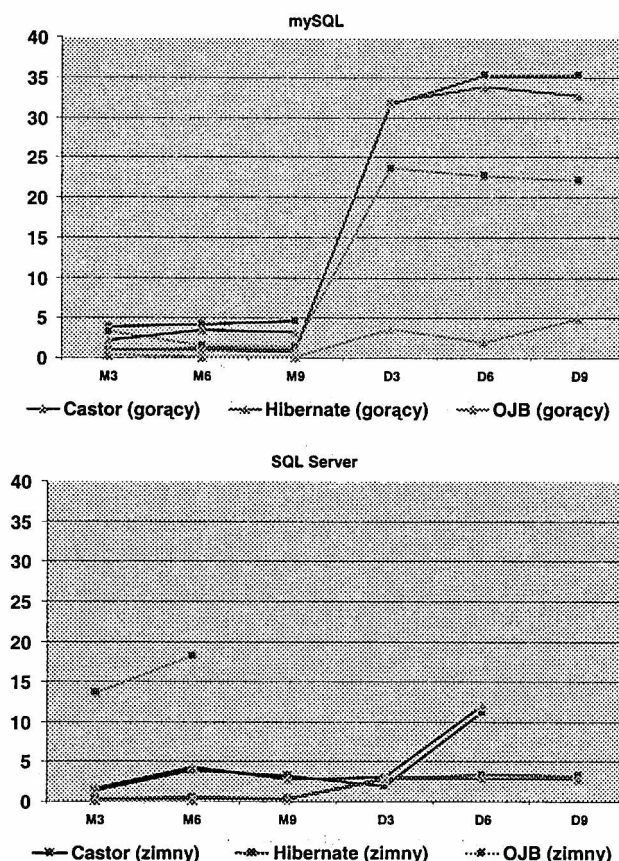
Patrząc na tę samą statystykę pod kątem RSZBD (rysunek 3) można zauważyć, że MySQL pozwolił na wykonanie większej liczby eksperymentów i to dla każdej bazy testowej. W sumie udało się przeprowadzić ponad 70% zaplanowanych testów, co jest wynikiem i tak bardzo dobrym zważywszy na to, że są to często eksperymenty o charakterze o wiele bardziej obiektowym, niż relacyjnym.

3.1. ZAPYTANIE 1. DOKŁADNE WYSZUKIWANIE

Przed wykonaniem zapytania 1 wygenerowano 10 identyfikatorów *części atomowych*. Dla każdego z wylosowanych identyfikatorów wykonywane jest zapytanie, które wyszukuje *część atomową* o zadanym identyfikatorze.

Zapytanie takie powoduje, że system musi przejrzeć wszystkie *części atomowe* i porównywać dla nich dwie wartości liczbowe, jakimi są identyfikatory. Dla każdej znalezionej *części atomowej* wywoływana jest metoda *nieNieRob()*. Zwracana jest liczba przetworzonych *części atomowych*.

Eksperyment ten dla żadnej z baz nie trwał dłużej niż 36 sekund. Jest to pierwsze z przeprowadzonych badań, w którym najlepszy nie jest system Hibernate. Dla MySQL trochę lepszy przy pustym i dużo lepszy przy pełnym buforze okazał się OJB. Niestety na SQL Serwerze było już inaczej – przy zimnym buforze OJB potrzebowało więcej czasu. Zapytanie 1 pokazało również, że Hibernate o wiele lepiej współpracuje z SQL Serwerem niż z mySQLeM. Wyniki tego zapytania, szczególnie dla większych baz, okazały się być o wiele lepsze we współpracy z bazą firmy Microsoft. Mogą istnieć dwie przyczyny. Po pierwsze Hibernate mógł wygenerować zapytanie bardziej zoptymalizowane dla dialektu SQL Servera (np. odpowiedni rodzaj złączenia), bądź też to sama baza danych działała po prostu lepiej dla większego zbioru danych testowych.

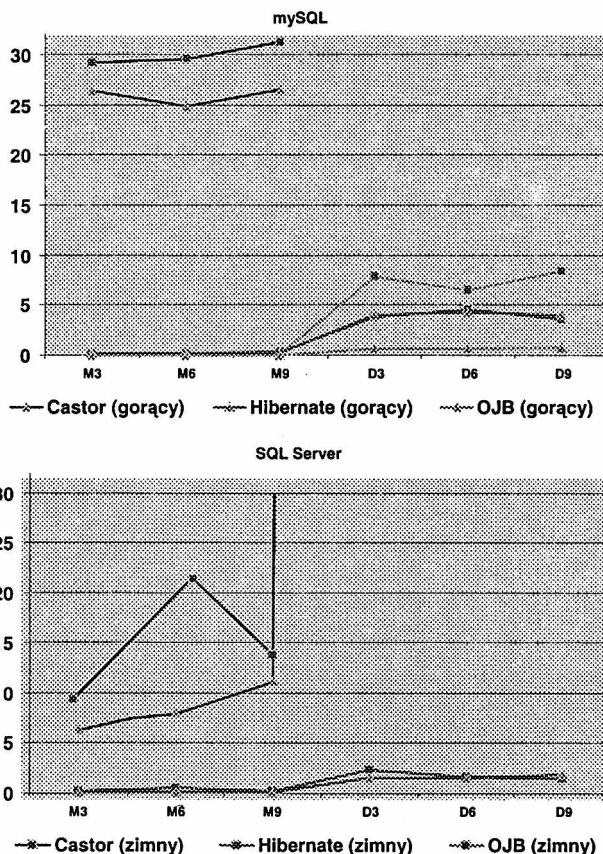


Rys. 4. Wyniki zapytania 1

3.2. ZAPYTANIE 2. WYSZUKANIE 1% CZĘŚCI ATOMOWYCH

Zapytanie 2 ma za zadanie wyszukać 1% części atomowych z datą od 1 stycznia 1997 do 31 grudnia 2005 roku. Dla znalezionych części atomowych wywoływana jest pusta metoda *nicNieRob()*. Zapytanie zwraca liczbę znalezionych obiektów.

Wyszukiwanie 1% części atomowych dla mniejszych (oznaczonych jako M3, M6 i M9) baz testowych zajęło najmniej czasu OJB. Dla większych baz przy pustym buforze wygrał Hibernate, jednak, gdy bufor był już pełny, system OJB okazał się najszybszy. Castor w każdym przypadku wypadł kilka razy gorzej od pozostałych systemów.

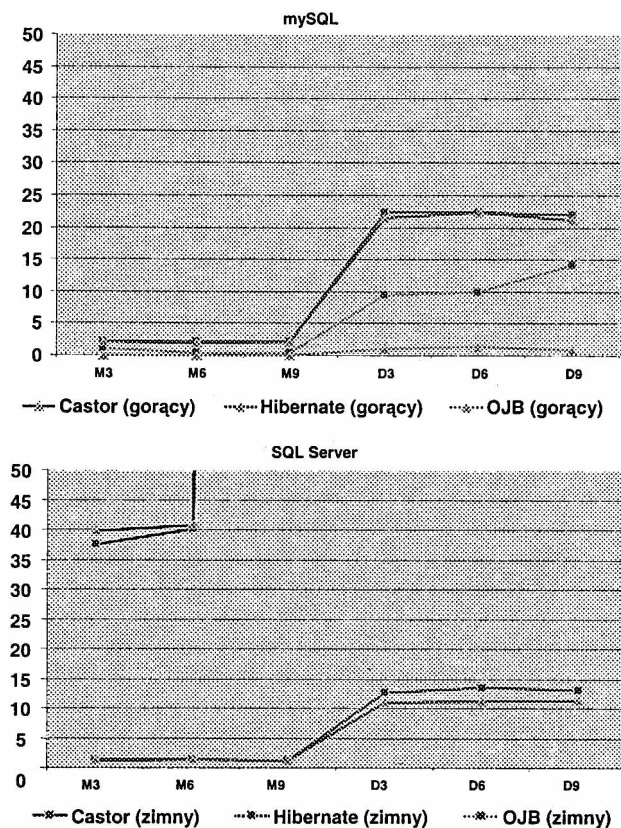


Rys. 5. Wyniki zapytania 2

3.3. ZAPYTANIE 3. WYSZUKANIE 10% CZĘŚCI ATOMOWYCH

Zapytanie 3. ma za zadanie wyszukać 10% części atomowych. Postępując analogicznie jak w zapytaniu 2., zapytanie zwraca liczbę znalezionych obiektów.

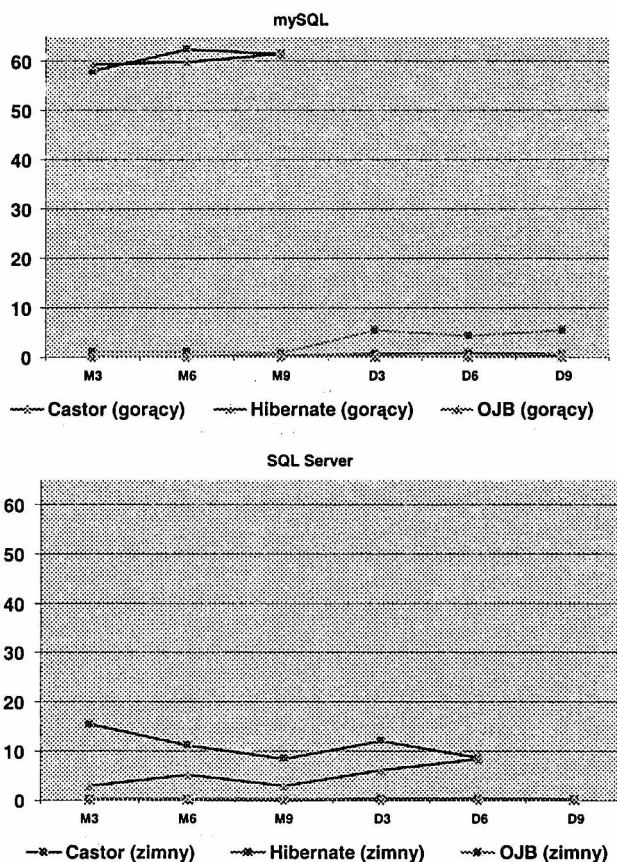
Zapytanie 3. wyszukuje około 10% części atomowych. W stosunku do poprzedniego eksperymentu, czas wzrósł około 4–5 razy dla Hibernate oraz dla Castora. System OJB okazał się nieco mniej wrażliwy na kryterium wybierające 10 razy więcej obiektów z dużej bazy danych (D3, D6 i D9). W jego przypadku szybkość zmalała mniej niż dwukrotnie. OJB było zdecydowanie szybsze od pozostałych systemów, a jego przewaga była jeszcze większa, gdy bufor stawał się pełny.



Rys. 6. Wyniki zapytania 3

3.4. ZAPYTANIE 4. PRZEGLĄDANIE ŚCIEŻKI

Zapytanie 4. rozpoczyna się wygenerowaniem tytułu 10 *dokumentów*. W kroku kolejnym wyszukiwane są *zestawy podstawowe*, których przynajmniej jedna z prywatnych (niedzielonych) części złożonych posiada połączenie z *dokumentem* o zadanym tytule. W ostatnim kroku, dla każdego ze znalezionych zestawów, wywoływana jest metoda pusta *nicNieRob()*, i zwracana jest liczba zestawów.

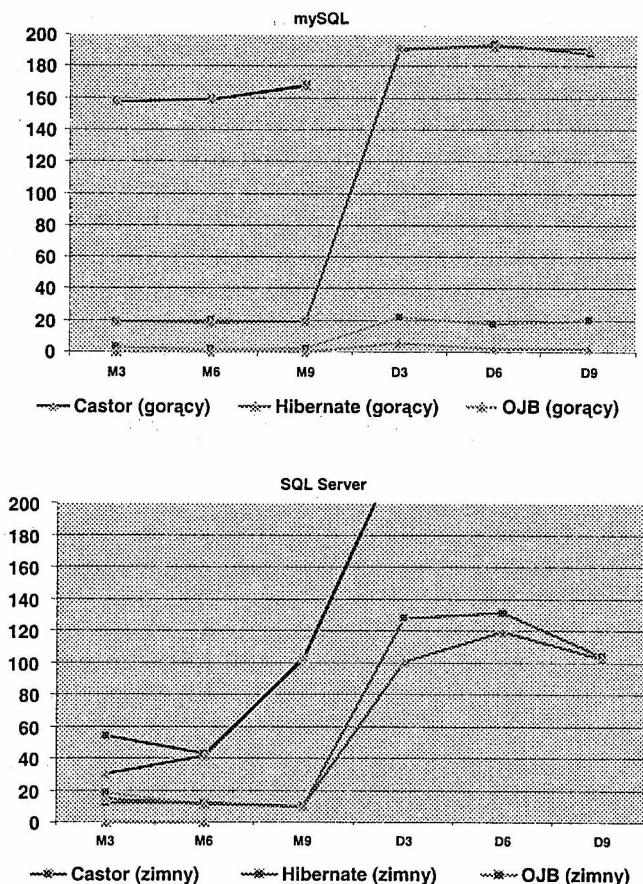


Rys. 7. Wyniki zapytania 4

Eksperyment ten nadaje się w szczególności do badania wydajności indeksu ścieżkowego w obiektowych bazach danych. Indeks taki pozwala podczas wykonywania zapytania nie pobierać wszystkich obiektów znajdujących się na jego ścieżce poszukiwań. W przypadku baz relacyjnych, jest to zapytanie zwracające stosunkowo niewielką liczbę obiektów. Najwięcej czasu na wykonanie eksperymentu potrzebował Castor. Także w jego przypadku zauważyć można największą różnicę między bazą mySQL, a SQL Server, gdzie zapytanie 4 trwało ponad 3 razy krócej.

3.5. ZAPYTANIE 5. WYSZUKANIE 100% CZĘŚCI ATOMOWYCH

Zapytanie 5. stanowi dopełnienie zapytań 2. i 3. Zapytanie 2. i 3. wyszukiwały odpowiednio 1% oraz 10% wszystkich części atomowych, w przypadku zapytania 5. wyszukiwane są wszystkie. Dla każdej odwiedzonej części atomowej wywoływana jest metoda *nicNieRob()*.



Rys. 8. Wyniki zapytania 5

To pierwszy eksperyment, który wykazuje, że któryś z systemów jest wyraźnie szybszy od Hibernate. Różnica między zapytaniami 2. i 3. jest taka, że w zapytaniu 5. zwracane są wszystkie części atomowe. OJB okazał się jedynym systemem, który jest prawie niewrażliwy na liczbę wyszukanych obiektów. Po raz kolejny, jedynie OJB

wydaje się prawidłowo współpracować z buforem. Tylko w jego przypadku zauważyć można wzrost szybkości przy kolejnych próbach.

3.6. ZAPYTANIE 6. ZAPYTANIE AD HOC O DOKUMENT

Zapytanie 6. to zapytanie typu ad hoc. Dane w bazie testowej wygenerowane są w taki sposób, że część *atomowa* posiada atrybut typu *Integer* z losową wartością. Teraz wykorzystywany jest ten fakt i z każdej części *atomowej* pobierana jest wartość tego atrybutu. Dalej następuje próba wyszukania *dokumentu* o identyfikatorze równym tej wartości. W wyniku działania zapytania zwracana jest liczba dokumentów, które zostały w ten sposób znalezione.

Zapytanie to ma typowy charakter obiektowy. Wymaga dla najmniejszej bazy M3 ponad 10 000 zapytań sparametryzowanych, z których każde zwraca duży obiekt *dokumentu*. Niestety dla żadnego z systemów nie udało się doczekać na wynik. Dowodzi to, że istnieją operacje, które nie będą mogły zostać w akceptowalnym czasie przetworzone przez żadne z testowanych narzędzi ORM.

4. PODSUMOWANIE

Podczas wykonywania tej pracy, przeprowadzono bardzo dużo eksperymentów. Wnioski z nich płynące nie wyznaczyły żadnego zdecydowanego lidera, co jest zgodne z oczekiwaniami autorów. Każde z badanych narzędzi cechuje się czymś innym. Spoglądając na poszczególne wyniki można zauważyć, że w OJB świetnie działa bufor pamięci podręcznej, Hibernate bardzo dobrze radzi sobie nawet z dużymi bazami danych, a Castor jest konkurencyjny jedynie dla najmniejszych baz. Wszystkie eksperymenty trwały ponad 70 godzin, 25% testów stanowią nawigacje i zapytania.

Uzyskane wyniki są uniwersalne. Jednak dynamika zmian narzędzi ORM jest bardzo duża i wyniki mogą okazać się aktualne tylko przez bardzo krótki czas, bo kolejna wersja danego narzędzia może działać zupełnie inaczej, jak to miało miejsce w przypadku systemu Castor.

W zdecydowanej większości eksperymentów pamięć podręczna działała na tyle źle, że niejednokrotnie jej obecność nie odbijała się korzystnie na wynikach. Wyjątkiem był system OJB, w którym pamięć podręczna działa najlepiej. Pozostałe narzędzia albo zbyt często porównywały zawartość bufora ze stanem faktycznym w bazie danych, albo też zbyt rzadko z niego korzystały.

Ważnym wnioskiem jest fakt, że gdy system jest zaimplementowany z użyciem któregośkolwiek z narzędzi mapujących i działa z konkretną relacyjną bazą danych, to przeniesienie go by działał z inną bazą danych nie stanowi z reguły problemu.

Wystarczy tylko zainstalować sterownik oraz zmienić informacje w pliku konfiguracyjnym.

Implementacja testu okazała się być prosta jedynie z teoretycznego punktu widzenia. Próba kodowania dla każdego z wymienionych narzędzi kończyła się napotkaniem nieprawidłowości w funkcjonowaniu. Świadczy to o tym, że narzędzia ORM są jeszcze w początkowej fazie rozwoju i posiadają wiele błędów. W trakcie badań zostały zgłoszone 4 błędy do autorów, a w wielu przypadkach praca wymagała szukania innego rozwiązania ze względu na niezaimplementowane funkcje.

Jak słusznie zauważają Szpunar i Wróblewski [SZPUNAR], rzutowanie modelu obiektowego na relacyjny model danych jest jedną z czynności, która dzięki automatyzacji skraca czas i wysiłek, jaki musi włożyć programista w swoją pracę. Z drugiej jednak strony rozwiązanie takie musi mieć pewien niekorzystny wpływ na wydajność działania aplikacji. Spostrzeżenie to zostało empirycznie udowodnione, ponieważ różnice między badanymi narzędziami były miejscami dość znaczne.

Problematyka związana z narzędziami ORM nie została w pracy w pełni wyczerpana. Dalsze badania mogłyby dotyczyć w szczególności:

1. Modyfikacji parametrów narzędzi ORM oraz analizy wpływu tych zmian na wyniki poszczególnych testów, optymalnego doboru tych parametrów (wielkość bufora, rodzaj buforowania, sterownik JDBC itp.).
2. Sprawdzenia, ile i jakiego rodzaju zapytań generują narzędzia dla tego samego eksperymentu (można to łatwo sprawdzić włączając opcję podglądu generowanego kodu SQL oraz liczyć liczbę poleceń SELECT, UPDATE oraz INSERT).
3. Opracowania i zaimplementowania testu, który zbadałby transakcyjne cechy, jakie oferowane są przez narzędzia ORM (blokowanie, brudny odczyt itp.). Testy mogłyby być przeprowadzone w środowisku rozproszonym.
4. Implementacji uniwersalnego testu OO7 dla narzędzi zgodnych ze specyfikacją JDO.
5. Sprawdzenia wydajności działania narzędzi ORM z plikami XML i porównania wyników z przedstawionymi w niniejszej pracy. Niektóre z narzędzi już teraz oferują wsparcie do danych składowanych w plikach XML. Z drugiej strony istnieje odmiana testu OO7, która opisywana jest jako XOO7 [XOO7] i służy ona do badania wydajności obiektowego dostępu do repozytoriów XML.

LITERATURA

- [ALUR] Alur D., *Core J2EE. Wzorce projektowe*. Gliwice, Helion, 2004.
- [ATKINSON] Atkinson M., *An orthogonally persistent Java*. ACM SIGMOD Record, 1996.
- [CAREY] Carey M., *The OO7 Benchmark*. Computer Science Tech. Report, University of Wisconsin-Madison, 1994.
- [JORDAN] Jordan M., *A comparative Study of Persistence Mechanisms for the Java Platform*, Sun SMLI TR, 2004.

- [KNABEL] Knabel B., *Uniwersalny dostęp do baz danych w środowisku J2EE*. Praca magisterska, Politechnika Wrocławska, Wrocław 2005.
- [KROL] Król D., Kojs Ł., *Wyszukiwanie danych w obiektowej bazie danych*. Prace Naukowe Wydziałowego Zakładu Informatyki Politechniki Wrocławskiej, Bazy Danych, Zeszyt 4, 2003, 19–40.
- [KRYNSKI] Kryński A., *Wyszukiwanie danych w obiektowej bazie danych ObjectStore*. Praca magisterska, Politechnika Wrocławska, 2001.
- [SZPUNAR] Szpunar Z., Wróblewski A., *Rzutowanie modelu obiektowego do relacyjnego modelu danych*. Prace Naukowe Instytutu Informatyki Stosowanej Politechniki Wrocławskiej, Bazy Danych, Zeszyt 6, 2005, 51–66.
- [XOO7] *The XOO7 Benchmark*. <http://www.comp.nus.edu.sg/~ebh/XOO7.html>, 2005.

QUERY EVALUATION FOR DATABASES ON THE J2EE PLATFORM

Most modern development projects use object technology such as Java to combine the application software and relational databases to store the data. There is an impedance mismatch between these two technologies. This problem can be overcome by mapping objects to relational tuples. There are plenty of tools helpful in developing persistent classes. For three of them (Castor, Hibernate and OJB) was implemented OO7 test – known benchmark designed to provide a comprehensive profile of the performance of an object-oriented database management system. This work contains results of OO7 running on Castor, Hibernate and OJB systems.

Kazimierz FRĄCZKOWSKI*
Zygmunt MAZUR**

SOA – ARCHITEKTURA ZORIENTOWANA NA USŁUGI

Współczesne przedsiębiorstwa często wykorzystują infrastrukturę informatyczną łączącą technologie różnych dostawców, kosztowną oraz trudną do utrzymania i rozwijania. Dla zasobów danych, które są często nieustrukturyzowane, są określane specyfikowane usługi czyli grupy zadań umożliwiających realizację wymagań biznesowych. Projektowanie i implementację takich systemów umożliwia wykorzystanie architektury zorientowanej na usługi (SOA), z zastosowaniem której wiąże się duże nadzieje. W artykule omówiono tę architekturę oraz oddziaływanie SOA na środowisko Web.

1. WPROWADZENIE

Projekty informatyczne wymagają ciągłej opieki i dopasowania do bieżących potrzeb, usuwanie tzw. bieżących ograniczeń oprogramowania stanowi ok. 70% kosztów utrzymania systemu. Ponadto w czasie tych zmian, klienci żądają stabilności platformy i zachowania sprawdzonych komponentów. Integracja projektu, którego złożoność wzrasta o nową aplikację stymulowaną nowymi potrzebami stoi w sprzeczności z oczekiwaniami redukcji kosztów działalności operacyjnej w części której *nowa usługa*, ma stanowić poprawienie nowych usług dla biznesu. W czasie eksploatacji często ulegają zmianom założenia biznesowe i wymagania funkcjonalne, których uwzględnienie w projekcie wymaga ponownej analizy i przeprojektowania modelu bazy danych i aplikacji bazodanowej. Dlatego też niezwykle istotne jest zorientowanie technologii informatycznych na elastyczne wprowadzanie zmian stymulowanych potrzebą realizacji nowych usług. W wielu przedsiębiorstwach wykorzystywane jest heterogeniczne środowisko IT (ang. *Information Technology*), pochodzące od różnych dostawców, w którym zachodzi konieczność eksploatacji oraz utrzymania zduplikowanych danych oraz funkcji biznesowych (usług), występujących w różnych aplikacjach i systemach a które można byłoby skonsolidować do niewielkiej liczby współ-

* Instytut Informatyki Stosowanej, Wydział Informatyki i Zarządzania Politechniki Wrocławskiej, 50-370 Wrocław, Wybrzeże Wyspiańskiego 27, kazimierz.fraczkowski@pwr.wroc.pl

** Instytut Informatyki Stosowanej, Wydział Informatyki i Zarządzania Politechniki Wrocławskiej, 50-370 Wrocław, Wybrzeże Wyspiańskiego 27, zygmunt.mazur@pwr.wroc.pl

użytkowanych usług. Pozwoliłoby to wyeliminować usługi nadmiarowe i obniżyć koszty utrzymania systemów poprzez zapewnienie jednego punktu wprowadzania zmian w warstwie aplikacji (i jednokrotnego wprowadzania danych). W tym celu można np. zastosować metodę ABC (ang. *Activity-based Costing*) zarządzania kosztami i efektywnością [CooKap 2000], według której można wywieść tezę sformułowaną przez Peter F. Drucker'a, że *Jedynym efektywnym sposobem redukcji kosztów jest wyeliminowanie działań. Próby ograniczenia kosztów rzadko bywają skuteczne. Nie ma większego sensu próba wykonywania tanio tego, co w ogóle nie powinno być wykonywane.* Przełomem w technologiach informatycznych, które sekundują tym wyzwaniom związanym z efektywnością zarządzania zmianami i skutecznym zarządzaniem projektami informatycznymi jest architektura SOA (ang. *Service Oriented Architecture*), która jest zorientowana na usługi, upraszcza środowiska informatyczne i aplikacji oraz eliminuje zduplikowane działania.

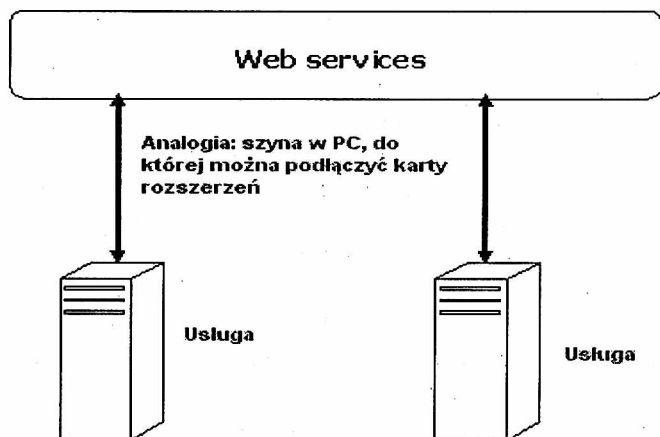
2. ODDZIAŁYWANIE SOA NA ŚRODOWISKO WEB

W ogólnym modelu SOA Web Service (WS) jest jednostkowym elementem logiki aplikacji, dostarczającym dane i usługi innym aplikacjom. Aplikacje komunikują się z WS za pomocą standardowych protokołów i formatów danych, niezależnie od sposobu implementacji danego WS. Najlepsze elementy oprogramowania opartego na komponentach oraz sieci WWW stanowią fundament modelu programistycznego środowiska .NET. Aplikacje korzystające z WS mogą być zaimplementowane w dowolnym języku i na dowolnej platformie, o ile tylko są w stanie wysyłać i odbierać komunikaty w formacie zdefiniowanym przez interfejs usługi. Ogólny model SOA przedstawia rysunek 1.

Web Services są to usługi sieciowe pracujące w architekturze wielowarstwowej, udostępniane przez standardowe protokoły internetowe i umożliwiają np. pobieranie danych meteorologicznych, wyników sportowych, kursów walut czy notowań giełdowych. Usługi oferowane przez Web Services mogą być zdalnie wywoływane przez programy tworzone w różnych technologiach (strony WWW, typowe aplikacje 4GL). Podstawowe cechy WS to:

- mechanizm niezależny od systemu operacyjnego i używanego języka programowania,
- SOAP (ang. *Simple Object Access Protocol*) – oparty o XML protokół do przysyłania wiadomości między klientem a serwerem,
- UDDI (ang. *Universal Description, Discovery and Integration*) czyli rejestr usług oraz mechanizm wyszukiwawczy oparty o XML (pewnego rodzaju „książka telefoniczna”), umożliwiają tworzenie oraz wyszukiwanie usług WS. Rejestr jest dostępny jako zasób dzielony np. w postaci odrębnego Web Service, udostępnia także odpowiednie API poprzez protokół SOAP, który służy do publikowania i odpytywania Web Service.

- udostępnianie i opisywanie usług poprzez mechanizm WSDL (ang. *Web Services Description Language*) – bazujący na XML dokument opisujący funkcjonalność danej usługi, lokalizację i sposób wywołania – dokładniej opisane są udostępnione metody, parametry, protokoły i tryb wywołania,
- dostęp w ramach Internetu (intranetu) przez standardowe protokoły – http.



Rys. 1. Ogólny model SOA

2.1. ŚRODOWISKO SYSTEMOWE I APLIKACYJNE

Rozważając architekturę SOA nasuwa się pytanie, czy można ją wykorzystać do istniejących systemów informatycznych, a jeśli tak, to jakie zmiany należałoby wprowadzić, aby dostosować je do SOA i jaki byłby koszt i czas realizacji takiego przedsięwzięcia.

Mając listę usług dostarczających określone dane, dostawca usług wspólnie z pracownikami z firmy ustala, jakie zadania ma wykonywać nowa aplikacja i według jakich procedur czyli zapoznaje się z logiką biznesową. Znając logikę biznesową oraz możliwości usług już zainstalowanych, dostawca buduje nową aplikację i obsługę interfejsu wymiany danych biorąc pod uwagę bezpieczeństwo, prostotę oraz obciążenie sieci i serwera komunikacyjnego. Architektura SOA może być wprowadzana stopniowo, początkowo dla wybranych obszarów, a następnie, wraz z rozwojem firmy, rozbudowywana, przyczyniając się tym samym do przekształcenia sposobu działania firmy w kierunku modelu „na żądanie” (ang. *on demand*). W przypadku, gdy model danych przedsiębiorstwa zdefiniuje wymagania na synchronizację danych pomiędzy systemami (kilka systemów utrzymuje takie same dane), celowe jest zastosowanie brokera integracyjnego realizującego automatyczną synchronizację danych. W przypadku, gdy na mapie usług przedsiębiorstwa pewne usługi występować będą w kilku

obszarach funkcjonalnych lub na granicy obszarów, celowe może być zarekomendowanie narzędzia klasy BPM (ang. *Business Process Management*), które wspomagać będzie nadzór nad realizacją procesów, wspieranych przez więcej niż jedną aplikację. W przypadku gdy użytkownicy korzystają w swojej pracy z więcej niż jednej aplikacji, celowe wydaje się zbudowanie jednego interfejsu użytkownika przy pomocy mechanizmów stosowanych w budowie portali.

Docelowy model aplikacji opisuje zarówno wszystkie aplikacje posiadane obecnie oraz zalecane do wdrożenia, jak również wszystkie zidentyfikowane komponenty dodatkowe infrastruktury takie jak centralny katalog użytkowników czy przywoływane narzędzia typu portal, broker integracyjny.

SOA, architektura ukierunkowana na usługi, jest złożona z komponentów i połączeń, które zapewniają wysoki stopień zgodności operacyjnej i przezroczystości lokalizacji. Architektura ta stanowi podstawę do integracji i współpracy z podmiotami zewnętrznymi przy użyciu otwartych standardów (np. usług sieciowych opartych na XML). Usługi komunikują się na zasadzie przekazywania danych i koordynacji usług.

W pracy [SkSiNi 2005] autorzy przedstawiają Wirtualny System Archiwizacji (VSS – *Virtual Storage System*) oparty na technologii SOA. Technologia ta jest odpowiednia przy tworzeniu bardzo złożonych, rozproszonych systemów w środowisku rozproszonym i heterogenicznym. Z tego powodu architektura SOA z powodzeniem może być wykorzystywana do tworzenia systemów gridowych, rozwijanych z reguły przez niezależne zespoły geograficznie rozproszone w różnych ośrodkach. Architektura SOA pozwala na precyzyjne rozdzielenie zakresu prac i odpowiedzialności za poszczególne usługi co z kolei ma pozytywny wpływ na zarządzanie dużymi projektami.

2.2. ŚRODOWISKO WEB ORAZ NARZĘDZIA

Nad rozwojem standardów związanych z usługami Web czuwa *The Web Services Interoperability Organization* (WS-I). Celem WS-I (www.ws-i.org) jest m.in. promowanie technologii Web Services oraz możliwości wykorzystania tej technologii przy współdziałaniu różnych platform, systemów operacyjnych i języków programowania, nadzorowanie opracowywania standardów w tym zakresie oraz koordynowanie prac tak, by produkty wytworzone przez różnych producentów korzystających ze standardu mogły ze sobą współpracować. Podstawową cechą współdziałania (ang. *interoperability*) jest wymiana danych pomiędzy aplikacjami napisanymi w różnych językach programowania i działającymi na różnych platformach. Wymiana danych musi być niezawodna i musi umożliwiać przekazywanie danych typów prostych i złożonych występujących być może tylko w jednym z użytych języków programowania.

Organizacja WS-I skupia zarówno wielkich producentów związanych z Web-usługami takimi jak BEA, IBM, Sun czy Microsoft, oraz małe firmy programistyczne.

WS-I opracowała tzw. profil podstawowy (ang. *Basic Profile*) usług Web czyli zestaw standardów obejmujących usługi podstawowe m.in.:

- SOAP (ang. *Simple Object Access Protocol*) czyli protokół wywoływania zdalnego dostępu do obiektów wykorzystujący język XML do kodowania wywołań i protokół HTTP do ich przenoszenia,
- WSDL (ang. *Web Services Description Language*) – język do definiowania usług webowych oparty na języku XML (opracowany przez Microsoft i IBM),
- UDDI (ang. *Universal Description, Discovery and Integration* – Uniwersalny Opis, Odkrywanie i Integracja) – usługa umożliwiająca oprogramowaniu automatycznie wykrywać i integrować Web Services w sieci Internet, zawiera tzw. *white pages*, czyli adresy i kontakty, *yellow pages* (klasyfikacje przemysłowe) i *green pages* (opisy usług).

Obecnie prace organizacji koncentrują się na tworzeniu elementów infrastruktury usług Web, które pozwoliłyby odwzorować skomplikowane mechanizmy komunikacji biznesowej. Poniżej zamieszczamy omówienie kilku najważniejszych zagadnień, nad którymi pracuje WS-I.

Dość długo słabo opracowanym elementem w WS było bezpieczeństwo (WS-security). Obecnie WSE 3.0 (ang. *Web Services Enhancements*) jest częścią dotyczącą bezpieczeństwa i wspiera m.in. szyfrowanie i integralność przesyłanych wiadomości, różne sposoby zabezpieczeń, zarządzanie tokenami, mechanizmy uwierzytelniania oparte na rolach oraz zintegrowane z Windows. WS-Policy jest częścią WSE 3.0, która pozwala w deklaratywny sposób określić wymagania dotyczące podpisywania oraz szyfrowania wychodzących i przychodzących wiadomości SOAP. WSE 3.0 upraszcza bezpieczne programowanie i wdrażanie Web Services. Pozwala deweloperom i administratorom zarządzać polityką bezpieczeństwa Web Services pracujących w oparciu o platformę .NET Framework 2.0. Przy pomocy WSE 3.0 komunikacja pomiędzy Web Services może być podpisana i szyfrowana przy pomocy żetonów Kerberos, certyfikatów X.509 i innych technik opartych o żetony binarne i XML-owe.

W poszukiwaniu jednolitego warsztatu wspomagającego wdrażanie architektury SOA, firma Oracle na przełomie lat 2005/06 zintegrowała kilka istniejących produktów warstwy pośredniczącej Fusion w pakiet Oracle SOA Suite. Produkty te zostały zintegrowane i mają wspólne narzędzia do instalowania i zarządzania. Według zapewnień Oracle, zestaw ten, wyposażony w produkty BPEL (ang. *Business Process Execution Language*) i ESP (ang. *Encapsulating Security Payload* – protokół zapewniający integralność i poufność), ma zapewniać kompleksowe podejście do wdrażania architektury SOA. Komponenty wchodzące w skład zestawu to: Oracle BPEL Process Manager – do odwzorowywania przepływu procesów biznesowych, Oracle Enterprise Service Bus – do łączenia istniejących systemów IT i partnerów biznesowych, Oracle Web Services Manager – konsola do ustawiania polityk dla Web Services, Oracle Rules Engine – do definiowania i zarządzania regułami biznesowymi.

Pozostałymi produktami warstwy pośredniczącej w zestawie są: Oracle Business Activity Monitoring – zapewniający wgląd w operacje biznesowe, Oracle Enterprise

Manager – do zarządzania aplikacjami zorientowanymi na usługi oraz Oracle Jdeveloper 10g – środowisko deweloperskie aplikacji Java.

Jak wynika z informacji podanej 03 marca 2006 r. na stronie WWW firmy Oracle, w najbliższym czasie firmy Hewlett Packard i Oracle zamierzają wprowadzić na rynek kompletną i w pełni zintegrowaną technologię, umożliwiającą klientom tworzenie i wdrażanie architektur SOA oraz zarządzanie nimi.

Obecnie firma HP zatrudnia ponad 6,5 tys. osób specjalizujących się we wdrażaniu architektury SOA i oferuje usługi oraz oprogramowanie do zarządzania zaspokajające potrzeby klientów na każdym etapie cyklu eksploatacji architektury usługowej – od opracowania koncepcji SOA poprzez zarządzanie cyklem eksploatacji, zarządzanie usługami, monitoring, audyt i analizy, aż po umowy o poziom usług (SLA). Firma HP rozwija architekturę i rozwiązania dla zarządzanych i bezpiecznych środowisk SOA poprzez łączenie usług, oprogramowania do zarządzania oraz najlepszych w branży serwerów i systemów pamięci masowej zgodnie z wymaganiami architektury SOA.

Firmy HP i Oracle oferują środowiska SOA dla najpopularniejszych platform usługowych warstwy pośredniej i infrastruktury, takich jak J2EE, .NET, UNIX, Windows, Linux, Open VMS i Non-Stop, wdrażane na urządzeniach klienckich, serwerach, w sieciach i systemach pamięci masowej. Środowiska te umożliwiają klientom wykorzystanie dotychczasowych inwestycji w informatyzację.

2.3. MODELE WYTWÓRCZE SYSTEMÓW INFORMATYCZNYCH A SOA

Wyróżnia się dwa zasadnicze cykle życia oprogramowania: kaskadowy i iteracyjny (zwany również spiralnym lub przyrostowym). Istotą procesu kaskadowego jest podział całego przedsięwzięcia ze względu na czynności na pewne zadania (np. analiza, projektowanie, implementacja, testowanie), które powinny być zrealizowane w określonym czasie. Realizacja kolejnego zadania rozpoczyna się po zakończeniu zadania wcześniejszego. W podejściu iteracyjnym planowany okres realizacji przedsięwzięcia dzieli się na pewne podokresy, w których są wykonywane wszystkie czynności (analiza, projektowanie, implementacja, testowanie) w celu uzyskania systemu oferującego zaplanowane funkcjonalności. Na zakończenie, po pewnym okresie stabilizacyjnym, w którym usuwane są dostrzeżone błędy, następuje integracja całego systemu i szkolenie użytkowników. Często przy podejściu iteracyjnym stosuje się ramy czasowe, które ograniczają czas przeznaczony na wykonanie każdej iteracji.

Podejście kaskadowe zapewnia większą przewidywalność co do celu, zakresu, czasu i kosztów budowy systemu. Planowanie przedsięwzięcia może mieć charakter predykcyjny lub adaptacyjny. W planowaniu predykcyjnym w pierwszym etapie wykonuje się jak najdokładniej plan całego przedsięwzięcia a w etapie drugim stara się go wiernie realizować. W podejściu adaptacyjnym z góry zakłada się zmienność planów i wymagań i w zasadzie nigdy nie można powiedzieć, że prace przebiegają zgodnie

z planem. Początkowy plan jest raczej bardzo szerokim zarysem całego przedsięwzięcia. Możliwość realizacji podejścia adaptacyjnego zapewniają kontrakty określające koszt i czas wykonania projektu nie precyzujące jednak wszystkie funkcjonalności systemu (ewentualnie tylko te podstawowe).

„Internetowe” tempo pracy powoduje, że ludzie nie mają czasu pisać ani czytać (zarówno projektanci jak i użytkownicy) dokumentacji systemów informatycznych. Stąd popularność zwinnych (ang. *agile*) metodyk projektowania oprogramowania, które nie zakładają niezmienności wymagań przez cały czas trwania projektu i nie wymagają tworzenia dodatkowej dokumentacji projektu, przyjmując, że sam kod jest wystarczającą dokumentacją.

Teoretycy SOA uważają, że zwinne metodyki projektowania, które z natury są procesami adaptacyjnymi, są odpowiednie do prowadzenia projektów w modelu SOA.

Aby upowszechnić podejście SOA należy opracować pewne nowe standardy cyklu życia oprogramowania. Na rynku stabilne i sprawdzone aplikacje jak również infrastruktura zachowa status standardu – komponentem, z których będą „zestawiane” nowe usługi stosownie do dynamicznie pojawiających się potrzeb użytkownika systemu informatycznego. Zwiększy się jego rola i możliwości użytkownika końcowego w zakresie modyfikacji modeli biznesowych oraz budowy aplikacji aż do automatycznego generowania kodu włącznie, z wykorzystaniem MDA (ang. *Model Driven Architecture*). Według twórców tego rozwiązania, architektura kierowana modelem jest podejściem do procesu wytwarzania oprogramowania dla biznesu, bazującym na automatycznych narzędziach, mającym na celu budowanie modeli niezależnych od technologii i przekształcanie ich w efektywne implementacje [Booch 2004]. Architektura MDA czyli „kierowanie modelem” oznacza, że udostępnia środki pozwalające użyć modeli do sterowania sposobem rozumienia, projektowania, rozwijania, wdrażania i modyfikacji systemów konstruowanych za jej pomocą. Zwiększa bardziej niż kiedykolwiek znaczenie modeli w procesie wytwarzania oprogramowania, dostarcza także wytycznych w jaki sposób strukturalizować specyfikacje wyrażone w postaci modeli.

3. PODSUMOWANIE

Dotychczas bazy danych wykorzystywały głównie dane strukturalne a aplikacje bazodanowe realizowały określone zadania. Dziś często wykorzystuje się nieustrukturyzowane zasoby danych, dla których są określane specyfikowane usługi czyli grupy zadań umożliwiających realizację wymagań biznesowych. Projektowanie i implementację takich systemów umożliwia wykorzystanie architektury zorientowanej na usługi (SOA). Z zastosowaniem modelu SOA wiąże się duże nadzieje. Informatycy spodziewają się w przyszłości obniżenia kosztów wytwarzania systemów informatycznych i zwiększenia ich elastyczności. Początkowo z pewnością budowa aplikacji w zgodzie z wytycznymi SOA nie będzie tańsza niż obecnie ale po pewnym czasie na pewno przyniesie znaczne korzyści finansowe. Tak jak i w innych dziedzinach wytwarzania

nastąpiła specjalizacja a następnie wykorzystywanie i łączenie poszczególnych elementów tak i wydaje się, że w informatyce jest to możliwe do zastosowania. Podstawowym założeniem SOA jest dostarczenie użytkownikowi zintegrowanego interfejsu, za pomocą którego będzie mógł samodzielnie modelować i budować różnorodne aplikacje wykorzystujące aplikacje pochodzące z różnych środowisk. Tak zbudowana aplikacja będzie umożliwiała realizację wymaganych procesów biznesowych. Zaletą takich aplikacji będzie również wykorzystywanie jednolitego środowiska. Według firmy Gartner, do 2008 roku około 80% rozwijanych projektów będzie wykorzystywało architekturę SOA.

Podejście SOA wymaga większego zwrócenia uwagi na modelowanie procesów biznesowych, dlatego też w istotny sposób ulegnie zmianie tworzenie aplikacji i praca programistów. Kod musi być pisany w sposób uniwersalny, umożliwiający jego wielokrotne użycie. Dzięki MDA zwiększy się bardziej niż kiedykolwiek znaczenie modeli w procesie wytwarzania oprogramowania, dostarczy to także nowych wytycznych, w jaki sposób strukturalizować specyfikacje wymagań wyrażone w postaci modeli.

LITERATURA

- [Booch 2004] Booch G. et al., *MDA Manifesto*. MDA Journal, May 2004.
- [CooKap 2000] Cooper R., Kaplan R.S., *Zarządzanie kosztami i efektywnością*. Oficyna Ekonomiczna, Kraków 2000.
- [SkSiNi 2005] Skitał Ł., Słota R., Nikolov D., Kitowski J. *Grid Enabled Virtual Storage System Using Services Oriented Architecture*. W: *Software Engineering: Evolution and Emergin Technologies*, Edited by: Krzysztof Zieliński, Tomasz Szmuc, IOS Press, 2005.

SOA – SERVICE ORIENTED ARCHITECTURE

The contemporary enterprises often use information infrastructure connecting technologies from different suppliers, which is expensive and difficult to maintain and develop. For data (which are often unstructured), there are defined specific services, i.e. groups of tasks making it the realization of business requirements possible. The design and implementation of these systems is possible thanks to SOA (Services Oriented Architecture), which has a lot of hope connected with its use. This paper presents SOA and its influence for the Web environment.

*niedostrzegalne podpisy cyfrowe,
niedostrzegalne sygnowanie danych,
protokół IP*

Arkadiusz LIBER*

Jarosław DYBOWSKI*

KONSTRUKCJA NIEDOSTRZEGALNYCH KANAŁÓW KOMUNIKACYJNYCH Z WYKORZYSTANIEM PROTOKOŁU IP I ICH ZASTOSOWANIE W PODPISYWANIU DANYCH

W ramach pracy przedstawiono wyniki badań autorów w zakresie wykorzystania protokołu IP do transmisji dodatkowych informacji. Dodatkowe informacje mogą być wykorzystane do umieszczania podpisów cyfrowych lub do wytworzenia dodatkowego, niezależnego kanału transmisyjnego. Wyniki badań przedstawione w pracy są kontynuacją i rozszerzeniem badań opisanych w pracy [LIBER 04] oraz w pracach [ROWLAND 97], [ASHAN 02]. Rezultaty otrzymane w pracy mogą być wykorzystane do konstrukcji systemów wymiany danych z ukrytymi kanałami informacyjnymi. W szczególności zaś w nowoczesnych systemach baz danych.

1. WPROWADZENIE

Problematyka ukrywania dodatkowych informacji w protokołach komunikacyjnych jest przedmiotem intensywnych badań. Za przełomowe prace wskazujące możliwość umieszczania dodatkowych informacji w protokołach TCP/IP należy uznać prace Handela i Sandforda [HANDEL 96] oraz Rowlanda [ROWLAND 97]. Mimo istnienia znacznej liczby publikacji ukazujących możliwość konstrukcji ukrytych kanałów informacyjnych w środowisku sieciowym, publikacje dotyczące umieszczania niedostrzegalnego podpisu w takich kanałach należą do rzadkości [LIBER 04].

W pracy Handela i Sandforda [HANDEL 96] dokonano analizy możliwości konstrukcji ukrytych kanałów informacyjnych we wszystkich warstwach modelu ISO/OSI. Dla warstwy sieciowej i transportowej zaproponowano jedynie wykorzystanie w tym celu pól nagłówek TCP i IP zarezerwowanych na przyszłość. W swojej publikacji Rowland [ROWLAND 97] prezentuje aplikację wysyłającą dane do odbiorcy wyłącznie

* Instytut Informatyki Stosowanej, Wydział Informatyki i Zarządzania Politechniki Wrocławskiej,
50-370 Wrocław, Skwer Idaszewskiego 1, arkadiusz.liber@pwr.wroc.pl

przy pomocy pola „Identyfikator” nagłówka IP oraz pól „Numer kolejny” i „Numer potwierdzenia” protokołu TCP. W [ABAD 01] do przesłania dodatkowej informacji wykorzystano zjawisko kolizji bitów w polu „Suma kontrolna” nagłówka IP. W pracy [ASHAN 02] autor proponuje wykorzystanie pól „Identyfikator”, „Fragmentacja zabroniona” oraz pola „Wskaźnik pilnych danych”. Autorzy pracy [FISK 02] dokonują analizy możliwości wykrycia ukrytego kanału informacyjnego podając jako potencjalne miejsca ukrycia informacji pola „Opcje”, „Typ usługi”, „Fragmentacja zabroniona”, „Wypełnienie”, „Wskaźnik pilnych danych” oraz „Numer kolejny”. W pracy [GIFFIN 02] jako nośnik dodatkowej informacji proponuje się pole „Opcje” natomiast w pracy [MURDOCH 05] rozpatrzono możliwość użycia pól „Typ usługi”, „Opcje”, „Identyfikator” oraz „Numer kolejny”. W Polsce badania nad ukrytymi kanałami informacyjnymi w warstwie łącza danych prowadzi Krzysztof Szczypiorski [SZCZYP 03]. Autorzy Liber i Kosmulska w pracy [LIBER 04] proponują nowy sposób konstrukcji niedostrzegalnego podpisu, który dzięki wykorzystaniu grup symetrii podobnych do grup symetrii punktowej, jest niewrażliwy na błędy występujące w sieciach.

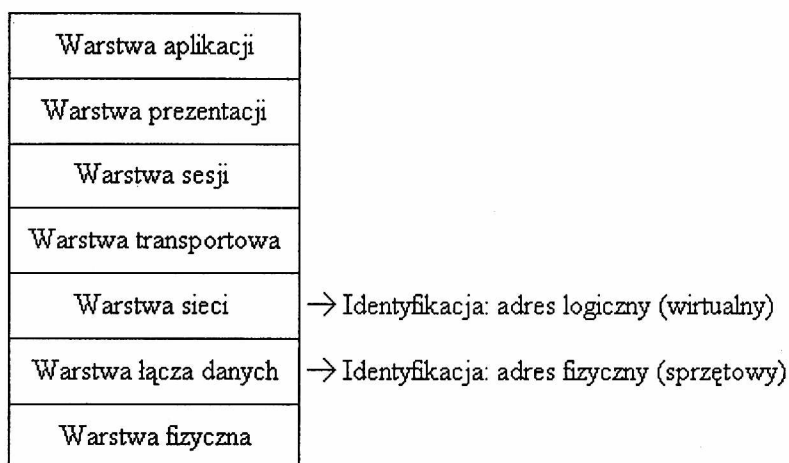
2. KONSTRUKCJA NIEDOSTRZEGALNYCH KANAŁÓW INFORMACYJNYCH PRZY WYKORZYSTANIU PROTOKOŁU IP

W ujęciu informatycznym, sygnatura (podpis) dowolnego rodzaju informacji (np. obrazu, sekwencji dźwięków, sekwencji obrazów, ramki sieciowej) spełnia dwie podstawowe funkcje:

- 1) zapewnia odbiorcę informacji, że pochodzi ona od autora, który ją podpisał (i jedynie od niego),
- 2) zapewnia odbiorcę informacji, że od momentu umieszczenia podpisu, podpisana informacja nie uległa zmianie.

Zgodnie z powyższym stwierdzeniem, w pracy podpis utożsamiany będzie z takim rodzajem informacji, który gwarantuje autentyczność (a w konsekwencji tego i niezaprzeczalność) oraz integralność innej informacji (podpisanej).

Za najmniejszą możliwą jednostkę mogącą być przedmiotem podpisu przyjęto pojedynczą ramkę protokołu sieciowego. Z racji mnogości i różnorodności stosowanych obecnie protokołów sieciowych, konieczny staje się wybór jednego lub kilku z nich, które będą przedmiotem analizy. Odnosząc się przykładowo do warstwowego modelu sieci ISO/OSI (rysunek 1), identyfikacja nadawcy jest możliwa już na poziomie protokołów warstwy łącza danych oraz sieciowej. Ze względu na warstwowość modelu sieci, podpis może być umieszczony już w którejś z tych dwóch warstw bądź w dowolnej wyższej.



Rys. 1. Ustandaryzowany warstwowy model sieci ISO/OSI
z zaznaczonymi warstwami, na poziomie, których możliwa jest identyfikacja nadawcy

Protokoły warstwy łącza danych są silnie zależne od własności zastosowanego fizycznego nośnika, a te różnią się bardzo między sobą w zależności od przeznaczenia połączenia (tj. np. docelowego zasięgu czy żądanej przepustowości). Stąd na poziomie warstwy łącza danych występuje duża różnorodność protokołów (np. Ethernet, WiFi, FDDI, Token Ring) i sposobów adresowania, przez co rozwiązania stosowane w tej warstwie mają charakter tylko lokalny. Dużo bardziej uniwersalne jest stosowanie podpisu na poziomie warstwy sieciowej, która w chwili obecnej zdominowana jest przez protokół IP (Internet Protocol) w wersji czwartej. Ze względu na to, że jednolita adresacja elementów sieci na poziomie protokołu IP obejmuje cały świat, użycie tego protokołu pozwala na przesłanie podpisu pomiędzy dwoma dowolnymi maszynami na Ziemi. Na rysunku 2 przedstawiono ilustrację powiązań, powszechnie używanych dziś protokołów aplikacji końcowych (HTTP, FTP, SMTP, POP3, SSH, ED2K) ze wspólnym protokołem w warstwie transportowej. Protokół TCP (ang. *Transmission Control Protocol*) jest w warstwie transportowej wykorzystywany równie często, co protokół IP (ang. *Internet Protocol*) w warstwie sieciowej i razem z nim dostarcza usług dużej liczbie protokołów aplikacji końcowych. Z racji powszechności zastosowania, przedmiotem badania możliwości umieszczenia niewidocznych podpisów pracy będzie protokół IP.

Przez niedostrzegalność podpisu będzie rozumiana niemożliwość jednoznacznego stwierdzenia obecności podpisu przy użyciu tylko tych elementów odbiorczych, dla których podpisana informacja została przeznaczona (np. „nieuzbrojone” ludzkie oko, „nieuzbrojone” ucho, standardowa implementacja programu odbiorczego). Oznacza to, że niedostrzegalność podpisu rozumiana jest jako jego niezauważalność nie tylko dla zamierzonego odbiorcy informacji, ale także dla dowolnego (zaufanego bądź nie)

pośrednika w jej dostarczeniu, jak również dla osób i urzędzeń zainteresowanych wejściem w jej posiadanie. Do odnalezienia niedostrzegalnego podpisu niezbędna jest dodatkowa wiedza o miejscu i sposobie jego umieszczenia.

DHCP	SNMP	NTP	DNS	HTTP	FTP	SMTP	POP3	SSH	ED2K
UDP				TCP					
IP									
protokoły warstwy łącza danych									

Rys. 2. Najczęściej stosowane protokoły warstwy transportowej i aplikacji korzystające z protokołu IP

Całkowita niedostrzegalność podpisu znacznie ogranicza liczbę miejsc, w których można go umieścić. Rozpatrując fragment typowej ramki sieciowej transportującej dane dowolnego protokołu warstwy aplikacji potencjalnymi miejscami ukrycia podpisu mogą być nagłówki protokołów IP i TCP oraz właściwe dane znajdujące się w części protokołu aplikacji, przesyłane zazwyczaj otwartym, czytelnym dla podsłuchującego tekstem.

Podpis nie może być ukryty w polach nagłówka protokołu IP w dowolny sposób, by nie spowodować błędów podczas przetwarzania datagramu w urządzeniach pośredniczących w transmisji danych (głównie ruterach (ang. *routers*) i zaporach sieciowych (ang. *firewalls*)). Podobnie, nie może być on umieszczony w dowolny sposób w polach nagłówka protokołu TCP, by nie spowodować utraty połączenia i przerwania sesji pomiędzy nadawcą i odbiorcą danych na skutek błędów przetwarzania po stronie odbiorcy lub w zaporze sieciowej. Całkowicie wykluczone jest przechowywanie podpisu we właściwych danych transportowanych przez ramkę, gdyż spowoduje to ich nieczytelność i może sugerować podsłuchującemu, że dane są chronione.

Zatem do umieszczenia niedostrzegalnego podpisu można stosować jedynie nie mające wpływu na przebieg połączenia, nadmiarowe w danej sytuacji pola nagłówków protokołów IP i TCP. Należy jednak mieć na uwadze, że istnieją pola w nagłówkach protokołów, których rozmiar pozwala na przechowanie dużej ilości informacji, jednak ze względu na konieczność zgodności ze specyfikacją protokołu, dozwolony dla nich jest tylko ograniczony zbiór wartości. Nie można używać takich pól do przechowywania nadmiarowej informacji, gdyż urządzenia i oprogramowanie pośredniczące w transmisji będą oczekiwały w ich miejscu wartości dozwolonych przez specyfikację i w przypadku wystąpienia innych wartości, nie przetworzą poprawnie pakietu.

Protokoły IP i TCP zostały zaprojektowane tak, by dostarczać funkcjonalności przy minimalnym rozmiarze nagłówków. Pomimo obsługi sytuacji występujących rzadko

przez opcjonalne pola nagłówka, procesy zachodzące w warstwie sieciowej oraz transportowej powodują, że poprawne przesyłanie danych może mieć miejsce nawet przy traktowaniu informacji obecnej w obowiązkowych polach nagłówka jako nadmiarowej. Przeprowadzone badania pozwoliły dokonać wyboru takich pól.

2.1. ANALIZA MOŻLIWOŚCI UMIESZCZENIA NIEWIDOCZNEGO PODPISU W NAGŁÓWKU PROTOKOŁU IP

Protokół IP dostarcza mechanizmów umożliwiających transport danych do adresata poprzez sieci o różnych mediach transmisyjnych, a więc i różnych protokołach warstwy łącza danych. Protokół IP wprowadza również jednolity system adresacji maszyn, niezależny od ich fizycznych adresów w sieci, w której przebywają [POSTEL 81]. Przy konstrukcji niedostrzegalnych kanałów transmisyjnych należy brać pod uwagę fakt, iż protokół IP nie gwarantuje:

- dotarcia ramki IP do miejsca przeznaczenia,
- zachowania oryginalnej zawartości danych w ramce IP, jeśli ramka dotarła do miejsca przeznaczenia,
- transmisji ramki najkrótszą możliwą trasą,
- identycznej trasy dla ramek wysyłanych do tego samego adresata,
- kolejności docierania ramek do odbiorcy zgodnej z kolejnością ich wysłania przez nadawcę,
- dotarcia danych w jednej ramce, jeśli były one wysłane w jednej ramce,
- transmisji fragmentów jednej ramki tą samą trasą,
- dotarcia do odbiorcy fragmentów ramki w prawidłowej kolejności.

Wybór dalszej trasy transmisji ramki jest dokonywany przez każde urządzenie pośredniczące w transmisji (ruter) niezależnie.

O konieczności przeprowadzenia procesu fragmentacji ramki decyduje węzeł pośredniczący w transmisji po wyznaczeniu dalszej trasy dla ramki. Decyzja podejmowana jest na podstawie wyniku porównania wielkości ramki z maksymalną jednostką transmisji (ang. *Maximum Transmission Unit*, *MTU*) obowiązującą w kolejnym segmencie sieci. Wartość maksymalnej jednostki transmisji jest charakterystyczna dla zastosowanego w aktualnym segmencie sieci protokołu warstwy łącza danych. W tabeli 1 przedstawiono typowe wartości maksymalnych jednostek transmisji dla najczęściej używanych protokołów w warstwie łącza danych. Wartość maksymalnej jednostki transmisji ma duży wpływ na szybkość transmisji.

Fragmentacja ramki IP jest procesem jednokierunkowym, tzn. może ona w trakcie transmisji zachodzić wielokrotnie, ale proces do niej odwrotny, tj. scalenie, zachodzi tylko raz na maszynie odbiorcy. Analizę możliwości w zakresie konstrukcji niedostrzegalnych kanałów informacyjnych na bazie protokołu IP należy rozpocząć od analizy nagłówka.

Tabela 1. Wartości maksymalnych jednostek transmisji (MTU)
dla najczęściej używanych protokołów w warstwie łącza danych

Nazwa protokołu warstwy łącza danych	Wartość maksymalnej jednostki transmisji
Maksymalne MTU	65535 B
Token Ring	17914 B
FDDI	4352 B
Ethernet	1500 B
Point-to-Point	1500 B
IEEE 802.3	1492 B
SLIP	1006 B
X.25	576 B
Minimalne MTU	68 B

Źródło: [MOGUL 90].

Widok poszczególnych pól tworzących nagłówek protokołu IP przedstawiono na rysunku 3.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	Version				IHL			Type of Service								Total length																
1	Identifycation																Flags		Fragment Offset													
2	Time to Live								Protocol								Header Checksum															
3	Source Address																															
4	Destination Address																															
5	Options																								Padding							

Rys. 3. Format nagłówka protokołu IP: Version – pole „Wersja protokołu”; IHL – pole „Długość nagłówka ramki”; Type of Service – pole „Typ usługi”; Total Length – pole „Długość całkowita ramki”; Identification – pole „Identyfikator”; R – flaga zarezerwowana; DF – flaga „Fragmentowanie zabronione”; MF – flaga „Więcej fragmentów”; Fragment Offset – pole „Przesunięcie fragmentu”; Time to Live – pole „Licznik czasu”; Protocol – pole „Następny protokół”; Header Checksum – pole „Suma kontrolna nagłówka”; Source Address – pole „Adres źródłowy”; Destination Address – pole „Adres docelowy”; Options – pole „Opcje”; Padding – pole „Wypełnienie”

2.1.1. POLE „WERSJA PROTOKOŁU”

Jako pierwsze rozważmy pole „Wersja Protokołu”. Wartość pola „Wersja Protokołu” oznacza numer wersji protokołu, przy użyciu, której nadawca uformował bieżącą ramkę (np. 4 dla rozpatrywanej w tej pracy wersji czwartej protokołu IP). Wersja protokołu determinuje format nagłówka ramki, tj. rozmieszczenie i znaczenie pól nagłówka. Pole to powinno zawsze przechowywać prawdziwą informację o zasto-

sowanej wersji protokołu i nie powinno być modyfikowane przez nadawcę w celu przechowania innej informacji, a więc i podpisu. Zmiana wartości tego pola spowoduje niemożliwość poprawnej obsługi ramki przez urządzenia pośredniczące w transmisji bądź przez odbiorcę.

2.1.2. POLE „DŁUGOŚĆ NAGŁÓWKA RAMKI”

Wartość pola „Długość nagłówka ramki” równa jest rozmiarowi nagłówka protokołu IP bieżącej ramki, wyrażonym w słowach 32-bitowych (rozmiar każdego poprawnego nagłówka musi być wielokrotnością 4 bajtów). Rozmiar najmniejszego poprawnego nagłówka wynosi 20 bajtów, a największego – 60 bajtów (każdy rozmiar większy od 20 bajtów oznacza obecność w nagłówku pola opcji). Zatem pole „Długość nagłówka ramki” może przyjmować wartości od 5 do 15. Ze względu na stosowanie pól opcji tylko w specjalnych przypadkach dotyczących badań struktury sieci, wartość tego pola prawie zawsze równa jest 5. Jeśli odbiorca mógłby być pewien, że nagłówki wszystkich ramek wysyłanych od nadawcy będą miały rozmiar 20 bajtów (niezależnie od wartości faktycznie wpisanej przez nadawcę w pole „Długość nagłówka ramki”), pole „Długość nagłówka ramki” mogłoby zostać nośnikiem ukrytego podpisu. Nie można jednak wykorzystać do przechowania nadmiarowej informacji faktu, że pole to nigdy nie przyjmuje wartości od 0 do 4, bowiem w każdym ruterze pośredniczącym w transmisji wartość tego pola jest sprawdzana pod kątem poprawności i ramka jest odrzucana, gdy wartość tego pola jest mniejsza od 5. Podobnie nie można w polu tym umieścić większej wartości niż faktyczna długość nagłówka ramki, gdyż spowoduje to zinterpretowanie przez ruter danych występujących bezpośrednio po nagłówku jako pól opcji i w konsekwencji przetworzenie ramki w sposób inny niż oczekiwany przez nadawcę (w przypadku zgodności tych danych z formatem pola opcji) albo odrzucenie ramki (w przypadku niezgodności tych danych z formatem pola opcji). W związku z tym wykorzystanie pola „Długość nagłówka ramki” do przechowania dodatkowej informacji ogranicza się do przypadku, gdy istnieje pewność, że każde połączenie pomiędzy nadawcą a odbiorcą jest połączeniem bezpośrednim. W przeciwnym razie nadawca powinien zawsze wpisywać do pola „Długość nagłówka ramki” prawdziwą informację o rozmiarze nagłówka i pole to nie może być nośnikiem żadnej dodatkowej informacji.

2.1.3. POLE „TYP USŁUGI”

Ustawiając wartość pola „Typ usługi”, nadawca może określić „klasę jakości” transmisji dla nadawanej ramki. Dzięki ustawieniu odpowiedniego bitu, w zależności od przeznaczenia ramki, nadawca może zażądać np. natychmiastowego przekazywania ramki zawsze między ruterami najmniej obciążonymi, ustalenia najpierw najkrót-

szej trasy, a dopiero potem wysłania ramki lub wysłania ramki trasą o największej niezawodności. Każda opcja ma swoje wady i zalety, a różne opcje skutkują zazwyczaj różnymi trasami.

Ze względu na konieczność opracowania skomplikowanych algorytmów estymacji najkrótszych tras oraz wydajnych sposobów wymiany informacji pomiędzy ruterami o ich bieżącym obciążeniu, dyskusje na temat ostatecznego znaczenia bitów tego pola cały czas trwają i do chwili obecnej brak jest oficjalnej specyfikacji w tym zakresie (ostatni dokument na ten temat, RFC 3260 z kwietnia 2002 roku, ma w dalszym ciągu charakter wyłącznie informacyjny). Sprawę pogarsza duża czasochłonność przeprowadzanych obliczeń i stały wzrost ruchu sieciowego, którym coraz trudniej jest sterować. Wszystko to skutkuje tym, że w ruterach publicznych pole „Typ usługi” nie jest w ogóle interpretowane, a popularne systemy operacyjne (np. Microsoft Windows i Linux) w normalnych warunkach generują dla tego pola wyłącznie wartość 0. Niektórzy prywatni dostawcy usług internetowych oferują płatną usługę ustawiania gwarantowanych parametrów transmisji w obszarze swoich sieci i pole to wtedy jest nadpisywane przez oprogramowanie uruchomione na ich ruterach.

Pomimo faktu ignorowania wartości pola „Typ usługi” przez zdecydowaną większość ruterów, nie jest ono dobrym nośnikiem ukrytego podpisu. Podsluchujący prawdopodobnie będzie zdawał sobie sprawę z tego, że pole to bardzo rzadko ma wartość różną od 0 i gdy taką zaobserwuje, może nabrać podejrzeń, co do obecności podpisu. Poza tym, do czasu oficjalnego ustalenia znaczenia tego pola, jest ono najlepszym (ze względu na rozmiar) potencjalnym miejscem przechowania niewidocznego podpisu.

Chcąc mimo wszystko wykorzystać pole „Typ usługi” do przesłania ukrytej informacji, należy zawsze upewnić się, czy wszystkie pośredniczące w transmisji routery na pewno ignorują wartość tego pola i czy nie nadpisują go nową wartością.

2.1.4. POLE „DŁUGOŚĆ CAŁKOWITA RAMKI”

Wartość pola „Długość całkowita ramki” wyraża rozmiar (w bajtach) całej ramki protokołu IP, łącznie z nagłówkiem. Zbiór poprawnych wartości tego pola zawiera się pomiędzy 20 a 65535. Wartość tego pola musi być zawsze prawidłowa, bowiem w każdym routerze pośredniczącym w transmisji wartość ta jest porównywana z liczbą bajtów faktycznie odebranych z sieci przez fizyczny interfejs urządzenia pośredniczącego, pomniejszoną o długość nagłówka protokołu warstwy łącza danych (np. nagłówka Ethernetu). W przypadku stwierdzenia niezgodności, ramka jest odrzucana, co wyklucza przechowywanie jakichkolwiek dodatkowych danych, modyfikujących nawet tylko nieznacznie wartość tego pola. W związku z tym wykorzystanie pola „Długość całkowita ramki” do przechowania dodatkowej informacji ogranicza się do przypadku, gdy istnieje pewność, że każde połączenie pomiędzy nadawcą a odbiorcą jest połączeniem bezpośrednim i, że nadawca albo tak będzie przygotowywać ramki, by

na drodze do odbiorcy nie musiały być fragmentowane albo tak ukryje informację w polu „Długość całkowita ramki”, by nie przeszkadzało to poinformowanemu o tym odbiorcy w poprawnym przeprowadzeniu procesu scalania. W przeciwnym wypadku nadawca powinien zawsze wpisywać do pola „Długość całkowita ramki” prawdziwą informację o rozmiarze ramki i pole to nie może być nośnikiem żadnej dodatkowej informacji.

2.1.5. POLE „IDENTYFIKATOR”

Wartość pola „Identyfikator” jednoznacznie identyfikuje ramkę (lub jej część będącą wynikiem fragmentacji) w sieci w dowolnej chwili, co pozwala np. poprawnie scalać jej fragmenty, bądź skierować prośbę do nadawcy o ponowną jej transmisję w przypadku wystąpienia błędu. Pole „Identyfikator” wchodzi w skład piątki pól („Adres docelowy”, „Adres źródłowy”, „Protokół”, „Identyfikator”, „Przesunięcie fragmentu”), która w dowolnej chwili powinna wskazywać, na co najwyżej jedną (pofragmentowaną bądź nie) ramkę w sieci. Specyfikacja protokołu nie dopuszcza obsługi większej ilości takich ramek w tym samym czasie i w takiej sytuacji trzeba liczyć się z możliwością niepoprawnego przebiegu procesu scalania fragmentów u odbiorcy (a więc i niepoprawnego rezultatu weryfikacji podpisu) na skutek wymieszania się fragmentów pochodzących od dwóch różnych ramek. Danej wartości pola „Identyfikator” nadawca może użyć po raz kolejny dopiero po upływie liczby sekund równej wartości wysyłanej standardowo przez nadawcę w polu „Licznik czasu” (czyli nawet po upływie 255 s). Stąd pole „Identyfikator” nie jest dobrym miejscem na przechowanie podpisu, gdyż może zdarzyć się, że nadawca w krótkim przedziale czasu podpisze dwie ramki o różnych zawartościach tym samym 16-bitowym słowem. W celu wyeliminowania takiego przypadku należałoby przeznaczyć kilka bitów tego pola (w ilości zależnej od prawdopodobieństwa powstania w danym przedziale czasu identycznego podpisu dla dwóch różnych ramek) na zapewnienie niepowtarzalności podpisu.

Pole „Identyfikator” jest natomiast dobrym nośnikiem podpisu w momencie, gdy z rozważań można wyeliminować proces fragmentacji w warstwie sieciowej. Proces fragmentacji nie zajdzie, gdy nadawca zbada wartości maksymalnych jednostek transmisji (MTU) na każdej z możliwych tras prowadzących do odbiorcy, spośród nich wybierze wartość najmniejszą i tak skonfiguruje ustawienia protokołów wyższych warstw (np. protokołu TCP), by te dostarczały modułowi nadawczemu IP dane w porcjach nie przekraczających tej wybranej minimalnej wartości. Pomimo, że scenariusz taki nie wyklucza nadania w krótkim odstępie czasu dwóch ramek o identycznym podpisie, zapobiega on występowaniu nieprawidłowości podczas procesu scalania, który w takim wypadku nie jest u odbiorcy w ogóle przeprowadzany. Jeśli bowiem odbiorca ma pewność, że żadna odebrana od nadawcy ramka nie została po drodze pofragmentowana, może on weryfikować podpis na bieżąco i zakończyć

przetwarzanie ramki, nim rozpocznie przetwarzanie innej ramki o identycznym identyfikatorze (mogącej w normalnych warunkach być przyczyną błędu).

Wadą takiego sposobu podpisywania jest dłuższy czas nawiązania transmisji, związany z koniecznością wykrycia minimalnej wartości MTU na wszystkich możliwych trasach przesyłu oraz dłuższy czas generowania ramki protokołu IP przez nadawcę, który musi odpowiednio dopasować jej rozmiar już na poziomie protokołów wyższych warstw modelu. Ponadto rozwiązanie to znacznie zwiększa liczbę danych przesyłanych w sieci, co negatywnie odbija się na przepustowości sieci.

2.1.6. POLE „ZNACZNIKI”

W polu „Znaczniki” są zawarte wartości trzech jednobitowych, niezależnych od siebie flag, które mają wpływ na działanie procesów fragmentacji i scalania ramki protokołu IP. Flagi te to:

- 1) flaga o wymaganej wartości 0, niezdefiniowanej nazwie i nieprzypisanym znaczeniu – zarezerwowana do wykorzystania w przyszłości, gdy zajdzie potrzeba rozszerzenia funkcjonalności protokołu. Aktualnie nie jest wykorzystywana w żadnym celu, jednak nie można dowolnie ustawiać jej wartości, ponieważ urządzenia pośredniczące w transmisji akceptują tylko wartość 0 (zgodnie z wymogami specyfikacji). Wartości 1 będzie można używać dopiero po nadaniu znaczenia tej fladze i zamieszczeniu w specyfikacji protokołu IP jej dokładnego opisu;
- 2) flaga „Fragmentowanie zabronione” – jej wartość równa 1 zabezpiecza ramkę przed pofragmentowaniem w maszynie nadawcy oraz w którymkolwiek ruterze pośredniczącym w transmisji. Ramka z ustawioną flagą „Fragmentowanie zabronione” nie dotrze do odbiorcy, jeśli podczas wyznaczania dalszej trasy dla tej ramki w którymkolwiek ruterze okaże się, że ramka jest za duża, by mogła być w całości przetransportowana przez którąkolwiek następną sieć. Nadawca ramki zostanie w takim wypadku poinformowany o wystąpieniu błędu podczas transmisji przez komunikat protokołu ICMP. Nadawca może ustawić tą flagę w celu wykrycia maksymalnej jednostki transmisji dla całej trasy prowadzącej do odbiorcy. Jest w stanie to zrobić wysyłając ramki o różnej wielkości i z ustawioną flagą „Fragmentowanie zabronione” oraz obserwując nadchodzące komunikaty protokołu ICMP (lub ich brak);
- 3) flaga „Więcej fragmentów” – jej wartość, razem z wartością pola „Przesunięcie fragmentu”, dostarcza odbiorcy informacji potrzebnej w procesie scalania fragmentów – informuje odbiorcę o tym, czy bieżąca ramka powstała w wyniku fragmentacji, a więc czy należy oczekiwać na inne ramki o tym samym polu „Identyfikator” i przystąpić do przeprowadzenia procesu scalenia. Jeśli tak, dostarcza też informacji czy bieżąca ramka powstała jako ostatnia w procesie fragmentacji. Znaczenie wszystkich możliwych kombinacji pól „Więcej fragmentów” oraz „Przesunięcie fragmentu” przedstawiono w tabeli 2.

Tabela 2. Znaczenie wszystkich możliwych kombinacji flagi „Więcej fragmentów” oraz pola „Przesunięcie fragmentu”

„Więcej fragmentów”	„Przesunięcie fragmentu”	Znaczenie kombinacji
0	0	Ramka nie powstała w wyniku procesu fragmentacji, jest samodzielną ramką i nie trzeba przeprowadzać procesu scalania.
0	>0	Ramka powstała w wyniku procesu fragmentacji jako ostatnia. W celu zrekonstruowania oryginalnej, niesfragmentowanej ramki należy skompletować wszystkie fragmenty o wartości pola „Przesunięcie fragmentu” mniejszej od wartości tego pola w tej ramce oraz je scalić. Wielkość oryginalnej ramki (bez nagłówka) to „Przesunięcie fragmentu” + „Długość całkowita ramki” – „Długość nagłówka ramki”.
1	0	Ramka powstała w wyniku procesu fragmentacji jako pierwsza.
1	>0	Ramka powstała w wyniku procesu fragmentacji nie jako pierwsza i nie jako ostatnia.

2.1.7. FLAGA „FRAGMENTOWANIE ZABRONIONE”

Flaga „Fragmentowanie zabronione” może być nośnikiem ukrytego podpisu, gdy ustawienie jej wartości na 1, nie spowoduje błędu transmisji wynikającego ze zbyt dużej wielkości ramki. Aby uniknąć takiego błędu, nadawca musi odpowiednio dobrać rozmiar wysyłanej ramki, by nie przekroczył minimalnej wartości spośród wszystkich maksymalnych jednostek transmisji występujących na trasie. Może to osiągnąć badając wielkości maksymalnych jednostek transmisji na wszystkich możliwych trasach do odbiorcy i tak konfigurując ustawienia protokołów wyższych warstw (np. protokołu TCP), by te dostarczały modułowi nadawczemu IP dane w porcjach nie przekraczających najmniejszej z tych wartości. Jednak w takim wypadku ilość informacji, jaką może pomieścić podpis ukryty we fładze „Fragmentowanie zabronione”, wynosi tylko jeden bit.

Dużo większą siłę podpisu umożliwia zezwolenie protokołom wyższych warstw na dostarczanie danych w porcjach o niezmiennionej wielkości i dokonanie dopasowania rozmiaru wysyłanej ramki w module nadawczym protokołu IP przy zastosowaniu procesu fragmentacji względem najmniejszej z wykrytych w badaniach wartości MTU. Części podpisu oryginalnej ramki można wtedy po jej pofragmentowaniu umieścić we flagach „Fragmentowanie zabronione” utworzonych fragmentów, co oznacza ilość bitów podpisu równą liczbie powstałych fragmentów. Przy wykorzystaniu techniki z użyciem fragmentacji na maszynie nadawcy, odbiorca powinien w trakcie procesu scalania fragmentów gromadzić również na bieżąco informację z flag „Fragmentowanie zabronione” i przystąpić do weryfikacji podpisu dopiero po zakończeniu procesu scalania. Odbiorca powinien znać maksymalną jednostkę transmisji wybraną przez nadawcę i podczas weryfikacji podpisu jako jedno z kryteriów poprawności podpisu uwzględniać również rozmiar każdego fragmentu.

Wadą powyższych rozwiązań jest dłuższy czas nawiązania transmisji, związany z koniecznością wykrycia minimalnej wartości MTU na wszystkich możliwych trasach przesyłu oraz dłuższy czas generowania ramki protokołu IP przez nadawcę, który musi odpowiednio zmniejszyć jej rozmiar bądź podczas procesu fragmentacji w warstwie sieciowej, bądź na poziomie protokołów wyższych warstw modelu. Ponadto rozwiązania te znacznie zwiększają liczbę danych przesyłanych w sieci, co negatywnie odbija się na przepustowości sieci.

2.1.8. FLAGA „WIĘCEJ FRAGMENTÓW”

Flaga „Więcej fragmentów” może być nośnikiem ukrytego podpisu, gdy odbiorca ma pewność, że ramka nie została nigdzie pofragmentowana lub nadawca ustalił wcześniej z odbiorcą stałą wartość wskazującą, na ile fragmentów podzieli każdą ramkę przed jej wysłaniem. W drugim przypadku informacje otrzymywane w polach „Przesunięcie fragmentu” całkowicie wystarczą do przeprowadzenia procesu scalenia fragmentów.

Pierwszy sposób ukrycia podpisu wymaga od nadawcy takiego skonfigurowania ustawień protokołów wyższych warstw niż sieciowa (np. protokołu TCP), by te dostarczały modułowi nadawczemu IP dane w porcjach nie przekraczających najmniejszej z wszystkich maksymalnych jednostek transmisji występujących na każdej możliwej trasie do odbiorcy. Jednak w takim wypadku ilość informacji, jaką może pomieścić podpis ukryty we fładze „Więcej fragmentów”, wynosi tylko jeden bit.

Dużo większe możliwości daje rozwiązanie z wykorzystaniem początkowej fragmentacji ramki w maszynie nadawcy na ustaloną z odbiorcą, stałą dla wszystkich ramek liczbę fragmentów. Ustalona liczba fragmentów powinna być na tyle duża, by żaden fragment nie podległ w dalszej drodze ponownemu procesowi fragmentacji, co da odbiorcy pewność, że odbiera dokładnie tyle fragmentów, ile wysłał do niego nadawca. Sposób ten wymaga, zatem również znalezienia najmniejszej z wszystkich maksymalnych jednostek transmisji występujących na każdej możliwej trasie do odbiorcy. Minimalna liczba fragmentów powinna być równa najmniejszej liczbie całkowitej większej od $(MAX_ROZMIAR_IP - ROZM_NAGŁ_IP) / (MTU_{min} - ROZM_NAGŁ_IP)$, gdzie $MAX_ROZMIAR_IP$ to maksymalny rozmiar porcji danych dla protokołu IP w bajtach (co najwyżej 65535, wartość konfigurowalna), $ROZM_NAGŁ_IP$ to używany przez nadawcę rozmiar nagłówka ramki IP w bajtach (co najmniej 20, co najwyżej 60, najczęściej występująca wartość to 20) a MTU_{min} to znaleziona najmniejsza maksymalna jednostka transmisji wyrażona w bajtach. Ilość bitów, na których można przechować podpis, równa jest liczbie ustalonych fragmentów, bowiem części podpisu oryginalnej ramki można po jej pofragmentowaniu umieścić we flagach „Więcej fragmentów” utworzonych fragmentów. Odbiorca powinien w trakcie procesu scalania fragmentów gromadzić również na bieżąco informację z flag „Więcej fragmentów” i przystąpić do weryfikacji podpisu dopiero po zakończeniu procesu scalania.

Wadą powyższych rozwiązań jest dłuższy czas nawiązania transmisji, związany z koniecznością wykrycia minimalnej wartości MTU na wszystkich możliwych trasach przesyłu oraz dłuższy czas generowania ramki protokołu IP przez nadawcę, który musi odpowiednio zmniejszyć jej rozmiar bądź podczas procesu fragmentacji w warstwie sieciowej, bądź na poziomie protokołów wyższych warstw modelu. Ponadto rozwiązania te znacznie zwiększają liczbę danych przesyłanych w sieci, co negatywnie odbija się na jej przepustowości.

2.1.9. POLE „PRZESUNIĘCIE FRAGMENTU”

Wartość pola „Przesunięcie fragmentu” wyznacza pozycję danych zawartych wewnątrz bieżącego fragmentu w oryginalnej ramce i umożliwia odbiorcy poprawne przeprowadzenie procesu scalania. Jeśli ramka nie była pofragmentowana, wartość w polu „Przesunięcie fragmentu”, jak i we fladze „Więcej fragmentów” wynosi 0.

Pole „Przesunięcie fragmentu” może być nośnikiem dodatkowej informacji, jeśli odbiorca ma pewność, że proces scalania nie będzie musiał być przeprowadzony, czyli że nadawca tak dobrał rozmiar ramki, żeby nie uległa fragmentacji. Wymaga to od nadawcy takiego skonfigurowania ustawień protokołów wyższych warstw niż sieciowa (np. protokołu TCP), by te dostarczały modułowi nadawczemu IP dane w porcjach nie przekraczających najmniejszej z wszystkich maksymalnych jednostek transmisji występujących na każdej możliwej trasie do odbiorcy. W tym przypadku rozmiar ukrytej informacji wynosi 13 bitów.

Rozmiar niedostrzegalnego podpisu można zwiększyć, jeśli nadawca uzgodni z odbiorcą maksymalny rozmiar każdej (niepofragmentowanej) ramki, którą będzie do niego wysyłał oraz dodatkowo każdą wysyłaną ramkę będzie fragmentował na swojej maszynie. Pole „Przesunięcie fragmentu” umożliwia poprawne adresowanie fragmentów ramki, jeśli jej długość nie przekracza 65535 bajtów. Jednak bardzo często ustawienia protokołów wyższych warstw (np. TCP) są tak skonfigurowane, by przekazywały modułowi nadawczemu protokołu IP dane w porcjach dużo mniejszych (ok. 1500 bajtów). Zatem kilka najbardziej znaczących bitów pola „Przesunięcie fragmentu” będzie (w przypadku np. użycia protokołu TCP) zawsze zerami i można je przeznaczyć na przechowanie dodatkowej informacji, jeśli tylko uprzednio poinformuje się odbiorcę o zmianie znaczenia tych bitów i ich liczbie. Zakładając, że nadawca zobowiąże się do wysyłania ramek o rozmiarze przed lokalną fragmentacją nie większym, niż k bajtów (bez nagłówka protokołu IP), w jednym fragmencie możliwe jest przeznaczenie na podpis ilości najbardziej znaczących bitów pola w liczbie równej największej liczbie całkowitej mniejszej od $f = 13 - \log_2(k) + 3$ (dodanie liczby 3 do obliczeń wynika z faktu, że pole „Przesunięcie fragmentu” wyznacza pozycje danych fragmentu w całej ramce z dokładnością do 8 bajtów). Dzięki możliwości podzielenia podpisu całej ramki na pola „Przesunięcie fragmentu” wszystkich wygenerowanych n fragmentów, dostępna liczba bitów podpisu wzrasta do $n \cdot f$. Zaletą tej metody ukry-

wania podpisu jest to, że liczba fragmentów (n) może być dowolna, jeśli tylko każdy fragment jest wystarczająco mały, by nie być pofragmentowanym ponownie na dalszej drodze do odbiorcy. W celu osiągnięcia silnego podpisu nadawca powinien też ustalić z odbiorcą minimalną liczbę fragmentów, na jaką będzie dzielił każdą wysłaną do niego ramkę. Odbiorca powinien w trakcie procesu scalania fragmentów gromadzić również na bieżąco części podpisu z pól „Przesunięcie fragmentu” i przystąpić do weryfikacji podpisu dopiero po zakończeniu procesu scalania.

Wadą powyższych rozwiązań jest dłuższy czas nawiązania transmisji, związany z koniecznością wykrycia minimalnej wartości MTU na wszystkich możliwych trasach przesyłu oraz dłuższy czas generowania ramki protokołu IP przez nadawcę, który musi odpowiednio zmniejszyć jej rozmiar bądź podczas procesu fragmentacji w warstwie sieciowej, bądź na poziomie protokołów wyższych warstw modelu. Ponadto rozwiązania te znacznie zwiększają liczbę danych przesyłanych w sieci, co negatywnie odbija się na przepustowości sieci.

2.1.10. POLE „LICZNIK CZASU”

Wartość pola „Licznik czasu” oznacza maksymalną liczbę sekund, przez jaką ramka może pozostawać w sieci. Celem wprowadzenia tego pola do nagłówka protokołu było zapobiegnięcie krążeniu po sieci w nieskończoność ramek z błędnym adresem odbiorcy, bądź ramek, których z różnych powodów nie można do odbiorcy dostarczyć. Nadawca ustala wartość tego pola tak, by być pewnym, że w określonej liczbie sekund ramka na pewno dotrze do odbiorcy. Każdy ruter pośredniczący w dostarczeniu ramki do odbiorcy zmniejsza wartość pola o tyle, ile sekund zajęło mu jej przetworzenie. Ponieważ routery przetwarzają ramki w czasie znacznie krótszym niż 1s (jednak i tak zmniejszają wtedy wartość pola o 1), wartość tego pola często rozumie się jako maksymalną liczbę ruterów, które mogą pośredniczyć w dostarczeniu ramki (tzw. maksymalna długość trasy). W przypadku, gdy któryś z ruterów nie może już zmniejszyć wartości tego pola (tj. wartość wynosi 0), ramka nie jest wysyłana przez niego dalej i uznaje się ją za zagubioną.

Najbezpieczniejszą wartością dla tego pola jest jego maksymalna wartość, czyli 255. Jest to jednak wartość o wiele za wysoka, bowiem obecna struktura światowej sieci Internet prawie zawsze pozwala na przekazanie ramki do dowolnego odbiorcy za pośrednictwem co najwyżej 25 ruterów. Znajduje to potwierdzenie w konkretnych implementacjach modułów nadawczych protokołu IP: w systemach operacyjnych Linux i Unix wartość ta wynosi standardowo 64, w systemie Windows – 128. Pole „Licznik czasu” dostarcza zatem miejsca na ukrycie dodatkowych danych. Wielkość wolnego miejsca zależy od tego, czy liczba ruterów pośredniczących w przesyłaniu danych od nadawcy do odbiorcy jest taka sama dla każdej możliwej trasy przesyłu czy nie i wynosi:

- 1) 7 bitów, jeśli liczba ruterów pośredniczących jest taka sama dla każdej możliwej trasy przesyłu. Sytuacja taka występuje wtedy, gdy ramki mogą być przesy-

lane od nadawcy do odbiorcy dokładnie jedną trasą lub wieloma trasami, ale wybór trasy (którego wyniku nadawca nie może przewidzieć) nie ma wpływu na ilość ruterów pośredniczących. W tym przypadku nadawca ustawia (na przykład) najbardziej znaczący bit pola „Licznik czasu” na 1 i wtedy niezależnie od wartości podpisu, który ukrywa na kolejnych siedmiu bitach, minimalna wartość tego pola wynosi 128, co zapewnia dotarcie ramki do odbiorcy. Aby zweryfikować podpis, odbiorca potrzebuje informacji o liczbie ruterów pośredniczących. Informację tę może zdobyć nadawca wykonując występujące w prawie każdym systemie operacyjnym polecenie *traceroute* (lub jego odpowiednik, np. *tracert*) na rzecz adresu IP odbiorcy, a następnie przekazać ją w dowolny sposób odbiorcy przed rozpoczęciem podpisanej transmisji. Informację taką może też zdobyć odbiorca, wykonując u siebie obecne w prawie każdym systemie operacyjnym polecenie *ping* na rzecz adresu IP nadawcy. Zakładając niezmiennosc w czasie długości tras, operację uzyskania liczby ruterów pośredniczących wystarczy wykonać raz. Jeśli jednak w przyszłości długości tras mogłyby ulec zmianie (na inną wartość stałą przez długi czas, jednakową dla wszystkich możliwych tras), wielkość dodatkowej informacji wynosi w dalszym ciągu 7 bitów. W takiej sytuacji należałoby tylko w porę wykryć zmianę liczby ruterów pośredniczących, np. wykonując okresowo polecenia *ping* lub *traceroute* i zaktualizować tę wartość po stronie odbiorcy;

- 2) 8- k bitów, jeśli liczba ruterów pośredniczących nie jest taka sama dla każdej możliwej trasy przesyłu, ale istnieje pewność, że nie przekroczy $2^k - 1$. Sytuacja taka występuje wtedy, gdy wybór trasy (którego wyniku nadawca nie może przewidzieć) ma wpływ na liczbę ruterów pośredniczących. W tym przypadku nadawca ustawia k najmniej znaczących bitów pola „Licznik czasu” na 1 i wtedy niezależnie od wartości podpisu, który ukrywa na $8 - k$ najbardziej znaczących bitach, minimalna wartość tego pola równa jest maksymalnej liczbie ruterów pośredniczących wybranej spośród wszystkich możliwych tras, co zapewnia dotarcie ramki do odbiorcy. Aby zweryfikować podpis, odbiorca potrzebuje informacji o maksymalnej długości trasy lub liczbie bitów wykorzystanych na podpis. Maksymalną długość trasy można wykryć wielokrotnie wykonując polecenia *ping* lub *traceroute* na maszynie odpowiednio odbiorcy lub nadawcy. Nadawca może też przekazać odbiorcy informację o liczbie bitów wykorzystywanych na podpis w dowolny sposób przed rozpoczęciem podpisanej transmisji. Zakładając niezmiennosc w czasie maksymalnej długości trasy, operację uzyskania maksymalnej liczby ruterów pośredniczących wystarczy wykonać raz. Jednak, gdy występuje możliwość wzrostu maksymalnej długości trasy w przyszłości (np. poprzez dodawanie z czasem nowych, dłuższych tras lub rozrastanie się obecnych podsięci), należy w porę wykryć taką zmianę (np. wykonując okresowo polecenia *ping* lub *traceroute*) i zaktualizować tę wartość po stronie odbiorcy. Pozytywnym zjawiskiem jest zmniejszenie maksymalnej długości trasy, co może pozwolić na zwiększenie ilości bitów przeznaczonych na podpis;

- 3) 2 bity, jeśli nie ma możliwości sprawdzenia liczby ruterów pośredniczących na wszystkich możliwych trasach. Może się tak stać w przypadku, gdy urządzenia pośredniczące w transmisji nie obsługują bądź nie zezwalają na propagowanie komunikatów takich programów jak *ping* lub *traceroute*. W tym przypadku nadawca ustawia 6 najmniej znaczących bitów pola „Licznik czasu” na 1 i wtedy niezależnie od wartości podpisu, który ukrywa na dwóch najbardziej znaczących bitach, minimalna wartość tego pola wynosi 63, co zapewnia dotarcie ramki do odbiorcy. Zaletą tego rodzaju podpisu jest to, że aby go zweryfikować, odbiorca nie potrzebuje żadnych dodatkowych informacji.

Proces fragmentacji ramki nie ma negatywnego wpływu na zaprezentowane wyżej techniki ukrywania podpisu, ponieważ podczas tego procesu aktualna wartość pola „Licznik czasu” fragmentowanej ramki jest kopiowana do tych samych pól w nowo utworzonych fragmentach. Fragmenty są też ramkami, w trakcie transmisji podlegają tym samym operacjom, co one i pomimo, że mogą dotrzeć do odbiorcy różnymi trasami i w różnej kolejności, nigdy nie przyjmą stanu, który nie byłby rozpatrzony tutaj dla ich ramki wyjściowej.

2.1.11. POLE „NASTĘPNY PROTOKÓŁ”

Wartość pola „Następny protokół” równa jest numerowi protokołu warstwy transportowej, którego nagłówek znajduje się w ramce zaraz za nagłówkiem protokołu IP. Numer ten jednoznacznie wyznacza, moduł odbiorczy, którego z protokołów warstwy transportowej powinien być użyty do dalszego przetwarzania odebranej ramki IP. Pole „Następny protokół” może być nośnikiem dodatkowej informacji w momencie, gdy nadawca uzgodni wcześniej z odbiorcą, że będzie wysyłał do niego dane przy użyciu tylko jednego, zawsze tego samego protokołu warstwy transportowej (np. TCP). Wtedy nadawca może zmienić znaczenie pola „Następny protokół” i na wszystkich jego 8 bitach umieścić niewidoczny podpis. Chcąc umożliwić komunikację przy użyciu większej ilości protokołów warstwy transportowej, można przeznaczyć k bitów tego pola na przechowanie informacji o użytym protokole (np. dwa bity w celu zakodowania informacji o trzech najczęściej używanych protokołach tej warstwy: TCP, UDP, ICMP i dowolnym czwartym protokole), umieszczając wtedy podpis na pozostałych bitach pola. W takim wypadku ilość bitów przeznaczonych na podpis jest równa największej liczbie całkowitej nie przekraczającej $8 - \log_2(k)$, a odbiorca musi zostać poinformowany o wybranym nowym zbiorze dostępnych protokołów i ich nowych numerach.

W polu „Następny protokół” można też ukryć podpis nie ograniczając w ogóle ilości dostępnych protokołów. Wynika to z faktu, że do chwili obecnej (grudzień 2005 r.) zdefiniowano tylko 141 numerów dla protokołów warstwy transportowej [IANA 06], a pole „Następny protokół” może przyjmować 256 różnych wartości. Zatem pozostałe 115 kombinacji można wykorzystać jako wartości ukrytego podpisu. Sytuacja taka

wymaga jednak uprzedniego uzgodnienia między odbiorcą a nadawcą, jakie nieużywane standardowo wartości odpowiadają, jakim protokołom, by odbiorca otrzymując w polu „Następny protokół” numer nie przyporządkowany żadnemu protokołowi, mógł w dalszym ciągu prawidłowo wybrać moduł odbiorczy do dalszego przetwarzania ramki w warstwie transportowej. Najprostszym rozwiązaniem może być tutaj ustalenie, że nadawca będzie podpisywał tylko ramki przenoszące dane protokołu TCP. Nowe numery protokołów dodawane są do standardu bardzo rzadko, co jeszcze przez długi czas gwarantuje dużą ilość nieużywanych kombinacji w polu „Następny protokół”.

Należy pamiętać, że pole „Następny protokół” wchodzi standardowo w skład piątki pól („Adres docelowy”, „Adres źródłowy”, „Protokół”, „Identyfikator”, „Przesunięcie fragmentu”), która w dowolnej chwili powinna wskazywać, na co najwyżej jedną (pofragmentowaną bądź nie) ramkę w sieci. Nadawca musi, zatem pamiętać o wyborze innego identyfikatora dla ramki, która w polach „Adres docelowy”, „Adres źródłowy”, „Przesunięcie fragmentu” i zmodyfikowanym na potrzeby przechowania ukrytego podpisu, polu „Następny protokół” ma już wartości równe wartościom tych pól w innej, wygenerowanej niedawno ramce.

Dla każdego powyższego scenariusza, rozmiar niedostrzegalnego podpisu jednej ramki można zwiększyć, jeśli nadawca będzie fragmentował wysyłaną ramkę na swojej maszynie. Wtedy ilość możliwej do ukrycia informacji wzrasta tyle razy, ile fragmentów zostało wygenerowanych przez nadawcę. Wynika to z faktu, że każdy wygenerowany fragment posiada własne pole „Następny protokół”, niezależne od takich samych pól w innych fragmentach. Zaletą tej metody ukrywania podpisu jest to, że liczba fragmentów może być dowolna, jeśli tylko każdy fragment jest wystarczająco mały, by nie być pofragmentowanym ponownie na dalszej drodze do odbiorcy.

Z powodu wykorzystywania pola „Następny protokół” w standardowym procesie scalania fragmentów, dopuszczenie do różnicowania pola „Następny protokół” w ramach fragmentów jednej ramki będzie powodować błędy w przeprowadzaniu tego procesu na maszynie odbiorcy. Dlatego należy zmodyfikować przebieg tego procesu tak, by w jego trakcie informacja z pola „Następny protokół” nie była w ogóle brana pod uwagę. Pociąga to za sobą zmianę znaczenia pojęcia klucza ramki, w którego skład w takim wypadku powinny wchodzić jedynie cztery pola: „Adres docelowy”, „Adres źródłowy”, „Identyfikator”, i „Przesunięcie fragmentu”. To z kolei powoduje, że nadawca musi pamiętać o wyborze innego identyfikatora dla ramki, która w polach „Adres docelowy”, „Adres źródłowy”, „Przesunięcie fragmentu” ma już wartości równe wartościom tych pól w innej, wygenerowanej niedawno ramce. Odbiorca natomiast powinien w trakcie procesu scalania fragmentów gromadzić również na bieżąco części podpisu z pól „Następny protokół” i przystąpić do weryfikacji podpisu dopiero po zakończeniu procesu scalania. W celu osiągnięcia silnego podpisu nadawca powinien ustalić z odbiorcą minimalną liczbę fragmentów, na jaką będzie dzielił każdą wysyłaną do niego ramkę.

Wadą rozwiązania z wykorzystaniem lokalnej fragmentacji do zwiększenia siły podpisu jest dłuższy czas nawiązania transmisji, związany z koniecznością wykrycia

minimalnej wartości MTU na wszystkich możliwych trasach przesyłu oraz dłuższy czas generowania ramki protokołu IP przez nadawcę, który musi odpowiednio zmniejszyć jej rozmiar podczas procesu fragmentacji w warstwie sieciowej. Ponadto rozwiązanie to znacznie zwiększa liczbę danych przesyłanych w sieci, co negatywnie odbija się na przepustowości sieci.

2.1.12. POLE „SUMA KONTROLNA NAGŁÓWKA”

Wartość tego pola równa jest sumie kontrolnej całego nagłówka ramki protokołu IP (łącznie z polem „Opcje”, jeśli występuje) i jest wyliczana na nowo po każdej modyfikacji dowolnego z pól nagłówka. Suma kontrolna jest najpierw wyliczana i zapisywana na maszynie nadawcy, po ustaleniu wartości wszystkich pól nagłówka. Potem suma kontrolna jest ponownie wyliczana i zapisywana w każdym ruterze pośredniczącym w transmisji na skutek modyfikacji przez te urządzenia niektórych pól nagłówka, np. pola „Licznik czasu”.

W każdym ruterze pośredniczącym, tuż po odebraniu ramki, na podstawie otrzymanego nagłówka wyliczana jest suma kontrolna i porównywana z tą przechowywaną w polu nagłówka. Jeśli wystąpiła niezgodność, ramka uznana jest za uszkodzoną, nie jest dalej przetwarzana i nie dociera do odbiorcy. Dlatego pole „Suma kontrolna nagłówka” może być nośnikiem ukrytego podpisu tylko wtedy, gdy wszystkie połączenia między nadawcą a odbiorcą są bezpośrednie oraz połączenie charakteryzuje się wysoką niezawodnością.

2.1.13. POLE „ADRES ŹRÓDŁOWY”

Wartość pola „Adres źródłowy” oznacza adres nadawcy i nie powinno być nigdy modyfikowane, ponieważ na podstawie tego pola nastąpi identyfikacja nadawcy w maszynie odbiorcy. Zmiana wartości tego pola, oprócz niepoprawnej identyfikacji uniemożliwi także poinformowanie nadawcy o ewentualnych błędach podczas transmisji przez urządzenia pośredniczące. Pole to w szczególnych okolicznościach może być wykorzystane do sygnowania sieci za pomocą liczby i rodzaju generowanych w niej błędów [LIBER 04].

2.1.14. POLE „ADRES DOCELOWY”

Wartość pola „Adres docelowy” oznacza adres odbiorcy i nie powinno być nigdy modyfikowane, gdyż uniemożliwi to dostarczenie ramki zamierzonemu odbiorcy. Pole to w szczególnych okolicznościach może być wykorzystane do sygnowania sieci za pomocą ilości i rodzaju generowanych w niej błędów [LIBER 04].

2.1.15. POLE „OPCJE”

Pole „Opcje” jest opcjonalnym polem nagłówka dodanym głównie w celu umożliwienia nadawcy wysyłania komunikatów diagnostycznych. Przykładowo, dzięki ustawieniu odpowiednich wartości w tym polu, nadawca może narzucić dokładną trasę transmisji ramki albo zażądać zgromadzenia informacji o wszystkich ruterach pośredniczących w transmisji w celu dalszego wykorzystania tych informacji przez odbiorcę ramki.

Pole „Opcje” nie jest dobrym nośnikiem ukrytej informacji. Obecność tego pola można łatwo wywnioskować z niestandardowej, większej od 5 wartości pola „Długość nagłówka ramki” w nagłówku. Z racji tego, że pole „Opcje” z założenia miało być wykorzystywane tylko w celach diagnostycznych, spotyka się je w sieci niezwykle rzadko i tylko wtedy, gdy zachodzi potrzeba wskazania przyczyny poważnego problemu o naturze sieciowej. W związku z tym pojawienie się tego pola w nagłówkach ramek, szczególnie wysyłanych nieustannie tylko przez jedną parę nadawca–odbiorca, może z dużym prawdopodobieństwem stać się źródłem podejrzeń podsłuchującego, co do obecności ukrytego podpisu. Tym bardziej, że pole „Opcje” oferuje dużo większą przestrzeń do umieszczenia podpisu od innych pól nagłówka – nawet do 40 bajtów.

Należy też mieć na uwadze, że obecność pola opcji znacznie zwiększa czas przetwarzania ramki w routerze oraz zwiększa liczbę danych przesyłanych w sieci, co negatywnie odbija się na przepustowości sieci. Dlatego wiele zapór sieciowych nie pozwala na przekazywanie ramek protokołu IP z nagłówkami zawierającymi pole „Opcje”. Decydując się mimo wszystko na stosowanie ruchu podpisanego przy wykorzystaniu pola „Opcje”, należy mieć na uwadze, że routery, zgodnie ze specyfikacją, samodzielnie nadpisują znaczne obszary tego pola.

2.1.16. POLE „WYPEŁNIENIE”

Pole „Wypełnienie” jest obecne tylko w przypadku użycia w nagłówku IP opcji. Służy do wypełnienia pozostałej części nagłówka tak, by jego długość była wielokrotnością czterech bajtów. Zgodnie ze specyfikacją protokołu, pole to powinno być wypełnione zerami. Pole może być nośnikiem dodatkowej informacji, ale należy liczyć się z tym, że zapory sieciowe mogą nie przekazywać pakietów z wartością tego pola różną od 0. Również nadmiarowej informacji nie można traktować jako niewidocznej dla podsłuchującego, bowiem podsłuchujący może nabrać podejrzeń, co do obecności ukrytego podpisu obserwując wartości pola „Wypełnienie” różne od 0.

2.2. PODSUMOWANIE WYNIKÓW ANALIZY

W wyniku przeprowadzonej analizy wykonano zestawienie tabelaryczne (tabela 3), w którym zawarto wyliczenia ilości nadmiarowych bitów informacji udostępnianych

do wykonania niedostrzegalnego podpisu przy wykorzystaniu poszczególnych pól nagłówka IP. Na rysunku 4 przedstawiono widok ramki protokołu IP, na której zaznaczono te pola, które mogą być (przynajmniej częściowo) dobrym nośnikiem ukrytej informacji.

Widoczne jest (tabela 3), że największa liczba bitów na przechowanie niewidocznego podpisu (około połowa objętości nagłówka IP) dostępna jest w sytuacji, w której nadawca i odbiorca znajdują się w tej samej sieci i nie komunikują się za pośrednictwem żadnego routera. Wystarcza ona na przechowanie silnego podpisu. Tak duża liczba nadmiarowych bitów wynika z braku obecności na trasie transmisji routera, który dokonywałby sprawdzania poprawności wartości niektórych pól nagłówka. Ponieważ w praktyce sytuacja, w której nadawca i odbiorca znajdują się w tej samej sieci jest spotykana bardzo rzadko, zostanie ona pominięta w dalszych rozważaniach.

Tabela 3. Liczba nadmiarowych bitów udostępnianych na niedostrzegalny podpis przez poszczególne pola nagłówka protokołu IP w różnych sytuacjach

Nazwa pola	Długość pola w bitach	Maksymalna liczba nadmiarowych bitów w jednej ramce	Maksymalna liczba nadmiarowych bitów przy pewności braku fragmentacji	Maksymalna liczba nadmiarowych bitów przy fragmentacji lokalnej i pewności braku dalszej fragmentacji	Maksymalna liczba nadmiarowych bitów w przypadku bezpośredniego połączenia i pewności braku fragmentacji
1	2	3	4	5	6
Wersja protokołu	4	–	–	–	4
Długość nagłówka ramki	4	–	–	–	4
Typ usługi	8	$_{-1)}$	$_{-1)}$	$_{-1)}$	$_{-1)}$
Długość całkowita ramki	16	–	–	–	16
Identyfikator	16	$16 - k^{10}) \approx 13$	16	$16 - k^{10}) \approx 13$	16
Fragmentacja zabroniona	1	–	1	$1 * n$	1
Więcej fragmentów	1	$1^{2)}$	1	$1 * n^{2)}$	1
Przesunięcie fragmentu	13	–	13	$(13 - \log_2(k) + 3) * n^{3)}) \approx 5 * n$	13
Licznik czasu	8	$\begin{matrix} 7^{4)}) \\ 8 - k^{5)}) \approx 3 \\ 2^{6)}) \end{matrix}$	$\begin{matrix} 7^{4)}) \\ 8 - k^{5)}) \approx 3 \\ 2^{6)}) \end{matrix}$	$\begin{matrix} 7 * n^{4)}) \\ (8 - k) * n^{5)}) \approx 3 * n \\ 2 * n^{6)}) \end{matrix}$	8

1	2	3	4	5	6
Następny protokół	8	8^7 $8 - \log_2(k)^{8)} \approx 6$ 115 nadm. komb. bitów ⁹⁾	8^7 $8 - \log_2(k)^{8)} \approx 6$ 115 nadm. komb. bitów ⁹⁾	8^7 $(8 - \log_2(k)) * n^{8)}$ $\approx 6 * n$ 115 * n nadm. komb. bitów ⁹⁾	8^7 $8 - \log_2(k)^{8)}$ ≈ 6 115 nadm. komb. bitów ⁹⁾
Suma kontr. nagłówka	16	–	–	–	16
Adres źródł.	32	–	–	–	–
Adres docel.	32	–	–	–	–
Opcje	0–320	– ¹⁾	– ¹⁾	– ¹⁾	– ¹⁾
Wypełnienie	0–24	– ¹⁾	– ¹⁾	– ¹⁾	– ¹⁾
Suma¹¹⁾	–	≈ 26	≈ 44	$\approx 32 * n$	≈ 85

¹⁾ Pole umożliwia przechowanie nadmiarowej informacji, ale nie w sposób niewidoczny dla podsłuchującego

²⁾ Gdy w wyniku fragmentacji powstaje stała, znana odbiorcy liczba fragmentów

³⁾ Przy założeniu, że rozmiar ramki przed lokalną fragmentacją nie przekracza k

⁴⁾ Gdy wszystkie możliwe trasy do odbiorcy mają tę samą długość

⁵⁾ Gdy najdłuższa z możliwych tras do odbiorcy ma długość nie większą niż 2^k

⁶⁾ Gdy nie można określić długości najdłuższej z możliwych tras do odbiorcy

⁷⁾ Gdy nadawca ustalił z odbiorcą stosowanie tylko jednego protokołu warstwy transportowej

⁸⁾ Gdy nadawca ustalił z odbiorcą stosowanie k protokołów warstwy transportowej

⁹⁾ Nadmiarowe kombinacje wynikają z przyporządkowania numerów tylko 141 protokołom warstwy transportowej; stan na grudzień 2005r.

¹⁰⁾ Przy założeniu, że k bitów przeznaczono na zapewnienie niepowtarzalności wartości pola „Identyfikator” w krótkich odstępach czasu

¹¹⁾ Podczas wyznaczania sumy ilości nadmiarowych bitów w poszczególnych sytuacjach, dla pól „Licznik czasu” i „Następny protokół” przyjęto warianty najbardziej prawdopodobne, tj. wariant pierwszy dla pola „Licznik czasu” oraz wariant drugi dla pola „Następny protokół”

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
0	Version			IHL			Type of Service						Total length																				
1	Identification															R	D	M	Fragment Offset														
2	Time to Live						Protocol						Header Checksum																				
3	Source Address																																
4	Destination Address																																
5	Options																							Padding									

Rys. 4. Pola nagłówka ramki protokołu IP, w których można ukryć niedostrzegalny podpis.

Pola te zaznaczono kolorem szarym. Kolorem jasnoszarym zaznaczono pola, w których można ukryć niewidoczny podpis przy pewności braku fragmentacji

Jeśli transmisja odbywa się przy współudziale ruterów, to w ogólności liczba bitów mogących być nośnikiem niedostrzegalnego podpisu jest niewielka. Gdy nie ma pewności co do wystąpienia podczas transmisji procesu fragmentacji, w najlepszym przypadku do dyspozycji pozostaje 26 bitów informacji.

Liczba nadmiarowych bitów zwiększa się do 44, jeśli tylko obydwie strony połączenia są pewne, że w czasie transmisji nie zajdzie proces fragmentacji. Sytuację taką zapewnia najczęściej nadawca, konfigurując ustawienia protokołów wyższych warstw niż sieciowa tak, by przekazywały one dane modułowi nadawczemu IP w porcjach nie większych od maksymalnej jednostki transmisji dla całej trasy. Przyprost liczby dostępnych bitów w takiej sytuacji wynika z możliwości wykorzystania do przechowania nadmiarowej informacji pól nagłówka stworzonych z myślą o obłudzie procesu fragmentacji – pola „Przesunięcie fragmentu” oraz flagi „Fragmentowanie zabronione”.

Silny podpis gwarantuje dopiero wielokrotna (przynajmniej trzykrotna), wymuszona fragmentacja ramki na maszynie nadawcy. Nadawca ma wtedy pełną kontrolę nad wartościami pól w nagłówkach powstałych fragmentów i może w nagłówku każdego fragmentu część pól przeznaczyć na przechowanie dodatkowej informacji. Jednocześnie lokalna fragmentacja zapobiega fragmentacji ramki w dalszej drodze, a taka mogłaby uszkodzić podpis. Zapory sieciowe tylko w nieznacznym stopniu mogą przeszkodzić w umieszczeniu niedostrzegalnej informacji w nagłówku IP. Wynika to z faktu, że wartości wielu z pól nagłówka zależą od warunków panujących w sieci na całej trasie od nadawcy do odbiorcy, a stan ramki może być modyfikowany przez urządzenia pośredniczące w transmisji. Zatem wartości w takich polach są trudne do przewidzenia.

Tabela 4. Pola nagłówka IP wykorzystane do przesłania dodatkowej informacji w poszczególnych eksperymentach. Kolorem szarym zaznaczono wszystkie pola, które mogą być wykorzystane do przesłania dodatkowej informacji w danej sytuacji

Nazwa pola	Długość pola w bitach	Eksperyment I (brak ograniczeń) – liczba nadmiarowych bitów w jednej ramce	Eksperyment II (pewność braku fragmentacji) – liczba nadmiarowych bitów w jednej ramce	Eksperyment III (fragmentacja lokalna i pewność braku dalszej fragmentacji) – liczba nadmiarowych bitów w jednej ramce (n – liczba fragmentów)
1	2	3	4	5
Wersja protokołu	4	–	–	–
Długość nagłówka ramki	4	–	–	–
Typ usługi	8	–	–	–
Długość całkowita ramki	16	–	–	–
Identyfikator	16	–	16	–

1	2	3	4	5
Fragmentacja zabroniona	1	–	–	$1 * n$
Więcej fragmentów	1	–	–	–
Przesunięcie fragmentu	13	–	8	$5 * n$
Licznik czasu	8	7	7	$7 * n$
Następny protokół	8	3	–	$3 * n$
Suma kontrolna nagłówka	16	–	–	–
Adres źródłowy	32	–	–	–
Adres docelowy	32	–	–	–
Opcje	0–320	–	–	–
Wypełnienie	0–24	–	–	–
Suma	–	10	31	$16 * n$

Eksperymenty z udziałem rutera potwierdziły, że w polach „Identyfikator”, „Licznik czasu”, „Następny protokół”, „Przesunięcie fragmentu”, „Fragmentowanie zabronione” i „Rozmiar okna” można ukryć dodatkową informację tak, by nie powodowała ona błędów podczas transmisji. W tabeli 4 zaznaczono te pola nagłówka IP, które zostały wykorzystane do przesłania nadmiarowej informacji.

Tabela 5. Pola nagłówka IP wykorzystane do przesłania dodatkowej informacji w poszczególnych eksperymentach. Kolorem szarym zaznaczono wszystkie pola, które mogą być wykorzystane do przesłania dodatkowej informacji w danej sytuacji

Nazwa pola	Długość pola w bitach	Eksperyment I (brak ograniczeń) – liczba nadmiarowych bitów w jednej ramce	Eksperyment II (pewność braku fragmentacji) – liczba nadmiarowych bitów w jednej ramce	Eksperyment III (fragmentacja lokalna i pewność braku dalszej fragmentacji) – liczba nadmiarowych bitów w jednej ramce (n – liczba fragmentów)
1	2	3	4	5
Wersja protokołu	4	–	–	–
Długość nagłówka ramki	4	–	–	–
Typ usługi	8	–	–	–
Długość całkowita ramki	16	–	–	–
Identyfikator	16	–	16	–
Fragmentacja zabroniona	1	–	–	$1 * n$
Więcej fragmentów	1	–	–	–

1	2	3	4	5
Przesunięcie fragmentu	13	–	8	$5 * n$
Licznik czasu	8	7	7	$7 * n$
Następny protokół	8	3	–	$3 * n$
Suma kontrolna nagłówka	16	–	–	–
Adres źródłowy	32	–	–	–
Adres docelowy	32	–	–	–
Opcje	0–320	–	–	–
Wypełnienie	0–24	–	–	–
Suma	–	10	31	$16 * n$

Tabela 6. Pola nagłówka IP wykorzystane do przesłania dodatkowej informacji w poszczególnych eksperymentach. Kolorem szarym zaznaczono wszystkie pola, które mogą być wykorzystane do przesłania dodatkowej informacji w danej sytuacji

Nazwa pola	Długość pola w bitach	Eksperyment I (brak ograniczeń) – liczba nadmiarowych bitów w jednej ramce	Eksperyment II (pewność braku fragmentacji) – liczba nadmiarowych bitów w jednej ramce	Eksperyment III (fragmentacja lokalna i pewność braku dalszej fragmentacji) – liczba nadmiarowych bitów w jednej ramce (n – liczba fragmentów)
Wersja protokołu	4	–	–	–
Długość nagłówka ramki	4	–	–	–
Typ usługi	8	–	–	–
Długość całkowita ramki	16	–	–	–
Identyfikator	16	–	16	–
Fragmentacja zabroniona	1	–	–	$1 * n$
Więcej fragmentów	1	–	–	–
Przesunięcie fragmentu	13	–	8	$5 * n$
Licznik czasu	8	7	7	$7 * n$
Następny protokół	8	3	–	$3 * n$
Suma kontrolna nagłówka	16	–	–	–
Adres źródłowy	32	–	–	–
Adres docelowy	32	–	–	–
Opcje	0–320	–	–	–
Wypełnienie	0–24	–	–	–
Suma	–	10	31	$16 * n$

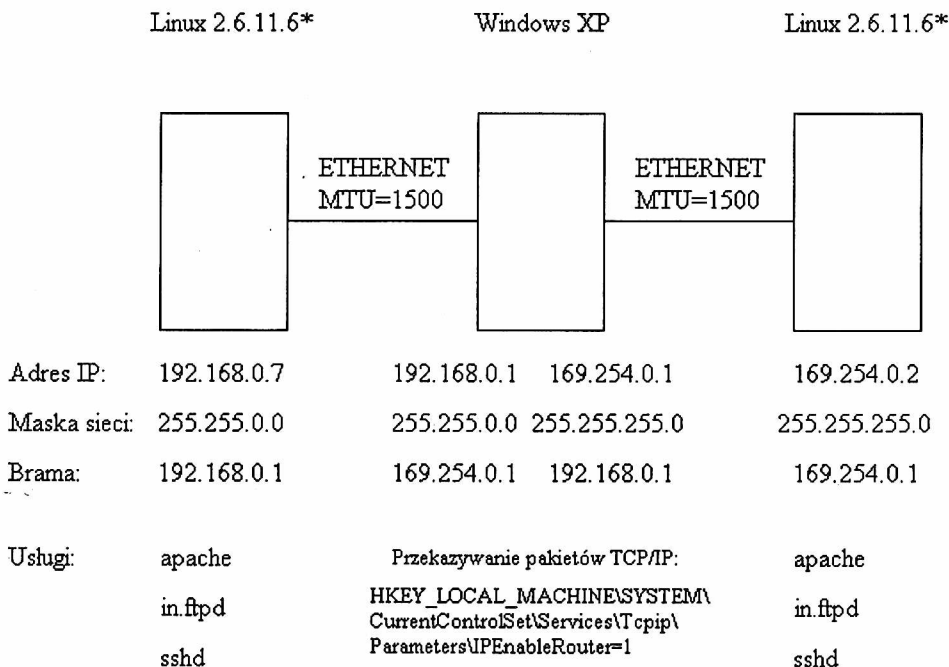
3.1. BADANIE MOŻLIWOŚCI WYKORZYSTANIA PÓL „LICZNIK CZASU” ORAZ „PROTOKÓŁ” NAGŁÓWKA IP DO PRZECHOWANIA NIEDOSTRZEGALNEGO PODPISU

W celu przeprowadzenia eksperymentów wykonano układ eksperymentalny, którego schemat przedstawiono na rysunku 5. Każdy z eksperymentów został wykonany niezależnie, tj. po każdym z nich następowało ponowne uruchomienie komputera ze specjalnie przygotowanej startowej płyty CD oraz zapewnione zostały identyczne warunki początkowe. Badania przeprowadzono przy wykorzystaniu protokołów HTTP, FTP oraz polecenia *ping* korzystających z zaimplementowanego oprogramowania modyfikującego nagłówki w protokole IP.

Przed wykonaniem eksperymentu wykonywana była inicjalizacja systemów na komputerach w modelowej sieci. Po zalogowaniu się jako użytkownik *root* i poprawnym skonfigurowaniu urządzeń sieciowych nastąpiło wykonanie poleceń inicjalizujących:

```
(na maszynie o adresie 169.254.0.2)
echo „117483712” > /proc/sys/net/ipv4/ip_inv_sign_rcv_src
(co oznacza 192.168.0.7)
echo „1 1 3” > /proc/sys/net/ipv4/ip_inv_sign_rcv_modes
echo „AAAAAAAAAAAAAAAAAAAA” >
/proc/sys/net/ipv4/ip_inv_sign_rcv_key
echo „117483712” > /proc/sys/net/ipv4/ip_inv_sign_snd_dst
(co oznacza 192.168.0.7)
echo „1 1 3” > /proc/sys/net/ipv4/ip_inv_sign_snd_modes
echo „BBBBBBBBBBBBBBBBBBBB” >
/proc/sys/net/ipv4/ip_inv_sign_snd_key
```

```
(na maszynie o adresie 192.168.0.7)
echo „33619625” > /proc/sys/net/ipv4/ip_inv_sign_rcv_src
(co oznacza 169.254.0.2)
echo „1 1 3” > /proc/sys/net/ipv4/ip_inv_sign_rcv_modes
echo „BBBBBBBBBBBBBBBBBBBB” >
/proc/sys/net/ipv4/ip_inv_sign_rcv_key
echo „ 33619625” > /proc/sys/net/ipv4/ip_inv_sign_snd_dst
(co oznacza 169.254.0.2)
echo „1 1 3” > /proc/sys/net/ipv4/ip_inv_sign_snd_modes
echo „AAAAAAAAAAAAAAAAAAAA” >
/proc/sys/net/ipv4/ip_inv_sign_snd_key
```



* - z modyfikacjami

Rys. 5. Schemat sieci użytej w eksperymentach

Powyższe sekwencje poleceń zapewniły ruch dwustronnie podpisywany przy wykorzystaniu pól „Licznik czasu” (7 najmniej znaczących bitów, odległość: 1 ruter) oraz „Protokół” (3 najbardziej znaczące bity).

3.1.1. BADANIA Z WYKORZYSTANIEM PROTOKOŁU HTTP

W celu rozpoczęcia eksperymentu na maszynie o adresie 192.168.0.7 wykonano polecenie:

```
lynx 169.254.0.2
```

Spowodowało to połączenie się przeglądarki WWW z adresu 192.168.0.7 z serwerem WWW *apache* na maszynie 169.254.0.2. Na stronie startowej wybrano ściągnięcie pliku o wielkości 8738766 bajtów, wciskając na klawiaturze klawisz „d” po podświetleniu odnośnika do niego.

Przebieg eksperymentu był obserwowany z komputera pośredniczącego w transmisji przy pomocy programu analizatora pakietów sieciowych Ethereal. Jak zostało

przedstawione na rysunku 6, pola „Licznik czasu” oraz „Protokół” przyjmują w czasie połączenia różne wartości, gdyż w rzeczywistości są nośnikiem ukrytego podpisu.

Id.	Time	Source	Destination	Protocol	Info
58	19.129776	192.168.0.7	169.254.0.2	TCP	46251 > http [ACK] Seq=5
59	19.130300	169.254.0.2	192.168.0.7	IP	Unknown (0xc6)
60	19.131031	169.254.0.2	192.168.0.7	IP	Unknown (0xa6)
61	19.131606	192.168.0.7	169.254.0.2	TCP	46251 > http [ACK] Seq=5
62	19.132375	169.254.0.2	192.168.0.7	IP	RSVP E2EI (0x86)
63	19.133605	169.254.0.2	192.168.0.7	IP	Control Message (0x26)
64	19.134314	192.168.0.7	169.254.0.2	TCP	46251 > http [ACK] Seq=5
65	19.134833	169.254.0.2	192.168.0.7	IP	Unknown (0xa6)
66	19.136063	169.254.0.2	192.168.0.7	IP	Unknown (0xa6)
67	19.136742	192.168.0.7	169.254.0.2	TCP	46251 > http [ACK] Seq=5
68	19.137291	169.254.0.2	192.168.0.7	IP	Unknown (0xa6)
69	19.138045	169.254.0.2	192.168.0.7	IP	Unknown (0xa6)
70	19.138633	192.168.0.7	169.254.0.2	TCP	46251 > http [ACK] Seq=5
71	19.139353	169.254.0.2	192.168.0.7	IP	RSVP E2EI (0x86)
72	19.140608	169.254.0.2	192.168.0.7	IP	Unknown (0xa6)
73	19.141321	192.168.0.7	169.254.0.2	TCP	46251 > http [ACK] Seq=5
74	19.141811	169.254.0.2	192.168.0.7	IP	VISA (0x46)
75	19.143125	169.254.0.2	192.168.0.7	IP	VISA (0x46)
76	19.143819	192.168.0.7	169.254.0.2	TCP	46251 > http [ACK] Seq=5
77	19.144329	169.254.0.2	192.168.0.7	IP	VISA (0x46)
78	19.145045	169.254.0.2	192.168.0.7	IP	PNNI over IP (0x66)

▶ Ethernet II, Src: 00:11:d8:14:c4:75, Dst: 00:c0:df:13:29:ab					
▼ Internet Protocol, Src Addr: 169.254.0.2 (169.254.0.2), Dst Addr: 192.168.0.7					
Version: 4					
Header length: 20 bytes					
▶ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)					
Total Length: 1004					
Identification: 0x5e36 (24118)					
▶ Flags: 0x04 (Don't Fragment)					
Fragment offset: 0					
Time to live: 139					
Protocol: Unknown (0xa6)					
Header checksum: 0x2286 (correct)					
Source: 169.254.0.2 (169.254.0.2)					
Destination: 192.168.0.7 (192.168.0.7)					
Data (984 bytes)					

Rys. 6. Zrzut ekranu z programu analizatora pakietów sieciowych Ethereal podczas podpisanej transmisji HTTP

Po zakończeniu transferu pliku porównano jego zawartość z zawartością pliku oryginalnego, który był dostępny wcześniej na maszynie klienta:

```
#diff plik_sciagniete plik_stary
#
```


Stwierdzono, że połączenie przebiegło bez zakłóceń i niewidostrzegalny podpis nie miał żadnego wpływu na przebieg połączenia. W ramach eksperymentu zbadano ilość wystąpień ciągów „PODPIS PRAWIDŁOWY” oraz „PODPIS NIEPRAWIDŁOWY” w plikach /var/log/kern.log na maszynie 192.168.0.7:

```
#grep -c „PODPIS PRAWIDŁOWY” /var/log/kern.log
2826
#grep -c „PODPIS NIEPRAWIDŁOWY” /var/log/kern.log
0
```

oraz na maszynie 169.254.0.2:

```
#grep -c „PODPIS PRAWIDŁOWY” /var/log/kern.log
481
#grep -c „PODPIS NIEPRAWIDŁOWY” /var/log/kern.log
0
```

Otrzymane rezultaty potwierdziły poprawną weryfikację podpisu w ramach, w których on wystąpił.

3.1.2. BADANIA Z WYKORZYSTANIEM PROTOKOŁU FTP

Eksperyment z wykorzystaniem protokołu FTP rozpoczęty został od inicjalizacji. Na maszynie o adresie 192.168.0.7 wykonano polecenie:

```
ftp 169.254.0.2
```

co spowodowało nawiązanie połączenia z serwerem FTP uruchomionym na maszynie o adresie 169.254.0.2. Po nawiązaniu połączenia i zalogowaniu się jako użytkownik *root*, rozpoczęto ściąganie pliku o rozmiarze 8738766 bajtów poleceniem:

```
get plik
```

Przebieg eksperymentu był obserwowany z komputera pośredniczącego w transmisji przy pomocy programu analizatora pakietów sieciowych Ethereal. Na rysunku 7, pola „Licznik czasu” oraz „Protokół” mają różne wartości w trakcie trwania połączenia, gdyż w rzeczywistości są nośnikiem ukrytego podpisu.

Po zakończeniu ściągania pliku przez program kliencki ftp został wyświetlony komunikat:

```
226 Transfer complete.
8738766 bytes received in 10.61 secs (804.0 kB/s)
```

oraz pojawił się typowy znak zachęty tego programu. Następnie porównano zawartość ściągniętego pliku z zawartością pliku oryginalnego, który był dostępny wcześniej na maszynie klienta:

```
#diff plik_sciagniety plik_stary
#
```

No.	Time	Source	Destination	Protocol	Info
44	5.825127	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Control Message 0x26, off=)
45	5.825493	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Unknown 0xc6, off=24576)
46	5.825544	192.168.0.7	169.254.0.2	TCP	32774 > ftp-data [ACK] Seq=1515465473 Ack=3525363362 w'
47	5.825678	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=176)
48	5.826047	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Unknown 0xe6, off=33120)
49	5.826232	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Unknown 0xa6, off=0)
50	5.826418	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=32944)
51	5.826470	192.168.0.7	169.254.0.2	TCP	32774 > ftp-data [ACK] Seq=1515465473 Ack=3525363862 w'
52	5.826608	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Control Message 0x26, off=)
53	5.826886	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Unknown 0xe6, off=32768)
54	5.827095	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=PNMI over IP 0x66, off=5)
55	5.827117	192.168.0.7	169.254.0.2	TCP	32774 > ftp-data [ACK] Seq=1515465473 Ack=3525364362 w'
56	5.827233	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Control Message 0x26, off=)
57	5.827424	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Unknown 0xa6, off=0)
58	5.827707	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Unknown 0xa6, off=24576)
59	5.827730	192.168.0.7	169.254.0.2	TCP	32774 > ftp-data [ACK] Seq=1515465473 Ack=3525364810 w'
60	5.827871	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Unknown 0xa6, off=16736)
61	5.828211	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=VISA 0x46, off=24576)
62	5.828342	192.168.0.7	169.254.0.2	TCP	32774 > ftp-data [ACK] Seq=1515465473 Ack=3525365310 w'
63	5.828441	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=57520)
64	5.828586	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=24928)
65	5.828852	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Unknown 0xc6, off=32768)

Internet Protocol, Src Addr: 169.254.0.2 (169.254.0.2), Dst Addr: 192.168.0.7 (192.168.0.7)	
Version: 4	
Header length: 20 bytes	
Differentiated Services Field: 0x08 (DSCP 0x02: Unknown DSCP; ECN: 0x00)	
Total Length: 196	
Identification: 0xec09 (60425)	
Flags: 0x06 (Don't Fragment) (More Fragments)	
Fragment offset: 24752	
Time to live: 219	
Protocol: Unknown (0xa6)	
Header checksum: 0x1bbc (correct)	
Source: 169.254.0.2 (169.254.0.2)	
Destination: 192.168.0.7 (192.168.0.7)	
Data (176 bytes)	

Rys. 7. Widok ekranu z programu analizatora pakietów sieciowych Ethereal podczas podpisanej transmisji FTP

Przeprowadzona transmisja pozbawiona była zakłóceń a po jej zakończeniu klient i serwer były w stanie uczestniczyć w kolejnych połączeniach, co oznacza to, że niedostrzegalny podpis nie miał żadnego wpływu na przebieg połączenia. Podobnie jak przy pozostałych eksperymentach zbadano ilość wystąpień ciągów „PODPIS PRAWDŁOWY” oraz „PODPIS NIEPRAWDŁOWY” w plikach /var/log/kern.log na maszynie 192.168.0.7. Wszystkie przeprowadzone eksperymenty potwierdziły poprawność niedostrzegalnego sygnowania informacji.

3.1.3. BADANIA Z WYKORZYSTANIEM POLECENIA PING

Eksperymenty z wykorzystaniem polecenia PING rozpoczyna się od inicjalizacji. Badaniu podlegały partie ok. 2 MB podpisanych danych wysyłanych przy pomocy

standardowego polecenia PING uruchamianego na komputerach ze zmodyfikowanym oprogramowaniem jądra:

```
ping -c 100 -s 20000 169.254.0.2
```

Przebieg eksperymentu był obserwowany z komputera pośredniczącego w transmisji przy pomocy programu analizatora pakietów sieciowych Ethereal. Jak zostało przedstawione na rysunku 8, pola „Licznik czasu” oraz „Protokół” mają w trakcie działania programu różne wartości, gdyż w rzeczywistości są nośnikiem niedostrzegalnego podpisu.

no.	Time	Source	Destination	Protocol	Info
8	8.952002	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=1480)
9	8.952395	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=2960)
10	8.952402	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=4440)
11	8.952415	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=5920)
12	8.952423	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=7400)
13	8.952431	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=8880)
14	8.952438	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=10360)
15	8.952535	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=11840)
16	8.952658	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=13320)
17	8.952780	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=14800)
18	8.952919	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=16280)
19	8.953028	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=17760)
20	8.953092	192.168.0.7	169.254.0.2	ICMP	Echo (ping) request
21	8.972020	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Kryptolan 0x41, off=0)
22	8.973212	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Kryptolan 0x41, off=148)
23	8.974405	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Kryptolan 0x41, off=296)
24	8.975670	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Kryptolan 0x41, off=444)
25	8.976936	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Kryptolan 0x41, off=592)
26	8.978133	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Kryptolan 0x41, off=740)
27	8.979355	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Kryptolan 0x41, off=888)
28	8.980585	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Kryptolan 0x41, off=103)

▾ Internet Protocol, Src Addr: 169.254.0.2 (169.254.0.2), Dst Addr: 192.168.0.7 (192.168.0.7)
 Version: 4
 Header length: 20 bytes
 ▸ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
 Total Length: 1500
 Identification: 0xc181 (49537)
 ▸ Flags: 0x02 (More Fragments)
 Fragment offset: 2960
 Time to live: 169
 Protocol: Kryptolan (0x41)
 Header checksum: 0xbe3d (correct)
 Source: 169.254.0.2 (169.254.0.2)
 Destination: 192.168.0.7 (192.168.0.7)
 Reassembled IP in frame: 34
 Data (1480 bytes)

Rys. 8. Widok ekranu programu analizatora pakietów sieciowych Ethereal podczas podpisanej transmisji żądania ECHO protokołu ICMP

Eksperyment przeprowadzono wielokrotnie badając liczbę wystąpień ciągów „PODPIS PRAWDŁOWY” oraz „PODPIS NIEPRAWDŁOWY” w plikach /var/log/kern.log na maszynie 192.168.0.7. Uzyskane wyniki świadczą o poprawnej weryfikacji podpisu w każdej z odebranych ramek.

3.2. BADANIE MOŻLIWOŚCI WYKORZYSTANIA PÓŁ „LICZNIK CZASU”, „PRZESUNIĘCIE FRAGMENTU” ORAZ „IDENTYFIKATOR” NAGŁÓWKA IP DO PRZECHOWANIA NIEDOSTRZEGALNEGO PODPISU W PRZYPADKU PEWNOŚCI BRAKU FRAGMENTACJI

W ramach pracy przeprowadzono badania stanowiskowe możliwości wykorzystania pól „Licznik Czasu”, „Przesunięcie fragmentu” oraz „Identyfikator” nagłówka IP do przechowywania niedostrzegalnego podpisu. Każdy z eksperymentów został wykonany niezależnie, tj. po każdym z nich następowało ponowne uruchomienie komputera z płyty CD oraz zapewnione zostały identyczne warunki początkowe. Badanie przeprowadzono na przykładzie protokołów HTTP, FTP oraz polecenia *ping*. Po zalogowaniu się jako użytkownik *root* i poprawnym skonfigurowaniu urządzeń sieciowych nastąpiło wykonanie poleceń inicjalizujących zapewniających ruch dwustronnie podpisanych ramek:

```
(na maszynie o adresie 169.254.0.2)
echo „117483712” >
/proc/sys/net/ipv4/ip_inv_sign_rcv_src (co oznacza
192.168.0.7)
echo          „2          1          1          1”          >
/proc/sys/net/ipv4/ip_inv_sign_rcv_modes
echo          „AAAAAAAAAAAAAAAAAAAAA”          >
/proc/sys/net/ipv4/ip_inv_sign_rcv_key
echo „117483712” >
/proc/sys/net/ipv4/ip_inv_sign_snd_dst (co oznacza
192.168.0.7)
echo          „2          1          1          1”          >
/proc/sys/net/ipv4/ip_inv_sign_snd_modes
echo          „BBBBBBBBBBBBBBBBBBBBB”          >
/proc/sys/net/ipv4/ip_inv_sign_snd_key
(na maszynie o adresie 192.168.0.7)
echo „33619625” >
/proc/sys/net/ipv4/ip_inv_sign_rcv_src (co oznacza
169.254.0.2)
echo          „2          1          1          1”          >
/proc/sys/net/ipv4/ip_inv_sign_rcv_modes
echo          „BBBBBBBBBBBBBBBBBBBBB”          >
/proc/sys/net/ipv4/ip_inv_sign_rcv_key
echo „ 33619625” >
/proc/sys/net/ipv4/ip_inv_sign_snd_dst (co oznacza
169.254.0.2)
```

```

echo      "2      1      1      1"      >
/proc/sys/net/ipv4/ip_inv_sign_snd_modes
echo      "AAAAAAAAAAAAAAAAAAAAAA"      >
/proc/sys/net/ipv4/ip_inv_sign_snd_key

```

3.2.1. BADANIA Z WYKORZYSTANIEM PROTOKOŁU HTTP

Eksperymenty z wykorzystaniem protokołu HTTP przeprowadzono zaczynając od wykonania na maszynie o adresie 192.168.0.7 polecenia:

```
lynx 169.254.0.2
```

co spowodowało połączenie się przeglądarki WWW z adresu 192.168.0.7 z serwerem WWW *apache* na maszynie 169.254.0.2. Na stronie startowej wybrano ściągnięcie pliku o wielkości 8738766 bajtów, wciskając na klawiaturze klawisz „d” po podświetleniu odnośnika do niego.

lo.	Time	Source	Destination	rotoco	Info
86	92.407363	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=65024)
87	92.408616	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=64768)
88	92.408978	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=58368)
89	92.409821	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=51968)
90	92.411049	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=24576)
91	92.411450	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=41216)
92	92.412282	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=31232)
93	92.413064	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=26368)
94	92.414402	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=42496)
95	92.415631	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=22784)
96	92.416907	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=37632)
97	92.417281	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=39680)
98	92.418172	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=55040)
99	92.419400	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=43008)
00	92.419787	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=46592)
01	92.420124	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=39936)
02	92.422364	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=44544)
03	92.462304	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=43008)
04	92.463710	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=52480)
05	92.464978	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=60928)
06	92.465343	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=34048)
07	92.466205	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=53760)
08	92.467521	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=64000)

▶ Frame 398 (1514 bytes on wire, 1514 bytes captured)
 ▶ Ethernet II, Src: 00:e0:4c:01:ed:1b, Dst: 00:50:ba:3a:ad:85
 ▼ Internet Protocol, Src Addr: 192.168.0.7 (192.168.0.7), Dst Addr: 169.254.0.2 (169.254.0.2)
 Version: 4
 Header length: 20 bytes
 ▶ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
 Total Length: 1500
 Identification: 0x9d4b (40267)
 ▶ Flags: 0x00
 Fragment offset: 55040
 Time to live: 246
 Protocol: TCP (0x06)
 Header checksum: 0x9c40 (correct)
 Source: 192.168.0.7 (192.168.0.7)

Rys. 9. Widok ekranu programu analizatora pakietów sieciowych Ethereal podczas podpisanej transmisji HTTP

transmisji plików dostępnego dla użytkownika. W ramach eksperymentu na maszynie o adresie 192.168.0.7 wykonano polecenie:

```
ftp 169.254.0.2
```

co spowodowało nawiązanie połączenia z serwerem FTP uruchomionym na maszynie o adresie 169.254.0.2. Po nawiązaniu połączenia i zalogowaniu się jako użytkownik *root*, rozpoczęto ściąganie pliku o rozmiarze 8738766 bajtów poleceniem:

```
get plik
```

Przebieg eksperymentu był obserwowany z komputera pośredniczącego w transmisji przy pomocy programu analizatora pakietów sieciowych Ethereal. Jak przedstawiono na rysunkach 11 i 12, pola „Licznik czasu”, „Przesunięcie fragmentu” oraz „Identyfikator” mają różne wartości dla kolejno wysyłanych pakietów, gdyż w rzeczywistości są nośnikiem ukrytego podpisu.

No.	Time	Source	Destination	Protocol	Info
31	18.468648	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=60416)
32	18.469319	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=51968)
33	18.469714	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=61184)
34	18.470053	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=42496)
35	18.470193	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=38912)
36	18.510067	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=31488)
37	18.561132	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=32768)
38	18.562383	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=4608)
39	18.562743	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=12032)
40	18.563336	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=11264)
41	18.564626	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=50432)
42	18.565555	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=32000)
43	18.565941	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=19456)
44	18.566860	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=18176)
45	18.575807	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=15872)
46	18.577059	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=11264)
47	18.577418	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=2304)
48	18.578290	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=3840)
49	18.578653	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=1536)
50	18.579408	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=23552)
51	18.579791	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=55040)
52	18.580843	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=61184)
53	18.581227	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=28160)

▶ Frame 45 (1514 bytes on wire, 1514 bytes captured)

▶ Ethernet II, Src: 00:e0:4c:01:ed:1b, Dst: 00:50:ba:3a:ad:85

▼ Internet Protocol, Src Addr: 192.168.0.7 (192.168.0.7), Dst Addr: 169.254.0.2 (169.254.0.2)

Version: 4

Header Length: 20 bytes

▶ Differentiated Services Field: 0x08 (DSCP 0x02: Unknown DSCP; ECN: 0x00)

Total Length: 1500

Identification: 0x099f (2463)

▶ Flags: 0x00

Fragment offset: 15872

Time to live: 248

Protocol: TCP (0x06)

Header checksum: 0x4105 (correct)

Source: 192.168.0.7 (192.168.0.7)

Rys. 11. Widok ekranu programu analizatora pakietów sieciowych Ethereal podczas podpisanego połączenia FTP

Połączenia dokonywane w ramach eksperymentu przebiegły bez zakłóceń, po ich zakończeniu klient i serwer były w stanie uczestniczyć w kolejnych połączeniach.

Oznacza to, że niewidoczny podpis nie miał żadnego wpływu na przebieg połączenia. W ramach eksperymentu zbadano ilość wystąpień ciągów „PODPIS PRAWDLOWY” oraz „PODPIS NIEPRAWDLOWY” w plikach /var/log/kern.log na maszynie 192.168.0.7 oraz na maszynie 169.254.0.2. Uzyskane wyniki świadczą o poprawnej weryfikacji podpisu w ramach, w których on wystąpił.

no.	Time	Source	Destination	protocol	Info
31	18.468648	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=60416)
32	18.469319	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=51968)
33	18.469714	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=61184)
34	18.470053	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=42496)
35	18.470193	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=38912)
36	18.510067	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=31488)
37	18.561132	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=32768)
38	18.562383	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=4608)
39	18.562743	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=12032)
40	18.563336	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=11264)
41	18.564626	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=50432)
42	18.565555	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=32000)
43	18.565941	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=19456)
44	18.566860	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=18176)
45	18.575807	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=15872)
46	18.577059	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=11264)
47	18.577418	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=2304)
48	18.578290	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=3840)
49	18.578653	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=1536)
50	18.579408	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=23552)
51	18.579791	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=55040)
52	18.580843	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=61184)
53	18.581227	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=TCP 0x06, off=28160)

```

> Frame 46 (1514 bytes on wire, 1514 bytes captured)
> Ethernet II, Src: 00:e0:4c:01:ed:1b, Dst: 00:50:ba:3a:ad:85
> Internet Protocol, Src Addr: 192.168.0.7 (192.168.0.7), Dst Addr: 169.254.0.2 (169.254.0.2)
  Version: 4
  Header length: 20 bytes
  > Differentiated Services Field: 0x08 (DSCP 0x02: Unknown DSCP; ECN: 0x00)
    Total Length: 1500
    Identification: 0xa067 (41063)
  > Flags: 0x00
    Fragment offset: 11264
    Time to live: 165
    Protocol: TCP (0x06)
    Header checksum: 0xff7c (correct)
    Source: 192.168.0.7 (192.168.0.7)

```

Rys. 12. Widok ekranu programu analizatora pakietów sieciowych Ethereal podczas podpisanej transmisji FTP

3.2.3. BADANIA Z WYKORZYSTANIEM POLECENIA PING

Podobnie jak w poprzednim eksperymencie z wykorzystaniem polecenia PING na maszynie o adresie 192.168.0.7 wykonano polecenie:

```
ping -c 100 -s 1400 169.254.0.2
```

co spowodowało wysłanie do maszyny o adresie 169.254.0.2 ponad 140 kB podpisanych danych. W wyniku wykonania polecenia również do nadawcy zwrócono ok. 140 kB podpisanych danych.

Przebieg eksperymentu był obserwowany z komputera pośredniczącego w transmisji przy pomocy programu analizatora pakietów sieciowych Ethereal. Jak przedstawiono na rysunkach 13 i 14, pola „Licznik czasu”, „Przesunięcie fragmentu” oraz „Identyfikator” mają różne wartości dla kolejno wysyłanych pakietów, gdyż w rzeczywistości są nośnikiem ukrytego podpisu.

Po zakończeniu działania programu *ping*, na maszynie nadawcy (192.168.0.7) wyświetlony został komunikat:

```
---169.254.0.2 ping statistics ---
```

```
100 packets transmitted, 100 packed received, 0% packet loss
```

co świadczy o braku ingerencji niewidocznego podpisu w działanie programu *ping*.

Przeprowadzono badania ilości wystąpień ciągów „PODPIS PRAWDLOWY” oraz „PODPIS NIEPRAWDLOWY” w plikach */var/log/kern.log* na maszynie 192.168.0.7 oraz na maszynie 169.254.0.2. Uzyskano wyniki świadczące o poprawnej weryfikacji podpisu w każdej z odebranych ramek.

No.	Time	Source	Destination	Protocol	Info
8	2.004689	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=64000)
9	3.002204	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=ICMP 0x01, off=48640)
10	3.005446	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=41984)
11	4.002967	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=ICMP 0x01, off=8192)
12	4.006214	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=55296)
13	5.003734	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=ICMP 0x01, off=58368)
14	5.006978	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=23552)
15	6.004491	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=ICMP 0x01, off=54272)
16	6.007748	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=5632)
17	7.005252	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=ICMP 0x01, off=57088)
18	7.008491	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=52736)
19	8.006003	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=ICMP 0x01, off=45312)
20	8.009247	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=27648)
21	9.006777	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=ICMP 0x01, off=56832)
22	9.010016	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=19200)
23	10.007546	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=ICMP 0x01, off=60672)
24	10.010789	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=40960)
25	11.008303	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=ICMP 0x01, off=1536)
26	11.011542	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=32512)
27	12.009071	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=ICMP 0x01, off=2048)
28	12.012313	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=13824)
29	13.009832	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=ICMP 0x01, off=11520)
30	13.013072	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=38400)
▶ Frame 16 (1442 bytes on wire, 1442 bytes captured)					
▶ Ethernet II, Src: 00:e0:4c:01:ed:1b, Dst: 00:50:ba:3a:ad:85					
▼ Internet Protocol, Src Addr: 192.168.0.7 (192.168.0.7), Dst Addr: 169.254.0.2 (169.254.0.2)					
Version: 4					
Header length: 20 bytes					
▶ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)					
Total Length: 1428					
Identification: 0xd3e (7486)					
▶ Flags: 0x00					
Fragment offset: 5632					
Time to live: 227					
Protocol: ICMP (0x01)					
Header checksum: 0x47bb (correct)					
Source: 192.168.0.7 (192.168.0.7)					

Rys. 13. Widok ekranu programu analizatora pakietów sieciowych Ethereal podczas podpisanej transmisji żądania ECHO protokołu ICMP

No.	Time	Source	Destination	Protocol	Info
8	2.004689	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=64000)
9	3.002204	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=ICMP 0x01, off=48640)
10	3.005446	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=41984)
11	4.002967	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=ICMP 0x01, off=8192)
12	4.006214	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=55296)
13	5.003734	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=ICMP 0x01, off=58368)
14	5.006978	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=23552)
15	6.004491	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=ICMP 0x01, off=54272)
16	6.007748	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=5632)
17	7.005252	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=ICMP 0x01, off=57088)
18	7.008491	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=57432)
19	8.006003	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=ICMP 0x01, off=45312)
20	8.009247	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=27648)
21	9.006777	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=ICMP 0x01, off=56832)
22	9.010016	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=19200)
23	10.007546	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=ICMP 0x01, off=60672)
24	10.010789	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=40960)
25	11.008303	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=ICMP 0x01, off=1536)
26	11.011542	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=32512)
27	12.009071	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=ICMP 0x01, off=2048)
28	12.012313	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=13824)
29	13.009832	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=ICMP 0x01, off=11520)
30	13.013072	192.168.0.7	169.254.0.2	IP	Fragmented IP protocol (proto=ICMP 0x01, off=38400)

▶ Frame 18 (1442 bytes on wire, 1442 bytes captured)
 ▶ Ethernet II, Src: 00:e0:4c:01:ed:1b, Dst: 00:50:ba:3a:ad:85
 ▶ Internet Protocol, Src Addr: 192.168.0.7 (192.168.0.7), Dst Addr: 169.254.0.2 (169.254.0.2)
 Version: 4
 Header length: 20 bytes
 ▶ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
 Total Length: 1428
 Identification: 0xa3ca (41930)
 ▶ Flags: 0x00
 Fragment offset: 52736
 Time to live: 138
 Protocol: ICMP (0x01)
 Header checksum: 0x032f (correct)
 Source: 192.168.0.7 (192.168.0.7)

Rys. 14. Widok ekranu programu analizatora pakietów sieciowych Ethereal podczas podpisanej transmisji żądania ECHO protokołu ICMP

3.3. WYKORZYSTANIE PÓŁ „PROTOKÓŁ” ORAZ „FRAGMENTOWANIE ZABRONIONE” NAGŁÓWKA IP DO PRZECHOWANIA NIEDOSTRZEGALNEGO PODPISU W PRZYPADKU ZASTOSOWANIA LOKALNEJ FRAGMENTACJI

W ramach pracy przeprowadzono badania stanowiskowe możliwości wykorzystania pól „Protokół” oraz „Fragmentowanie Zabronione” nagłówka IP do przechowywania niedostrzegalnego podpisu. Każdy z eksperymentów został wykonany niezależnie, tj. po każdym z nich następowało ponowne uruchomienie komputera z płyty CD oraz zapewnione zostały identyczne warunki początkowe. Badanie przeprowadzono na przykładzie protokołów HTTP, FTP oraz polecenia *ping*. Po zalogowaniu się jako użytkownik *root* i poprawnym skonfigurowaniu urządzeń sieciowych nastąpiło wykonanie poleceń inicjalizujących zapewniających ruch dwustronnie podpisanych ramek. Proces badawczy przeprowadzono analogicznie do opisanych w poprzednich rozdziałach.

No.	Time	Source	Destination	Protocol	Info
44	0.012000	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=unknown 0xe6, off=57696)
45	0.012136	169.254.0.2	192.168.0.7	HTTP	Continuation or non-HTTP traffic
46	0.012443	192.168.0.7	169.254.0.2	TCP	35286 > http [ACK] Seq=2950643983 Ack=636231539 Win=3782 Len=0
47	0.041614	192.168.0.7	169.254.0.2	TCP	35286 > http [FIN, ACK] Seq=2950643983 Ack=636231630 Win=3782 Len=0
48	0.041886	169.254.0.2	192.168.0.7	TCP	http > 35286 [ACK] Seq=636231630 Ack=2950643984 Win=1716 Len=0
49	7.132603	192.168.0.7	169.254.0.2	TCP	35287 > http [SYN] Seq=2950162008 Ack=0 Win=5840 Len=0 MSS=5760
50	7.132964	169.254.0.2	192.168.0.7	TCP	http > 35287 [SYN, ACK] Seq=636560131 Ack=2950162009 Win=5760 Len=0
51	7.133233	192.168.0.7	169.254.0.2	TCP	35287 > http [ACK] Seq=2950162009 Ack=636560132 Win=1460 Len=0
52	7.134364	192.168.0.7	169.254.0.2	HTTP	GET /et2 HTTP/1.0
53	7.134952	169.254.0.2	192.168.0.7	TCP	http > 35287 [ACK] Seq=636560132 Ack=2950162296 Win=1698 Len=0
54	7.135518	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Unknown 0xe6, off=16384)
55	7.135703	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Unknown 0xe6, off=8368)
56	7.135892	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=VISA 0x46, off=41312)
57	7.136077	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Control Message 0x26, off=0)
58	7.136262	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=RSVP 0xe1 0x86, off=16560)
59	7.136447	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Unknown 0xe6, off=16384)
60	7.136524	192.168.0.7	169.254.0.2	TCP	35287 > http [ACK] Seq=2950162296 Ack=636560632 Win=1728 Len=0
61	7.136683	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=RSVP 0xe1 0x86, off=57344)
62	7.136823	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=16560)
63	7.136975	192.168.0.7	169.254.0.2	TCP	35287 > http [ACK] Seq=2950162296 Ack=636561132 Win=1978 Len=0
64	7.137146	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Unknown 0xc6, off=16736)
65	7.137414	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=VISA 0x46, off=40960)

▼ Internet Protocol, Src Addr: 169.254.0.2 (169.254.0.2), Dst Addr: 192.168.0.7 (192.168.0.7)	
Version: 4	
Header length: 20 bytes	
Differenciated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)	
Total Length: 200	
Identification: 0xecac (60588)	
Flags: 0x04 (Don't Fragment)	
Fragment offset: 16736	
Time to live: 130	
Protocol: unknown (0xe6)	
Header checksum: 0x97c7 (correct)	
Source: 169.254.0.2 (169.254.0.2)	
Destination: 192.168.0.7 (192.168.0.7)	
Data (180 bytes)	

Rys. 15. Widok ekranu programu analizatora pakietów sieciowych Ethereal podczas podpisanej transmisji HTTP

Przebieg eksperymentu był obserwowany z komputera pośredniczącego w transmisji przy pomocy programu analizatora pakietów sieciowych Ethereal. Jak przedstawiono na rysunku 15 (dla transmisji HTTP), pola „Licznik czasu”, „Protokół”, „Przesunięcie fragmentu” oraz „Fragmentowanie zabronione” mają różne wartości (nawet we fragmentach tej samej ramki), gdyż w rzeczywistości są nośnikiem ukrytego podpisu.

Przebieg eksperymentu dla transmisji z wykorzystaniem protokołu FTP był obserwowany z komputera pośredniczącego w transmisji przy pomocy programu analizatora pakietów sieciowych Ethereal. Jak przedstawiono na rysunku 16, pola „Licznik czasu”, „Protokół”, „Przesunięcie fragmentu” oraz „Fragmentowanie zabronione” mają różne wartości (nawet we fragmentach tej samej ramki), gdyż w rzeczywistości są nośnikami ukrytego podpisu.

Przebieg eksperymentu przeprowadzonego przy użyciu polecenia PING był obserwowany z komputera pośredniczącego w transmisji przy pomocy programu analizatora pakietów sieciowych Ethereal.

Jak przedstawiono na rysunku 17, pola „Licznik czasu”, „Protokół”, „Przesunięcie fragmentu” oraz „Fragmentowanie zabronione” mają różne wartości (nawet we fragmentach tej samej ramki), gdyż w rzeczywistości są nośnikami zaszyfrowanego podpisu. Uzyskane wyniki świadczą o poprawnej weryfikacji podpisu w każdej z odebranych ramek.

No.	Time	Source	Destination	Protocol	Info
44	5.825127	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Control Message 0x26, off=)
45	5.825493	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Unknown 0xc6, off=24576)
46	5.825544	192.168.0.7	169.254.0.2	TCP	32774 > ftp-data [ACK] Seq=1515465473 Ack=3525363362 w'
47	5.825678	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=176)
48	5.826047	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Unknown 0xa6, off=33120)
49	5.826232	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Unknown 0xa6, off=0)
50	5.826418	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=32944)
51	5.826470	192.168.0.7	169.254.0.2	TCP	32774 > ftp-data [ACK] Seq=1515465473 Ack=3525363862 w'
52	5.826608	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Control Message 0x26, off=)
53	5.826886	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Unknown 0xa6, off=32768)
54	5.827095	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=PNNI over IP 0x66, off=5)
55	5.827117	192.168.0.7	169.254.0.2	TCP	32774 > ftp-data [ACK] Seq=1515465473 Ack=3525363362 w'
56	5.827233	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Control Message 0x26, off=)
57	5.827424	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Unknown 0xa6, off=0)
58	5.827707	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Unknown 0xa6, off=24752)
59	5.827730	192.168.0.7	169.254.0.2	TCP	32774 > ftp-data [ACK] Seq=1515465473 Ack=3525364810 w'
60	5.827871	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Unknown 0xa6, off=16736)
61	5.828211	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=VISA 0x46, off=24576)
62	5.828342	192.168.0.7	169.254.0.2	TCP	32774 > ftp-data [ACK] Seq=1515465473 Ack=3525365310 w'
63	5.828441	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=57520)
64	5.828586	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=TCP 0x06, off=24928)
65	5.828852	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Unknown 0xc6, off=32768)

▾ Internet Protocol, Src Addr: 169.254.0.2 (169.254.0.2), Dst Addr: 192.168.0.7 (192.168.0.7)
 Version: 4
 Header Length: 20 bytes
 ▸ Differentiated Services Field: 0x08 (DSCP 0x02: Unknown DSCP; ECN: 0x00)
 Total Length: 196
 Identification: 0xec09 (60425)
 ▸ Flags: 0x06 (Don't Fragment) (More Fragments)
 Fragment offset: 24752
 Time to live: 219
 Protocol: Unknown (0xa6)
 Header checksum: 0x1bbc (correct)
 Source: 169.254.0.2 (169.254.0.2)
 Destination: 192.168.0.7 (192.168.0.7)
 Data (176 bytes)

Rys. 16. Widok ekranu programu analizatora pakietów sieciowych Ethereal podczas podpisanej transmisji FTP

No.	Time	Source	Destination	Protocol	Info
8	0.506367	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Ether in IP 0x61, off=41312)
9	0.506552	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=ICMP 0x01, off=57872)
10	0.506739	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Ether in IP 0x61, off=704)
11	0.506925	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=ICMP 0x01, off=880)
12	0.507109	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=ICMP 0x01, off=1056)
13	0.507294	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=ICMP 0x01, off=1232)
14	1.504606	192.168.0.7	169.254.0.2	ICMP	Echo (ping) request
15	1.506347	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Unknown 0xa1, off=0)
16	1.506718	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Unknown 0xa1, off=24752)
17	1.506903	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=IPLT 0x81, off=41312)
18	1.507089	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Ether in IP 0x61, off=25104)
19	1.507274	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Unknown 0xa1, off=17088)
20	1.507460	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=ICMP 0x01, off=880)
21	1.507645	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=ICMP 0x01, off=1056)
22	1.507829	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=ICMP 0x01, off=1232)
23	2.505307	192.168.0.7	169.254.0.2	ICMP	Echo (ping) request
24	2.507299	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Ether in IP 0x61, off=57344)
25	2.507360	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Unknown 0xa1, off=8384)
26	2.507637	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Unknown 0xc1, off=352)
27	2.507823	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=IPLT 0x81, off=33296)
28	2.508010	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=Ether in IP 0x61, off=58048)
29	2.508195	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=ICMP 0x01, off=880)
30	2.508380	169.254.0.2	192.168.0.7	IP	Fragmented IP protocol (proto=ICMP 0x01, off=1056)

▾ Internet Protocol, Src Addr: 169.254.0.2 (169.254.0.2), Dst Addr: 192.168.0.7 (192.168.0.7)
 Version: 4
 Header Length: 20 bytes
 ▸ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
 Total Length: 196
 Identification: 0x785c (30812)
 ▸ Flags: 0x06 (Don't Fragment) (More Fragments)
 Fragment offset: 8368
 Time to live: 225
 Protocol: Unknown (0xa1)
 Header checksum: 0x9176 (correct)
 Source: 169.254.0.2 (169.254.0.2)
 Destination: 192.168.0.7 (192.168.0.7)
 Data (176 bytes)

Rys. 17. Widok ekranu programu analizatora pakietów sieciowych Ethereal podczas podpisanej transmisji żądania ECHO protokołu ICMP

WNIOSKI

W ramach pracy przeprowadzono badania w zakresie możliwości umieszczenia niedostrzegalnych podpisów w nagłówkach protokołu IP. Pokazano, że możliwość przesłania niewidocznych danych zachodzi zawsze tylko przy spełnieniu dodatkowych warunków, jednak w rzeczywistych transmisjach w większości przypadków mogą one zostać zapewnione przy odpowiedniej współpracy nadawcy i odbiorcy. Wyjątkiem jest wystąpienie na drodze transmisji zaawansowanych zapór sieciowych, których obecność może znacznie ograniczyć ilość dodatkowej informacji możliwej do przesłania.

Konieczność zmiany interpretacji przez odbiorcę pól nagłówka o znaczeniu ściśle określonym przez specyfikację może prowadzić do nieprawidłowego przebiegu transmisji i w efekcie do przerwania połączenia. Dlatego ostateczna decyzja o liczbie bitów informacji przyjętych jako nadmiarowe powinna być podjęta po serii testów przeprowadzonych w konkretnej rzeczywistej sieci i przy uwzględnieniu jej specyfiki, tj. parametrów takich jak przepustowość, niezawodność czy maksymalna jednostka transmisji (MTU). Ponadto odbiorca, po ekstrakcji bitów podpisu oraz jego weryfikacji, musi zawsze przywrócić ramkę do takiego stanu, by była czytelna dla modułów odbiorczych protokołów wyższych warstw.

Jak wykazują badania, ilość bitów dostępnych w celu przesłania ukrytej informacji pozwala na przechowanie silnego podpisu (tj. o długości przynajmniej 80 bitów przy użyciu klucza symetrycznego) dopiero przy zastosowaniu trzykrotnej fragmentacji na poziomie warstwy sieciowej lub przy jednoczesnym wykorzystaniu wielu pól nagłówków protokołów IP. Obydwa zabiegi mogą jednak obudzić czujność podsłuchującego, ponieważ wielokrotna fragmentacja na fragmenty o niestandardowej długości jest zjawiskiem rzadkim, a zwiększenie ilości wykorzystanych pól nagłówków zwiększa prawdopodobieństwo zauważenia obecności w nich nietypowych wartości. Niemożliwość umieszczenia silnego podpisu nie jest jednak dużym problemem, bowiem siła niedostrzegalnego podpisu tkwi nie w trudności jego przetwarzania, a w jego niedostrzegalności. Dlatego najlepszym rozwiązaniem jest zadbanie o jak najmniejszą wykrywalność podpisu kosztem jego odporności na złamanie. Umiejętne wykorzystanie pól „Identyfikator” i „Czas życia” nagłówka protokołu IP. Rezultaty uzyskane w ramach pracy pozwalają na przeprowadzenie dodatkowych badań nad zastosowaniem ukrytych podpisów w kanałach komunikacyjnych do sygnowania wysokopoziomowych transakcji np. przeprowadzanych w bazach danych. Jak pokazano w pracy [LIBER 04] niedostrzegalne informacje zapisane w nagłówkach protokołu IP mogą być wykorzystane do wytworzenia sieci o sygnowaniu strukturalnym ruchu pakietów. W sieci takiej możliwe jest np. wykonanie bezpiecznego serwera baz danych odpornego na ataki z sieci znajdujących się poza siecią podpisaną strukturalnie. Prowadzone są dalsze badania nad konstrukcją dynamicznych niedostrzegalnych sygnatur protokołów sieciowych, w tym np. protokołu TCP.

LITERATURA

- [ABAD 01] Abad C.: *IP Checksum Covert Channels and Selected Hash Collision*, 2001.
<http://downloads.securityfocus.com/library/ipccc.pdf> (02.02.2006).
- [ASHAN 02] Ahsan K.: *Covert channel analysis and data hiding in TCP/IP*. Master's thesis. University of Toronto, 2002. <http://ee.tamu.edu/~deepa/theses/ahsan02.pdf> (02.02.2006).
- [FISK 02] Fisk G., Fisk M., Papadopoulos C., Neil J.: *Eliminating Steganography in Internet Traffic with Active Wardens*. Proceedings of Information Hiding – 5th International Workshop, IH 2002. Vol. 2578 of Lecture Notes in Computer Science, Springer-Verlag Inc., str. 29-46. Noordwijkerhout, The Netherlands, October 7-9, 2002. <http://woozle.org/~mfisk/papers/ih02.pdf> (02.02.2006).
- [GIFFIN 02] Giffin J., Greenstadt R., Litwack P., Tibbetts R.: *Covert Messaging through TCP Timestamps*. Massachusetts Institute of Technology, 2002.
<http://web.mit.edu/greenie/Public/CovertMessaginginTCP.ps> (02.02.2006).
- [HANDEL 96] Handel T. G., Sandford M. T.: *Hiding data in the OSI network model*. Information Hiding, Lecture Notes in Computer Science. Springer-Verlag, 1996.
- [IANA 06] <http://www.iana.org/assignments/protocol-numbers> (02.02.2006)
- [LIBER 04] Liber A., Kosmulska – Bochenek E.: *Niewidoczne sygnatury symetryczne protokołów komunikacyjnych*. Współczesne problemy sieci komputerowych, zastosowanie i bezpieczeństwo, str. 369-378. Wydawnictwa Naukowo – Techniczne. Warszawa, 2004.
- [MOGUL 90] Mogul J., Deering, S.: *Path MTU discovery*. IETF, 1990.
- [MURDOCH 05] Murdoch S. J., Lewis S.: *Embedding Covert Channels into TCP/IP*, 2005.
<http://www.cl.cam.ac.uk/users/sjm217/papers/ih05coverttcp.pdf> (02.02.2006).
- [POSTEL 81] Postel J.: *Internet Protocol*. IETF, 1981.
- [ROWLAND 97] Rowland C. H.: *Covert channels in the TCP/IP protocol suite*. Tech. Rep. 5, First Monday, Peer Reviewed Journal on the Internet, July 1997.
http://www.firstmonday.dk/issues/issue2_5/rowland/ (02.02.2006).
- [SZCZYP 03] Szczypiorski K.: *HICCUPS: Hidden communication system for corrupted networks*. International Multi-Conference on Advanced Computer Systems, 2003.
<http://krzysiek.tele.pw.edu.pl/pdf/acs2003-hiccups.pdf> (02.02.2006).

THE CONSTRUCTION OF IMPERCEPTIBLE CHANNELS OF COMMUNICATION WITH THE UTILIZATION OF THE IP PROTOCOL AND THEIR USE IN THE DATA SIGNING

In frames of the work one represented results of research of authors in the range of the utilization of the IP protocol to the transmission of the additional information. The additional information can be put-upon to placing of digital signatures or to producing of the additional, independent broadcasting channel. Results of research introduced on the job are a continuation and an enlargement of research described on the job [LIBER 04] and in works [ROWLAND 97], [ASHAN 02]. Received results on the job can be put-upon to the construction of systems of the exchange given with hidden inquiry channels. Particularly while in modern systems of databases.

Łukasz PŁAWIAK*

Artur WILCZEK**

DANE PRZESTRZENNE WEDŁUG SPECYFIKACJI OpenGIS

Systemy określane mianem Geograficznych Systemów Informacyjnych (ang. *Geographic Information Systems*) pozwalają przechowywać, przetwarzać, analizować, modelować oraz prezentować dane geograficzne. Dane geograficzne zawierają dwa typy informacji: informację przestrzenną (lokalizującą obiekty geograficzne na powierzchni ziemi) i towarzyszące jej dane opisowe. Systemy GIS wykorzystują systemy zarządzania bazami danych (SZBD) do przechowywania danych przestrzennych wraz z ich opisami. Przechowywanie danych przestrzennych wymaga dużych zasobów systemowych w związku z tym istotną cechą systemów GIS jest zapewnienie możliwości współdzielenia danych. Współdzielenie danych geograficznych wymaga zapewnienia możliwości współdziałania oprogramowania różnych dostawców. Oprogramowanie GIS, aby spełnić te wymagania, powinno być projektowane i budowane w oparciu o powszechnie zaakceptowane standardy. W ostatnich latach powstały powszechnie zaakceptowane międzynarodowe normy (ISO) i specyfikacje (OpenGIS) dotyczące systemów GIS. Normy ISO (grupa ISO 19100) opracowano zachowując wysoki stopień ogólności. Ich praktyczne zastosowanie wymaga wielu prac adaptacyjnych. Niezależnie własne prace standaryzacyjne prowadzi konsorcjum OGC (ang. *Open Geospatial Consortium*), które opracowało własne specyfikacje o nazwie OpenGis. Wynikiem porozumienia obu organizacji (ISO i OGC) jest stan aktualny, w którym specyfikacje OpenGis pełnią rolę specyfikacji implementacyjnych uwzględniających specyficzne wymagania konkretnego środowiska narzędzi programistycznych.

1. WPROWADZENIE

Geograficzne Systemy Informacyjne to systemy komputerowe, na które składają się ludzie, oprogramowanie, sprzęt oraz dane [HARMON 03]. Systemy GIS mają bardzo szerokie zastosowanie w wielu instytucjach prywatnych oraz publicznych. Dane przestrzenne rozumiane są jako liczby i fakty odzwierciedlające obiekty przestrzenne w dowolny układ współrzędnych [WIKI]. Istnieją dwa znane i powszechnie akceptowane modele danych przestrzennych: rastrowy i wektorowy. Model rastrowy

* Instytut Informatyki Stosowanej, Wydział Informatyki i Zarządzania Politechniki Wrocławskiej, 50-370 Wrocław, Wybrzeże Wyspiańskiego 27, student, lukasz_plawiak@wp.pl

** Instytut Informatyki Stosowanej, Wydział Informatyki i Zarządzania Politechniki Wrocławskiej, 50-370 Wrocław, Wybrzeże Wyspiańskiego 27, artur.wilczek@pwr.wroc.p

prezentuje dane przestrzenne w postaci rastra, czyli siatki punktów lub innych wielokątów. Wartość każdego elementu, z jakich składa się siatka, reprezentuje jakąś konkretną daną związaną z rzeczywistym miejscem świata realnego. Model wektorowy prezentuje dane jako ciągi liczb. Ciągi te, związane z geometrią, symbolizują punkty, linie oraz wielokąty. Geometryczność ułatwia przetwarzanie danych oraz przedstawianie ich bardzo istotnej cechy, jaką jest topologiczność (relacje sąsiedztwa i/lub zależności między obiektami przestrzennymi).

Nie istnieje uniwersalne kryterium pozwalające ocenić, który z modeli, rastrowy czy wektorowy, jest lepszy. Użycie każdego z nich wiąże się z odpowiednimi korzyściami oraz niedogodnościami dla systemu. W praktyce systemy GIS stosują odpowiednie połączenie obu tych modeli [HARMON 03].

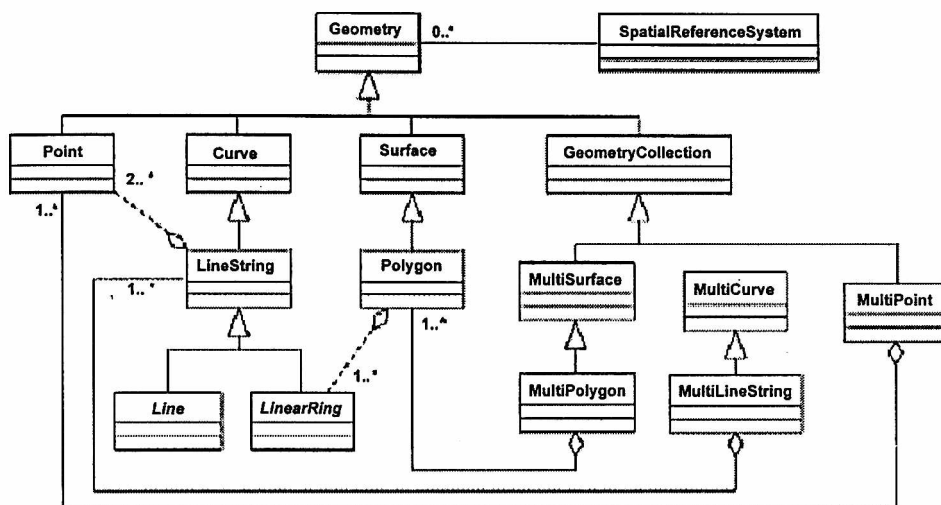
Istnieje wiele metod definiowania i przechowywania danych wektorowych w bazie danych. Jedną z nich zaproponowało konsorcjum OGC (ang. *Open Geospatial Consortium*) w specyfikacjach OpenGIS. OpenGIS to zbiór specyfikacji określających kompleksowe rozwiązania w zakresie tworzenia oraz wymiany informacji geograficznej pomiędzy różnymi aplikacjami. Standard danych przestrzennych według OpenGIS nie stawia wymagań czy zaleceń odnośnie sposobów implementacji przechowywania danych przestrzennych w bazach danych. Określa jedynie co system zarządzania bazą danych (SZBD) powinien zawierać i jakie wymogi spełniać [SFAP2 05].

W dalszej części przedstawimy propozycję metody przechowywania danych przestrzennych w relacyjnej bazie danych zawartą w specyfikacji OpenGIS Implementation Specification for Geographic Information – Simple Feature Access (SFA). Dokument ten zawiera propozycję wspólnego modelu pojęciowego danych geograficznych (Part 1: Common architecture) oraz proponuje metody przechowywania danych przestrzennych w bazach SQL. Dokument zawiera dwie propozycje. Pierwsza zakłada wykorzystanie języka SQL (w wersji SQL:92), druga wprowadza definicje typów geometrycznych jako abstrakcyjnych typów danych (SQL:92 + ADT).

2. WSPÓLNY MODEL POJĘCIOWY

Głównym celem wprowadzenia wspólnego modelu pojęciowego jest wyspecyfikowanie pewnej grupy powiązanych obiektów geometrycznych, wraz z możliwymi operacjami na nich, które będą wykorzystywane w pozostałych dokumentach standardu. Zdefiniowana została hierarchia klas widoczna na rysunku 1 [SFAP1 05].

Główną klasą jest abstrakcyjna, bazowa klasa *Geometry* (Geometria). Została ona zdefiniowana w celu ujednolichenia i utworzenia jednego przodka dla wszystkich pozostałych klas geometrycznych. Wszystkie klasy konkretne (klasa konkretna w odróżnieniu od klas abstrakcyjnej, może mieć bezpośrednie wystąpienia obiektów) dziedziczą własności *Geometry*, są zawarte w 0, 1 lub 2 wymiarach i są topologicznie zamknięte (ich obiekty uwzględniają swoje granice).



Rys. 1. Hierarchia klas geometrycznych [SFAP1 05]

Klasami, które bezpośrednio dziedziczą z klasy abstrakcyjnej *Geometry* są:

- Klasa *Point* (Punkt) – jest 0-wymiarową klasą konkretną określającą, pojedynczą lokalizację w przestrzeni. Punkt posiada współrzędne X, Y .
- Klasa *Curve* (Krzywa) – jest 1-wymiarową klasą abstrakcyjną zawierającą sekwencję punktów, które określają w przybliżeniu pewien kształt krzywej (interpolacja). Sposób przybliżenia nie jest zdefiniowany w klasie *Curve*, lecz w instancyjnej klasie dziedziczącej po niej – *LineString* (Linia Wielopunktowa). Klasa *LineString* definiuje wyłącznie interpolacje linową.
- Klasa *Surface* (Powierzchnia) – jest 2-wymiarową klasą abstrakcyjną, po której dziedziczy tylko klasa *Polygon* (Wielokąt). Klasa *Polygon* posiada tylko jedną granicę zewnętrzną i żadnej wewnętrznej.
- Klasa *GeometryCollection* (Kolekcja Geometryczna) – jest to klasa konkretna określająca kolekcję jednego lub więcej obiektów geometrycznych. Wszystkie obiekty kolekcji muszą być oparte na tym samym Przestrzennym Układzie Odniesienia (ang. *Spatial Reference System*).

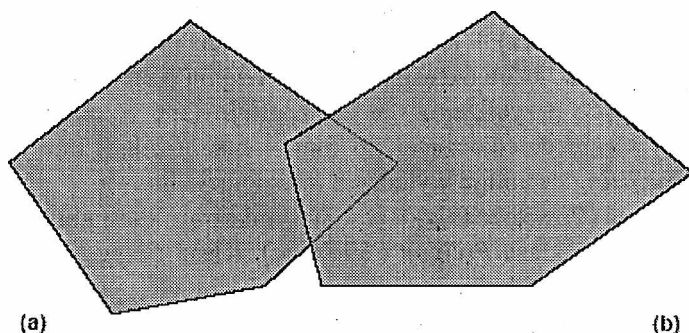
Klasa *LineString* jest klasą bazową dla dwóch innych klas:

- Klasa *Line* (Linia) – to *LineString* z dokładnie dwoma punktami.
- Klasa *LinearRing* (Pierścień) – to prosta (ang. *Simple*) i zamknięta (ang. *Closed*) Linia Wielopunktowa czyli *LineString*. Prosty *LineString* to taki, który po interpolowaniu nie przecina sam siebie. Natomiast zamknięty to taki, którego punkt początkowy i końcowy jest dokładnie ten sam.

Klasa *GeometryCollection* jest to klasa bazowa dla wielu innych kolekcji dziedziczących bezpośrednio po niej:

- Klasa *MultiSurface* (Kolekcja Przestrzeni) – jest 2-wymiarową klasą abstrakcyjną służącą do tworzenia kolekcji powierzchni (*Surface*). Istnieje jedna klasa dziedzicząca po *MultiSurface* i jest nią *MultiPolygon* (Kolekcja Wielokątów), której elementami mogą być takie przestrzenie, które są wielokątami.
- Klasa *MultiCurve* (Kolekcja Krzywych) – jest 1-wymiarową klasą abstrakcyjną służącą do tworzenia kolekcji krzywych (*Curve*). Jest to inaczej mówiąc *GeometryCollection* posiadająca dodatkowe ograniczenie mówiące, że elementami kolekcji są obiekty 1-wymiarowe. Istnieje jedna klasa dziedzicząca po *MultiCurve* i jest to *MultiLineString*, czyli klasa służąca do tworzenia kolekcji tylko linii typu *LineString*.
- Klasa *MultiPoint* (Kolekcja Punktów) – jest 0-wymiarową klasą konkretną służącą do tworzenia kolekcji Punktów (*Point*). Jest to, inaczej mówiąc, klasa *GeometryCollection* posiadająca dodatkowe ograniczenie mówiące, że elementami kolekcji są jedynie obiekty 0-wymiarowe.

Ostatnią istotną klasą hierarchii jest klasa Przestrzenny Układ Odniesienia (ang. *Spatial Reference System*). Jest to jedyna klasa, która nie dziedziczy po klasie *Geometry* a posiada z nią jedynie asocjacje. Jej głównym zadaniem jest określenie układu odniesienia (współrzędnych) z jakimi będą związane wszystkie obiekty, które są instancją klas dziedziczących po *Geometry*.



	Interior	Boundary	Exterior
Interior	2	1	2
Boundary	1	0	1
Exterior	2	1	2

Rys. 2. Przykład wymiarów części wspólnych dwóch Wielokątów [SFAP1 05]

Wspólny geometryczny model pojęciowy definiuje także szereg możliwych operacji, które można wykonać na obiektach opisanych wyżej klas. Operacje te reprezentuje się poprzez operatory relacyjne, których wynikiem jest zawsze wartość logiczna (ang. *Boolean*). Operacje te polegają na sprawdzaniu topologicznych zależności między obiektami. Podstawowe operacje to: *Disjoint* (są rozłączne), *Touches* (dotykają się), *Crosses* (przecinają się), *Within* (jest wewnątrz), *Overlaps* (pokrywa się), *Contains* (zawiera się), *Intersects* (przecina się). Każda operacja ma określone typy argumentów. Na przykład operator *Overlaps* (pokrywa się) może być stosowany wyłącznie dla obiektów o tym samym wymiarze.

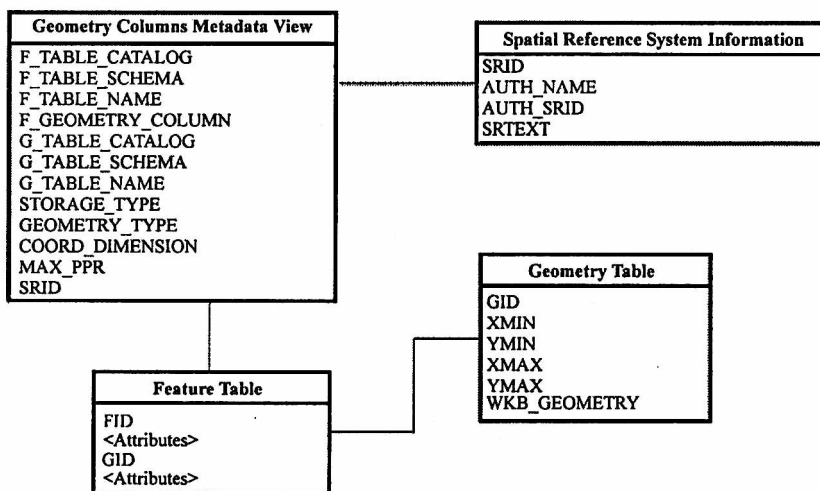
Sprawdzanie topologicznych zależności obiektów polega na określeniu części wspólnej podstawowych obszarów każdego z obiektów. Podstawowe obszary obiektów to *Interior* (wnętrze), *Boundary* (granica) i *Exterior* (część zewnętrzna). Na przykład częścią wspólną granicą jednego wielokąta i wnętrza drugiego może być linia, punkt lub może jej nie być wcale. Każda część wspólna obszarów obiektów definiuje nowy obiekt o nowych wymiarach. Na rysunku 2. pokazany jest przykład wymiarów, jakie tworzą części wspólne wnętrza, granic i części zewnętrznych dwóch wielokątów.

3. IMPLEMENTACJA W SQL:92

Implementacja standardu OpenGIS w języku SQL:92 polega na zdefiniowaniu schematu relacyjnej bazy danych odpowiedzialnego za przechowywanie danych przestrzennych (relacji i ich atrybutów). Implementacja w SQL:92 nie definiuje funkcji SQL służących do zarządzania i indeksowania danych. Ponadto nie korzysta z typów geometrycznych, lecz wyłącznie z typów wbudowanych SQL bądź typów binarnych. Na rysunku 3. zaprezentowany jest schemat relacyjnej bazy danych zgodny z proponowaną w specyfikacji implementacją w SQL:92.

Schemat proponowany w specyfikacji OpenGIS zawiera następujące tabele [SFAP2 05]:

- Tabela *Spatial Reference System Information* – zawiera informacje o wszystkich przestrzennych systemach odniesienia.
- Tabela *Geometry Table* – zawiera dane geometryczne.
- Tabela *Feature Table* – zawiera dane opisowe (*Attribute*), odwołania do tablic geometrycznych (*GID*) oraz odwołania do własnej tablicy w celu zdefiniowania zależności, topologii.
- Perspektywa *Geometry Columns Metadata View* – widok udostępniający informacje o danych geometrycznych w bazie danych (złączenie tabel *Feature Table* i *Geometry Table*). Dostępna jest informacja o tym, z jakim przestrzennym systemem odniesienia związane jest połączenie *Feature Table* z *Geometry Table*.



Rys. 3. Schemat relacyjnej bazy danych dla danych przestrzennych [SFAP2 05]

Implementacja w SQL:92 opisuje jakiego typu pola będą w zdefiniowanych tabelach oraz jakie będą dopuszczalne dla nich wartości. Zdefiniowany jest zakres obiektów geometrycznych dostępnych dla SQL:92. Dostępne są następujące typy obiektów geograficznych:

- 0 = GEOMETRY,
- 1 = POINT,
- 2 = CURVE,
- 3 = LINESTRING,
- 4 = SURFACE,
- 5 = POLYGON,
- 6 = COLLECTION,
- 7 = MULTIPOINT,
- 8 = MULTICURVE,
- 9 = MULTILINESTRING,
- 10 = MULTISURFACE,
- 11 = MULTIPOLYGON.

W tabeli *Feature Table* możemy określić dwa rodzaje przechowywanych danych przestrzennych. Pierwszym typem jest typ binarny (obiekt przechowywany jest binarnie), a drugim typem jest typ znormalizowany (oparty na typach wbudowanych SQL:92).

4. IMPLEMENTACJA W SQL:92 + ADT

Drugą metodą implementacji proponowaną w standardzie OpenGIS jest wykorzystanie języka SQL:92 i abstrakcyjnych typów danych (ADT). W tym rozwiązaniu nie wprowadza się tablicy *Geometry Table*, czyli tablicy zawierającej dane przestrzenne. Zamiast tego dostarczone są geometryczne typy danych. W tablicach *Feature Table* brak jest odwołania do *Geometry Table* (*GID*), ale występuje w zamian kolumna o typie

geometrycznym. Typ geometryczny jest zgodny z klasami zdefiniowanymi we wspólnym modelu pojęciowym [SFAP1 05].

W porównaniu do implementacji w SQL:92 inna jest także struktura tablicy *Geometry Columns Metadata View*, usunięto kolumny *G_TABLE_CATALOG*, *G_TABLE_NAME* i *G_TABLE_SCHEMA*.

Implementacja w SQL:92 z ADT opisuje także sposób, w jaki należy formułować instrukcje języka SQL w celu przetwarzania danych przestrzennych opartych na typach geometrycznych. Przykłady takich zapytań przedstawimy w dalszej części.

Przykładowe instrukcje SQL:92 tworzące obiekty geometryczne przy zastosowaniu proponowanych w specyfikacji OpenGIS typów geometrycznych mogą mieć postać [SFAP2 05]:

```
// Point (Punkt):
POINT (10 10)
// LineString (Linia Wielopunktowa):
LINESTRING (10 10, 20 20, 30 40)
// Polygon(Wielokąt):
POLYGON ((10 10, 10 20, 20 20, 20 15, 10 10))
// MultiPoint(Kolekcja Punktów):
MULTIPOINT (10 10, 20 20)
// MultiLineString(Kolekcja Linii Wielopunktowych):
MULTILINESTRING ((10 10, 20 20),(15 15, 30 15))
// MultiPolygon(Kolekcja Wielokątów):
MULTIPOLYGON (((10 10, 10 20, 20 20, 20 15, 10 10)), ((60
60, 70 70, 80 60, 60 60 ) ))
```

Specyfikacja proponuje także zestaw funkcji operujących na typach geometrycznych w instrukcjach języka SQL [SFAP2 05]:

- Zbudowanie obiektu na podstawie tekstu:
GeomFromText(geometryTaggedText String, SRID Integer): Geometry
- Określenie wymiaru obiektu:
Dimension(g Geometry) : Integer
- Określenie długości krzywej:
Length(c Curve) : Double Precision
- Pobranie obszaru powierzchni:
Area(s Surface) : Double Precision
- Określenie, czy dwa obiekty dotykają się:
Touches(g1 Geometry, g2 Geometry) : Integer
- Sprawdzenie, czy obiekt g1 zawiera się w obiekcie g2:
Contains(g1 Geometry, g2 Geometry) : Integer

Poniżej zamieszczamy przykładowe instrukcje języka SQL:92 zgodne ze specyfikacją [SFAP2 05]:

```
// Umieszczenie w tabeli 'Countries'
// nazwy i lokalizacji państwa:
INSERT INTO Countries (Name, Location)
```

```
VALUES ('Kenya', PolygonFromText('POLYGON ((x y, x y, x y,
..., x y))', 14));
// Pobranie z tabeli 'Countries' nazwy i lokalizacji
// w formie tekstowej wszystkich państw na literę 'K':
SELECT Name, AsText(Location)
FROM Countries WHERE Name LIKE 'K';
// Pobranie nazwy i id wszystkich parceli z tabeli
// 'Parcels', spełniających warunek przecięcia:
SELECT Parcel.Name, Parcel.Id
FROM Parcels WHERE Intersects(Parcels.Location, Polygon-
FromWKB(:wkb, : srid))=1;
```

5. PODSUMOWANIE

Standard przechowywania danych przestrzennych w relacyjnych bazach danych według specyfikacji OpenGIS odnosi się przede wszystkim do wektorowego modelu danych. W standardzie tym istnieje model pojęciowy, w którym zdefiniowano typy danych geometrycznych i operacje sprawdzania ich zależności.

Specyfikacja OpenGIS proponuje także dwie metody implementacji standardu w bazach danych. Pierwsza zakłada wykorzystanie wyłącznie podstawowych konstrukcji języka SQL. Druga wprowadza definicje geometrycznych typów danych, które można implementować w bazach danych wspierających abstrakcyjne typy danych języka SQL.

LITERATURA

- [SFAP1 05] *OpenGIS® Implementation Specification for Geographic information – Simple feature access - Part 1: Common architecture*, Open Geospatial Consortium Inc., 2005.
- [SFAP2 05] *OpenGIS® Implementation Specification for Geographic information – Simple feature access - Part 2: SQL option*, Open Geospatial Consortium Inc., 2005.
- [HARMON 03] John E. Harmon, Steven J. Anderson, *Design and Implementation of GIS*, John Wiley & Sons, Inc., 2003.
- [WIKI] http://pl.wikipedia.org/wiki/Dane_przestrzenne.

GEOSPATIAL DATA BY OpenGIS

OpenGIS specifications propose methods for storing geospatial data in relational database systems. This paper provides a brief description of those methods.

*bazy danych,
dostęp do baz danych,
programowanie w C#, ADO.NET*

Zbigniew FRYŻLEWICZ*
Marcin APOSTOLUK

GENERYCZNA WARSTWA POŚREDNIA DOSTĘPU DO BAZY DANYCH

Artykuł przedstawia opis generycznej warstwy pośredniej dostępu do baz danych ułatwiającej programowanie logiki biznesowej, zapobiegającej dublowaniu kodu, porządkującej aplikację. Na prostym przykładzie dokładnie przedstawia architekturę oraz implementację takiego mechanizmu.

1. WPROWADZENIE

Technologia ADO.NET zapewnia bardzo szybki i łatwy dostęp do baz danych. W prosty sposób można zadać zapytanie do bazy i przetworzyć otrzymany wynik. Stosując ADO.NET kod jest na tyle mało skomplikowany, że bardzo często wpłata się go pomiędzy wiersze odpowiedzialne za logikę biznesową aplikacji. Takie podejście sprawdza się jednak tylko w przypadku niewielkich aplikacji o małej złożoności. W większych projektach problemem staje się dublowanie kodu odpowiedzialnego za budowanie zapytań i korzystanie z bazy danych. Również korzystanie z bazy danych przeplatane z kodem logiki biznesowej zaczyna przeszkadzać podczas debugowania całej implementacji, gdy priorytetem jest sprawdzenie samej logiki biznesowej.

Potrzebne jest rozwiązanie, które sprawdzi się w przypadku każdego projektu, pomoże oddzielić logikę biznesową od bezpośredniego dostępu do bazy danych, nie będzie zmuszać programistę do ciągłego budowania zapytań za każdym razem, gdy potrzebny będzie wynik tylko nieco innego od poprzedniego, zapewni obiektowość i wprowadzi porządek w aplikacji. Takim rozwiązaniem jest właśnie generyczna warstwa pośrednia.

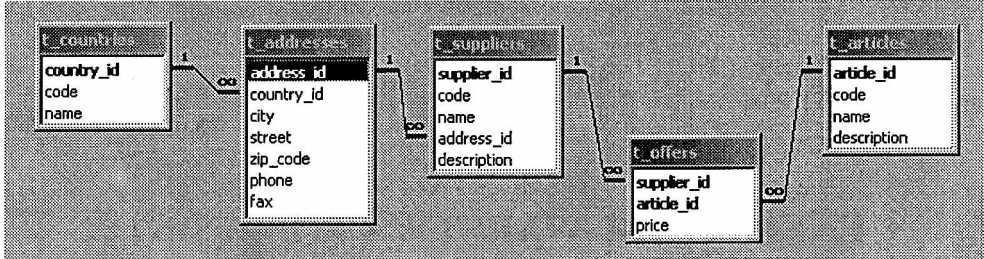
Przy stosowaniu warstwy pośredniej należy wyróżnić w aplikacji cztery poziomy:

- bezpośredni dostęp do bazy danych,
- warstwa pośrednia,
- logika biznesowa,
- interfejs użytkownika.

* Instytut Informatyki Stosowanej, Wydział Informatyki i Zarządzania Politechniki Wrocławskiej,
50-370 Wrocław, Skwer Idaszewskiego 1, zbigniew.fryzlewicz@pwr.wroc.pl,
marcin.apostoluk@student.pwr.wroc.pl

W dalszej części artykułu zbudujemy przykładową aplikację korzystającą z bazy danych i na jej przykładzie szczegółowo omówimy powyższą architekturę.

Rozważmy model bazy danych z rysunku 1.



Rys. 1. Przykładowy model bazy danych

Baza przechowuje informacje o dostawcach. Każdy z dostawców jest powiązany ze swoim adresem, każdy z adresów przydzielony jest do jednego państwa. W bazie znajdują się także informacje o artykułach. Każdy z dostawców może dostarczać wybrane artykuły po ustalonej cenie.

2. OBIEKTY

Stwórzmy klasy odpowiadające tabelom z bazy danych. Przykładowo:

```
public class Country
{
    public int countryId;
    public string code;
    public const int LENGTH_CODE = 5;

    public string name;
    public const int LENGTH_NAME = 50;
}
```

Statyczne pola przechowują dane o długości pól tekstowych w bazie danych.

Wszystkie przedstawiane definicje klas powinny oczywiście posiadać pola o dostępie `protected` i właściwości o dostępie publicznym. Zostało to pominięte w niniejszym artykule w celu uproszczenia przykładów.

Analogicznie budujemy klasę `Address`:

```
public class Address
{
    public int addressId;
    public int countryId;
    public Country country;
```

```

public string city;
public const int LENGTH_CITY = 50;

...
}

```

Klasa `Address` zawiera klucz obcy do tabeli `t_countries` - `countryId`. Oprócz niego posiada także obiekt klasy `Country`, gdyż każdy obiekt klasy `Address` powiązany jest z dokładnie jednym elementem klasy `Country`. Zawierania obiektów tych klas nie należy mylić z agregacją tych obiektów w rzeczywistości – tutaj związek ten reprezentuje asocjację. Tak więc fakt, że obiekt klasy `Address` zawiera w sobie obiekt klasy `Country` oznacza, że istnieje asocjacja pomiędzy tymi obiektami. Równie dobrze moglibyśmy napisać, że obiekt klasy `Country` zawiera obiekt klasy `Address`. Dla uproszczenia zamodelujemy tutaj związek jednostronny.

Załóżmy także, że asocjacja taka nie zawsze musi być uwzględniona w obiekcie tzn. obiekt państwa w klasie adresu „wskazywać będzie” na powiązany z adresem kraj tylko wtedy, gdy wymagać będziemy tego od warstwy pośredniczącej, czyli gdy zażądamy:

„Podaj wszystkie adresy, które, łącznie z państwami”.

Równie dobrze będziemy mogli żądać:

„Podaj wszystkie adresy, które”.

Zrozumienie asocjacji obiektów jest kluczem do zrozumienia całej warstwy pośredniczącej.

Pozostałe klasy (odpowiedniki schematów relacji z bazy danych) należy zbudować w analogiczny sposób.

3. FILTRY

W ten sposób otrzymujemy klasy, które przechowywać będą dowolnie połączone dane aplikacji. Potrzebujemy teraz generycznego rozwiązania, które pozwoli na pobranie dowolnie przefiltrowanych i połączonych danych z bazy i umieszczenie ich w obiektach. Warstwa pośrednicząca musi umożliwiać dowolny poziom zagnieżdżenia żądań, przykładowo musi umieć odpowiedzieć na następujące pytania:

„Pobierz wszystkie adresy, w których miasto to Warszawa”,

„Pobierz wszystkie adresy, w których miasto to Warszawa, a nazwa kraju to Polska”,

„Pobierz wszystkich dostawców, których kod to „DK-1”, a których miasto adresu to Warszawa, nazwa państwa to Polska”.

Warstwa pośrednicząca zapewnić musi także **poprawne wielokrotne wykorzystanie tej samej relacji**.

Założmy hipotetyczną tabelę:

```
t_meetings(meeting_id, #partner1_id, #partner2_id)
```

oraz tabelę:

```
t_partners(partner_id, name)
```

Spotkanie powiązane jest z dwoma partnerami poprzez dwa klucze obce. Dwóch partnerów może spotkać się ze sobą dowolną ilość spotkań. Musimy mieć możliwość zadania zapytania:

„Pobierz wszystkie spotkania, które odbyła osoba o imieniu Jan z osobą o imieniu Tomasz”.

Czasem będziemy chcieli, aby pytania zadawać w dowolną ze stron powiązania np. *„Pobierz wszystkich partnerów, z którymi odbył spotkanie partner o identyfikatorze 1”.*

Aby zapewnić tak generyczne rozwiązanie wprowadzamy **filtry**. Filtry to klasy pośredniczące, służące do budowy dowolnych zapytań *SELECT*.

Przeanalizujemy przykładowe proste zapytanie (do bazy SQL Server 2000):

```
SELECT * FROM t_addresses
```

Aby przekształcić to zapytanie w zapytanie w pełni generyczne należy dodać możliwość edycji poszczególnych jego części:

```
SELECT * FROM {0} t_addresses {1} WHERE ( 1 = 1 ) {2}
```

gdzie markery oznaczają kolejno:

{0} – uzupełnienie nawiasów otwierających niezbędnych przy zapytaniach używających klauzuli *INNER JOIN* (wymagane w przypadku niektórych baz danych)

– {1} – uzupełnienie aliasów tabel oraz połączeń z innymi tabelami (klauzula *INNER JOIN*)

{2} – warunek zawężający otrzymaną ilość rekordów (string “(1 = 1)” został dodany dla prostszego „doklejania kolejnych warunków poprzedzonych łącznikiem *AND*), klauzula *ORDER BY*.

Należy zauważyć, że wszystkie markery są podatne na *SQL Injection* i przed wstawieniem należy je odpowiednio sprawdzić.

Rozważmy przykładowe zapytanie:

```
SELECT * FROM ( t_addresses T0 INNER JOIN t_countries T1
ON T1.country_id = T0.country_id ) WHERE ( 1 = 1 )
AND T0.city LIKE 'Warszawa' AND T1.name LIKE 'Polska'
ORDER by T0.street, T1.country_id
```

Znając problem, zdefiniujmy budowę abstrakcyjnej klasy filtru:

```
public abstract class AbstractFilter
{
    public string aliasName;
    public bool attachObject = false;
    public string orderBy;

    public AbstractFilter()
    {
        this.aliasName = AliasCreator.getNextAlias( );
    }
}
```

```

public abstract string generateSubWhereSQL( );
public abstract string generateSubFromSQL( );

public string generateWhereSQL( )
{
    return this.convertString( this.generateSubWhereSQL( ) );
}

public string generateFromSQL( )
{
    return this.convertString( this.generateSubFromSQL( ) );
}

public string fillQueryString( string query )
{
    ....
}

public string getOrderBy( )
{
    ....
}

protected String appendSQL( string sql )
{
    return sql + " AND ";
}

protected string convertString( string sql )
{
    return sql.Replace("?", "%");
}
}

```

Pole `aliasName` przechowuje nazwę aliasu relacji, do której odnosi się filtr (pamiętajmy, że jeden filtr odnosi się bezpośrednio tylko do jednej relacji z bazy danych). Alias generowany jest automatycznie przy tworzeniu filtru. Służy do tego pomocnicza klasa `AliasCreator`, która generuje kolejnym filtrom aliasy: "T0", "T1", "T2", ...

Zapewnia to unikalność nazw i pozwala na użycie nazw kwalifikowanych. Jest to rozwiązanie problemu wielokrotnych złączeń (zobacz relacje `t_meetings` i `t_partners`).

Funkcja `convertString(string sql)` zamienia w zapytaniu znaki `*` na `%`. Funkcja jest bardzo przydatna, gdy w GUI aplikacji używamy wyrażeń regularnych (zawierających znak `*`, który jest bardziej intuicyjny dla użytkownika niż znak `%`).

Funkcje `generateFromSQL()` i `generateWhereSQL()` odpowiedzialne są za generowanie opisanych wyżej znaczników tekstu {1} oraz {2} w tekście zapytania. Funkcje wywołują polimorficzne abstrakcyjne funkcje z klas dziedziczących generujące fragmenty zapytań odpowiadające używanym filtrom.

Funkcja `getOrderBy()` wypełnia część zapytania związaną z klauzulą `ORDER BY` wyrażenia SQL. Funkcja zwraca wyrażenie ("`ORDER BY`" + `this.orderBy`) jeśli string `this.orderBy` został zdefiniowany. Mimo, że rozwiązanie takie wydaje się częściowo ograniczać użytkownika poprzez swą statyczność, pokazane zostanie w dalszej części artykułu prawidłowe użycie parametru `orderBy` w celu generowania dowolnych klauzul `ORDER BY` z danymi pochodzącymi z różnych tabel. Implementacja funkcji została pominięta w celu zwiększenia czytelności kodu.

Funkcja `fillQueryString(string query)` służy do wypełnienia odpowiednich części zapytań wynikami funkcji `generateFromSQL()`, `generateWhereSQL()`, `getOrderBy()` oraz brakującymi znakami nawiasów. Implementacja funkcji została pominięta w celu zwiększenia czytelności kodu.

Funkcja `appendSQL(string sql)` dodaje łącznik `AND` na końcu zapytania SQL. Służy zwiększeniu poziomu abstrakcji w dalszej części generowania zapytań.

Znacznik `attachObject` decyduje, czy dołączać do wyniku obiekt odpowiadający danemu filtrowi (np. przedstawione uprzednio dołączenie obiektu klasy `Country` do obiektu klasy `Address`).

Zbudujmy teraz wszystkie filtry na podstawie klasy `AbstractFilter`, np.:

```
public class CountryFilter: AbstractFilter
{
    public PInt id;
    public string code;
    public string name;

    public override string generateSubWhereSQL( )
    {
        string sql = "";

        if ( this.id != null )
        {
            sql = this.appendSQL( sql ) + this.aliasName +
                ".country_id = " + this.id.Value;
        }
        If ( this.code != null )
        {
            sql = this.appendSQL( sql ) + this.aliasName +
                ".code LIKE '" + StringConverter.convertToDb(
                    this.code, Country.LENGTH_CODE ) + "'";
        }

        If ( this.name != null )
        {
            sql = this.appendSQL( sql ) + this.aliasName +
                ".name LIKE '" + StringConverter.convertToDb(
                    this.name, Country.LENGTH_NAME ) + "'";
        }
        return sql;
    }
}
```

```

public override string generateSubFromSQL( )
{
    return "";
}

```

Klasa zawiera dokładnie takie same pola jak klasa obiektu przechowującego dane. Są one jednak nieco innego typu. Spowodowane jest to ważnym założeniem: **pole filtru jest aktywne wtedy i tylko wtedy, gdy jego wartość jest różna od NULL**. Pozwala to na filtrowanie tylko po wybranych przez nas polach. Dlatego też zmienna `countryId` jest typu `PInt` a nie `int` (który nie może przyjąć wartości `NULL`) [w dialekcie C# 2.0 problem ten już nie istnieje].

```

public class PInt
{
    public int Value;
    public PInt( int Value )
    {
        this.Value = Value;
    }
}

```

Najbardziej rozbudowaną częścią klasy `CountryFilter` jest funkcja `generateSubWhereSQL()`. Dla każdego z pól klasy (różnego od `NULL`) generuje ona ułamkową część zapytania SQL ograniczającego tabelę wynikową. Przy generacji wartości typu tekstowego użyte zostaje słowo kluczowe `LIKE` w celu umożliwienia obsługi wyrażeń regularnych zawierających '%'. Dodatkowo wykorzystana zostaje klasa `StringConverter`. Klasa posiada metody:

```

public static string convertToDb(string text, int length)
public static string convertFromDb(string text)

```

Użycie tych metod zapewnia poprawną obsługę wszystkich ciągów tekstu w zapytaniu SQL.

Funkcja `convertToDb` zamienia wszystkie znaki powodujące problemy w zapytaniach SQL: `'`, `\`, `@` na mało prawdopodobne do wystąpienia w tekście ciągi znaków `"@"`, `"@1"`, `"@2"`.

W ten sposób zwiększa się długość tekstów – funkcja ogranicza ją do wielkości danego pola w bazie danych. W ten sposób unikamy otrzymania wyjątku od bazy danych informującego o przekroczeniu dozwolonej długości słowa. Innym wyjściem jest zamiana na pojedynczy mało prawdopodobny do wystąpienia znak np. `'~'` – nie wymaga to zmiany długości pól.

Funkcja `convertFromDb` wykonuje konwersję odwrotną.

Wróćmy jeszcze do klasy `CountryFilter`. Metoda `generateSubFromSQL` zwraca pusty ciąg, gdyż klasa `Country` nie jest powiązana w naszym przykładzie asocjacją z żadną inną klasą (mówiąc o aliasach pamiętajmy, że są one jednostronne – klasa

Address posiada odwołanie do klasy Country, lecz nie na odwrót. Brakujące odwołanie można oczywiście uzupełnić). Spójrzmy na przykładową definicję klasy filtra, który posiada taką asocjację:

```
public class AddressFilter: AbstractFilter
{
    public PInt CountryId;
    public CountryFilter countryFilter;
    public string City;
    ...

    public override string generateSubWhereSQL( )
    {
        string sql = "";

        if( this.CountryId != null )
        {
            sql = this.appendSQL( sql ) + this.aliasName +
                ".country_id = " + this.CountryId.Value;
        }

        if( this.countryFilter != null )
        {
            sql += this.countryFilter.generateSubWhereSQL();
        }

        if( this.City != null )
        {
            sql = this.appendSQL( sql ) + this.aliasName +
                ".city LIKE '" +
                StringConverter.convertToDb( this.City,
                Address.LENGTH_CITY ) + "'";
        }

        ...
        return sql;
    }

    public override string generateSubFromSQL( )
    {
        string sql = "";
        if( this.countryFilter != null )
        {
            sql += " INNER JOIN t_countries " +
                this.countryFilter.getAliasName() +
                " ON " + this.aliasName + ".country_id = " +
                this.countryFilter.getAliasName()
                + ".country_id )";
            sql += this.countryFilter.generateSubFromSQL();

            AliasCreator.bracketCount++;
        }
    }
}
```

```

        return sql;
    }
}

```

Atrybut `countryFilter` klasy `AddressFilter` to odpowiednik obiektu klasy `Country` dla klasy `Address`. Jest to zagnieżdżony filtr pozwalający na wygenerowanie części zapytania wiążącej tabelę `t_addresses` z tabelą `t_countries` poprzez klauzulę `INNER JOIN`, a także części zapytania odnoszącej się do tabeli `t_countries` i ograniczającej wynikowy zbiór rekordów.

Zgodnie z powyższym w metodzie `generateSubWhereSQL` dodano wpis:

```

if( this.countryFilter != null )
{
    sql += this.countryFilter.generateSubWhereSQL();
}

```

a w metodzie `generateSubFromSQL` wpis:

```

if( this.countryFilter != null )
{
    sql += " INNER JOIN t_countries " +
    this.countryFilter.getAliasName() +
    " ON " + this.aliasName + ".country_id = " +
    this.countryFilter.getAliasName()
    + ".country_id )";
    sql += this.countryFilter.generateSubFromSQL();
    AliasCreator.bracketCount++;
}

```

tworzący połączenie pomiędzy tabelami i używający aliasów tabel w celu uniknięcia konfliktów nazw. Zagnieżdżenie jest rekurencyjne i obiekt `countryFilter` generuje swoją część zapytania w miejscu `FROM`. Zwiększona zostaje także liczba nawiasów zamykających następujących po klauzuli `INNER JOIN` w celu wygenerowania równoważnej liczby w znaczniku `{0}` zapytania.

Filtry mogą być w sobie dowolnie zagnieżdżane, nie ma ograniczenia głębokości zagnieżdżenia.

Znając powyższe możemy teraz łatwo zbudować strukturę filtrów generującą zdefiniowane wcześniej zapytanie:

```

SELECT * FROM ( t_addresses T0 INNER JOIN t_countries T1 ON
T1.country_id = T0.country_id ) WHERE ( 1 = 1 ) AND T0.city LIKE
'Warszawa' AND T1.name LIKE 'Polska' ORDER by T0.street, T1.country_id

```

Kod programu budujący strukturę filtrów na postać:

```

AddressFilter addressFilter = new AddressFilter();

```

```

addressFilter.city = "Warszawa";
CountryFilter countryFilter = new CountryFilter();
countryFilter.name = "Polska";
addressFilter.countryFilter = countryFilter;
addressFilter.orderBy = addressFilter.aliasName + ".street " + ", " +
countryFilter.aliasName + ".country_id";

```

Takie zagnieżdżenie filtrów da w wyniku funkcji `fillQueryString` klasy `AbstractFilter` następująco połączone części zapytania:

```

SELECT * FROM ( t_addresses T0 INNER JOIN t_countries T1 ON T1.country_id
= T0.country_id ) WHERE ( 1 = 1 ) AND T0.city LIKE 'Warszawa' AND T1.name
LIKE 'Polska' ORDER BY T0.street, T1.country_id

```

gdzie:

TEKST – bazowe zapytanie przekazane jako parametr do funkcji `fillQueryString`

TEKST – część wygenerowana przez `AddressFilter`

TEKST – część wygenerowana przez `CountryFilter`

Pomińmy tymczasowo definicję pól `attachObject`. Przyjrzyjmy się natomiast klauzuli `ORDER BY` i jej obsłudze w kodzie następującego programu:

```

addressFilter.orderBy = addressFilter.aliasName + ".street " + ", " +
countryFilter.aliasName + ".country_id";

```

Działanie jest oczywiste i proste. Do parametru `orderBy` głównego filtru dostawiamy zbudowany na podstawie aliasów oraz nazw kolumn łańcuch uporządkowania. Tak proste rozwiązanie pozwala na wygenerowanie dowolnego uporządkowania obiektów bardzo małym nakładem pracy. Kolejna funkcjonalność zostaje zamknięta.

4. FABRYKI

Fabryki to klasy służące do budowania obiektów. W rozważanym programie jest potrzebna jeszcze generyczna klasa, która budować będzie złożone obiekty definiowane przez zagnieżdżoną strukturę filtrów.

Zdefiniujmy następująco abstrakcyjną klasę `AbstractFactory`:

```

public abstract class AbstractFactory
{
    public abstract Object createObject( DataSet dataSet,
        AbstractFilter filter, int rowNumber );
}

```

Abstrakcyjna klasa fabryki posiada metodę `createObject` zdolną wygenerować dowolny obiekt na podstawie danych z wiersza o numerze `rowNumber` z obiektu typu `DataSet`, o strukturze opisanej przez obiekt klasy dziedziczącej po klasie `AbstractFilter`.

`AbstractFactory` jest klasą bazową dla wszystkich fabryk w aplikacji. Stwórzmy te klasy. Przykładowo:

```
public class CountryFactory: AbstractFactory
{
    public override Object createObject( DataSet dataSet,
        AbstractFilter filter, int rowNumber )
    {
        Country country = new Country();
        country.countryId = (int)dataSet.Tables[0].Rows[
            rowNumber ].ItemArray[
                this.getId( "country_id", filter.getAliasName(), dataSet)
            ];
        country.code = StringConverter.convertFromDb(
            (string)dataSet.Tables[0].Rows[
                rowNumber ].ItemArray[ this.getId( "code",
                    filter.getAliasName(), dataSet) ] );
        country.name = StringConverter.convertFromDb(
            (string)dataSet.Tables[0].Rows[
                rowNumber ].ItemArray[ this.getId( "name",
                    filter.getAliasName(), dataSet) ] );
        return country;
    }
}
```

Metoda `createObject` tworzy obiekt klasy `Country`, wypełnia jego dane oraz zwraca go jako wynik.

Pomocnicza metoda `getId( string columnName, string aliasName, DataSet dataSet)` pobiera indeks z tablicy, w którym odwzorowana jest kolumna `aliasName.columnName`. Implementacja została pominięta w celu uproszczenia przykładu.

Metoda `convertFromDb` klasy `StringConverter` została już wcześniej omówiona.

Rozpatrzmy teraz nieco bardziej skomplikowaną fabrykę:

```
public class AddressFactory: AbstractFactory
{
    public override Object createObject( DataSet dataSet,
        AbstractFilter filter, int rowNumber )
    {
        Address address = new Address();
        address.addressId = (int)dataSet.Tables[0].Rows[
            rowNumber ].ItemArray[
                this.getId( "address_id", filter.getAliasName(), dataSet)
            ];
    }
}
```

```

address.countryId = (int) dataSet.Tables[0].Rows[
rowNumber ].ItemArray[
this.getId( "country_id", filter.getAliasName(), dataSet)
];
address.city = StringConverter.convertFromDb(
( string) dataSet.Tables[0].Rows[rowNumber ].ItemArray[
this.getId( "city", filter.getAliasName(), dataSet) ] );
...
AddressFilter addressFilter = (AddressFilter) filter;
if( addressFilter.countryFilter != null &&
addressFilter.countryFilter.attachObject )
{
    CountryFactory countryFactory = new
CountryFactory();
address.country = (Country)
countryFactory.createObject( dataSet,
addressFilter.countryFilter, rowNumber );
}

return address;
}
}

```

Metoda `createObject` sprawdza dodatkowo, czy do filtru adresu dodano filtr państwa oraz czy zażądano by dołączyć ten obiekt do struktury wynikowej. Jeśli tak, tworzony jest obiekt państwa i dodany do obiektu klasy adresu.

Tutaj właśnie kluczową rolę odgrywa flaga `attachObject`. Jeśli dodamy obiekt klasy `CountryFilter` do obiektu klasy `AddressFilter`, to możliwe staje się wyszukiwanie adresów także po parametrach krajów. Jeśli dodatkowo ustawiona zostanie flaga `attachObject = true`, to w wyniku obiekt klasy `Country` zostanie dołączony do obiektu klasy `Address`.

Naszkicowane rozwiązanie pozwala na budowanie dowolnych zapytań i tworzenie dowolnych struktur na ich podstawie.

5. DATASOURCE

Ostatnią klasą generycznej warstwy pośredniej, którą stworzymy jest `DataSource` – źródło danych wychodzących z warstwy pośredniej. Klasa ma postać:

```

public class DataSource
{
    MethodInvocation methodInvoker = new MethodInvocation();

    public ArrayList getCountryList( CountryFilter countryFilter )
    {
        String query = countryFilter.fillQueryString(
        DataBase.Queries.getCountryList );
    }
}

```

```

        return this.methodInvoker.getObjectList( query, new
        CountryFactory(), countryFilter );
    }

    public ArrayList getAddressList( AddressFilter addressFilter )
    {
        String query = addressFilter.fillQueryString(
        DataBase.Queries.getAddressList );
        return this.methodInvoker.getObjectList( query, new
        AddressFactory(), addressFilter );
    }

    ...
}

```

Klasa służy do faktycznego pobierania danych przez warstwę logiki biznesowej aplikacji z warstwy pośredniej. DataSource zapewnia metody pobrania dowolnej listy obiektów.

Klasa MethodInvocationer pochodzi z warstwy bezpośredniego dostępu do bazy danych. Jej metoda getObjectList(string query, AbstractFactory abstractFactory, AddressFilter addressFilter) wywołuje zapytanie (query) i jego wyniki – w postaci obiektu klasy DataSet – przekazuje wraz z filtrem do fabryki w celu utworzenia obiektów, które są końcowo zwracane.

Dodatkowo, dla ułatwienia korzystania z DataSource dodajmy do niej metody pobierające nie listy lecz pojedyncze obiekty. Są to:

```

public Country getCountry( CountryFilter countryFilter )
{
    String query = countryFilter.fillQueryString(
    DataBase.Queries.getCountryList );
    return (Country)this.methodInvoker.getObject( query, new
    CountryFactory(), countryFilter );
}

public Address getAddress( AddressFilter addressFilter )
{
    String query = addressFilter.fillQueryString(
    DataBase.Queries.getAddressList );
    return (Address)this.methodInvoker.getObject( query, new
    AddressFactory(), addressFilter );
}

```

6. CREATE/UPDATE/DELETE

Otrzymaliśmy generyczną klasę dającą duże możliwości pobierania obiektów odpowiadającym relacjom baz danych, tworzenia ich skomplikowanych struktur oraz filtrowania według wyspecyfikowanych kryteriów.

Pobieranie obiektów to jednak tylko część zadań, które spełniać powinna warstwa pośrednicząca. Pozostałe zadania to dodanie, zmiana i usunięcie danych. Te funkcjonalności są jednak znacznie prostsze w implementacji.

Dla przykładu zbudujmy generyczne zapytania SQL dla klasy Country:

```
INSERT INTO t_countries( code, name ) VALUES ( '{0}', '{1}' )
UPDATE t_countries SET code = '{0}', name = '{1}' WHERE country_id = {2}
DELETE FROM t_countries WHERE country_id = {0}
```

Rozwiązanie okazuje się banalne. W miejsca znaczników {0}, {1}, ... wystarczy podstawić odpowiednie wartości z obiektu klasy Country. Zastrzeżenia budzić może to, że przy akcji UPDATE nadpisywane są wszystkie pola. Jednak biorąc pod uwagę to, że zmiany w bazie danych dokonywane są raczej pojedynczo, możemy zauważyć, że rozwiązanie to wymaga niewiele większego nakładu obliczeń, a w zamian zapewnia bardzo dużą wygodę w posługiwaniu się warstwą pośredniczącą.

W efekcie dodajmy do klasy DataSource dla każdego z obiektów proste funkcje wypełniające znaczniki i wywołujące powstałe zapytania. Są to:

```
public bool createCountry( Country country )
{
    string query = String.Format( DataBase.Queries.createCountry,
        StringConverter.convertToDb( country.code, Country.LENGTH_CODE),
        StringConverter.convertToDb( country.name, Country.LENGTH_NAME)
    );
    return this.methodInvoker.execute( query );
}
public bool updateCountry( Country country )
{
    string query = String.Format( DataBase.Queries.updateCountry,
        StringConverter.convertToDb( country.code, Country.LENGTH_CODE),
        StringConverter.convertToDb( country.name, Country.LENGTH_NAME),
        country.countryId );
    return this.methodInvoker.execute( query );
}
public bool deleteCountry( Country country )
{
    string query = String.Format( DataBase.Queries.deleteCountry,
        country.countryId );
    return this.methodInvoker.execute( query );
}
```

Funkcja execute(string query) klasy MethodInvoker służy do wywołania zapytania przekazanego w parametrze.

Możemy dodatkowo zmodyfikować metodę createCountry tak, by zwracała wartość typu int dla tabel z kluczem autonumeracyjnym (wartość klucza).

Na koniec wydzielmy wszystkie zapytania SQL do osobnej klasy Queries w celu łatwiejszego ich odnajdywania i modyfikacji. Klasa Queries powinna zawierać dla każdej z tabel cztery zapytania, przykładowo:

```

public static String getCountryList = "SELECT * FROM {0} t_countries
{1} WHERE ( 1 = 1 ) {2}";
public static String createCountry = "INSERT INTO t_countries( code,
name ) VALUES ( '{0}', '{1}' )";
public static String updateCountry = "UPDATE t_countries SET code =
'{0}', name = '{1}' WHERE country_id = {2}";
public static String deleteCountry = "DELETE FROM t_countries WHERE
country_id = {0}";

```

Stworzona warstwa pośrednicząca składa się z elementów:

- Obiekty reprezentujące dane
- Filtry
- Fabryki
- *DataSource*
- *StringConverter*
- *Queries*
- Definicje rozszerzonych typów (*PLnt*, *PDouble* ...)

6.1. WARSTWA BEZPOŚREDNIEGO DOSTĘPU DO BAZ DANYCH

Zajmijmy się teraz strukturą warstwy bezpośredniego dostępu do bazy danych. Warstwa ta powinna składać się co najmniej z dwóch klas:

MethodInvoker

Klasa wspomniana wcześniej, zawiera definicje metod *getObject* – pobierającą pojedynczy obiekt, *getObjectList* – pobierającą listę obiektów, *execute* – wywołującą dowolne zapytanie SQL (*INSERT*, *UPDATE*, *DELETE*).

ConnectionHolder

Klasa odpowiedzialna za przechowywanie połączenia/połączeń z bazą danych, wykorzystywana przez *MethodInvoker* przy wywoływaniu zapytań. Składowa klasy *MethodInvoker*.

6.2. WARSTWA LOGIKI BIZNESOWEJ

W tej warstwie zaimplementowana jest cała logika biznesowa aplikacji. Do operowania na danych z bazy wykorzystywane są funkcje klasy *DataSource* – pobierające, dodające, zmieniające i usuwające dane. Warstwa ta abstrahuje od sposobu budowania zapytań oraz połączenia z bazą danych. Programistom łatwiej jest skupić się na rzeczywistych problemach logiki biznesowej.

6.3. WARSTWA INTERFEJSU UŻYTKOWNIKA

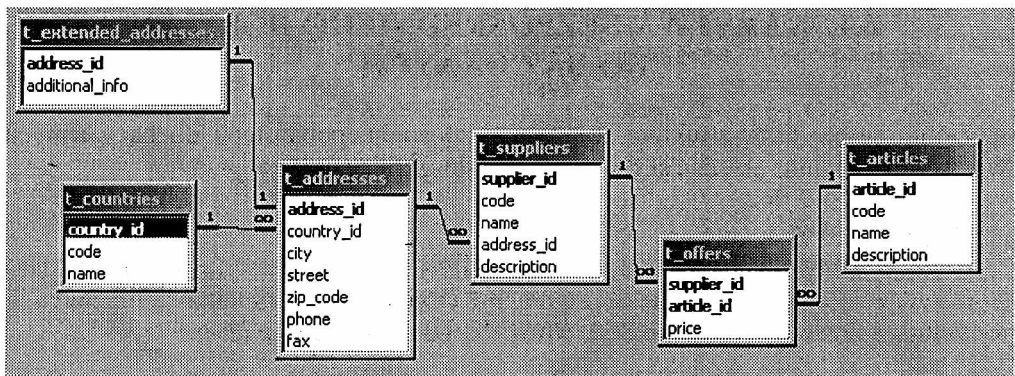
Klasy z tej warstwy bezpośrednio wywołują gotowe funkcje odpowiedzialne za logikę biznesową z warstwy logiki biznesowej.

7. DODATKI/ULEPSZENIA/MODYFIKACJE

7.1. IMPLEMENTACJA DZIEDZICZENIA

Do warstwy pośredniczącej bardzo łatwo dodać funkcjonalność dziedziczenia klas odpowiadającą rozszerzaniu tabel bazy danych (rysunek 2).

Związek pomiędzy tabelami `t_addresses` i `t_extended_addresses` odzwierciedla dziedziczenie.



Rys. 2. Przykładowy rozszerzony model bazy danych

Implementację można przedstawić w następujących krokach:

1. Tworzymy klasę `ExtendedAddress` dziedziczącą po klasie `Address`.
2. Tworzymy klasę `ExtendedAddressFilter` dziedziczącą po klasie `AddressFilter`. Dodajemy tej klasie dodatkowy parametr `newAliasName` i generujemy dla niego alias; do funkcji `getSubFromSQL()` wstawiamy stały wpis łączący tabelę `t_extended_addresses` z tabelą `t_addresses`. Parametru `newAliasName` używamy zawsze przy polach z tabeli `t_extended_addresses`; `aliasName` używamy natomiast zawsze przy odniesieniach do tabeli `t_addresses`.
3. Tworzymy osobną klasę `ExtendedAddressFactory`. W implementacji tej klasy, na początku metody `createObject` tworzymy fabrykę typu `AddressFactory` oraz wywołujemy jej metodę `createObject` w celu utworzenia obiektu bazowego. Jego

parametrami inicjujemy nowo tworzony obiekt `ExtendedAddress`, pozostałe parametry `ExtendedAddress` pobieramy z danych obiektu klasy `DataSet`.

Przy połączeniach z innymi filtrów, zawsze używamy `newAliasName` z filtru `ExtendedAddressFilter` w celu definicji parametrów łączących klauzuli `INNER JOIN`.

7.2. PULE POŁĄCZEŃ

Stworzony mechanizm równie dobrze nadaje się do korzystania z pojedynczego połączenia jak i z puli połączeń. Pojedyncze połączenie z klasy `ConnectionHolder` wystarczy zastąpić pulą połączeń, a odniesienia do połączenia – wywołaniem odpowiednich metod puli połączeń.

7.3. TRANSAKCJE

Nic nie stoi na przeszkodzie dodania transakcyjności do warstwy pośredniczącej.

W przypadku pojedynczego połączenia wystarczy dodać funkcje inicjalizujące oraz kończące transakcje w klasie `DataSource`, będą one następnie wywoływane przez warstwę logiki biznesowej przed i po pewnym ciągu zadań.

Dla puli połączeń implementacja wygląda podobnie, z tym że dodajemy do wszystkich metod z `DataSource` parametr połączenia z bazą danych, tak by warstwa logiki biznesowej przekazywała połączenie do tych metod. Oczywiście pozostawić można dodatkowo metody nie pobierające połączeń.

7.4. SZABLONY

Wraz z nowym frameworkiem pojawiają się szablony. Ich wykorzystanie może jeszcze poprawić architekturę, gdyż wprowadza ono silną typizację. Aby dodać szablony do warstwy pośredniczącej należy wszystkie klasy fabryk uczynić szablonowymi (*dwa parametry szablonu*: klasa obiektu zawierająca dane, klasa filtru). Wymusi to utworzenie wielu obiektów klasy `MethodInvoker` – również szablonowych, z identycznymi parametrami.

7.5. ROZSZERZENIE FILTRÓW DLA RELACJI JEDEN DO WIELU, WIELE DO WIELU

Rozważmy tabelę `t_orders` przechowującą dane zamówień oraz tabelę `t_order_positions` przechowującą dane pozycji zamówień z tabeli `t_orders`. Możemy rozszerzyć funkcjonalność filtrów, aby możliwe było zadanie zapytania

postaci: „Wybierz wszystkie zamówienia, w których chociaż jedna pozycja jest na artykuł nazwany Monitor”.

W klasie `OrderPositionFilter` należy dodać atrybut `parentAliasName` i nadać mu wartość równą aliasowi z obiektu klasy `OrderFilter`. W metodzie `generateWhereSQL()` należy dodać sprawdzenie, czy został ustawiony atrybut `parentAliasName`. Jeśli tak, musimy wygenerować klauzulę `EXISTS`, a w niej selekcję rekordów z połączenia pozycji i zamówienia (przy użyciu `parentAlias` i pozostałych parametrów obiektu klasy `OrderPositionFilter`).

8. PODSUMOWANIE

Język C# wraz z technologią ADO.NET doskonale nadaje się do oprogramowania generycznej warstwy pośredniczącej. Warstwa pośrednicząca stworzona w pierwszej fazie implementacji projektu może znacznie przyspieszyć dalszy jego rozwój. Atutem jest, że w zależności od wielkości oraz złożoności projektu, implementacja warstwy pośredniczącej może być różna. Czasem wystarczy stworzyć podstawową funkcjonalność z generycznymi filtrami, czasem zaś przydatne mogą okazać się dodatki i ulepszenia.

Generyczna warstwa pośrednicząca jest mechanizmem dającym duże możliwości. Pozwala programistom przejść na wyższy poziom abstrakcji i skupić się na poprawnym implementowaniu logiki biznesowej aplikacji, zapobiega dublowaniu kodu i porządkuje aplikację.

LITERATURA

- [PETZOLD 2002] Charles Petzold, *Programowanie Microsoft Windows w Języku C#*. ReadMe, 2002.
- [LIBERTY 2001] Jessie Liberty, *Programming C#*. O'Reilly, 2001.
- [MSDN 2003] MSDN, 2003.

GENERIC DATABASE MIDDLEWARE

The article presents description of database middleware that makes programming business logic much easier, prevents code doubling and cleans the application. Using simple example the article presents the architecture and implementation of such mechanism.

*saga, transakcja długa,
metoda znaczników czasowych,
metoda wielowersyjna*

Zygmunt MAZUR*

ZARZĄDZANIE DŁUGIMI TRANSAKCJAMI

W artykule zdefiniowano pojęcie długiej transakcji oraz zaprezentowano problemy, jakie występują w systemach baz danych obsługujących takie transakcje. Zaprezentowano metody obsługi transakcji, które pozwalają zniwelować lub ominąć niektóre z przedstawionych problemów.

1. WPROWADZENIE

Jedną z podstawowych cech współczesnych systemów zarządzania bazami danych jest wielodostęp, umożliwiający wielu użytkownikom na jednoczesną pracę na danych zgromadzonych w bazie danych. W celu zapewnienia poprawności wykonywania złożonych operacji, ze szczególnym uwzględnieniem jednoczesnego operowania na tych samych danych, stosuje się operacje transakcji. Tradycyjne metody przetwarzania transakcji zakładają pomijalnie krótki czas wykonania transakcji. Jednak w praktyce często poziom złożoności przetwarzania lub liczba danych, jakie należy przetworzyć powoduje, że pomimo rozwoju nowoczesnych technologii, nie można wykonać przetwarzania transakcji w wymaganym krótkim czasie. Należy wtedy wziąć pod uwagę problemy i komplikacje, jakie stwarza różnym użytkownikom systemu bazodanowego długotrwałe przetwarzanie transakcji.

Oprócz dużej liczby danych i złożonego procesu ich przetwarzania, powodem długiego przetwarzania transakcji może być źródło danych lub miejsce docelowego przechowywania danych. Współczesne systemy informatyczne mogą korzystać z wielu różnych źródeł danych. Może to być inny system komputerowy (także odległy), urządzenia różnego typu, czy po prostu użytkownik. Może się zdarzyć (szczególnie w przypadku kontaktu z użytkownikiem), że transakcja wykonuje się w krótkim czasie, ale kontakt ze źródłem danych jest o kilka rzędów wolniejszy od czasu operowania na danych. Wtedy czas wykonania transakcji wydłuża się i można wtedy mówić o długiej transakcji przy czym przymiotnik „długa” jest względny.

* Instytut Informatyki Stosowanej, Wydział Informatyki i Zarządzania Politechniki Wrocławskiej, 50-370 Wrocław, Wybrzeże Wyspiańskiego 27, zygmunt.mazur@pwr.wroc.pl

2. POJĘCIE TRANSAKCJI I TRANSAKCJI DŁUGIEJ

Transakcja (ang. *transaction*) to zbiór operacji na bazie danych, które mogą być wykonane albo wszystkie albo żadna i w tym sensie jest niepodzielna. W systemach bazodanowych pożądanym jest, aby transakcja trwała jak najkrócej. Transakcja składa się z etapu rozpoczęcia, wykonania i zamknięcia. Każdy etap transakcji jest zapisywany (logowany) w specjalnych dziennikach transakcji tzw. logach. W razie awarii systemu, dzięki zawartości logów, można odtworzyć stan bazy danych sprzed rozpoczęcia transakcji, która nie została prawidłowo zamknięta.

Współbieżne wykonywanie transakcji polega na wykonywaniu operacji zapisu i odczytu różnych transakcji tak, aby efekt był taki sam jak w przypadku niezależnego wykonania transakcji. Ta cecha nazywa się **szeregowalnością** (ang. *serializability*). Dwie operacje z różnych transakcji są w konflikcie, jeśli mają dostęp do tych samych danych i co najmniej jedna z nich jest operacją zapisu. Jeśli dwie operacje nie są w konflikcie, to można zmienić ich kolejność.

Przy dokładnej analizie zagadnienia przetwarzania współbieżnych transakcji, zauważamy, że główne problemy wynikają z tego, że transakcje trwają przez jakiś określony przedział czasu. Jeśli transakcja wykonywałaby się na bazie danych wystarczająco krótko, tak krótko, aby nie mogły w tym czasie zajść jakiegokolwiek inne zdarzenia, np. działanie innego użytkownika, to nie byłoby anomalii wynikających ze współbieżności transakcji. Analogicznie, gdyby transakcja była fizycznie jedną operacją, trwającą tak krótko, że czas jej wykonania mógłby być pomijany, to nie byłoby ryzyka nie wykonania części transakcji i konieczności odtwarzania po awarii. Można, zatem wnioskować, że długie transakcje niosą ze sobą nowe problemy, a ich obsługa może być bardziej skomplikowana [GaUIWi 2003, GoGh 1999, KiBeLe 2003]. Często stosuje się metodę wstrzymania wykonania transakcji, jeśli operacja wykonana w ramach danej transakcji miałaby spowodować konflikt. Wykonanie jest kontynuowane, gdy zakończy się transakcja, z którą mogło dojść do konfliktu. Jeśli wszystkie transakcje mogą zostać wstrzymane do zakończenia transakcji T, to transakcja T jest transakcją krótką. Jeśli natomiast istnieją w systemie transakcje, które mogą doprowadzić do konfliktu z transakcją T i powinny one być wykonane w czasie, który nie może zależeć od czasu wykonania transakcji T, to T jest **transakcją długą**. Długa transakcja (ang. *long live transaction* lub *long transaction*) może trwać kilka sekund, minut, kilka godzin czy kilka dni. Jest to zależne od złożoności transakcji, ilości danych oraz dostępnych zasobów. Dla systemów czasu rzeczywistego nie dozwolone są często nawet sekundowe oczekiwania. W systemach obsługi klienta, np. w systemie bankowym, opóźnienie rzędu sekundy jest nie zauważalne i możliwe do zaakceptowania. W systemach wspomagania podejmowania decyzji nawet oczekiwanie kilku minutowe jest czasami całkowicie akceptowalne.

Długie transakcje najczęściej występują w trzech obszarach:

- Tradycyjne bazy danych. Zazwyczaj występują tutaj krótkie transakcje, ale mogą zdarzyć transakcje na tyle skomplikowane, że ich wykonanie będzie

długotrwałe. Może to wynikać np. z dużej złożoności obliczeniowej problemu lub z dużej liczby danych do przeanalizowania. Czas trwania transakcji może być długi ze względu na dużą liczbę operacji wykonywanych w transakcji.

- Systemy projektowe. Projekt z założenia jest procesem tworzenia czegoś nowego i wymaga procesu twórczego myślenia projektanta, co zazwyczaj trwa długi, jak na czas wykonania transakcji w bazie danych, okres czasu. Przykładem tego zastosowania może być wykorzystanie bazy danych jako repozytorium dla wyników dużego projektu, nad którym pracuje wiele osób. Projekt składa się z wielu elementów, które są przechowywane jako dane w bazie danych (np. mogą być to pliki). Z założenia do projektu powinni mieć zapewniony równoczesny dostęp różni projektanci. Z założenia powinni oni mieć możliwość przeglądania i modyfikacji projektu. O ile z udostępnianiem danych do odczytu nie ma problemów, to w przypadku zapisu zmian w projekcie sytuacja się komplikuje. Jeśli dwóch projektantów będzie pracowało nad tą samą częścią projektu i obaj zechcą zapisać wynik swojej pracy, to rezultaty jednego z nich mogą zostać zaprzepaszczone. Prace nad projektem mogą trwać kilka godzin lub kilka dni. Jeśli modyfikacja przez jednego projektanta wiązałaby się z zablokowaniem dostępu do tej części przez innych projektantów, to mogłoby to spowodować opóźnienie w realizacji innych, powiązanych części projektu.
- Systemy przepływu czynności. W przypadku takich rozwiązań procesy składają się z wielu, często skomplikowanych, czynności. O ile są to procesy nie wymagające udziału człowieka, to są to systemy, które równie dobrze możemy zakwalifikować do pierwszej wymienionej grupy obszarów zastosowań długich transakcji. Jednak zazwyczaj przepływy czynności wymagają udziału czynnika ludzkiego, a wtedy czas wykonywanych czynności może się wydłużać do rozmiarów godzin, dni, tygodni. Przykładem zastosowania tego typu rozwiązań są np. systemy elektronicznego obiegu dokumentów. Dokumenty mogą wymagać zatwierdzenia przez różne osoby, których np. może aktualnie nie być w pracy, które mogą zaniedbywać swoje obowiązki itp.

W wymienionych powyżej obszarach można wyróżnić dwie przyczyny powstawania długich transakcji. Pierwszą przyczyną powodującą, że transakcje są długie jest ich złożoność przetwarzania. Skomplikowane operacje na dużej ilości danych mogą czasami trwać za długo, aby można było pozwolić pozostałym transakcjom czekać na ich zakończenie. Czas trwania takich transakcji nie zawsze można skrócić poprzez zastosowanie wydajniejszego komputera, optymalizacji zapytań kierowanych do bazy danych czy zmianę urządzeń przechowujących dane. Są takie zastosowania, dla których nie ma dostępnych rozwiązań technicznych i których zastosowanie pozwoliłoby skrócić czas wykonania transakcji, tak aby był on akceptowalnym czasem na jaki można zablokować dostęp do danych przez inne transakcje. Jeżeli natomiast długość transakcji jest spowodowana obciążeniem serwera bazy danych można starać się odciążyć serwer przez rozproszenie przetwarzania transakcji na kilka serwerów. Są także przypadki, gdzie rozwiązaniem problemu długich transakcji jest przesunięcie wyko-

niania takich transakcji do chwili, kiedy może ona być wykonana jak zwykła transakcja poprzez blokowanie dostępu do danych. Jeśli dostępność systemu nie jest wymagana stale, ale tylko w wyznaczonych godzinach np. w godzinach pracy przedsiębiorstwa, to można długie transakcje wykonać w czasie przerw w pracy systemu, np. w nocy. W artykule nie będziemy rozważać przypadków, w których można uzyskać efekt skrócenia czasu przetwarzania transakcji za pomocą modyfikacji sprzętowych lub dopuszczenia wykonywania transakcji z blokowaniem danych poza czasem dostępności systemu dla użytkowników.

Drugim czynnikiem, który często powoduje długi czas trwania transakcji, jest interakcja z użytkownikiem. Użytkownikiem może być zarówno człowiek, jak i również inny system komputerowy, bądź inne urządzenie. W przypadku, gdy transakcja komunikuje się z człowiekiem, to czas wykonywania operacji na danych może być znacznie krótszy (pomijalnie mały) w porównaniu do czasu reakcji użytkownika. Zwłaszcza w przypadku prac projektowych, jak to było opisane powyżej. Jeśli mówimy o współpracy transakcji z innym systemem komputerowym bądź innym urządzeniem, to musimy wziąć pod uwagę potrzebę komunikacji sieciowej, która także może trwać znaczącą ilość czasu, zwłaszcza, jeśli przesyłamy duże ilości danych.

Przyczyna decydująca o tym, że transakcja jest długa, niesie ze sobą charakterystykę danego typu transakcji. Jeżeli transakcja wymaga interakcji z użytkownikiem, to zazwyczaj będzie się ona sprowadzała do okresowych, stosunkowo krótkich operacji na bazie danych. Czas operacji na danych będzie zazwyczaj pomijalnie mały w porównaniu do czasu trwania całej transakcji. Natomiast transakcje, których długi czas wykonania wynika z ich dużej złożoności obliczeniowej, są ciągiem wykonywanych operacji. Operują one najczęściej na dużych zbiorach danych i nie występują przerwy w ich wykonywaniu. Mogą się również zdarzyć sytuacje, gdzie długi czas wykonania transakcji będzie spowodowany zarówno złożonością obliczeniową, jak i interakcją z użytkownikiem. Wtedy cechy takiej transakcji będą złożeniem cech charakterystycznych długich transakcji opisanych powyżej.

3. PROBLEMY ZWIĄZANE Z DŁUGIMI TRANSAKCJAMI

Problemy z transakcjami krótkimi wiążą się z koniecznością odtwarzania stanu systemu sprzed transakcji w przypadku awarii bądź wycofania transakcji oraz z anomaliami, jakie występują przy równoczesnym dostępie do danych przez kilka transakcji. Nie znikają one w przypadku długich transakcji, co więcej, komplikują się one jeszcze bardziej lub pojawiają się nowe. Problemy występujące w przypadku długich transakcji zależą od rodzaju tych transakcji. Interakcja z użytkownikiem pociąga za sobą inne ograniczenia i kłopoty, niż gdy długi czas transakcji wynika ze złożoności obliczeniowej. Rozpatrzmy teraz transakcje długie, złożone obliczeniowo.

W przypadku problemów z wykonaniem transakcji, transakcja powinna zostać wycofana. Można się spodziewać, że wykonanie operacji wycofania transakcji długiej będzie bardziej czasochłonne, niż w przypadku krótkich transakcji. Jeśli transakcja po wycofaniu będzie ponownie uruchomiona, to spowoduje to dodatkowe obciążenie systemu. Jeśli transakcja składa się z wielu operacji i za każdym razem będzie inny powód przerwania transakcji, to może się zdarzyć, że próba wykonania transakcji będzie podejmowana wielokrotnie. Transakcja może przez długi czas obciążać system zanim zostanie ukończona, a jej czas wykonania może znacznie przekraczać czas jej wykonania bez zakłóceń.

Problem pojawia się także w przypadku długich transakcji z udziałem użytkownika, którego należy poinformować o fakcie wycofania transakcji oraz w przypadku, gdy sposób, w jaki użytkownik postrzega system, nie łączy w całość wszystkich operacji składających się na transakcję. Problem może się pojawić także wówczas gdy użytkownik równocześnie uruchamia wiele takich samych długich transakcji. Wtedy trzeba zadbać, aby był on jasno poinformowany, która z długich transakcji się nie powiodła.

W przypadku ponownego wykonania wycofanej transakcji z udziałem użytkownika problemem może być powtarzalność wprowadzonych danych. Jeśli podejmowane w ramach transakcji akcje, lub podawane do transakcji dane wynikają ze zwróconych wcześniej danych, to ponowne wykonanie może mieć inny przebieg, bądź operować na innych danych. Jeśli transakcja pobiera od użytkownika dane, które użytkownik po prostu wpisuje, to może on je wprowadzać ponownie. Jeśli jednak wprowadzane dane są częścią prac projektowych, to ich pozyskanie może wymagać dodatkowej pracy, np. ponownych obliczeń przez projektanta.

Komplikuje się także przetwarzanie transakcyjne równoległych transakcji, jeśli któraś z nich jest długa. Sama definicja długich transakcji odrzuca ich realizację za pomocą blokowania dostępu do danych. W dalszej części pracy zostaną opisane metody realizacji dostępu do danych. Jednak w przypadku konfliktów przy realizacji szeregowalności transakcji wykorzystują one wycofanie transakcji, która powoduje konflikt i rozpoczęcie na nowo jej wykonania. Problemy wynikające z wycofania transakcji zostały już wcześniej opisane.

W przypadku długich transakcji z udziałem użytkownika ponowne uruchomienie transakcji może wymagać, aby transakcja była widziana jako nowa długa transakcja z aktualnym czasem zgłoszenia do systemu, aby użytkownik miał dostęp do najbardziej aktualnych danych – zmodyfikowanych od poprzedniego uruchomienia tej transakcji. Wtedy może się zdarzyć, że transakcja nigdy nie zostanie zakończona pomyślnie, gdyż za każdym razem pojawi się transakcja powodująca konflikt z daną transakcją. Może wówczas wystąpić zjawisko zagłodzenia (transakcja nieustannie jest w stanie oczekiwania na wykonanie). Pomijając nawet zjawisko zagłodzenia, nie jest poprawną sytuacją, w której użytkownik próbuje wielokrotnie przeprowadzić transakcję, podejmuje decyzje, wprowadza dane a transakcja zostaje za każdym razem wycofywana i wykonywanie jej zaczyna się od początku.

Można zauważyć, że obsługa długich transakcji ma także wpływ na krótkie transakcje pojawiające się w systemie. Przykładem takiej zależności może być wstrzymanie zatwierdzenia krótkiej transakcji, aż do chwili zakończenia wszystkich transakcji (w tym długich), które rozpoczęły się przed rozpoczęciem tej transakcji. Warunek taki może być wymagany w celu uniknięcia anomalii współbieżnego wykonania i zachowania pełnej izolacji transakcji. Zatem obsługa długich transakcji w sposób tradycyjny może powodować wydłużenie krótkich transakcji.

4. METODY ZARZĄDZANIA DŁUGIMI TRANSAKCJAMI

Do przetwarzania długich transakcji należy zastosować takie metody, które umożliwiają wyeliminowanie problemów z nimi związanych. Głównym problemem transakcji długich jest blokowanie dostępu do danych, co jest domeną pesymistycznych metod sterowania współbieżnymi transakcjami. Dlatego też w przypadku obsługi długich transakcji należy zastosować metody optymistyczne, które nie wykorzystują blokowania dostępu do danych. Do zarządzania długimi transakcjami można wykorzystać sagi lub metodę znaczników czasowych, które omówimy w niniejszej pracy lub metodę wielowersyjną, której opis można znaleźć w literaturze, np. [Batt2002, Bern 1987, BuSi 1993, Kosz 2000, WeSc 1992].

4.1. SAGA

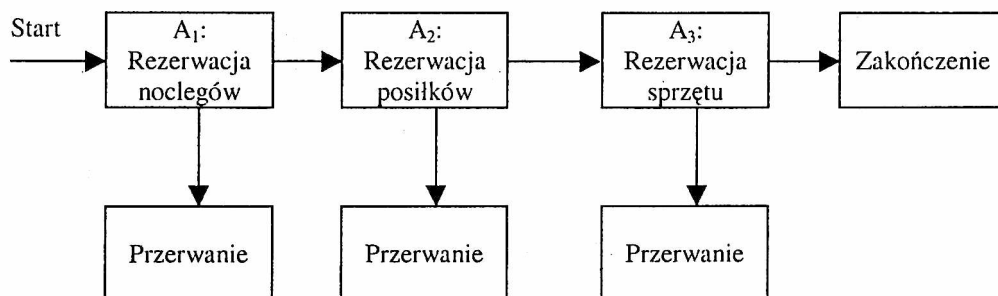
Saga (ang. *saga*) jest przypadkiem transakcji zagnieżdżonej zawierającej tylko dwa poziomy zagnieżdżenia. Pierwszym najwyższym poziomem jest sama saga, a na nią składa się sekwencja niezależnych od siebie podtransakcji. Każda z podtransakcji posiada transakcję kompensacyjną na wypadek potrzeby wycofania zmian przez nią wprowadzonych [CoBe 2004]. Podtransakcje wykonywane w ramach danej sagi są uruchamiane w sposób sekwencyjny, jedna po drugiej. Jednak tak, jak to jest w przypadku transakcji zagnieżdżonych, może być jednocześnie uruchomionych wiele transakcji. Mogą to być inne sagi, bądź też zwykłe transakcje. Wtedy takie transakcje nie są izolowane na poziomie sagi, lecz na poziomie podtransakcji. Zatem można się zdarzyć, że kolejna podtransakcja nie będzie operowała na wynikach poprzednio wykonanej podtransakcji, w ramach danej sagi, lecz na wartościach, które wynikają z innej transakcji, która nie ma nic wspólnego z przetwarzaną sagą.

Wyniki zatwierdzonych podtransakcji widoczne są poza sagą bezpośrednio po ich zatwierdzeniu, a nie dopiero po zakończeniu sagi. Oznacza to, że sagi są transakcjami zagnieżdżonymi typu **otwartego**. Jest to związane z opisanym powyżej brakiem izolacji pomiędzy sagami.

Saga kończy się w sposób normalny w chwili, gdy zostaną wykonane wszystkie podtransakcje składające się na nią. Jednak jest też możliwość, że któraś z podtransakcji nie wykona się poprawnie np. wystąpi błąd lub potrzeba jej wycofania. Wtedy saga zakończy się w sposób anormalny, tzn. wykonanie kolejnych podtransakcji jest wstrzymane, natomiast wykonywane są transakcje kompensacyjne dla podtransakcji już zatwierdzonych.

Definicja, nie korzystająca z pojęcia transakcji zagnieżdżonej, określa sagę jako graf skierowany, którego wierzchołkami są czynności lub są to wierzchołki specjalne (przerwanie, zakończenie) [GaUIWi 2003]. Krawędzie tego grafu wyznaczają kolejność wykonania operacji. Graf ten ma wyróżniony wierzchołek, od którego rozpoczyna się saga. Nazywany jest on wierzchołkiem startowym.

Rysunek 1. przedstawia przykładową transakcję rezerwacji turystycznej, jako sagę w postaci grafu. Składa się ona z trzech podtransakcji A_1 , A_2 , A_3 , które mają być wykonane w podanej kolejności. Każda z transakcji może zakończyć się powodzeniem i saga przejdzie do wykonania kolejnej transakcji. Może też pojawić się problem z dokonaniem rezerwacji, wtedy saga jest przerywana i wykonywane są odpowiednie transakcje kompensacyjne. Jeśli problem nastąpi przy transakcji A_1 , to nie ma potrzeby wywoływania żadnych transakcji kompensacyjnej. Jeśli problem wystąpi przy A_2 , to należy wykonać transakcję kompensacyjną dla transakcji A_1 , a przy problemie z A_3 dla transakcji A_1 i A_2 .



Rys. 1. Graf reprezentujący sagę

Definicja sagi jako grafu powoduje w rezultacie dwie zasadnicze różnice w porównaniu z pierwszą definicją. Graf ten dopuszcza istnienie rozgałęzień oraz cykli. Zatem poprawnie wykonana saga nie musi obejmować wszystkich podtransakcji na nią się składających. Natomiast istnienie cykli pozwala, aby niektóre transakcje były wielokrotnie wykonane w ramach jednej sagi.

Także w znaczeniu pierwszej definicji istnieje możliwość wykonania sagi, która zawiera rozgałęzienia, czy powtarza wykonanie wielokrotnie tych samych operacji. Takie rozwiązanie wymaga, aby zapętlenie, czy rozgałęzienie umieścić w dokładnie jednej podtransakcji. W rozumieniu sagi jako grafu skierowanego może to być wydzielone poza podtransakcje.

Jeśli długa transakcja jest sekwencją operacji, które pozostawiają bazę w spójnym stanie i dla tych operacji można utworzyć operacje odwrotne, to można ją przedstawić jako sagę, gdzie wspomniane operacje utworzą podtransakcje. Wtedy podtransakcje mogą być przetwarzane w dowolny sposób, np. za pomocą blokowania dostępu do danych. Tak samo można przetwarzać transakcje kompensacyjne.

Podstawowym problemem w przypadku długich transakcji było blokowanie dostępu do danych dla innych transakcji. W przypadku sag nie ma potrzeby blokowania użytych danych do zakończenia transakcji. Blokowanie danych odbywa się na poziomie podtransakcji.

Nie wszystkie transakcje można z powodzeniem przekształcić na sagi. Jest kilka wymagań, które muszą być spełnione. Warunek pierwszy, to możliwość podziału transakcji na mniejsze niezależne jednostki. Chodzi o to, aby można było wyodrębnić takie operacje na danych, które można umieścić w jednej podtransakcji. Podtransakcja powinna pozostawiać bazę danych w stanie spójnym, gdyż wynik jej działania będzie widoczny poza sagą bezpośrednio po zakończeniu tej podtransakcji. Tworzenie z długiej transakcji sagi jest celowe wtedy, gdy wydzielone podtransakcje nie będą już transakcjami długimi. Chodzi o to, aby przekształcić długą transakcję w wiele transakcji, które nie będą powodowały blokowania dostępu do danych na czas zbyt długi dla oczekujących na rezultat współbieżnych transakcji. Wtedy saga nie będzie odczuwana jako długa transakcja.

Kolejnym wymaganiem jest możliwość utworzenia dla prawie każdej podtransakcji transakcji kompensacyjnej. Wiążą się z tym problemy, które były wcześniej opisane. Z analizy wywoływania transakcji kompensacyjnych można wywnioskować, że transakcja kompensacyjna nie jest wymagana dla ostatniej podtransakcji, gdyż po zatwierdzeniu ostatniej podtransakcji jest zatwierdzana saga, więc nie będzie nigdy potrzeby wycofania tej podtransakcji.

W związku z możliwością przeplatania się podtransakcji należących do sagi z innymi transakcjami pojawiają się problemy omówione wcześniej. Oprócz anomalii, jakie występują przy współbieżnych transakcjach pojawia się problem związany z transakcjami kompensacyjnymi. Jeśli pojawi się błąd przy wykonaniu podtransakcji lub potrzeba jej wycofania, to jest określony sposób postępowania. Co jednak jeśli problem wydarzy się podczas wykonywania transakcji kompensacyjnej? Jeśli wycofamy transakcję kompensacyjną, to baza nie powróci do stanu sprzed rozpoczęcia wykonywania sagi. Przykład tego typu problemu został omówiony już wcześniej.

Kolejny problem, wynikający ze współbieżności sagi i innych transakcji i mający swoje źródło w transakcjach kompensacyjnych, może się przydarzyć, gdy transakcja nie należąca do sagi zmodyfikuje zbiór danych, który powinien być zmodyfikowany w ramach transakcji kompensacyjnej. Przykładem może być następująca sytuacja: długa transakcja daje podwyżkę wszystkim pracownikom wydziału W_1 o 10%. Po tej podtransakcji jest wykonywana inna transakcja zmieniająca miejsce pracy pracownika P_1 z wydziału W_2 na W_1 oraz pozostałe podtransakcje danej sagi. Jeśli jednak nie powiedzie się wykonanie sagi i zajdzie potrzeba wycofania podtransakcji dającej pod-

wyżkę, to transakcja kompensacyjna obniży także pensję pracownika P_1 , chociaż nie powinna tego zrobić, bo saga nie nadawała podwyżki temu pracownikowi, gdyż nie był on jeszcze wtedy pracownikiem wydziału W_1 .

Transakcje kompensacyjne mogą także być przyczyną braku trwałości innych transakcji. Przy tworzeniu sagi należy ocenić występujące w bazie ograniczenia, aby móc ocenić możliwość wystąpienia opisanych problemów. Może się okazać, że możliwe jest rozluźnienie ograniczeń, jakie są nałożone na bazę danych. Jeśli długą transakcją modyfikującą stan konta jest transakcja odpowiadająca za dopisanie odsetek do aktualnego stanu konta, to może nie warto pilnować, czy klient nie chce wypłacić odsetek, które mu dodała jeszcze nie zatwierdzona saga. Można dopuścić do chwilowego nieprawidłowego stanu konta, gdyż nawet jeśli wskutek tej operacji pojawi się debet, to najprawdopodobniej, po prawidłowym naliczeniu odsetek i zatwierdzeniu sagi odpowiedzialnej za ten proces, debet ten zostanie zlikwidowany. Nie zawsze jednak takie rozluźnienie ograniczeń jest dopuszczalne.

Oprócz ograniczeń należy ocenić możliwe do wystąpienia w bazie danych operacje na danych. Jeśli z bazy danych korzysta tylko i wyłącznie specjalizowana aplikacja, to nie ma większego problemu ze sprecyzowaniem wykonywanych operacji, gdyż jest to stały zbiór. Jednak można znaleźć systemy, gdzie operacje pojawiające się w bazie danych są tworzone *ad hoc* i nie sposób je przewidzieć. Aby zapewnić poprawność wykonania sagi należy wykluczyć możliwość operacji na danych, które są wykorzystywane przez sagę, aż do jej zakończenia. Jednak gdyby taka ewentualność miała miejsce, to nie byłyby wymagane sagi, ale można by zastosować blokowanie.

Stosując sagę dla transakcji, której długi czas trwania wynika ze złożoności obliczeniowej, możemy zdecydować, czy saga ma być już wycofana, czy nie chcemy tracić rezultatów obliczeń i jeszcze raz podjąć próbę normalnego zakończenia sagi. Nie tracimy wyników wykonanej części sagi np. w przypadku wystąpienia awarii sprzętu czy oprogramowania. Możemy kontynuować sagę od ostatnio zatwierdzonej podtransakcji.

4.2. SAGI A WŁASNOŚCI ASIT

Wszystkie transakcje i protokoły zarządzania transakcjami muszą spełniać podstawowy postulat ASIT* czyli własności: Atomowość, Spójność, Izolacja, Trwałość. Te cztery własności nie są całkowicie niezależne od siebie, zwykle występują między nimi zależności [Bern 1987, KiBeLe 2003].

Nie zawsze w sagach są ściśle zachowane wszystkie własności ASIT. Jedynie cecha spójności zostaje zachowana bez uproszczeń. Pozostałe cechy tracą na ogół coś ze swojego charakteru, np. rezygnujemy z pełnej atomowości na poziomie sagi, zachowana jest ona tylko na poziomie podtransakcji. Wynika to z faktu, że może się zda-

* Ang. *ACID properties* – *Atomicity, Consistency, Isolation, Durability*.

rzyć, że po wykonaniu części sagi dojdzie do konieczności jej wycofania. Wtedy nie są wycofywane wykonane podtransakcje, lecz uruchamiane są dla nich transakcje kompensacyjne. Może się więc zdarzyć, że saga się zakończyła, ale tylko część z podtransakcji została wykonana. Oczywiście skutek powinien być taki sam, jakby saga nie była wcale wykonywana – powinien być zachowany ten sam stan danych. Zatem, jeśli spojrzeć na atomowość pod kątem stanu bazy, to można powiedzieć, że atomowość jest zachowana także na poziomie sagi, gdyż po jej zakończeniu albo jest wykonana w całości, albo stan bazy jest taki, jak przed jej rozpoczęciem (przy założeniu, że żadna współbieżna transakcja nie zmieniła tego stanu).

W związku z tym, że sagi są transakcjami zagnieżdżonymi typu otwartego nie ma zapewnionej izolacji na poziomie sagi. Izolacja występuje na poziomie podtransakcji. Zmiany wprowadzone przez podtransakcję są od razu po jej zatwierdzeniu widoczne dla innych transakcji. Zatem mogą pojawić się wszystkie omawiane anomalie wynikające ze współbieżnego wykonania nieizolowanych transakcji.

Podobnie jak na atomowość także i na trwałość danych można spojrzeć na dwa sposoby. Patrząc przez pryzmat operacji jakie zostały wykonane można powiedzieć, że modyfikacje wprowadzone przez podtransakcje są trwałe, gdyż podtransakcja jest zatwierdzana i dane zostają zapisane. Jednak, jeśli spojrzymy na stan bazy danych, to się może okazać, że modyfikacje wprowadzone przez zatwierdzoną transakcję nie będą widoczne, gdyż została wykonana transakcja kompensacyjna, która przywróciła stan sprzed wykonania danej podtransakcji. Na poziomie sagi jak najbardziej trwałość danych jest zachowana. Modyfikacje wprowadzone przez zatwierdzoną sagę pozostają trwałe w bazie danych.

Poza merytorycznymi rozważaniami na temat znaczenia atomowości i trwałości sagi należy zwrócić uwagę na bardzo ważne konsekwencje, jakie wynikają z braku izolacji pomiędzy sagą a innymi transakcjami. Brak izolacji powoduje opisane wcześniej kłopoty z wykonaniem transakcji kompensacyjnych, gdyż transakcja kompensacyjna może działać na innym stanie bazy, niż był przy zakończeniu danej podtransakcji, która ma zostać wycofana. Także w przypadku długich transakcji z udziałem aktywnym użytkownika można wykorzystać sagi. W przypadku wycofania sagi, transakcje kompensacyjne dają możliwość poinformowania użytkownika, jakie akcje są podejmowane. Należy poinformować użytkownika, które przeprowadzone przez niego operacje nie będą odzwierciedlone w bazie danych.

5. METODA ZNACZNIKÓW CZASOWYCH W ZARZĄDZANIU DŁUGIMI TRANSAKCJAMI

Metoda znaczników czasowych została zaproponowana do obsługi długich transakcji jako metoda nie wykorzystująca blokowania odstępu do danych [Bern 1987, GaUIWi 2003].

Znacznik czasowy (stempel) *ts* (ang. *time stamp*) jest przypisywany do transakcji. Ciąg znaczników czasowych pozwala określić porządek, w jakim powinny być wykonane transakcje w systemie. Pozwalają na tworzenie uszeregowanych historii przetwarzania transakcji. Znacznikiem czasowym transakcji oznaczane są wszystkie operacje wykonane na danych przez każdą z transakcji.

W dalszej części rozdziału będzie używane pojęcie operacji konfliktowych. Operacje konfliktowe są to takie dwie operacje, których kolejność wykonania jest istotna czyli wykonanie ich w odwrotnej kolejności spowoduje, że przynajmniej jedna z nich da inny wynik. Aby można było mówić o konflikcie operacji, obie muszą działać na tych samych danych. Dwie operacje odczytu danej nie są konfliktowe.

Przetwarzanie transakcji w oparciu o znaczniki czasowe działa zgodnie z regułą, że dla każdej pary konfliktowych operacji na danych, operacje te zostaną wykonane według kolejności wyznaczonej przez znaczniki czasowe, czyli według kolejności w jakiej pojawiły się w systemie zarządzania bazą danych transakcje, do których należą te operacje. Reguła ta określa w jakiej kolejności powinny być wykonane operacje jednak nie określa jak należy postępować, aby zagwarantować taką kolejność. Nie ma problemów z zastosowaniem tej reguły w przypadku, gdy znane są wszystkie operacje jakie będą wykonywane w ramach wszystkich współbieżnych transakcji oraz wiadomo na jakich danych będą operowały. W rzeczywistości zazwyczaj jednak system zarządzania bazą danych nie posiada takiej wiedzy i może się zdarzyć, że pojawi się sytuacja, w której na danych została już wykonana pewna operacja O_1 i pojawia się konfliktowa operacja O_2 , której nie można wykonać. W tego typu sytuacjach można wybrać jedną z metod:

- metodę prostą,
- metodę ścisłą,
- metodę zachowawczą.

Metoda prosta

W metodzie prostej (ang. *basic time stamp ordering*) operacje na danych wykonywane są w kolejności pojawiania się. W przypadku wystąpienia sytuacji operacji konfliktowej jest ona odrzucana. Wynika to, z faktu, że nie można wykonać i -tej transakcji tak, aby historia wykonania transakcji była historią uszeregowaną. Skoro transakcja nie może być wykonana, więc zostaje wycofana i rozpoczęta na nowo. Przy ponownym rozpoczęciu transakcji generowany jest nowy znacznik czasowy, gdyż wykonanie tej transakcji z poprzednim znacznikiem czasowym przyniosłoby konflikt w dokładnie tym samym miejscu.

Likwidacja możliwości wystąpienia tego samego konfliktu likwiduje możliwość wystąpienia blokady pomiędzy współbieżnymi transakcjami. Jednak pozostaje problem ewentualnego zagłodzenia transakcji. Generowanie za każdym razem nowego znacznika czasowego dla transakcji powoduje, że za każdym razem może istnieć transakcja wykonująca operację konfliktową.

Metoda prosta znaczników czasowych ma pewną wadę, którą można wyeliminować na poziomie śledzenia kolejności wykonania operacji na danych stosując metodę ścisłą znaczników czasowych.

Metoda ścisła

Przedstawiona metoda prosta gwarantuje, że historia wykonania transakcji będzie historią uszeregowaną. Jednak może się okazać, że nie będzie to historia umożliwiająca wycofanie transakcji. Oto przykład takiej historii:

$$H = (w_1[y], r_1[x], w_1[x], r_2[x], w_2[x], \text{COMMIT}(2), \text{ROLLBACK}(1)),$$

gdzie przyjęto, że znacznik czasowy transakcji T_i ma wartość i co zapisujemy jako $ts(T_i) = i$, $w_i[x]$ – oznacza operację zapisu danej x w transakcji T_i , $r_i[x]$ – oznacza operację odczytu danej x w transakcji T_i .

Jak widać żadne z operacji na danej x nie są z sobą w konflikcie. Historia jest uszeregowana, najpierw wykonują się wszystkie operacje z transakcji T_1 , a następnie operacje z transakcji T_2 . Jednak w przypadku takiej historii może dojść do nieprawidłowości. Jeśli zostanie wykonana operacja $\text{ROLLBACK}(1)$, to nie będzie widoczny wynik operacji $w_2[x]$, gdyż dana x powinna przyjąć wartość sprzed rozpoczęcia transakcji T_1 . W takim przypadku wynik działania transakcji nie jest trwały w bazie danych, naruszona jest cecha trwałości transakcji. Jeśli natomiast nie zostanie wykonany $\text{ROLLBACK}(1)$, to w bazie danych może dojść do niespójności danych, np. dana y będzie miała nieprawidłową wartość.

Aby zapobiec takim sytuacjom, wprowadza się pewną modyfikację do algorytmu odpowiedzialnego za przekazywanie operacji przeznaczonych do wykonania. Jest to **metoda ścisła znaczników czasowych** (ang. *strict time stamp ordering*), która polega na tym, że licznik przechowujący liczbę operacji zapisu oznaczany jako $w_in_transit[x]$ nie jest zerowany po zakończeniu operacji zapisu, ale dopiero po zakończeniu transakcji (niezależnie od tego czy jest to operacja COMMIT , czy ROLLBACK). Zatem każda operacja zakończenia transakcji powoduje wyzerowanie liczników $w_in_transit[x]$ dla wszystkich danych x , które były modyfikowane w ramach kończącej transakcji. Można to zinterpretować następująco: blokada na danej x założona przez zapis tej danej przez operację należącą do transakcji T_i będzie zwolniona dopiero, gdy transakcja ta zostanie zatwierdzona bądź wycofana. Takie postępowanie, oprócz wykonania operacji zapisu w kolejności wynikającej z wartości znaczników czasowych transakcji, gwarantuje że modyfikacje danych będą zatwierdzane także w kolejności wynikającej z wartości znaczników czasowych.

Metoda zachowawcza

Jedną z największych wad metody znaczników czasowych przedstawionych do tej pory jest konieczność wycofania transakcji. Jeśli transakcja ma wykonać operację, która jest konfliktowa z operacją, która została już wykonana i należy do transakcji, która była rozpoczęta później, to należy odrzucić wykonanie takiej operacji i wycofać całą transakcję, do której ta operacja należy. Wycofanie takie może być bardzo kosz-

towne, zużywać zasoby systemu komputerowego. Dodatkowo istnieje ryzyko tego, że transakcja nigdy się nie zakończy, gdyż za każdym razem ma ona przyznawany nowy znacznik czasowy. Dlatego też należy maksymalnie ograniczyć liczbę wycofywanych transakcji.

Metoda zachowawcza znaczników czasowych (ang. *conservative time stamp ordering*) polega na zmniejszeniu liczby wycofywanych transakcji przez opóźnienie wykonania operacji. Operacja nie jest obsługiwana bezzwłocznie, lecz jest ona wstrzymywana na jakiś okres czasu. Jeśli w czasie opóźnienia operacji $o_j[x]$, pojawi się konfliktowa operacja $p_i[x]$, taka, że $ts(T_i) < ts(T_j)$, to operacje te nie zostaną wykonane w kolejności w jakiej się pojawiły w systemie zarządzania bazą danych, lecz w kolejności wynikającej z uszeregowania operacji. Zatem najpierw wykonana będzie operacja $p_i[x]$, a następnie $o_j[x]$, co pozwoli uniknąć konfliktu i wycofania transakcji T_i .

Jednak opóźnianie operacji ma także swoje słabe strony. Powoduje spowolnienie wykonania transakcji, będą zdarzały się odcinki czasu, kiedy system zarządzania bazą danych będzie bezczynny, chociaż będą operacje do wykonania. Dlatego też należy wybrać optymalny czas opóźnienia operacji, który spowoduje zmniejszenia liczby transakcji wycofywanych i rozpoczynanych ponownie, ale nie będzie powodował zbyt dużego spowolnienia przetwarzania transakcji.

Korzystając z opóźnienia wykonania operacji można sterować wykonaniem operacji, tak, aby żadna z transakcji nie była wycofywana z powodu próby wykonania operacji konfliktowej. Jest to metoda skrajnie zachowawcza (ang. *ultimate conservative time stamp ordering*).

Tylko prosta metoda znaczników czasowych może nadawać się do wykonania długich transakcji. Zastosowanie metody prostej wiąże się jednak z pewnymi ograniczeniami.

Rozważmy sytuację, gdy system zarządzania bazą danych obsługuje dwie transakcje, T_1 oraz T_2 , gdzie transakcja T_1 jest transakcją długą (trwającą znacznie dłużej niż T_2). Biorąc pod uwagę kolejność rozpoczęcia tych transakcji oraz kolejność wykonania operacji na danej x możemy wyróżnić 4 przypadki. Jeśli $ts(T_1) > ts(T_2)$, czyli gdy długa transakcja rozpoczyna się podczas trwania zwykłej transakcji, oraz operacje wykonywane są w kolejności $w_2[x]$, $w_1[x]$, to operacje pomimo tego, że są konfliktowe wykonają się bez problemów. Nie ma potrzeby wycofywania żadnej transakcji, gdyż operacje wykonane są zgodnie z kolejnością wynikającą ze znaczników czasowych. Jeśli natomiast kolejność wykonania operacji będzie odwrotna, czyli $w_1[x]$, $w_2[x]$, to będzie wymagane wycofanie transakcji T_2 . Jest to sytuacja normalna przy tej metodzie.

Jeśli natomiast podczas długiej transakcji zostanie rozpoczęta zwykła transakcja ($ts(T_1) < ts(T_2)$), to wykonanie sekwencji operacji $w_1[x]$, $w_2[x]$, nie będzie stanowiło żadnego problemu. Obie operacje wykonają się prawidłowo. W przypadku, gdy kolejność będzie odwrotna, czyli $w_2[x]$, $w_1[x]$, to zgodnie z zasadami działania metody znaczników czasowych operacja $w_1[x]$ powinna zostać odrzucona, a transakcja T_1 , wycofana i uruchomiona ponownie. Jednak jest to długa transakcja i jej wycofanie

i ponowne wykonanie może być niepożądane. Należy się zatem spodziewać, że nie zawsze będzie to skuteczna metoda przy przetwarzaniu długich transakcji. Zauważmy, że nie nastąpiło jednak blokowanie dostępu do danych, krótka transakcja mogła się zakończyć w sposób normalny, bez opóźnienia. Oczywiście problem jest analogiczny w przypadku dwóch współbieżnych długich transakcji. Z tym, że wtedy wycofanie którejkolwiek transakcji wiąże się z wycofaniem długiej transakcji.

Zatem przy zastosowaniu metody znaczników czasowych problem występuje, gdy dochodzi do sytuacji wycofania długiej transakcji. Czyli metoda ta będzie się nadawała do obsługi długich transakcji, które nie są złożone obliczeniowo. Wtedy wycofanie transakcji i jej ponowne wykonanie nie będzie miało większego wpływu na przebieg transakcji i nie będzie to w sposób znaczący obciążało systemu zarządzania komputerowego obsługującego tą transakcję.

Kolejnym zastosowaniem mogą być długie transakcje, których zakończenie nie jest wymagane w jakimś określonym czasie. Wtedy można pozwolić, aby próba wykonania długiej transakcji była podejmowana wielokrotnie, aż zostanie zakończona powodzeniem. Przy tego typu rozwiązaniach należy określić jakie inne transakcje są możliwe w systemie, które mogą spowodować konieczność wycofania i rozpoczęcia na nowo długiej transakcji. Należy następnie określić częstotliwość wywołania tych transakcji i oszacować prawdopodobieństwo, że transakcja będzie jednak wykonana.

Problemem przy obsłudze długich transakcji za pomocą metody znaczników czasowych jest obciążenie systemu, który wykonuje tą transakcję, gdyż wielokrotne próby wykonania transakcji obciążają system i mogą powodować opóźnienia w wykonywaniu innych zadań. Przykładem takiego zastosowania może być transakcja, która okresowo modyfikuje ustawienia kont użytkowników (np. po zmianie regulaminu obsługi kont). Jeśli żaden użytkownik nie będzie modyfikował ustawień podczas działania transakcji, to zakończy się ona poprawnie. Jeśli któryś z użytkowników zmodyfikuje swoje ustawienia zanim zrobiła to transakcja aktualizująca, to transakcja aktualizująca dane zostanie wycofana i rozpoczęta na nowo. Jeśli użytkownicy rzadko modyfikują ustawienia swoich kont w porównaniu do długości trwania transakcji aktualizującej, to prawdopodobieństwo, że transakcja się nie powiedzie jest niewielkie. Biorąc pod uwagę, że wywoływanie transakcji aktualizacyjnej jest ponawiane, aż do skutku, można być pewnym, że aktualizacja będzie wykonana, chociaż nie można określić dokładnie kiedy.

Natomiast przeciwwskazaniem do wykorzystania z tej metody do obsługi długich transakcji jest czynny udział użytkownika w działaniu transakcji. Jeśli użytkownik ma podawać w trakcie trwania transakcji jakieś wartości niezbędne do jej działania, to w przypadku wycofania transakcji powinien je podać ponownie, co może być powodem różnorodnych problemów, zwłaszcza przy kilkukrotnym niepowodzeniu wykonania transakcji.

Problemem przy korzystaniu z metody znaczników czasowych do obsługi transakcji jest konieczność przechowywania dla każdej danej wartości znaczników $\text{max_ts_r}[x]$ oraz $\text{max_ts_w}[x]$ (znacznik maksymalny dla odczytu i zapisu). W związku z możliwą

dużą liczbą danych zarządzanie wszystkimi znacznikami może być złożone. Dlatego też nie są przechowywane wszystkie znaczniki czasowe, lecz jedynie znaczniki wygenerowane później niż określony znacznik ts_{min} . Okresowo usuwane są znaczniki czasowe wygenerowane wcześniej niż ts_{min} . W przypadku konieczności porównania $ts(T_i)$ ze znacznikiem $max_ts_r[x]$ lub $max_ts_w[x]$ wyszukiwana jest wartość spośród przechowywanych znaczników. Jeśli nie ma określonej wartości, to przyjmowana jest wartość ts_{min} . Jeśli $ts_{min} < ts(T_i)$, to znaczy, że przechowywanie dokładnej wartości nie było konieczne do tej operacji. W przeciwnym wypadku pojawia się wątpliwość, czy rzeczywiście odrzucenie operacji powinno być wykonane. Oczywiście operacja jest odrzucana i transakcja jest wycofywana, choć może się zdarzyć, że jest to zbyteczne. Dlatego też ts_{min} należy dobrać tak, aby sporadycznie zdarzała się sytuacja, kiedy $ts_{min} > ts(T_i)$. Przyjmuje się zatem wartość $t - \delta$, gdzie t to aktualny czas operacji usuwającej znaczniki czasowe, a δ to czas trwania najdłuższej transakcji.

Jednak w przypadku długich transakcji metoda taka jest nieskuteczna z dwóch powodów. Po pierwsze znacznik czasowy ts_{min} , byłby zazwyczaj tak mały w porównaniu do większości znaczników czasowych aktualnie przetwarzanych transakcji, że prawdopodobnie mała część znaczników $max_ts_r[x]$, $max_ts_w[x]$ mogła by być usunięta. Większość z nich musiałaby być i tak przechowywana. Po drugie usuwanie części znaczników może być niekorzystne, gdyż może prowadzić do niepotrzebnego wycofywania transakcji, co zwłaszcza w przypadku długich transakcji jest mocno niewskazane. Dlatego też w przypadku zastosowania metody znaczników czasowych do zarządzania długimi transakcjami należy przechowywać znaczniki czasowe dla wszystkich danych (co powoduje dodatkowe obciążenie systemu i wymaga dodatkowej pamięci).

6. WŁASNOŚCI ASIT W METODZIE ZNACZNIKÓW CZASOWYCH

Własności ASIT transakcji przetwarzanych przy wykorzystaniu metody znaczników czasowych, zależą od zastosowanego wariantu tej metody. Oczywiście najbardziej interesujące są właściwości przy metodzie prostej, która jako jedyna daje możliwość obsługi transakcji o długim czasie trwania. Dlatego też uwaga będzie skupiona na własnościach tej metody.

Atomowość gwarantuje, że transakcja będzie wykonana w całości, albo w ogóle nie będzie efektów jej działania. Metoda prosta daje możliwość wycofania transakcji, bądź jej zatwierdzenia, zatem atomowość jest gwarantowana.

Problem pojawia się w przypadku konieczności wycofania transakcji, która operowała na danych modyfikowanych przez inną zatwierdzoną już transakcję. Rozważmy historię prezentującą ten problem:

$H = (w_1[y], r_1[x], w_1[x], r_2[x], w_2[x], COMMIT(2), ROLLBACK(1)).$

Jaką wartość powinna mieć dana x ? Z jednej strony transakcja T_2 została zatwierdzona i dana x powinna mieć wartość wynikającą z operacji $w_2[x]$. Z drugiej strony operacja $w_1[x]$, należąca do wycofywanej transakcji T_1 , powinna nie mieć wpływu na stan bazy, zatem dana x powinna mieć wartość sprzed wykonania tej operacji. Są dwa wyjścia. Można wycofać zmiany danej x (także tą wprowadzoną przez T_2) i wtedy pojawia się problem utraty trwałości. Dzieje się tak, gdyż nie będzie widoczna modyfikacja wprowadzona przez zatwierdzoną transakcję. Innym rozwiązaniem jest pozostawienie wartości danej x bez zmian, czyli tak jak ją ustaliła transakcja T_2 . Wtedy nie mamy zagwarantowanej spójności danych. Problem ten można zobrazować następującym przykładem. Jeśli transakcja T_2 przypisuje danej x wartość y , to po wycofaniu transakcji T_1 dana x pomimo wykonania T_2 będzie miała wartość różną od wartości danej y . Zatem metoda prosta znaczników czasowych nie pozwala zachować jednocześnie obu cech transakcji jakimi są spójność i trwałość.

Ostatnią cechą do omówienia jest izolacja. Ponieważ historie wykonania współbieżnych transakcji są historiami uszeregowanymi a więc ich wykonanie jest równoważne z sekwencyjnym wykonaniem transakcji. Zatem izolacja w przypadku metody prostej powinna być zachowana. Jednak jest ona zachowana tylko wtedy, gdy patrzymy na transakcje z poziomu ich historii wykonania. Jeśli spojrzymy na problem globalnie, biorąc pod uwagę stan bazy, to nie jest to już takie oczywiste. Chodzi mianowicie o przyznawanie transakcjom nowych znaczników czasowych w przypadku wystąpienia konfliktu. W związku z takim postępowaniem może się okazać, że wynik działania współbieżnych transakcji może nie zgadzać się z sekwencyjnym wykonaniem transakcji, pomimo tego, że historia wykonania transakcji będzie uszeregowana. Jest to możliwe, gdyż transakcja, która została jako pierwsza, nie musi w efekcie końcowym być widziana jako transakcja, która została przyjęta do realizacji jako pierwsza. Stanie się tak, gdy zostanie ona wycofana i rozpoczęta ponownie z nowym znacznikiem czasowym. Co więcej może się okazać, że wykonanie współbieżnych transakcji może za każdym razem dać inny efekt, gdyż wystąpienie konfliktu pomiędzy transakcjami zależy od kolejności operacji, a ta może być za każdym razem inna.

7. ALGORYTM WIELOWERSYJNY W ZARZĄDZANIU DŁUGIMI TRANSAKCJAMI

W przypadku zastosowania algorytmów wielowersyjnych [Batt 2002, Bern 1987, BuSi 1993, WeSc 1992] każda z przechowywanych danych może występować w kilku wersjach. Nowe wersje danych powstają w przypadku zapisu danej przez transakcję. Zarządzanie transakcjami sprowadza się nie tylko do dbania o kolejność wykonywania operacji ale również do zapewnienia, by odczyt i zapis danych dotyczył odpowiedniej wersji. Poszczególne wersje danych oznaczone są przez transakcję, która jako ostatnia dokonała zapisu tej wersji. W metodzie tej z każdą transakcją związany jest znacznik czasowy (stempel), który pozwala na oznaczanie poszczególnych wersji.

Jeśli chodzi o tradycyjne systemy zarządzania bazami danych, to poszczególne wersje nie powinny być widoczne na zewnątrz bazy danych. Wersje danych są tylko elementem implementacji zarządzania transakcjami, więc tylko wewnętrzne mechanizmy systemów zarządzania bazami danych mają do nich dostęp. Użytkownik widzi bazę danych w postaci tylko pojedynczych wersji danych. Są to wersje, które są widoczne dla transakcji wykonywanych przez tego użytkownika.

Są jednak systemy, w których użytkownik musi widzieć rozróżnienie na poszczególne wersje danych. Dodatkowo powinien on mieć możliwość modyfikacji dowolnie wybranej wersji danej. Takimi systemami są bazy danych, które odpowiadają za przechowywanie projektów. Wtedy użytkownik systemu musi widzieć wszystkie możliwe wersje projektu. Powinien więc mieć możliwość wybrania wersji projektu, nad którą chce pracować. Natomiast po skończeniu pracy może on zapisać efekt swojej pracy jako kolejną wersję projektu.

7.1. REALIZACJA ALGORYTMU WIELOWERSYJNEGO

W związku z tym, że może istnieć wiele wersji tej samej danej przyjmijmy następującą notację: zapis $w_3[x_2]$ należy rozumieć, że transakcja T_3 zapisuje wersję danej x , która poprzednio była zapisana przez transakcję T_2 .

Podstawową zaletą zastosowania wielowersyjnego algorytmu jest unikanie wycofania transakcji i rozpoczynania jej na nowo. Przykładem takiej sytuacji może być następująca historia:

$$H_1 = (w_1[y], w_1[x], r_2[x], w_2[x], r_1[x], \text{COMMIT}(2), \text{ROLLBACK}(1)),$$

gdzie $ts(T_1) < ts(T_2)$. Operacja $r_1[x]$ w takim przypadku nie powinna być wykonana, gdyż odczytuje ona wartość ustaloną przez transakcję T_2 . Operacja ta powinna być odrzucona, a transakcja T_1 powinna być wycofana i rozpoczęta ponownie. W przypadku wielowersyjnego zarządzania transakcjami nie będzie takiej potrzeby, gdyż pomimo tego, że transakcja T_2 dokonała zapisu danej x , w bazie istnieje wartość, która powinna być odczytana przez transakcję T_1 . W przypadku systemu wykorzystującego algorytm wielowersyjny powyższa historia (nazwijmy ją historią wielowersyjną) będzie miała następującą postać:

$$H_2 = (w_1[y_1], w_1[x_1], r_2[x_2], w_2[x_2], r_1[x_1], \text{COMMIT}(2), \text{ROLLBACK}(1)).$$

W przypadku zwykłych transakcji można łatwo określić, czy historia jest uszeregowana. Jednak, żeby można było powiedzieć to samo o historii wielowersyjnej, musi ona operować na identycznych danych jak odpowiadająca jej historia nie wielowersyjna. Aby można było powiedzieć, że transakcja operuje na danych identycznych należy porównać wszystkie odpowiadające sobie operacje z obu historii. Jeśli każda operacja $r_i[x_i]$ odczytuje wartość identyczną, jak odpowiadająca jej operacja $r_i[x]$ z historii nie wielowersyjnej oraz każda operacja $w_i[x_i]$ zapisuje identyczną wartość, jak

odpowiadająca jej operacja $r_i[x]$, to można powiedzieć, że utworzona historia wielowersyjna jest uszeregowana.

Zgodnie z tym, wszystkie operacje zapisu i odczytu z postaci $w_i[x]$ oraz $r_i[x]$ są odpowiednio transformowane do operacji $w_i[x_j]$ oraz $r_i[x_j]$. Odbywa się to zgodnie z poniższymi zasadami. Operacja odczytu $r_i[x]$ zostanie przekształcona do postaci $r_i[x_j]$, gdzie x_j jest wersją danej x , która została zapisana przez transakcję o największym znaczniku czasowym spośród znaczników transakcji mniejszych równych znacznikowi transakcji T_i . Inaczej mówiąc transakcja T_i odczyta tylko wersję, która została zapisana przez transakcję rozpoczętą przed transakcją T_i . Dokładnie będzie to wersja stworzona przez najstarszą (najpóźniej rozpoczętą) z tych transakcji, włączając w to oczywiście transakcję T_i .

W przypadku operacji zapisu $w_i[x]$ istnieją dwa warianty. Jeśli została już wykonana jakakolwiek operacja zapisu $r_j[x_k]$, taka że $ts(T_k) < ts(T_i) < ts(T_j)$, to operacja ta zostaje odrzucona, a transakcja wycofana i uruchomiona ponownie. W przeciwnym wypadku operacja zostaje przekształcona do postaci $w_i[x_i]$. Sprawdzenie istnienia operacji odczytu $r_j[x_k]$ służy wyeliminowaniu operacji konfliktowych. Zapobiega to wystąpieniu niespójności, gdyż warunki, jakie musi spełniać wspomniana operacja odczytu oznaczają, że nie mogło dojść do odczytu wersji starszej niż zapisywana wersja przez transakcję rozpoczętą później niż transakcja zapisująca daną. Brak sprawdzenia tego warunku powodowałby brak uszeregowania historii wykonania transakcji.

W przypadku wycofania transakcji usuwane są wszystkie wersje danych, które zostały zapisane przez wycofywaną transakcję. Jeśli chodzi natomiast o zatwierdzenie transakcji, to w celu zapewnienia możliwości wycofania transakcji, należy opóźnić do chwili zakończenia wszystkich transakcji, które zapisały dane czytane przez zatwierdzoną transakcję. Problem ten jest analogiczny, jak w przypadku metody prostej znaczników czasowych.

Problemem w przypadku metody wielowersyjnej może być narzut czasu wymagany na obsługę wielu wersji oraz rozmiar pamięci potrzebnej na przechowywanie wielu wersji. Jeśli chodzi o pierwszy problem, to większość systemów zarządzania bazami danych realizuje wycofanie transakcji poprzez zapamiętywanie wartości sprzed transakcji, czyli w pewnym sensie wersjonowanie też jest wykonywane (istnieją dwie wersje), należy je tylko rozszerzyć. Oczywiście pociąga to za sobą dodatkowe narzuty obliczeniowe. Rozwiązaniem problemu związanego z rozmiarem pamięci potrzebnej na przechowywanie kolejnych wersji danych jest czasowe usuwanie zbędnych wersji. Operacja ta będzie dodatkowym obciążeniem, gdyż oczywiście będzie wymagała zasobów systemowych. Oprócz tego, że podczas wycofania transakcji usuwane są wszystkie wersje utworzone przez wycofywaną transakcję, to można usunąć też wszystkie takie wersje x_i danej, dla których istnieje wersja x_j , taka, że $ts(T_i) < ts(T_j)$ oraz zakończone są już wszystkie transakcje o znaczniku czasowym mniejszym od $ts(T_j)$. Wynika to z faktu, że nie istnieje już aktywna transakcja, która mogłaby odczytać wartość x_i .

Jeśli chodzi o przydatność metody wielowersyjnej do zarządzania długimi transakcjami, to podobnie, jak w przypadku metod znaczników czasowych należy zrezygno-

wać z zagwarantowania możliwości odtwarzania transakcji. Związane jest, to z faktem, że zatwierdzenie transakcji, przy zagwarantowaniu możliwości jej wycofania, może zostać odroczone, aż do momentu zakończenia innych transakcji. Rozwiązanie takie jest nie dopuszczalne, gdyż operacja, na zakończenie której oczekuje transakcja zatwierdzania, może trwać długo.

Metoda ta, przez fakt, że w przypadku wystąpienia konfliktu powoduje wycofanie transakcji i rozpoczęcie jej na nowo, jest podobna do metody znaczników czasowych. Dlatego też wszystkie uwagi zawarte w punkcie dotyczącym przydatności metody znaczników czasowych do obsługi długich transakcji, które odnosiły się do problemów wynikających z wycofywania transakcji mają zastosowanie także w przypadku metody wielowersyjnej. Należy jednak podkreślić, że metoda ta jest znacznie bardziej przydatna, jeśli chodzi o obsługę współbieżnych transakcji. Jest to spowodowane, tym, że znacznie rzadziej wymagane będzie wycofanie transakcji spowodowane wystąpieniem konfliktu pomiędzy operacjami.

Zauważmy jeszcze, że w związku z podobieństwem metody wielowersyjnej do metody znaczników czasowych, także własności ASIT są analogiczne. Metoda wielowersyjna gwarantuje atomowość. Przez brak wymaganego oczekiwania na zakończenie innych transakcji przy zatwierdzaniu transakcji, jedna z cech trwałość lub spójność będzie musiała zostać niezrealizowana w pełni. Przyznawanie transakcji, przy każdej próbie jej wykonania, nowego znacznika czasowego ma swoje konsekwencje dla izolacji transakcji. Są one identyczne, jak dla metody znaczników czasowych.

5. PODSUMOWANIE

W pracy przedstawiono trzy metody umożliwiające obsługę długich transakcji. Jednak żadna z nich nie gwarantuje zachowania wszystkich cech ASIT transakcji. Dlatego też należy, decydując się na wykorzystanie któregoś z rozwiązań, mieć świadomość wiążących się z nim ograniczeń. Trzeba przeanalizować mogące pojawić się kłopotów oraz możliwe anomalie.

Nie wszystkie metody będą możliwe do zaimplementowania dla operacji występujących w danym systemie. Na przykład może się zdarzyć, że nie można określić transakcji kompensacyjnej dla danej transakcji. Dodatkowo nie wszystkie metody mogą być skuteczne w danym systemie. Mogą przykładowo stwarzać zagrożenie naruszenia istniejących w bazie danych ograniczeń.

Wykonana analiza właściwości i ograniczeń poszczególnych rozwiązań powinna pomóc w określeniu ich przydatności w konkretnym zastosowaniu. Opis możliwych do wystąpienia problemów i anomalii pozwala na podjęcie próby zapobieżenia im lub ewentualnego pogodzenia się z ich występowaniem.

Analiza sposobów realizacji długich transakcji, pokazuje, że transakcje takie muszą być obsługiwane w inny sposób, niż tradycyjne, krótkie transakcje. Niedopuszczalne jest wykorzystanie najczęściej stosowanej metody tzn. blokowania dostępu do danych.

A inne metody (wielowersyjna oraz znaczników czasowych) nie mogą być stosowane w ich wszystkich wariantach, ale tylko w niektórych.

Transakcje były zawsze oceniane poprzez cechy ASIT. Jak zostało zaprezentowane w artykule, obsługa długich transakcji zawsze wiąże się z utratą pewnych cech ASIT. Czasami nie jest to utrata całkowicie cechy, ale częściowa. Tak jest np. z izolacją podtransakcji w przypadku sag. Izolacja zresztą jest tą cechą, która jest najbardziej kłopotliwa w przypadku długich transakcji. Jeśli nie można zaakceptować rezygnacji z pełnej izolacji, to pojawia się konieczność zastosowania blokad i nie ma wówczas możliwości wykonania długich transakcji w sposób, który pozwoliłby na współbieżne wykorzystanie danych.

LITERATURA

- [Batt 2002] Batty P. M., *Version management revisited*, Englewood 2002.
- [Bern 1987] Bernstein P. A., *Concurrency control and recovery in database system*, Reading 1987.
- [BuSi 1993] Buckley G. N., Silberschatz A., *Obtaining Progressive Protocols for a Simple Multiversion Database Model*, Austin 1993.
- [CoBe 2004] Connolly T., Begg C., *Systemy baz danych - projektowanie, wdrażanie i zarządzanie w praktyce*. Tom 2, RM, Warszawa, 2004.
- [GaUIWi 2003] Garcia-Molina H., Ullman J. D., Widom J., *Implementacja systemów baz danych*, Warszawa 2003.
- [GoGh 1999] Gore M. M., Ghosh R. K., *Recovery in Distributed Extended Long-lived Transaction Models*, Kanpur 1999.
- [KiBeLe 2003] Kifer M., Bernstein A., Lewis P.M., *Database Systems: An Application Oriented Approach*, Addison 2003.
- [Kosz 2000] Koszłajda T., *Zarządzanie współbieżnością transakcji*, VI Konferencja PLOUG, Zakopane, 2000.
- [WeSc 1992] Weikum G., Schek H. J., *Concepts and Applications of Multilevel Transactions and Open Nested Transactions*, Zurych 1992.

A LONG LIVE TRANSACTIONS MANAGEMENT

This research applies long lived transactions in database domain. It introduces the definition the long lived transaction and presents the problems which occur in the systems supporting such transactions. As a respond to the problem presented in the research it displays the methods how operate long live transactions, which make it possible to diminish or omit the problems mentioned.

Zbigniew FRYŻLEWICZ*
Konrad KUNICKI

WSTĘP DO WS-POLICY

Artykuł zawiera omówienie standardu *Web Services Policy Framework (WS-Policy)*, który dostarcza model oraz syntaktykę umożliwiającą definiowanie polis dla usług webowych (*Web Services*). Częściowa implementacja tego standardu znajduje się w *Web Services Enhancements 2.0* i artykuł dotyczy głównie tej implementacji.

1. WPROWADZENIE

WSE 2.0 (*Web Services Enhancements 2.0*) [HOR04, WSE] to rozszerzenie dostarczające dodatkowej funkcjonalności do usług webowych (*Web Services*) [CUR05]. Rozszerzenie jest zaimplementowane w postaci bibliotek zintegrowanych ze środowiskiem programistycznym Visual Studio .NET 2003. WSE 2.0 obejmuje implementacje następujących standardów: *WS-Security*, *WS-SecurityPolicy*, *WS-Trust*, *WS-SecureConversation*, *WS-Policy*, *WS-Referral*, *WS-Addressing*, *DIME*, *WS-Attachments*.

Nad tymi oraz wieloma innymi standardami czuwa *Web Services Interoperability Organization (WS-I)* [WSI]. Organizacja zrzesza zarówno największych producentów oprogramowania związanych z usługami webowymi, jak i małe firmy programistyczne i użytkowników zainteresowanych współdziałaniem technologii opartych o usługi webowe. WS-I, poza opracowywaniem kolejnych standardów, gwarantuje, że produkty wytworzone przez różnych producentów korzystających ze standardu będą ze sobą współpracować.

Na razie większość specyfikacji wytworzonych przez WS-I istnieje jedynie na papierze. Wyjątkiem są specyfikacje dotyczące bezpieczeństwa. Dużą ich część dostarcza już WSE 2.0 i jest ona zaimplementowana w wymienionym środowisku programistycznym. Wspierane jest m.in. szyfrowanie i integralność przesyłanych wiadomości, różne sposoby zabezpieczeń, zarządzanie tokenami, mechanizmy uwierzytelniania oparte na rolach oraz zintegrowane z Windows.

* Instytut Informatyki Stosowanej, Wydział Informatyki i Zarządzania Politechniki Wrocławskiej, 50-370 Wrocław, Skwer Idaszewskiego 1, zbigniew.fryzlewicz@pwr.wroc.pl, konrad.kunicki@student.pwr.wroc.pl

2. PLIK KONFIGURACYJNY WS-POLICY

WS-Policy jest częścią WSE 2.0 i pozwala w deklaracyjny sposób określić wymagania dotyczące podpisywania oraz szyfrowania wychodzących i przychodzących wiadomości SOAP. Również pozwala określić rodzaj stosowanych zabezpieczeń, np. bilety systemu Kerberos, certyfikaty standardu X.509 etc.

Przed pojawieniem się WS-Policy, tworzenie usługi webowej korzystającej z WS-Security oznaczało konieczność napisania kodu sprawdzającego, czy oczekiwane tokeny występują w przesyłanej wiadomości i czy wiadomość jest podpisana oraz zaszyfrowana w oczekiwany sposób. WS-Policy umożliwia osiągnięcie tego samego celu w dużo prostszy sposób. Wystarczy utworzyć plik konfiguracyjny w formacie XML, który zawiera wszystkie wymagania dotyczące przesyłanych wiadomości. Biblioteczne programy implementujące WSE 2.0 zweryfikują zadeklarowane wymagania i w przypadku ich niespełnienia wygenerują stosowny wyjątek.

W pliku konfiguracyjnym może występować wiele polis, czyli zasad dotyczących przesyłania komunikatów. Mogą być one zdefiniowane dla trzech różnych typów wiadomości: żądania (*request*), odpowiedzi (*response*) oraz błędu (*fault*). Często są wzajemnie powiązane, np. polisa może wymagać, aby żądanie było podpisane cyfrowo certyfikatem X.509, który następnie posłuży do zaszyfrowania odpowiedzi.

Listing 1 przedstawia strukturę pliku XML zawierającego wymagania dotyczące przesyłania wiadomości SOAP.

Głównym elementem pliku konfiguracyjnego jest `policyDocument` pochodzący z przestrzeni nazw <http://microsoft.com/wse/2003/06/PolicyDocument>. Element `policyDocument` dzieli się na dwie główne sekcje: `mappings` i `policies`. Sekcja `mappings` zawiera jeden lub więcej elementów `endpoint`, czyli węzłów końcowych, zawierających adres URI do usługi webowej i identyfikatory nałożonych na nią polis. Natomiast sekcja `policies` zawiera jeden lub więcej elementów `Policy`, w których zdefiniowano wymagania nałożone na przesyłane wiadomości SOAP.

Główne elementy w deklaracji `Policy` wspierane przez WSE 2.0, to:

- *Integrity* (cyfrowy podpis),
- *Confidentiality* (szyfrowanie),
- *MessagePredicate* (wykaz obligatoryjnych elementów XML w wiadomości SOAP),
- *MessageAge* (czas wygaśnięcia wiadomości),
- *SecurityToken* (wymagany token bezpieczeństwa).

Są to oczywiście tylko wybrane elementy główne, ale dobrze ilustrują idee zawarte w WS-Policy i WSE 2.0.

```
<policyDocument
xmlns="http://schemas.microsoft.com/wse/2003/06/Policy">
  <mappings>
    <endpoint uri="http://www.develop.com/someapp/someservice.asmx">
      <operation requestAction=
```

```

        "http://www.develop.com/someapp/someservice/someop">
        <request policy="#policy1" />
        <response policy="#policy2"/>
        <fault policy="#policy3"/>
    </operation>

    <defaultOperation>
        <request policy="#policy4" />
        <response policy="#policy5" />
        <request fault="#policy6" />
    </defaultOperation>
</endpoint>
</mappings>

<policies
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
              oasis-200401-wss-wssecurity-utility-1.0.xsd"
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy">

  <wsp:Policy wsu:Id="policy1">
    <!--tu należy wstawić asercje polityki 1 -->
  </wsp:Policy>

  <wsp:Policy wsu:Id="policy2">
    <!--tu należy wstawić asercje polityki 2 -->
  </wsp:Policy>

  <!--pozostałe polityki pominięto ... -->

</policies>
</policyDocument>

```

Listing 1. Plik konfiguracyjny – fragment wymagań
dotyczących przesyłania wiadomości SOAP

3. PRZYKŁAD WYKORZYSTANIA MECHANIZMU WS-POLICY

Aby wykorzystać mechanizm WS-Policy w projektowanej usłudze webowej, należy wcześniej zainstalować pakiet biblioteczny z implementacją WSE 2.0 [WSE].

Poniżej przedstawiono kolejne etapy budowy usługi webowej oraz aplikacji klienckiej wykorzystujących w komunikacji podstawowe mechanizmy bezpieczeństwa [HOR04, ROS04]. Przykład ma na celu ukazanie elastyczności mechanizmu WS-Policy i możliwości zaoszczędzenia pracy programisty dzięki WS-Policy.

Na początek należy stworzyć w VS.NET projekt z prostą usługą webową zawierającą metodę *HelloWorld()*. Kolejnym krokiem jest utworzenie aplikacji klienckiej, dodanie w niej referencji do wcześniej stworzonej usługi i wywołanie jej metody, np.

w obsłudze jakiegoś zdarzenia. Na tym etapie można już w pełni korzystać ze stworzonej usługi webowej. Niestety nie jest ona w żaden sposób zabezpieczona, czyli przechodząc przez sieć jest narażona na różnego typu ataki. Osoba niepowołana bez problemu może podsłuchać taką wiadomość, bądź przejąć ją i następnie podszywać się za serwer lub aplikację kliencką. Tego typu zagrożeń twórcy usług webowych muszą być świadomi i muszą im przeciwdziałać. Z pomocą w tego typu wyzwaniach przychodzi właśnie WSE 2.0 wraz z WS-Policy.

Mechanizm WS-Policy wymusza określone zabezpieczenia dotyczące przesyłanych wiadomości. Innymi słowy, doprecyzowuje kontrakt pomiędzy usługodawcą a usługobiorcą w sferze zabezpieczeń przesyłanych komunikatów.

Kolejne kroki pokażą, w jaki sposób można wykorzystać ten standard we wcześniej utworzonej usłudze. Chcemy, aby żądanie, czyli komunikat wychodzący od klienta był podpisany przy użyciu *X509SecurityToken*, natomiast usługa WWW, generując odpowiedź, wykorzystywała do podpisu *UsernameToken*.

Przyjmijmy, że budowana usługa webowa będzie się nazywała *PolicyService* a aplikacja kliencka *PolicyClient*, ułatwi to dalszy opis wykonywanych działań.

Po pierwsze należy rozszerzyć stworzoną usługę do WSE 2.0. Można to wykonać, klikając prawym przyciskiem myszy na projekcie *PolicyService* i następnie wybierając opcję „WSE Settings 2.0...”. Opcja jest dostępna dopiero po zainstalowaniu WSE 2.0. Po otwarciu formularza z ustawieniami WSE 2.0 na pierwszej zakładce pokażą się dwa pola typu *CheckBox*. Należy je oba zaznaczyć i zatwierdzić zmiany. Dzięki temu automatycznie zostanie dodana referencja do biblioteki *Microsoft.Web.Services2.dll*, która zawiera wszystkie klasy dostarczające funkcjonalność WSE 2.0 oraz zostanie dodany dodatkowy wpis w pliku *Web.config*.

Kolejnym etapem jest budowa i dodanie do usługi webowej pliku konfiguracyjnego WS-Policy. Niech ten plik ma nazwę *policeCache.config*. Podane działania można wykonać samodzielnie lub skorzystać z generatora znajdującego się w ustawieniach WSE na zakładce *Policy*.

Listing 2. przedstawia kod, który powinien zawierać plik konfiguracyjny spełniający wcześniej zdefiniowane założenia.

Na początku w sekcji *mappings* został zdefiniowany węzeł końcowy, czyli stworzona usługa *PolicyService* oraz zostały przypisane polisy dla jej operacji żądania i odpowiedzi. Następnie w sekcji *policies* zostały te polisy zdefiniowane. Pierwsza wymaga, aby wiadomość była podpisana przy użyciu *UsernameToken*, natomiast druga, aby wiadomość była podpisana wykorzystując *X509SecurityToken*.

```
<?xml version="1.0" encoding="utf-8"?>
<policyDocument
  xmlns="http://schemas.microsoft.com/wse/2003/06/Policy">
  <mappings>
    <endpoint uri="http://localhost/PolicyService/PolicyService.asmx">
      <defaultOperation>
```

```

        <request policy="#Sign-X.509" />
        <response policy="#Sign-Username" />
        <fault policy="" />
    </defaultOperation>
</endpoint>
</mappings>
<policies xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
            oasis-200401-wss-wssecurity-utility-1.0.xsd"
            xmlns:wssp="http://schemas.xmlsoap.org/ws/2002/12/secext"
            xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy">
    <!-- wiadomość będzie podpisana przy użyciu UsernameToken -->
    <wsp:Policy wsu:Id="Sign-Username">
        <wssp:Integrity wsp:Usage="wsp:Required">
            <wssp:TokenInfo>
                <wssp:SecurityToken>
                    <wssp:TokenType>http://docs.oasis-open.org/wss/2004/01/
                        oasis-200401-wss-username-token-profile-1.0#UsernameToken
                    </wssp:TokenType>
                </wssp:SecurityToken>
            </wssp:TokenInfo>
            <wssp:MessageParts Dialect=
                "http://schemas.xmlsoap.org/2002/12/wsse#part">wsp:Body()
            </wssp:MessageParts>
        </wssp:Integrity>
    </wsp:Policy>
    <!-- wiadomość będzie podpisana przy użyciu certyfikatu X509 -->
    <wsp:Policy wsu:Id="Sign-X.509">
        <wssp:Integrity wsp:Usage="wsp:Required">
            <wssp:TokenInfo>
                <wssp:SecurityToken>
                    <wssp:TokenType>http://docs.oasis-open.org/wss/2004/01/
                        oasis-200401-wss-x509-token-profile-1.0#X509v3
                    </wssp:TokenType>
                </wssp:SecurityToken>
            </wssp:TokenInfo>
            <wssp:MessageParts Dialect=
                "http://schemas.xmlsoap.org/2002/12/wsse#part">wsp:Body()
            </wssp:MessageParts>
        </wssp:Integrity>
    </wsp:Policy>
</policies>
</policyDocument>

```

Listing. 2. Plik konfiguracyjny z przykładowymi elementami WS-Policy

Mając plik konfiguracyjny, należy jeszcze sprawić, aby zaczął on obowiązywać. W tym celu należy zmodyfikować plik *Web.config* dodając do elementu *microsoft.web.services2* następujące elementy podrzędne:

```

<policy>
    <cache name="policyCache.config" />

```

```

</policy>
<security>
  <x509 allowTestRoot="true" allowRevocationUrlRetrieval="false"
    verifyTrust="false" storeLocation="LocalMachine"/>
</security>

```

Element *policy* łączy plik konfiguracyjny *policyCache.config* z usługą webową. Natomiast element *security* jest niezbędny ze względu na wykorzystanie do cyfrowego podpisu *X509SecurityToken*. Brak tego elementu spowodowałby wystąpienie błędu SOAP.

Tak zbudowana usługa webowa korzysta już z mechanizmu WS-Policy. Jeśli teraz zostałaby uruchomiona aplikacja klienta i próbowałaby skorzystać ze tej usługi, to WSE wygeneruje wyjątek, bo niespełnione są jeszcze wymagania zdefiniowane w pliku *policyCache.config*.

Aby można było korzystać z usługi *PolicyService*, należy jeszcze zaimplementować zdefiniowane mechanizmy bezpieczeństwa. Dlatego wymagana jest modyfikacja przykładowej metody *HelloWorld* tak, aby korzystała ona z podpisu *UsernameToken*. Modyfikacja jest przedstawiona poniżej.

[Kod C#]

```

[WebMethod]
public string HelloWorld()
{
    UsernameToken token = new UsernameToken("Username", "Password",
        PasswordOption.SendPlainText);
    ResponseSoapContext.Current.Security.Tokens.Add(token);
    ResponseSoapContext.Current.Security.Elements.Add(new
        MessageSignature(token));
    return "Hello World";
}

```

Należy jeszcze zmienić atrybuty *Username* i *Password* w konstruktorze *UsernameToken* na zgodne z danymi użytkownika na lokalnym komputerze (hasło w przykładzie jest przesyłane otwartym tekstem; można to łatwo zmienić, ale jest to poza kwestią omawianą w artykule). Po tym zabiegu, zbudowana usługa webowa spełnia już wszystkie postawione wcześniej wymagania. Pozostało jedynie dostosować aplikację klienta, ponieważ w pliku *Policy* określono, że żądanie musi być podpisane przy użyciu *X509SecurityToken*. Aby to wykonać, trzeba wcześniej posiadać jakiś certyfikat, przy pomocy którego będzie podpisywane żądanie. W omawianym przykładzie wykorzystano jeden z certyfikatów dołączonych do WSE 2.0. Proces instalacji certyfikatów jest opisany w dokumentacji [WSE].

Posiadając certyfikaty, można przejść do modyfikacji aplikacji klienckiej. Po pierwsze, klient również musi mieć włączone WSE 2.0. Wykonuje się to podobnie jak

w przypadku usługi webowej. Należy kliknąć prawym przyciskiem na projekt klienta i wybrać opcję „*WSE Settings 2.0...*”. Po otwarciu formatki zaznacza się dostępne pole typu *CheckBox*.

Następnie należy dodać odpowiednie dyrektywy do kodu głównego programu, zbudować metodę, zwracającą wymagany token i trochę zmodyfikować wywołanie metody udostępnianej przez usługę webową. Wymagane modyfikacje są następujące:

[Kod C#]

```
using Microsoft.Web.Services2.Security;
using Microsoft.Web.Services2.Security.Tokens;
using Microsoft.Web.Services2.Security.X509;
```

Metoda zwracająca token *X509SecurityToken* pobrany z *X509CertificateStore*:

[Kod C#]

```
private X509SecurityToken GetX509Token()
{
    // Klucz pochodzi z certyfikatu dołączonego do WSE 2.0
    string keyId = "bBwPfItvKp3b6TNDq+14qs58VJQ=";
    X509SecurityToken token = null;
    X509CertificateStore store = X509CertificateStore.
        CurrentUserStore(X509CertificateStore.MyStore);

    if (!store.OpenRead())
        return null;

    X509CertificateCollection certificates = sto-
re.FindCertificateByKeyIdentifier(Convert.FromBase64String(keyId));

    if (certificates.Count > 0)
        token = new X509SecurityToken(certificates[0]);
    return token;
}
```

Wywołanie usługi ma postać:

[Kod C#]

```
try
{
    PolicyService.PolicyServiceWse serviceProxy =
        new PolicyClient.PolicyService.PolicyServiceWse();
    X509SecurityToken token = GetX509Token();
    if (token == null)
        throw new Exception("Couldn't find X509 Certificate.");
    service-
Proxy.RequestSoapContext.Security.Tokens.Add(token);
```



```

        serviceProxy.RequestSoapContext.Security.Elements.Add(new
            MessageSignature(token));

        MessageBox.Show(serviceProxy.HelloWorld());
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.ToString());
    }
}

```

Od tego momentu wszystkie żądania generowane przez aplikację kliencką są już podpisane cyfrowo. Będą również spełnione zadeklarowane wymagania i klient powinien móc bez problemu korzystać z usługi *PolicyService*. Generalnie, podpisanie wiadomości polega na dodaniu uzyskanego *SecurityToken* do kolekcji *RequestSoapContext.Security.Tokens*, a następnie dołączeniu *MessageSignature*, bazującego na *SecurityToken*, do kolekcji *RequestSoapContext.Security.Elements*.

4. PODSUMOWANIE

WS-Policy to bardzo rozbudowany i elastyczny mechanizm automatyzujący i upraszczający budowę aplikacji opartych o usługi webowe. W przykładzie, definiując w pliku konfiguracyjnym wymagania na podpisy cyfrowe, wykorzystano tylko jedynie fragment tych możliwości. Inne zastosowania WS-Policy, to deklaracja dotycząca szyfrowania wiadomości, opcjonalnych i wymaganych elementów wiadomości SOAP, czasu ważności komunikatu etc. Należy podkreślić, że wszystkie te wymagania nie wymuszają napisania ani jednej linijki kodu i zawarte są jedynie w pliku konfiguracyjnym, który można łatwo modyfikować.

Uogólniając, WS-Policy dostarcza elastyczny i rozszerzalny język, umożliwiający wyrażenie możliwości, ograniczeń i ogólnych charakterystyk dotyczących wiadomości SOAP przesyłanych w systemach opartych na usługach webowych. Dodatkowo, definiuje architekturę i model do wyrażenia tych właściwości jako zasad (polis) w pliku konfiguracyjnym.

Dzięki wykorzystaniu WS-Policy, kod aplikacji jest bardziej przejrzysty, ponieważ definicja nałożonych zasad i wymagań oraz późniejsze sprawdzanie ich przestrzegania odbywa się poza aplikacją. W każdej chwili można zmienić te zasady nie ingerując w kod aplikacji, co pozwala na późniejszą ich modyfikację przez administratora aplikacji.

LITERATURA

- [CUR05] Curbera F., Leymann F., Storey T., Ferguson D.F., Weerawarana S., *Web Services Platform Architecture : SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall PTR 2005.

- [HAS04] Hasan J., *Expert Service-Oriented Architecture in C#: Using the Web Services Enhancements 2.0*. APRESS 2004.
- [HOR04] Horrell S., *Web Services Enhancements 2.0 Support for WS-Policy*.
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwse/html/wse2wspolicy.asp>.
- [KOP04] Kopacz T., *Świat jako usługa*. Computerworld nr 30-2004.
- [ROS04] Rosenberg J., Remy D.L., *Securing Web Services With WS-Security*. SAMS 2004.
- [SKO03] Skonnard A., *Understanding WS-Policy*.
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/understwsapol.asp>.
- [WSE] *Web Services Enhancements 2.0 Documentation*.
- [WSI] <http://www.ws-i.org/>
- [WSP] *Web Services Policy Framework Specification*.

AN INTRODUCTION TO WS-POLICY

The paper presents the Web Services Policy Framework (WS-Policy) which provides a general purpose model and corresponding syntax to describe and communicate the policies of a Web service. Partial implementation of this standard is provided by WSE 2.0 and the paper presents features of this implementations.

Adam PIECHOWIAK*

Artur WILCZEK**

STANDARD SQLJ

Organizacje standaryzacyjne ISO oraz ANSI opracowały metody zagnieżdżania instrukcji języka SQL (ang. *Structured Query Language*) w językach programowania takich jak FORTRAN, COBOL, C. Język SQL jest niekwestionowanym standardem języka zapytań relacyjnych baz danych. Swoją popularność SQL zawdzięcza głównie prostocie podstawowych konstrukcji języka, które pozwalają deklaratywnie formułować złożone zapytania. Wszystkie dostępne obecnie relacyjne systemy zarządzania bazami danych implementują język SQL. Język SQL nie jest językiem proceduralnym, nie jest wystarczający aby budować przyjazne użytkownikom aplikacje. Twórcy aplikacji stosując uniwersalne języki programowania mogą stosować różne technologie dostępu do danych przechowywanych w relacyjnej bazie danych. Konsekwencją stosowania dwóch różnych języków: języka programowania (np. Javy) i języka dostępu do danych (SQL) jest niekorzystny efekt określany jako niezgodność impedancji. Standard SQLJ (SQL for Java) jest próbą ograniczenia negatywnych konsekwencji wynikających z osadzania instrukcji języka SQL w aplikacjach języka Java.

1. WPROWADZENIE

Język Java jest obecnie jednym z najczęściej stosowanych do budowy aplikacji. Dzięki swojej przenaszalności, wydajności oraz bogactwu dostępnych, specjalizowanych bibliotek pozwala on programistom na budowę nowoczesnego oprogramowania współpracującego z różnymi systemami baz danych.

Powszechnie stosowaną technologią dostępu do danych aplikacji języka Java jest JDBC (ang. *Java Database Connectivity*). Standard JDBC jest interfejsem programistycznym (API) dostępnym w postaci klas języka Java. Producenci systemów baz danych mogą dostarczać swoje implementacje standardu w postaci własnych sterowników JDBC. Standard JDBC opisuje wymagane interfejsy klas reprezentujących np. instrukcję języka SQL lub połączenie implementujących bazą danych. Treść instrukcji SQL jest przekazywana odpowiednim metodom jako parametr teks-

* Instytut Informatyki Stosowanej, Wydział Informatyki i Zarządzania Politechniki Wrocławskiej, 50-370 Wrocław, Wybrzeże Wyspiańskiego 27, adam.piechowiak@e-informatyka.pl, student

** Instytut Informatyki Stosowanej, Wydział Informatyki i Zarządzania Politechniki Wrocławskiej, 50-370 Wrocław, Wybrzeże Wyspiańskiego 27, artur.wilczek@pwr.wroc.pl

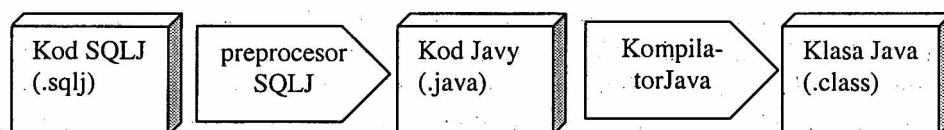
towy. Składnia SQL nie jest weryfikowana na etapie kompilacji programu i dlatego ewentualne błędy w instrukcjach SQL są wykrywane dopiero w trakcie działania programu. Tekstowe przekazywanie treści instrukcji SQL ma jednak pewną zaletę, a mianowicie umożliwia dynamiczne budowanie treści polecenia SQL w trakcie pracy programu.

Pierwotnie główną intencją twórców standardu SQLJ było zaproponowanie rozwiązania umożliwiającego weryfikację osadzonych instrukcji SQL na etapie kodu języka Java. Ostatecznie standard objął także zagadnienia związane z wykorzystaniem języka Java do tworzenia procedur składowanych bazy danych oraz definiowaniem abstrakcyjnych typów danych jako klas języka Java [EISEN]. Obecnie standard SQLJ składa się z trzech części:

- Część 0: (*SQLJ Part 0: Embedded SQL*) Stanowi specyfikację składni oraz semantyki zagnieżdżania kodu SQL w Javie. Potocznie ta część nazywana jest *Embedded SQL*.
- Część 1: (*SQLJ Part 1: Stored Procedures and UDFs*) Dotyczy wykorzystywania statycznych metod Java jako procedur i funkcji składowanych, jako alternatywy dla procedur tworzonych rozszerzeniach w proceduralnych rozszerzeniach języka SQL.
- Część 2: (*SQLJ Part 2: Data Types*) Definiuje rozszerzenia standardu SQL pozwalające na definiowanie abstrakcyjnych typów danych na podstawie klas Javy.

2. ZAGNIEŻDŻANIE KODU SQL W JAVIE

Standard SQLJ umożliwia zagnieżdżanie konstrukcji języka SQL w kodzie Javy, pozwalając na weryfikację poprawności zagnieżdżonych w programie instrukcji SQL na etapie kompilacji programu. Programista implementując aplikację wymagającą dostępu do bazy danych zawiera bezpośrednio w kodzie konstrukcje języka SQL. Taki kod zostanie później przetransformowany poprzez preprocesor SQLJ na w pełni przenaszalny kod Javy. Poniższy diagram obrazuje kolejne kroki tego procesu (rys. 1).



Rys. 1. Fazy kompilacji SQLJ [STEM]

W pierwszej fazie translator SQLJ parsuje plik sqlj: sprawdza poprawność gramatyczną instrukcji Javy, następnie postępuje podobnie z konstrukcjami SQLJ (są one poprzedzone frazą #sql) oraz dokonuje sprawdzenia poprawności odwoływania się do zmiennych i metod Javy z wewnątrz instrukcji SQLJ. Jeśli składania została pomyslnie zweryfikowana proces przechodzi do następnej fazy.

Druga faza polega na sprawdzeniu poprawności wbudowanych konstrukcji. Na tym etapie translator może połączyć się z bazą danych w celu weryfikacji zgodności zadeklarowanych typów danych (weryfikacja online). Typom bazodanowym odpowiadają określone typy Javy (tab. 1). Podstawowymi czynnościami translatora podczas tej fazy są:

- 1) Analiza typów w wyrażeniach zawartych w instrukcji SQLJ. Sprawdzana jest dostępność każdej zadeklarowanej klasy Javy – poprzez przeszukiwanie CLASSPATH.
- 2) W przypadku weryfikacji online przeprowadzone jest sprawdzanie poprawności zawartych operacji DML (SELECT, INSERT, UPDATE, DELETE ...).
- 3) Dodatkowo sprawdzane są typy danych (Java – SQL) wobec typów zadeklarowanych w argumentach procedur składowanych a także w zwracanych wartości w przypadku funkcji składowanych.

Tabela 1. Typy danych SQL – Java

TYP SQL	TYP JAVA
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	double
BINARY	byte[]
VARBINARY	byte[]
LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

Po pomyslnym przejściu wcześniejszych faz kod SQLJ jest zamieniany na kod Javy używający odwołań JDBC w celu komunikowania się z bazą danych. Kod środowiska

SQLJ używa domyślnego połączenia JDBC zdefiniowanego przez programistę, bądź oddzielnych połączeń, które muszą być zdefiniowane przez programistę jawnie.

Ostatnim krokiem jest wspólna kompilacja kodu programisty oraz wygenerowanego przez środowisko SQLJ. Kompilacja może być dokonana każdym kompilatorem Javy (standard 1.4). Powstaje plik klas (.class) przenaszalny na różne platformy sprzętowe i systemowe korzystający ze zdefiniowanego i skonfigurowanego wcześniej sterownika JDBC.

Wspomniany mechanizm sprawdzania poprawności typów online pozwala na zwerifikowanie ich poprawności jeszcze przed uruchomieniem kodu co pozwala na szybsze dostrzeżenie popełnionych błędów – zwiększając efektywność programisty.

Należy pamiętać, że SQLJ wspiera jedynie statyczne konstrukcje języka SQL. JDBC natomiast dostarcza dynamicznego interfejsu dostępu do bazy danych poziomu poziomu Javy [MEL 03A].

2.1. ZAPYTANIA W SQLJ

Aby móc korzystać z możliwości zagnieżdżania SQL w kodzie Javy należy zaimportować pakiet klas środowiskowych SQLJ:

```
import sqlj.runtime.*;
```

Jak wspomniano wyżej konstrukcje SQLJ rozpoznawane są przez translator poprzez poprzedzenie ich słowem kluczowym `#sql`, dodatkowo wyrażenia SQL muszą zostać ujęte w nawiasy klamrowe (`{, }`).

Wywołanie procedury składowanej o nazwie *MyProcedure* z dwoma parametrami przekazanymi jako wartości stałe, może mieć postać:

```
#sql { CALL MyProcedure('par1', 'par2') ; }
```

Aby uzyskać dostęp do wyników zapytania możemy wykorzystać iteratory nazwane lub pozycyjne. Przykładowy kod SQLJ wykorzystujący integrator pozycyjny jest następujący:

```
#sql public iterator
    ActiveAuthors (Integer authorID, Integer opublikowanych);
...

ActiveAuthotrs authors ;
#sql authors = {
    SELECT author.id, COUNT(*) AS opublikowanych
    FROM articles, authors
```

```
WHERE journalId = :journal and authors.id = articles.authorId
GROUP BY 1  };
```

```
while (sc.next())
    sendReminder(sc.authorId());
authors.close();
```

Definicja iteratora `ActiveAuthors` mówi, że będzie on pozwalał iteracyjnie przejść po elementach zbioru danych. Każdy z elementów ma dwa atrybuty typu `Integer`. Instrukcja przypisania (`#sql authors =`) powoduje powiązanie iteratora `authors` z wynikami zapytania. Kolejne wiersze wyniku pobieramy poprzez wywołanie metody `next()`. Dostęp do wartości w kolejnych kolumnach odbywa się poprzez nazwy zadeklarowane w definicji iteratora.

Definicja iteratora pozycyjnego wymaga jedynie znajomości typów wartości zwracanych przez iterator. W jego definicji brak jest nazw kolumn. Przechodzenie iteratora po elementach wyniku zapytania odbywa się w odmienny sposób niż w przypadku iteratora nazwanego. Wartości z wiersza wyniku pobierane są instrukcją `FETCH` oraz przypisywane zmiennym zadeklarowanym w bloku Javy. Podobnie jak iterator nazwany, iterator pozycyjny także posiada metodę, której wywołanie spowoduje sprawdzenie czy osiągnęliśmy już koniec kolekcji.

```
// deklaracja iteratora pozycyjnego
#sql public iterator ByPos (String, int);
...
ByPos positer;
String name = null;
int year = 0;
// populate it
#sql positer = { SELECT fullname, birthyear FROM people };

while ( true ) {
    if (positer.endFetch())
        break;
    #sql { FETCH :positer INTO :name, :year };
    System.out.println(name + " was born in " + year);
}
```

2.2. KONTEKST I POŁĄCZENIA

SQLJ automatycznie korzysta z domyślnie zdefiniowanego kontekstu i połączenia z bazą danych poprzez sterownik JDBC. Istnieje jednak możliwość podłączenia osobnego kontekstu oraz stworzenia odrębnego połączenia i używania go do wykonywania poleceń SQLJ.


```
#sql context MyContext
...
MyContext my = new MyContext(connectionString, true);
#sql [my] {
    UPDATE shipments
    SET periodLength = periodLength + 10
    WHERE realized = false
}
my.close();
```

Istnieje kilka konstruktorów kontekstu definiowanego przez użytkownika (interfejs `sqlj.runtime.ConnectionContext`). W powyższym przykładzie otworzyliśmy połączenie z bazą (zawarte w `connectionString`) oraz ustawiliśmy flagę `autoCommit` na `true`. Deklaracja wykorzystania nowego połączenia polega na dopisaniu napisu: `[my]` przed wywoływana instrukcją języka SQL.

3. PROCEDURY SKŁADOWANE JAVY

Sposób użycia statycznych metod Javy jako procedur i/lub funkcji składowanych oraz zwracania wyników definiuje pierwsza część standardu SQLJ (pakiet SQL/JRT standardu SQL:2003) [MEL 03B].

Każda publiczna, statyczna metoda, której typ argumentów oraz zwracanego wyniku mają odpowiadające im typy SQL może być wykorzystana jako procedura lub funkcja składowana (tab. 1).

Ładowanie, usunięcie czy zmiana klas Javy w bazie danych następuje poprzez procedury opisane w standardzie. Standard wymaga, aby baza danych zawierała dodatkowy schemat o nazwie `sqlj`, w którym zawarte są wszystkie procedury i funkcje środowiska SQLJ. Rozszerzono także składnię poleceń SQL: tworzenia i usuwania procedur oraz nadawania uprawnień do klas Javy.

3.1. ZARZĄDZANIE BIBLIOTEKAMI JAVY

W celu zarządzania środowiskiem wprowadzono nowe procedury dodające/zmieniające/usuwające biblioteki Javy załadowane do bazy danych.

Aby załadować bibliotekę do bazy danych wykorzystujemy polecenie `sqlj.install_jar`, tak jak w poniższym przykładzie:

```
sqlj.install_jar ('file:/myjar.jar', 'MyJar', 0);
```

Pierwszy parametr wskazuje na położenie pliku (w tym wypadku `myjar.jar`) z którego mają być załadowane klasy. Drugim parametrem jest kwalifikowana nazwa (wg standardów SQL), którą zostanie oznaczony wskazany plik `.jar`. Nazwa ta jest używana następnie do jednoznacznego wskazania źródła pochodzenia klasy (polecenia `sqlj.remove_jar`, `grant`, `create` ...). Trzecim parametrem jest liczba całkowita, która informuje środowisko SQL czy po załadowaniu pliku mają zostać wykonane operacje zdefiniowane w deskrytorze tego pliku (wartość różna od zera), czy nie (wartość 0).

Aby odinstalować załadowaną wcześniej bibliotekę Javy stosujemy polecenie `sqlj.remove_jar`. Plik `.jar`, który ma zostać usunięty z środowiska jest identyfikowany przez zdefiniowaną dla niego nazwę. Dodatkowo określa się również, czy mają zostać wykonane akcje zdefiniowane w deskrytorze pliku `.jar`.

```
sqlj.remove_jar ('MyJar', 1);
```

Powyższe polecenie spowoduje usunięcie z pamięci klas zawartych w pliku `.jar` oznaczonym jako *MyJar* oraz uruchomienie akcji związanych z jego usunięciem zdefiniowanych w deskrytorze tego pliku (wartość 1 drugiego parametru procedury). **UWAGA:** usunięcie biblioteki zakończy się niepowodzeniem, jeśli w systemie SQL znajdują się jakiekolwiek do niej odwołania (np. powiązanie nazwy procedury z metodą kasy zawartej w pliku).

Wykonywanie poleceń: `sqlj.remove_jar` oraz `sqlj.install_jar` pozwala dokonać aktualizacji pliku `jar`, a co za tym idzie metod w niej zawartych, niestety napotykamy problem: wszystkie powiązane procedury SQL z metodami pliku `.jar` (oraz powiązania związane z częścią 2 standardu) muszą zostać usunięte przed wykonaniem kombinacji tych instrukcji. W związku z tym wprowadzono instrukcję `sqlj.replace_jar`.

```
sqlj.replace_jar('file:/MyJar_ver2.jar', 'MyJar');
```

Wykonanie tej procedury pozwala na zaktualizowanie pliku `.jar` i metod w niej zawartych. Podczas tego procesu sprawdzana jest zgodność typów danych nowo instalowanych procedur ze zdefiniowanymi procedurami/funkcjami SQL oraz dokonywane jest sprawdzenie czy wszystkie metody do których odnoszą się konstrukcje SQL zawarte we wcześniejszej wersji pliku `.jar` są także obecne w jego nowo instalowanej wersji.

Gdy klasy załadowane z pliku `.jar` korzystają z klas, które nie są w nich zdefiniowane – środowisko poszukuje ich w zdefiniowanych ścieżkach. Ścieżki takie ustala się osobno dla każdego pliku `.jar`.

```
sqlj.install_jar('file:/SomeNewJar.jar', 'NewJar', 0);
sqlj.alter_java_path ('NewJar', '(prefix/*, MyJar)');
```

W powyższym przykładzie załadowaliśmy nowy plik .jar o nazwie *NewJar*, który korzysta z klas zdefiniowanych we wcześniej załadowanym przez nas pliku. W tym przypadku należy zdefiniować dla niego ścieżkę poszukiwania brakujących definicji klas stosując procedurę `sqlj.alter_java_path`. Pierwszym parametrem jest nazwa pliku .jar dla którego chcemy ustalić ścieżkę, następnie podajemy listę par. (oddzielonych przecinkami). Każda para (ujęta w nawiasach okrągłych) reprezentuje wzór poszukiwania oraz nazwę pliku .jar w którym poszukiwane są brakujące zależności.

3.2. WYKORZYSTANIE BIBLIOTEK JAVY

Mając zdefiniowane metody:

```
package examples;
public class ExClass {
    // przykładowa metoda
    public static String concat (String a, String b) {
        return a + " " + b;
    }
    // przykładowa metoda
    public static void happyBirthday() {
        Connection conn = DriverManager.
ger.getConnection("jdbc:default:connection");
        PreparedStatement pstmt = conn.prepareStatement(
            "UPDATE workers" +
            "SET soup = soup * 1.1" +
            "WHERE birthmonth = ? AND birthday = ?");

        pstmt.setInt(1, Calen-
dar.getInstance().get(Calendar.MONTH));
        pstmt.setInt(2, Calen-
dar.getInstance().get(Calendar.DAY_OF_MONTH));
        pstmt.executeUpdate();
        pstmt.close();
        conn.close();
    }
}
```

oraz załadowany plik .jar z zawartą klasą *ExClass* nazwany *methods* możemy zdefiniować procedurę łączącą dwa napisy (typ SQL mapowany na String).

```
CREATE FUNCTION concat
    (first VARCHAR(20), second VARCHAR(20)) RETURNS VARCHAR(40)
NO SQL
EXTERNAL NAME 'methods:examples.ExClass.concat'
LANGUAGE JAVA PARAMETER STYLE JAVA;
```

Polecenie zawiera frazę: `NO SQL`, która oznacza że zawarte w metodzie `concat` operacje nie odwołują się do danych zawartych w bazie. Tak zdefiniowana metoda może zostać użyta jak zwykła funkcja składowana:

```
CALL concat('jan', 'kowalski');
```

Korzystając ze zdefiniowanej wcześniej metody napisanej w Javie *happyBirthday* stworzymy procedurę `moresoup`, różnica w definicji nazwy metody widoczna jest w drugiej linii – występuje w niej fraza `MODIFIES SQL DATA` – co oznacza, iż metoda *happyBirthday* modyfikuje dane zawarte w bazie. Dokonuje tego przy użyciu domyślnego połączenia JDBC. Oprócz `NO SQL`, oraz `MODIFIES SQL DATA` dostępne są także dwa inne kwalifikatory dostępu: `CONTAINS SQL` oraz `READS SQL`.

```
CREATE PROCEDURE moresoup
  MODIFIES SQL DATA
  EXTERNAL NAME 'methods:examples.ExClass.happyBirthday'
  LANGUAGE JAVA PARAMETER STYLE JAVA;
```

```
CALL moresoup;
```

4. ABSTRAKCYJNE TYPY DANYCH

Możliwość definiowania abstrakcyjnych typów danych w oparciu o kod Javy została zdefiniowana w trzeciej części standardu SQLJ. Podobnie jak podczas używania statycznych konstrukcji języka Java, także i w tym przypadku należy posługiwać się z procedur zdefiniowanych w schemacie `sqlj`.

W celu umożliwienia tworzenia nowych typów danych rozszerzono składnię poleceń SQL: `CREATE TYPE` oraz `DROP TYPE`, które pozwalają na powiązanie nowych typów danych z istniejącymi klasami w Javie.

Prostą klasę Javy przechowującą informacje o kwadracie - jego długości i nazwie możemy zdefiniować następująco:

```
package mypackage;
public class Square implements java.Serializable {
    public int side;
    public String name;

    public Square (int side, String name) {
        this.side = side;
        this.name = name;
    }
}
```

```

    }
    public int field () {
        return this.side * this.side;
    }
}

```

Klasa implementuje interfejs `Serializable`, co umożliwia konwersję typów Javy na ich SQL'owe odpowiedniki (konwersja jest również możliwa po zaimplementowaniu interfejsu `java.sql.SQLData`). Po pomyślnym zakończeniu dołączenia biblioteki zawierającej naszą klasę do środowiska SQL (nazwa biblioteki: *MyTypes*) możemy zdefiniować typ SQL:

```

CREATE TYPE Square EXTERNAL NAME 'MyTypes:mypackage.Square'
LANGUAGE Java AS
    ( side INTEGER EXTERNAL NAME 'side', name VARCHAR(30)
    EXTERNAL NAME 'name' )

METHOD Square(side INTEGER, name VARCHAR(30)
    RETURNS Square EXTERNAL NAME 'mypackage.Square',

METHOD field()
    RETURNS INTEGER EXTERNAL NAME 'field' ;

```

Tak zdefiniowany typ możemy użyć wszędzie tam gdzie można używać abstrakcyjnych typów danych SQL. Na przykład podczas definicji schematu tabeli. Ustawianie wartości odbywa się poprzez konstruktor, wywołanie metody na tak zdefiniowanym typie wygląda również tak jak w przypadku innych typów.

```

CREATE TABLE farmers (
    name VARCHAR(30) PRIMARY KEY,
    culture Square NOT NULL );

INSERT INTO farmers VALUES ('kowalski', new Square(100, 'ziemiaki'));

SELECT name, culture.field()
FROM farmers
ORDER BY 2 DESC;

```

5. PODSUMOWANIE

Bardzo ważną zaletą standardu SQLJ jest jego przenaszalność, wynika ona z faktu przenośności samej Javy, jak również ustandaryzowania instrukcji SQLJ. Kod skom-

piłowany na jednej maszynie w środowisku Linux (część 0) może być dowolnie przeniesiony na inne platformy sprzętowe. Jedynym ograniczeniem jest konieczność zapewnienia tej samej infrastruktury bazodanowej (w przypadku definiowania połączeń niestandardowych z bazą danych). Dodatkowo składania standardu została opublikowana oraz zaprojektowana tak, aby była niezależna od silnika bazy danych. Bardzo dużą zaletą SQLJ jest możliwość dokonywania sprawdzenia poprawności kodu SQL na etapie kompilacji. Pozwala to na bardzo szybkie wyszukiwanie błędów.

LITERATURA

- [CLINE] Owen Cline, *Develop your applications using SQLJ*
<http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0412cline/>, 2004.
- [MEL 03A] Jim Meliton. (ISO-ANSI Working draft) *Object Language Bindings (SQL/OLB)*. ISO/ANSI, 2003-07-25.
- [MEL 03B] Jim Meliton. (ISO-ANSI Working draft) *Routines and Types using the Java Programming Language(SQL/JRT)*. ISO/ANSI, 2003-07-25.
- [EISEN] Andrew Eisenberg, Jim Melton, *SQLJ Part 0, now known as SQL/OLB (Object-Language Bindings)*, SIGMOD Record, Volume 2, 1998.
- [STEM] Shirley Ann Stem. *The open sesame of java database applications*.
http://www.javaworld.com/javaworld/jw-05-1999/jw-05-sqlj_p.html, 1999.

SQLJ STANDARD

Standard SQLJ stands for Structured Query Language (SQL) embedded in the programming language Java. This paper provides the rationale for SQLJ and a brief tutorial of its syntax and semantics.

Zygmunt MAZUR*
Przemysław MIELCAREK*
Marcin PARCZEWSKI*

HSSE – TRANSPARENTNA EWOLUCJA SCHEMATU W RELACYJNYCH BAZACH DANYCH

Aplikacje biznesowe wykorzystujące relacyjne systemy bazodanowe coraz częściej spotykają się z problemem dostępności schematu w czasie jego modyfikacji. Zazwyczaj nawet najmniejsza zmiana struktury schematu wiąże się ze wstrzymaniem pracy komponentów na czas aktualizacji schematu oraz ich aktualizacją. Rozwiązaniem problemu „wymuszenia aktualizacji komponentu” może być wersjonowanie schematu. Niestety istniejące mechanizmy mają wady, które uniemożliwiają ich powszechną implementację. Dodatkowo nie nadają się do zastosowania w istniejących systemach o wieloaplikacyjnej architekturze bazodanowej.

Artykuł opisuje propozycje modelu dedykowanego do rozwiązania problemów dostępności schematu w czasie modyfikacji, dostosowania komponentów po ewolucji schematu oraz możliwości wykorzystania w istniejących systemach.

1. WPROWADZENIE

W czasie pracy ze schematem relacyjnym coraz częściej mamy do czynienia z problemem dostępności schematu w czasie jego aktualizacji. Nawet najmniejsza zmiana struktury schematu wiąże się zazwyczaj:

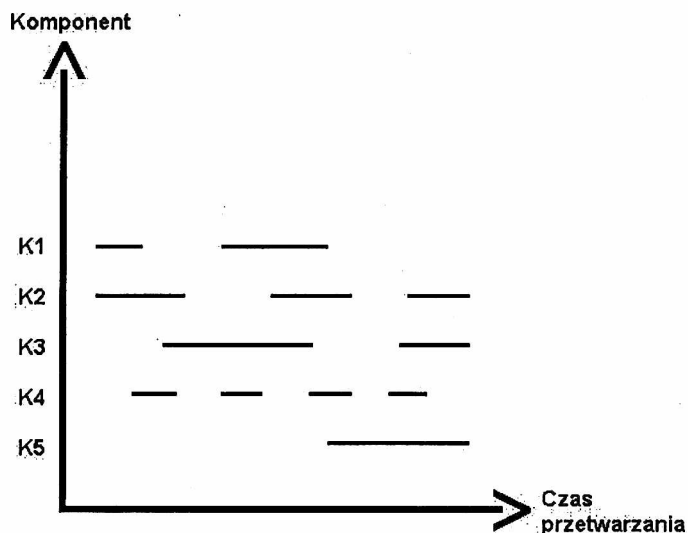
- 1) ze wstrzymaniem pracy komponentów na czas aktualizacji,
- 2) z aktualizacją komponentów, które z niego korzystają.

Z powyższego wynika, że zmiana w schemacie może zostać wykonana tylko w sytuacji, w której każdy komponent zakończy pracę ze schematem i są gotowe aktualizacje komponentów zgodne ze zmienionym schematem.

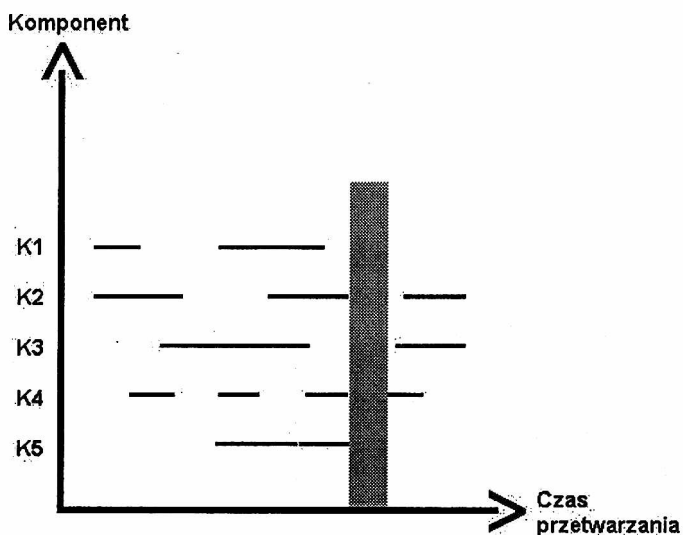
Duże systemy informatyczne składające się z wielu komponentów korzystających niezależnie ze schematu, mogą powodować problemy w zarządzaniu zmianami w bazie

* Instytut Informatyki Stosowanej, Wydział Informatyki i Zarządzania Politechniki Wrocławskiej, 50-370 Wrocław, Wybrzeże Wyspiańskiego 27, zygmunt.mazur@pwr.wroc.pl, przemyslaw.mielcarek@pwr.wroc.pl, marcin.parczewski@pwr.wroc.pl

utrzymania. Łatwo jest wyobrazić sobie sytuację, gdy nie jest możliwe znalezienie przestrzeni czasowej, w której wszystkie komponenty systemu zakończyły pracę ze schematem (rysunek 1). W takich sytuacjach wprowadza się procesy i przepływy, które sztucznie synchronizują przepływ sterowania komponentów tak, aby znaleźć interwał, w którym można wprowadzić zmiany w schemacie (wyróżniony obszar na rysunku 2).



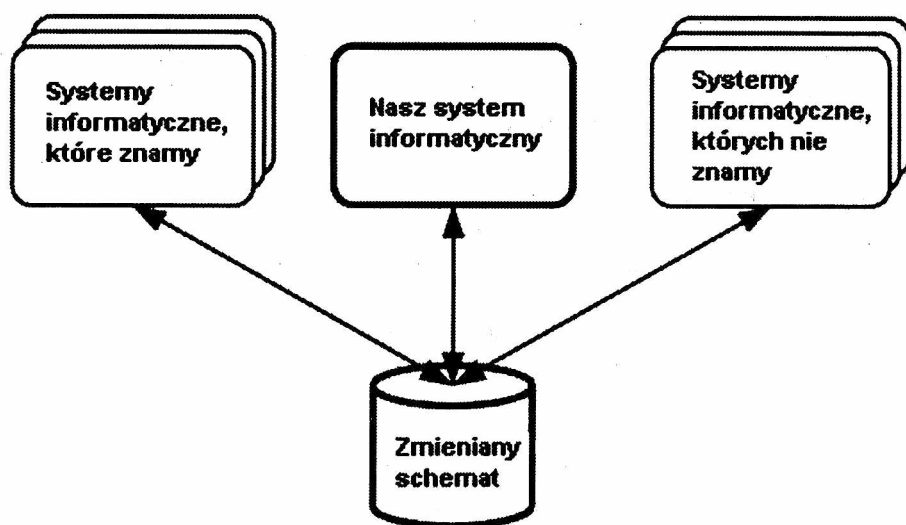
Rys. 1. Przepływ sterowania komponentów bez możliwości zmian schematu



Rys. 2. Przepływ sterowania komponentów z wolnym interwałem czasowym

Takiego rozwiązania w wielu przypadkach nie można jednak zastosować, np. w systemach informatycznych z dziedziny finansowej, gdzie wymaganiem może być ciągła dostępność systemu dla użytkownika. Oznacza to, że niektóre komponenty muszą korzystać ze schematu przez 24 godziny, przez 7 dni w tygodniu.

Kolejnym problemem spotykanym w czasie aktualizacji schematu jest wymuszenie dostosowania wszystkich komponentów do zmienionego schematu w jednym czasie. Istnieją sytuacje, w których nie mamy wpływu na zmiany komponentów. Przykładem mogą być systemy spadkowe czy istnienie niezależnych systemów korzystających ze schematu, których aktualizacja jest utrudniona. Mówimy wtedy o wieloaplikacyjnej architekturze bazodanowej [AMBLER 06] (rysunek 3). Taka sytuacja bardzo ogranicza możliwość zmiany struktury schematu.



Rys. 3. Wieloaplikacyjna architektura bazodanowa

2. CHARAKTERYSTYKA MODELU

Proponowany w artykule model zwany dalej HSSE (ang. *Hot Swap Schema Evolution*) oparty jest na wersjonowaniu schematu relacyjnego oraz na opcjonalnym rozszerzeniu języka zapytań. Większość prac traktujących o wersjonowaniu schematu skupia się na modelu temporalnym oraz obiektowym np. [GRANDI 00]. Pojęcia i definicje wprowadzone dla tych modeli dotyczące zmiany struktury schematu są prawdziwe dla modelu relacyjnego z ograniczeniem jednak do modeli temporalnych wywodzących się z modelu relacyjnego [RODDICK 95]. Pojęcia te będą wykorzystywane w dalszej części artykułu i zostaną przytoczone na podstawie pracy [RODDICK 95].

Modyfikacja schematu

Modyfikacja schematu ma miejsce, gdy system bazodanowy pozwala na zmianę definicji schematu, który zawiera dane.

Ewolucja schematu

Ewolucja schematu ma miejsce, gdy system bazodanowy pozwala na zmianę definicji schematu bez utraty danych, które zawiera.

Wersjonowanie schematu

Wersjonowanie schematu ma miejsce, gdy system bazodanowy pozwala na dostęp do wszystkich danych z różnych schematów, zwanych wersjami schematu.

Częściowe wersjonowanie schematu

Częściowe wersjonowanie schematu oznacza dostęp do wszystkich danych z różnych schematów, zwanych wersjami, w trybie do odczytu. Tylko ostatnia wersja schematu ma dostęp do danych w trybie modyfikacji.

Pełne wersjonowanie schematu

Pełne wersjonowanie schematu oznacza dostęp do wszystkich danych z różnych schematów, zwanych wersjami w trybie do odczytu i modyfikacji.

Od początku pojęcie wersjonowania schematu było związane z aplikacjami typu CAD, gdzie występuje ciągła zmiana struktury encji. Nie jesteśmy w stanie przewidzieć przyszłości, dlatego trudno jest zamodelować takie zmiany w statycznym schemacie relacyjnym [KATZ 90]. Powstało wiele ciekawych modeli wersjonowania, które są odpowiedzią na problem zmieniającej się encji. Największym problemem istniejących modeli jest założenie permanentnego istnienia wersji historycznych schematu (*completed schema* [CASTRO 97], *global schema* [RUNDE 96]). Wynika to z charakteru modelu danych, dla którego modele wersjonowania były tworzone. Model temporalny u podstaw zakłada trwałe istnienie danych (nawet po logicznym usunięciu), a co za tym idzie istnienie wersji schematu, w którym daną składowano. Prowadzi to do szybkiego przyrostu rozmiaru bazy, która musi przechowywać informacje zarówno o aktualnej wersji schematu jak i wersjach historycznych.

Z tego powodu wykorzystanie istniejących temporalnych modeli wersjonowania dla modelu relacyjnego na przemysłową skalę jest ograniczone.

Proponowany przez nas model ogranicza istnienie wersji historycznej do czasu istnienia komponentu, który z niej korzysta. Tak więc aktualizacja wszystkich komponentów do ostatniej wersji schematu powoduje usunięcie historycznych wersji.

Kolejnym problem istniejących modeli jest brak możliwości zastosowania w istniejących, działających systemach informatycznych. Wynika to przede wszystkim z faktu obligatoryjnego rozszerzenia języka zapytań. W najprostszym przypadku roz-

szerzenie wymusza na komponencie wskazanie wersji schematu, z której ma zamiar korzystać w zapytaniu. Proponowany model HSSE wprowadza pewne rozszerzenia języka zapytań. Są to jednak rozszerzenia natury konfiguracyjnej i można je wprowadzić transparentnie dla komponentów. Rozszerzeniami tymi są:

- 1) wstrzymania przetwarzania zapytań do schematu,
- 2) wskazanie wersji schematu.

Wstrzymanie przetwarzania występuje w sytuacji zmiany struktury schematu. Nie oznacza to, że rozpoczęte przetwarzanie nie zostanie zakończone, oznacza jedynie wstrzymanie na okres aktualizacji schematu. Po aktualizacji schematu przetwarzanie komponentu zostanie zakończone. W popularnych obecnie językach programowania spotykamy się z pojęciem sterownika bazy danych. Przykładem są sterowniki oparte na standardzie JDBC. Standard zakłada szeroką konfigurowalność, dzięki której możemy osiągnąć wstrzymanie przetwarzania na czas aktualizacji schematu.

Rozszerzenie specyfikacji zapytań o wskazanie wersji schematu jest opcjonalne i występuje w sytuacji, kiedy implementacja modelu HSSE nie ma możliwości, aby jednoznacznie wskazać wersję schematu z zapytania. Wnioskowanie takie może być automatyczne. Kolejny raz posłużymy się obiektowym językiem programowania, gdzie występuje pojęcie puli połączeń. Załóżmy, że istnieje wiele puli połączeń, z których każda połączona jest z inną wersją schematu. Nie stanowi problemu, aby konfiguracyjnie wymusić na komponencie wykorzystanie odpowiedniej puli połączeń. Drugim rozwiązaniem jest przypisanie wersji schematu na podstawie specyfikacji zapytania. Zapytanie niesie ze sobą informacje o strukturze schematu (atrybuty w relacji, relacje).

2.1. CECHY HSSE

Najważniejszymi cechami proponowanego modelu są:

- 1) transparentna ewolucja schematu z wykorzystaniem wersjonowania,
- 2) ograniczenie miejsca wykorzystanego do wersjonowania w odniesieniu do innych modeli wersjonowania,
- 3) przezroczystość wdrożenia w istniejących systemach nie korzystających z wersjonowania,
- 4) pełne wersjonowanie schematu,
- 5) możliwość późniejszego dostosowania komponentów do aktualnej wersji schematu,
- 6) transparentne usunięcie nadmiarowych, nieaktualnych i redundantnych wersji schematu.

2.2. OPIS MODELU

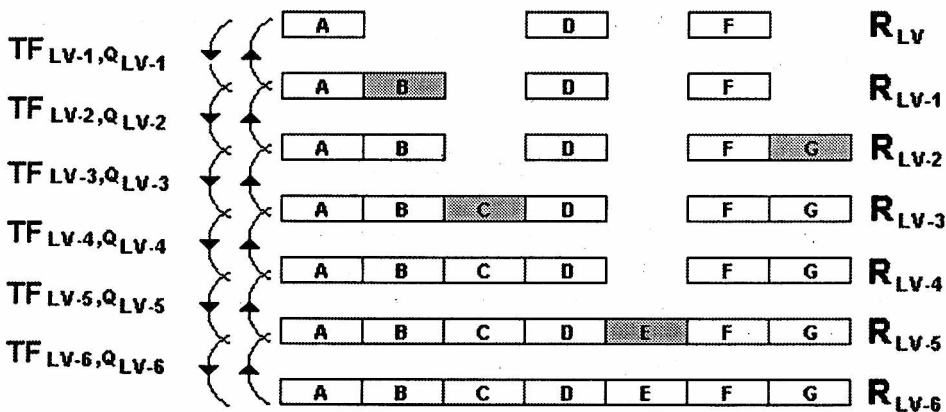
W modelu HSSE możemy wyróżnić schemat AS , funkcje przekształceń TF , reguły rzutowania zapytań QMR_n , reguły rzutowania odpowiedzi RMR_n i tymczasowe składki TP_n , gdzie n oznacza wersję schematu, $n \in (0..LV) \cap n \in N^+$, a $LV \in N^+$ oznacza aktualną wersję schematu.

W modelu fizycznie istnieje tylko jeden schemat o nazwie AS i jest on zawsze odzwierciedleniem rzeczywistości. Wersje historyczne schematu są pojęciem abstrakcyjnym i są budowanym na podstawie TP i TF .

Każda wersja schematu skojarzona jest z tymczasowym składem. Służy on do przechowywania danych niezbędnych do poprawnej pracy wersji schematu. Jest wykorzystywany w czasie przetwarzania zapytań. Przykładowymi danymi przechowywanymi w TP_n mogą być usunięte w wersji schematu $n + 1$ atrybuty, struktura widoku konceptualnego, reguły rzutowania zapytań czy reguły rzutowania odpowiedzi dla wersji n . Z założenia o transparentnym usuwaniu nadmiarowych wersji schematu wynika, że TP_{n+1} nie może być zależny od TP_n .

Reguła QMR_n definiuje sposób rzutowania zapytań DML między schematem w wersji n i $n + 1$. Wykorzystywana jest przez TF w czasie przetwarzania zapytań.

Reguła RMR_n definiuje sposób rzutowania odpowiedzi zapytań DML między schematem w wersji $n + 1$ i n . Wykorzystywana jest przez TF w czasie przetwarzania rezultatu zapytań.



Rys. 4. Przetwarzanie zapytania w modelu HSSE

Kwerenda na wersji schematu n wykonywana jest przy wykorzystaniu funkcji przekształceń o sygnaturze $TF(n, Q)$, gdzie Q oznacza kwerendę. Zadaniem funkcji przekształceń jest zwrócenie wyniku dla zadanej kwerendy. W procesie przetwarzania kwerendy funkcja przekształcająca wykorzystując reguły QMR wykonuje zapytanie na AS . Zwrócony wynik przy wykorzystaniu reguł RMR jest propagowany do wszystkich

TP w celu ich aktualizacji. Przykład działania funkcji przedstawia rysunek 4. Relacja *R* posiada sześć wersji historycznych, w których się zmieniała. Atrybuty *E*, *C*, *G* i *B* były usuwane z relacji w kolejnych wersjach (puste miejsce po atrybucie). W czasie usunięcia atrybutu zostaje przeniesiony do odpowiedniego *TP* (szare obszary). W przypadku atrybutu *E* został on przeniesiony do *TP_{LV-5}*. Zapytanie zostało wykonane na *LV-6* wersji schematu. Funkcja *TF* wykorzystując kolejne *QMR* modyfikuje zapytanie do wersji *LV* (strzałki skierowane do góry). Po wykonaniu zapytania na schemacie, rezultat wraz z dodatkowymi danymi jest propagowany do historycznych wersji schematu (strzałki skierowane w dół) i zwrócony użytkownikowi. Propagacja odbywa się z wykorzystaniem kolejnych *RMR*. Dodatkowe dane zwrócone przez zapytanie są wykorzystywane do aktualizacji *TP*.

3. PRZYSZŁE KIERUNKI BADAŃ

W pracy została przedstawiona pierwsza propozycja modelu wymagająca w wielu miejscach uściśleń, przyszłe badania nad modelem będą dotyczyły aspektów:

- 1) restrykcji dla pełnego wersjonowania schematu,
- 2) budowania reguł rzutowania,
- 3) wpływu wersjonowania na ograniczenia schematu (klucze główne itp.),
- 4) propozycji i implementacji modelu,
- 5) własności modelu.

3.1. RESTRYKCJE W PEŁNYM WERSJONOWANIU SCHEMATU

Istnieją sytuacje, kiedy w prosty sposób reguły rzutowania mogą nie poradzić sobie z przekształceniem zapytania do kolejnej wersji schematu. Przykładem może być zmiana domeny atrybutu. Temat ten został już poruszony w [VENTRONE 91] i wzbogacony wieloma przykładami. W istniejących modelach zaproponowano wiele rozwiązań dla tego problemu. Pozostaje to jednak problem, który trzeba zbadać w kontekście proponowanego modelu.

3.2. BUDOWA REGUŁ RZUTOWANIA

Kolejnym punktem badań są reguły rzutowania, a dokładnie metodyka ich budowania. Najprostszym sposobem wydaje się uzależnienie reguły od semantyki zmiany schematu. Niestety będzie to prowadziło do ingerencji czynnika ludzkiego w każdą zmianę schema-

tu. Należy się zastanowić, czy każdą zmianę w schemacie można rozpatrywać w kategoriach syntaktyki i wykorzystać to do budowania reguł rzutowania [VENTRONE 91].

3.3. WPŁYW WERSJONOWANIA NA OGRANICZENIA SCHEMATU

Zmiana danych przy wykorzystaniu wersji historycznej schematu może powodować naruszenia klucza głównego lub innych ograniczeń. Rozwiązaniem tego problemu może być zarządzanie ograniczeniami przy wykorzystaniu samego modelu, a nie SZBD.

3.4. PROPOZYCJA IMPLEMENTACJI MODELU HSSE I JEGO WŁASNOŚCI

Powstała pierwsza propozycja modelu [MAZUR 06]. Implementacja i zbadanie powstałego na jego podstawie mechanizmu dostarczy odpowiedzi na pytania dotyczące własności modelu oraz możliwości jego masowego zastosowania.

4. PODSUMOWANIE

Rozwijając systemy informatyczne często mamy do czynienia ze zmianami, które trzeba wprowadzić w fazie utrzymania systemu. Zmiany takie wynikają przede wszystkim:

- z dodania nowej funkcjonalności do systemu,
- z błędów implementacji w rozwijanym systemie,
- z refaktoringu,
- ze zmiany otoczenia, w którym system istnieje,
- z błędnych założeń funkcjonalnych wobec systemu.

Proponowany model może więc stanowić znakomite wsparcie dla fazy utrzymania. Pozostaje jednak jeszcze wiele pracy w dziedzinie implementacji i uszczegółowienia modelu. W artykule wskazano przyszłe kierunki badań. Wyniki badań będą stanowiły o sile proponowanego modelu.

LITERATURA

- [AMBLER 06] Scott W. Ambler, Pramod J. Sadalage, *Refactoring Databases: Evolutionary Database Design*, Addison Wesley Professional, 2006.

- [BALDWIN 04] Douglas Baldwin, Greg W. Scragg, *The Object Primer*, Third.
- [BOUNIF 06] Hassina Bounif, *Data Model Versioning and Database Evolution*, Idea Group, 2006.
- [CASTRO 97] Cristina de Castro, Fabio Grand and Maria Ritascalas, *Schema versioning for multi-temporal relational databases*, 1997.
- [GRANDI 00] Fabio Grandi, Federica Mandreoli and Maria Rita Scalas, *A Generalized Modeling Framework for Schema Versioning Support*, 2000.
- [GRANDI 02] Fabio Grandi, Federica Mandreoli, *A Formal Model for Temporal Schema Versioning in Object-Oriented Databases*, 2002.
- [KATZ 90] Randy H. Katz, *Toward a Unified Framework for Version Modeling in Engineering Databases*, 1990.
- [KIM 96] Kim W., *Wprowadzenie do obiektowych baz danych*, WNT, Warszawa 1996.
- [MAZUR 06] Mazur Z., Mielcarek P., Parczewski M., *HSSE – mechanizm dynamicznych zmian schematu relacyjnej bazy danych*, WKŁ, Warszawa, 2006.
- [RODDICK 95] John F. Roddick, *A Survey of Schema Versioning Issues for Database Systems*, 1995.
- [RUNDE 96] Taube and Elke A. Rundensteiner, Viviane Crestana, *Consistent View Removal in Transparent Schema Evolution Systems*, 1996.
- [VENTRONE 91] Ventrone, V. and Heiler, S., *Semantic Heterogeneity as a Result of Domain Evolution*, SIGMOD, 1991. Edition, Cambridge University Press, 2004.

HSSE – THE TRANSPARENT EVOLUTION OF SCHEMA IN RELATIONAL DATABASES

Business Applications based on relational model often meet problems of schema availability during its modification. Even the smallest change of schema structure is a risk of making existing application programs obsolete when they run against the modified schema. Schema versioning can be used as a solution of “forced component evolution” problem. Unfortunately, existing mechanisms have defects, which make impossible to use them widely as common implementation.

This article describes model that addresses the problem by integrating schema evolution with schema versioning facilities.





BIBLIOTEKA GŁÓWNA

329273 L/M

Wydawnictwa Politechniki Wrocławskiej
są do nabycia w księgarni
„Tech”
plac Grunwaldzki 13, 50-377 Wrocław
budynek D-1 PWr., tel. (0 71) 320 29 35
Prowadzimy sprzedaż wysyłkową

ISBN 83-7085-990-9