kowe
Instytutu Informatyki
Politechniki Wrocławskiej

10 BAZY DANYCH

Redakcja naukowa

Zygmunt Mazur

Oficyna Wydawnicza
Politechniki Wrocławskiej


EX LIBRIS
IZ
BIBLIOTEKI
POLITECHNIKI
WROCŁAWSKIEJ

Prace Naukowe
Instytutu Informatyki
Politechniki Wrocławskiej

10 BAZY DANYCH

Redakcja naukowa

Zygmunt Mazur

Oficyna Wydawnicza Politechniki Wrocławskiej
Wrocław 2009

SPIS TREŚCI

CONTENTS

Jakość to doskonałość, której nie da się osiągnąć,
lecz do której trzeba uporczywie dążyć

Lao Tsu

WSTĘP

Z prawdziwą radością oddajemy w ręce Czytelników kolejny, tym razem jubileuszowy, dziesiąty numer Prac Naukowych serii *„Bazy danych"*. Gdy w 2000 roku wydawany był pierwszy numer liczyliśmy na to, że będą jeszcze następne. I rzeczywiście, dzięki Autorom, którzy dzielą się wynikami swoich prac, od roku 2000 systematycznie corocznie publikowany jest jeden numer z tej serii. Autorami prac są głównie pracownicy, doktoranci a czasami studenci ostatnich lat studiów Wydziału Informatyki i Zarządzania Politechniki Wrocławskiej kierunku informatyka. Tematyka badawcza wszystkich prac jest związana z szeroko rozumianą problematyką baz danych i systemów baz danych takimi jak architektura baz danych, projektowanie, użytkowanie i bezpieczeństwo baz danych i systemów informatycznych, nowoczesne systemy zarządzania bazami danych, rozproszone systemy informatyczne, bazy danych w środowisku internetowym, mobilne i aktywne bazy danych, dziedzinowe bazy danych (medycyna, nauka, administracja publiczna), zagadnienia z wyszukiwania informacji itd. Wszystkie prace zgłoszone do czasopisma są recenzowane. Wydane dziesięć numerów czasopisma liczą łącznie 1400 stron i zawierają 78 artykułów.

Problematyka prac zamieszczonych w numerze dziesiątym dotyczy:

- zagadnień praktycznych związanych z hurtowniami danych i procesami ETL,
- wytwarzania aplikacji bazodanowych z wykorzystaniem technologii Hibernate (podejście obiektowo-relacyjne) oraz technologii SimpleJdbcTemplate z interfejsem proceduralnym,
- zagrożeń dla danych w aspekcie wad oprogramowania systemów komputerowych,
- cyfrowych archiwów danych,
- jakości danych zintegrowanych,
- możliwości stosowania protokołu TCP do konstrukcji niedostrzegalnych kanałów transmisyjnych dla celów kryptograficznych,
- zabezpieczeń oprogramowania przed nielegalnym rozpowszechnianiem,

- mechanizmów transakcji w środowisku usług webowych,
- usług sieciowych REST.

Dane gromadzone w hurtowniach danych powinny być trafnie wyselekcjonowane pod względem informacyjnym, tak by mogłyby stanowić podstawę do wykonywania poprawnych analiz. Zadania zasilania hurtowni danymi realizują procesy ETL (ang. *Extract*, *Transform*, *Load*). Tematyka pierwszej pracy dotyczy hurtowni danych i procesów ETL. Zostały przedstawione wybrane dedykowane im narzędzia programistyczne w SZBD Oracle 10g i Microsoft SQL Server 2005. Wszystkie rozpoznane mechanizmy zostały zaimplementowane i przetestowane w hurtowni danych zrealizowanej w obu środowiskach. Praca zakończona jest wnioskami z przeprowadzonych badań.

Kolejna praca zawiera porównanie dwóch sposobów wytwarzania aplikacji bazodanowych: technologii Hibernate (podejście obiektowo-relacyjne, logika biznesowa umiejscowiona jest po stronie aplikacji) i technologii SimpleJdbcTemplate z interfejsem proceduralnym (logika biznesowa jest po stronie bazy danych). Zaproponowano odpowiednie metryki i przy pomocy TKPROF – narzędzia ORACLE do śledzenia poleceń SQL i PL/SQL, porównywano czasy działania obu aplikacji, przedstawiono i omówiono wyniki przeprowadzonych testów.

Bezpieczeństwo systemów baz danych zależy od poziomu ochrony środowiska, w którym są eksploatowane. Autorzy trzeciej pracy poruszają problemy zagrożenia bezpieczeństwa systemów informatycznych spowodowane błędami w oprogramowaniu (w systemach operacyjnych, programach i aplikacjach użytkowych). Skala zjawiska występowania wad, które mogą być wykorzystane w celu bezprawnego działania w sieciach komputerowych, jest ogromna. Instalacja dostarczanych przez producenta nakładek systemowych nie rozwiązuje w pełni istniejącego problemu ponieważ nieustannie są wykrywane nowe wady, a nie wszyscy użytkownicy dbają o aktualizację wykorzystywanego oprogramowania.

Obecnie każda instytucja ma problem z ogromnym przyrostem danych. Gromadzone i przechowywane są coraz większe zasoby dokumentów, danych graficznych, tekstowych, multimedialnych. Dane cyfrowe łatwo utracić. Ręczne sterowanie wykonywaniem kopii zapasowych i archiwalnych narażone jest na ludzkie błędy, kosztowne i często nieskoordynowane w ramach firmy. Dodatkowym problemem jest właściwa klasyfikacja informacji. W czwartym artykule przedstawiono zagadnienia i problemy związane z archiwizacją danych, a szczególności z digitalizacją zasobów i elektronicznymi archiwami danych.

Konieczność wykorzystywania danych z różnych źródeł wymusiła powstawanie i rozwój systemów integracji danych. Potrzeba integracji danych szczególnie widoczna jest w dynamicznie rozwijających się instytucjach (np. w bankach) oraz w instytucjach rządowych. Potrzeba pomiaru jakości danych pojawia się wraz z napływem coraz większej ich ilości. Autor artykułu piątego przedstawia podstawowe zagadnienia związane z jakością danych zintegrowanych.

W kolejnej pracy przedstawiono szczegółową analizę protokołu TCP pod kątem konstrukcji niedostrzegalnych kanałów oraz wyniki badań autorów w zakresie konstruk-

cji niedostrzegalnych kanałów informacyjnych osadzonych w znormali-zowanych protokołach komunikacyjnych. Kanały takie mogą być wykorzystywane nie tylko do wytworzenia dodatkowych niezależnych kanałów przesyłowych, lecz również mogą być wykorzystane do realizacji protokołów związanych z ochroną danych lub weryfikacją użytkownika. Praca stanowi kontynuację i rozszerzenie wcześniejszych badań autorów nad protokołami komunikacyjnymi.

W następnej pracy zostały przedstawione wyniki badań Autorów w zakresie konstrukcji zabezpieczeń programów komputerowych przed nielegalnym rozpowszechnianiem. Zbadane i opisane zostały implementacje zabezpieczeń oprogramowania oparte na algorytmach: weryfikacji numeru seryjnego, weryfikacji numeru seryjnego i nazwy, weryfikacji pliku klucza, weryfikacji plikowego licznika uruchomień, weryfikacji rejestrowego licznika uruchomień, weryfikacji daty instalacji w rejestrze systemowym, identyfikacji klucza sprzętowego. Wybór algorytmów podyktowany był porównaniem implementacji podstawowych zabezpieczeń programów w języku C# oraz w językach C, Java, Assembler opisywanych we wcześniejszych pracach autorów. Wszystkie implementacje zostały zrealizowane na platformie .NET w języku C# w postaci samodzielnych aplikacji przygotowanych do szczegółowych badań porównawczych. W pracy przedstawiono również zaproponowany przez autorów algorytm obfuskacyjny wraz z opisem implementacji w języku C# oraz jego analizę.

W przypadku współpracujących ze sobą usług webowych transakcje trwają o wiele dłużej niż w tradycyjnych bazach danych. W siódmym artykule przedstawione są koncepcje architektoniczne wspierające koordynację i realizację transakcji w środowisku usług webowych a opartych na paradygmacie SOA. Autor szczegółowo przeanalizował model architektoniczny opracowany w ramach prac OASIS, W3C, WS-I i zdefiniowany w specyfikacjach WS-Coordination, WS-Atomic Transaction, WS-Business Activity oraz języka BPEL.

Duża złożoność protokołów sieciowych i standardów rodziny WS-* wykorzystywanych w budowie usług sieciowych spowodowała wzrost popularności architektury usług sieciowych wykorzystującej bezstanowe mechanizmy protokołu HTTP – REST (ang. *REpresentational State Transfer*), który jest dobrym rozwiązaniem do zastosowań w urządzeniach mobilnych. Autorzy ostatniego artykułu omawiają wady i zalety zastosowania wzorca architektury oprogramowania REST do budowy usług sieciowych.

Gorąco dziękuję wszystkim Autorom, którzy w ciągu minionych dziesięciu lat dzielili się z Czytelnikami wynikami swoich prac za pośrednictwem cyklu *Bazy danych – Prace Naukowe Instytutu Informatyki Politechniki Wrocławskiej*. Gorąco dziękuję wszystkim Recenzentom za ich wnikliwą i rzetelną ocenę tych prac.

Mam nadzieję na dalsze owocne lata współpracy.

Zygmunt Mazur

Prace Naukowe Instytutu Informatyki
Politechniki Wrocławskiej

Zeszyt 10 2009

hurtownie danych,
procesy ETL,
SZBD Oracle i MS SQL Server

Zbigniew STASZAK*,
Piotr OCHNIK

PORÓWNANIE MOŻLIWOŚCI PROCESÓW ETL W HURTOWNIACH DANYCH ORACLE I MICROSOFT

W pracy przedstawiono wybrane zagadnienia dotyczące hurtowni danych i, związanych z nimi, procesów ETL. Pierwsza część pracy opisuje zagadnienia teoretyczne takie jak architektura hurtowni oraz reprezentacja danych. Druga część pracy to opis zagadnień praktycznych związanych z hurtowniami danych i procesami ETL. Przedstawiono w niej rozpoznane mechanizmy ETL oraz wybrane dedykowane im narzędzia programistyczne w badanych SZBD Oracle 10g i Microsoft SQL Server 2005. Wszystkie rozpoznane mechanizmy zostały zaimplementowane w testowej hurtowni danych zrealizowanej w obu środowiskach. Zbadano ich działanie i możliwości. Praca zakończona jest wnioskami z przeprowadzonych badań.

1. WPROWADZENIE

Jedną z podstaw funkcjonowania większości średnich i dużych przedsiębiorstw na rynku jest dostęp do informacji. Aby zwiększyć swoją konkurencyjność wobec innych podmiotów i organizacji, wymaga się aby przy podejmowaniu kroków czy decyzji biznesowych możliwe było posiadanie wiarygodnych informacji. Informacje te powinny odzwierciedlać rzeczywisty obraz całego przedsiębiorstwa. Uzyskanie tak kompletnych danych w przypadku systemów transakcyjnych jest bardzo żmudne z powodu konieczności pobierania danych z wielu źródeł. Konieczność ciągłej aktualizacji danych, brak czasowego punktu odniesienia oraz brak optymalizacji przetwarzania analitycznego spowodował powstanie hurtowni danych (ang. *data warehouse*). Istnieje wiele definicji hurtowni danych. Poniżej przedstawiono kilka wybranych definicji.

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

„Hurtownia danych jest to analityczna baza danych wykorzystywana jako podstawa systemu wspomagającego podejmowanie decyzji" [Poe 2000].

„Hurtownia danych inaczej składnica danych, przechowywanych w postaci ujednoliconej i sprawdzonej, niezbędnych do sprawnego wspomagania decyzji" [Hurtownia 2008].

„Hurtownia danych jest wydzieloną centralną bazą danych zbierającą informacje służące do zarządzania organizacją. Baza ta jest odizolowana od baz operacyjnych a jej struktura i użyte do jej budowy narzędzia powinny być zoptymalizowane pod kątem przetwarzania analitycznego" [Wolski 2008].

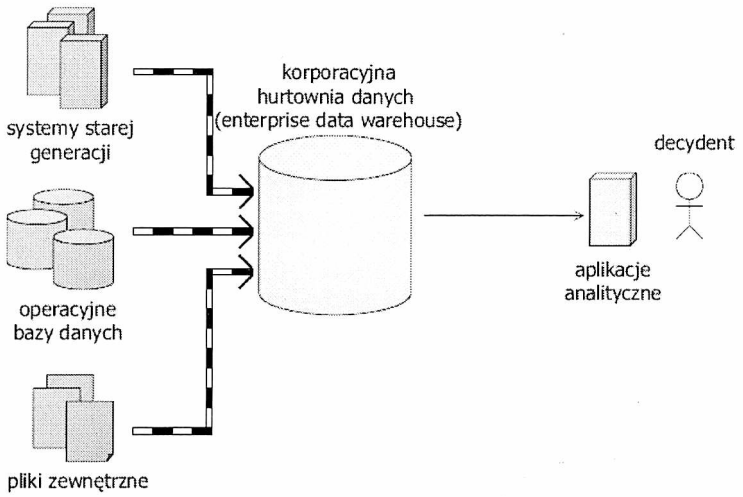
Pierwotne pojęcie hurtowni danych zostało zdefiniowane przez Billa Inmona – „ojca hurtowni", który określa hurtownię jako „zintegrowany, przedmiotowo-zorientowany, zmienny w czasie, nieulotny, zbiór danych służący do wspomagania podejmowania decyzji" [Inmon 1994]. Zintegrowany zbiór oznacza, że dane pobierane są z różnych źródeł, które finalnie tworzą określoną całość. Przedmiotowo-zorientowany oznacza przetwarzanie danych dla konkretnego wycinka rzeczywistości. Nieulotność powoduje, iż pobrane dane nie są modyfikowane lub usuwane. Zmienność w czasie mówi o danych, które powiązane są z określonym punktem w czasie lub z danym okresem.

Aby umożliwić uzyskiwanie informacje z hurtowni, należy uprzednio zasilić ją danymi. Dane te powinny być trafnie wyselekcjonowane pod względem informacyjnym, przez co mogą stanowić podstawę do wykonywania poprawnych analiz. Zadania zasilania hurtowni danymi realizują procesy ETL (ang. *Extract*, *Transform*, *Load*). Ekstrakcja (wydobywanie) danych jest etapem, w ramach którego pozyskiwane są dane z różnego rodzaju źródeł. Możliwe są źródła zewnętrzne (Internet, zewnętrzne bazy danych), wewnętrzne (dokumenty, wewnętrzne bazy danych), produkcyjne (operacyjne bazy danych) oraz zarchiwizowane (dane historyczne, zarchiwizowane). Podczas transformacji danych następuje ich integracja oraz weryfikacja pod względem poprawności (możliwa jest także ich walidacja). Wydobyte dane muszą być „czyszczone". Podlegają one indeksowaniu, nadawana jest im etykieta. W rezultacie uzyskiwane są z pobranego obrazu spójne dane. Wydobyte i przetransformowane dane ładowane są do utworzonej wcześniej hurtowni danych. Procesy ETL, w zależności od środowiska, w którym zaimplementowano hurtownię, wspomagane są przez różnorodne narzędzia programistyczne.

2. ARCHITEKTURA HURTOWNI DANYCH

Wyróżniane są trzy główne rodzaje architektur hurtowni danych. Pierwszym z nich jest model podstawowy (rys. 1). Składa się on z trzech elementów. Pierwszym elementem są dane ładowane ze źródeł, którymi mogą być między innymi pliki zewnętrzne, operacyjne bazy danych, inne systemy. Drugim elementem modelu jest hurtownia danych. Dane są bezpośrednio ładowane do i pobierane z hurtowni. Ostatnim elementem są aplikacje analityczne umożliwiające interakcję pomiędzy użytkownikiem a hurtownią. Są to narzędzia, dzięki którym użytkownik w łatwy i wygodny

dla siebie sposób przedstawia interesujące go informacje. Tworzone są zestawienia, raporty w zależności od zapotrzebowania w danej chwili na analizę. Cechą wyróżniającą ten model jest bezpośrednie ładowanie danych. Jest to rzadko stosowany model z powodu braku możliwości przetwarzania danych wprowadzanych do hurtowni danych [Hurtownia 2008], [Kimball 2002].

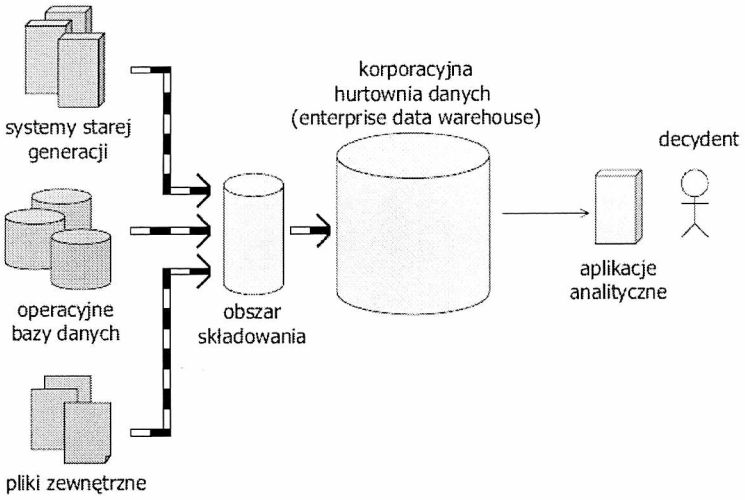


Rys. 1. Model podstawowy [PLOUG 2008]

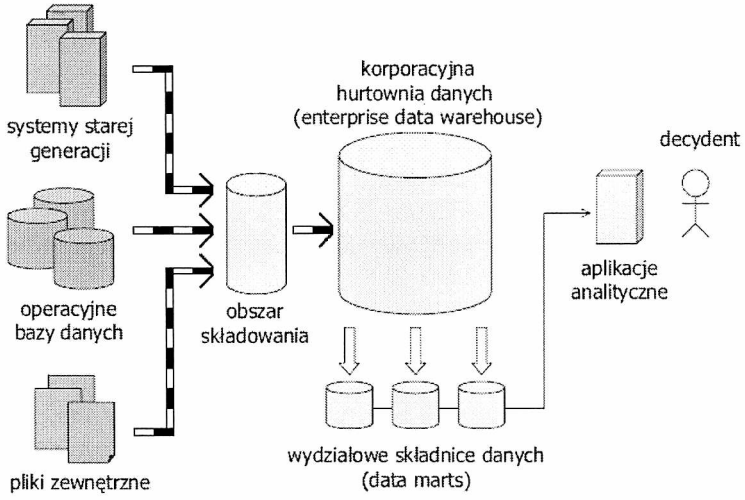
Model drugi (rys. 2) oprócz elementów analogicznych do poprzedniego, czyli: danych, hurtowni danych oraz aplikacji analitycznych posiada obszar składowania danych. Ładowane są do niego dane o różnej postaci, należące do różnych typów. W obszarze składowania, dane są wstępnie przetwarzane i podlegają kontroli. Wykorzystywane są w tym kroku procesy ETL [Walker 2006]. Integracja danych umożliwia wprowadzenie danych do hurtowni. Użytkownik końcowy ma w ten sposób dostęp do wyselekcjonowanych danych [Gorawski 2008].

Ostatnim modelem jest architektura zawierająca jak w poprzednim przykładzie: dane, obszar składowania, hurtownie danych i aplikacje analityczne (rys. 3). Różnicą jest dodanie do tej architektury składnic danych (ang. *data marts*). Składnica danych jest to wyszczególnione, przedmiotowo-zorientowane repozytorium danych, mające na celu wyszczególnienie konkretnych danych związanych z konkretną częścią działalności firmy. W przedsiębiorstwach stosuje się politykę, zgodnie z którą każdy rozbudowany dział firmy (marketing, kontrola, sprzedaż) ma swoją oddzielną składnicę danych. Składnice danych zazwyczaj zorganizowane są w postaci schematu gwiazdy, zbudowanej z jednej tabeli faktów i wielu tabel wymiarów. Pojęcia te zostaną omówione w dalszej części pracy. Składnice danych zawierają dane, które są mocno zagregowane i zdenormalizowane. Ich źródłem jest zazwyczaj główna hurtownia danych. Jako że są one tworzone dla konkretnego użytkownika (działu), ich treści są

powtarzane (dane są dublowane) w innych składnicach [Connolly 2004], [Gorawski 2004], [Gorawski 2008].



Rys. 2. Architektura z obszarem składowania [PLOUG 2008]

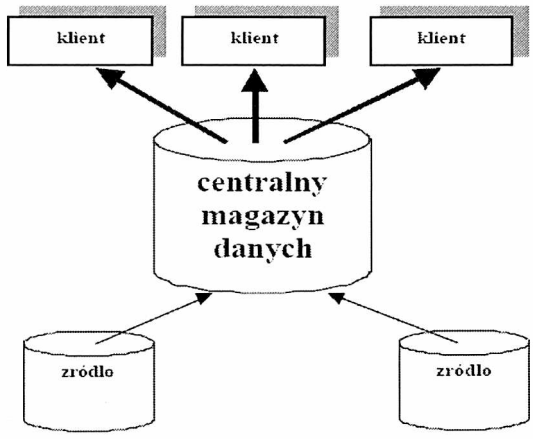


Rys. 3. Architektura ze składnicami danych [PLOUG 2008]

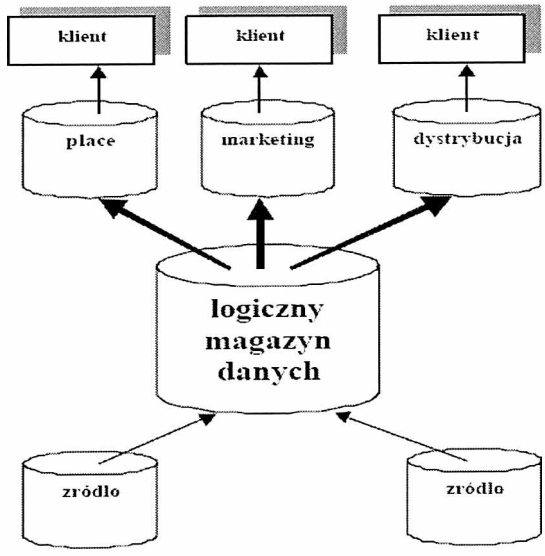
Hurtownie mogą być zbudowane na trzy sposoby. Pierwszym jest architektura scentralizowana, drugim federacyjna a trzecim warstwowa [Jarke 2000]. W pierwszym przypadku (rys. 4) występuje jeden magazyn danych (hurtownia), co oznacza gromadzenie

danych w jednym miejscu. Dzięki temu ułatwiony jest dostęp do interesujących danych, ale w konsekwencji czas dostępu do nich może ulec znacznemu wydłużeniu.

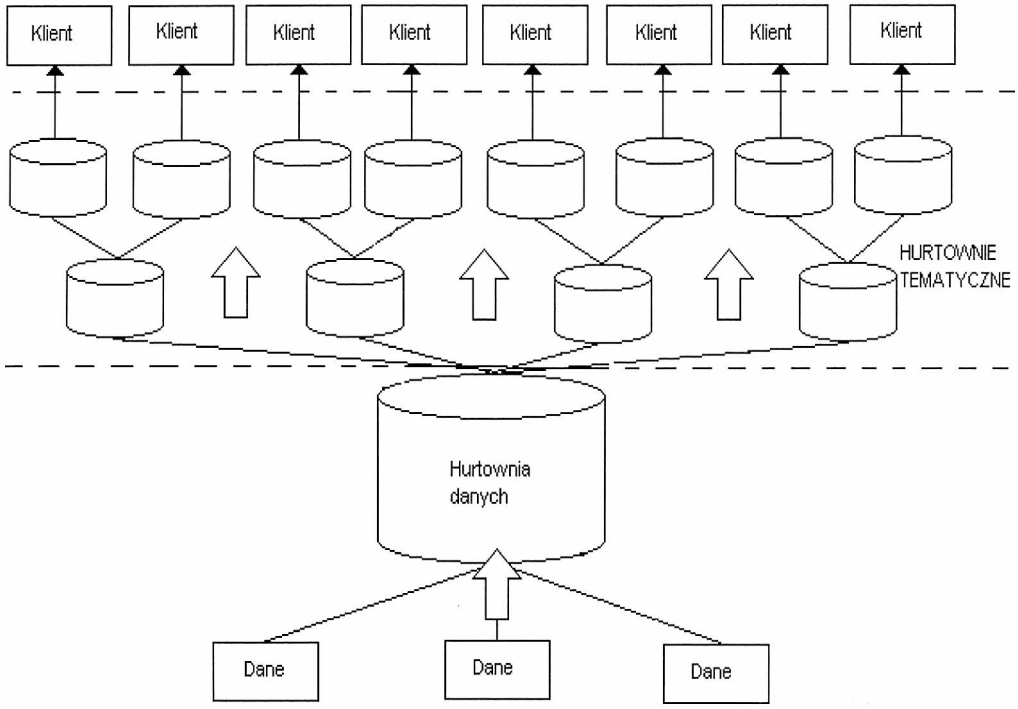
Federacyjna hurtownia danych (rys. 5) polega na utworzeniu logicznej hurtowni przy fizycznym przechowywaniu danych w oddzielnych składnicach danych. Na rysunku 5 przedstawiono podział, w którym składnice danych przechowują dane dla konkretnych działów w przykładowej firmie (płace, marketing, dystrybucja).



Rys. 4. Scentralizowana hurtownia danych [Wrembel 2000]



Rys. 5. Federacyjna hurtownia danych [Wrembel 2000]



Rys. 6. Warstwowa hurtownia danych

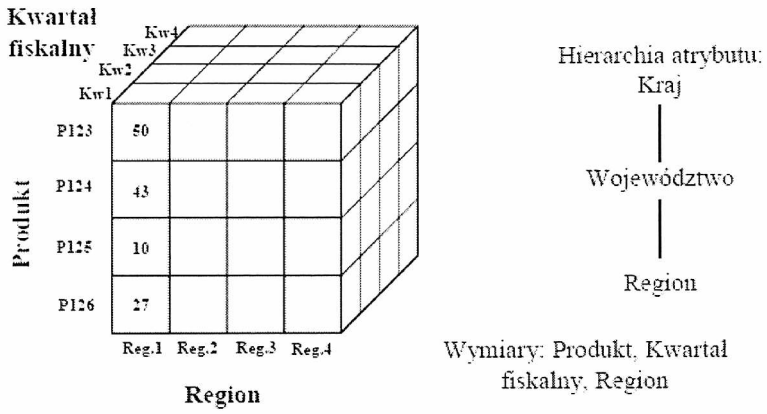
Warstwowa hurtownia danych (rys. 6) zbudowana jest z głównej hurtowni danych wraz z uzupełniającymi ją warstwowo połączonymi tematycznie hurtowniami. Wszystkie hurtownie tematyczne niezależnie od poziomu są zmaterializowane. Wraz ze wzrostem poziomu, na którym znajduje się hurtownia tematyczna, występuje wzrost stopnia agregacji danych. Dane obliczane są na podstawie danych z hurtowni umieszczonej na poprzedzającym ją poziomie [Inmon 2005], [Kimball 2002].

3. REPREZENTACJA DANYCH

W hurtowniach danych występuje kilka typów danych. Mogą to być dane operacyjne podlegające zmianom w zależności od momentu czasowego. Ich aktualizacja następuje w sposób ciągły. Mogą to być dane sumaryczne (dane operacyjne w postaci zagregowanej) lub dane referencyjne ograniczające rozmiar hurtowni oraz wspomagające zarządzanie. Mogą to w końcu być metadane czyli „dane o danych". Przedstawiają one zależności między strukturami i dodatkowo opisują je.

Z procesem modelowania danych łączy się pojęcie faktów. Są to ilościowe dane scharakteryzowane przez miary, których miejscem przechowywania są tablice faktów.

Reprezentują one jakiś wycinek danej rzeczywistości. Wyróżniane są także wymiary, przez które rozumiane są wielkości (dane), które opisują fakty według sposobu ich zaistnienia. Pojedynczy wymiar dodatkowo ma przypisane atrybuty. Fakty i wymiary tworzą model wielowymiarowy (rys. 7).

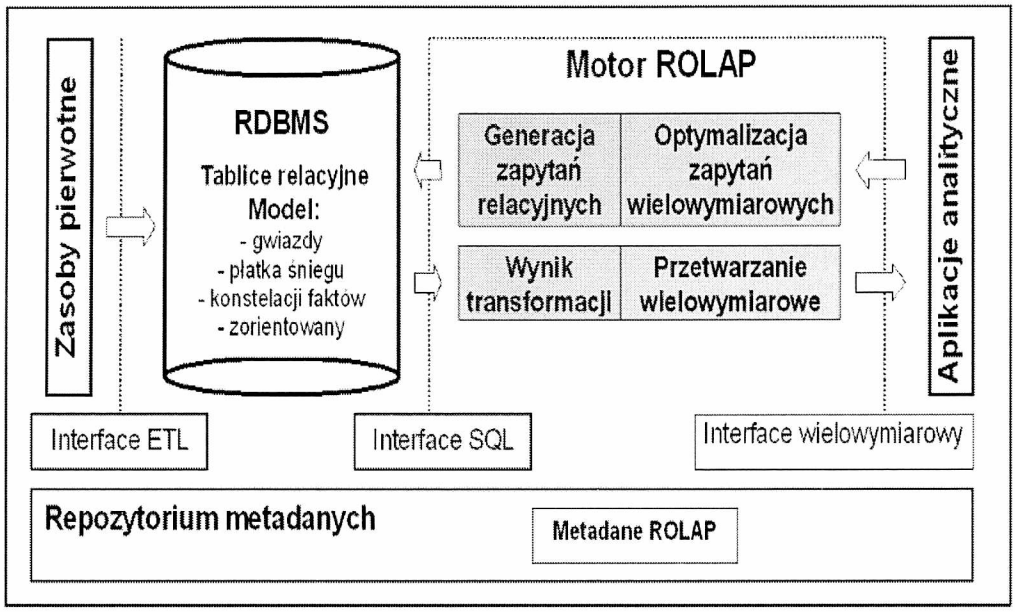


Rys. 7. Kostka wielowymiarowa [Bembenik 2002]

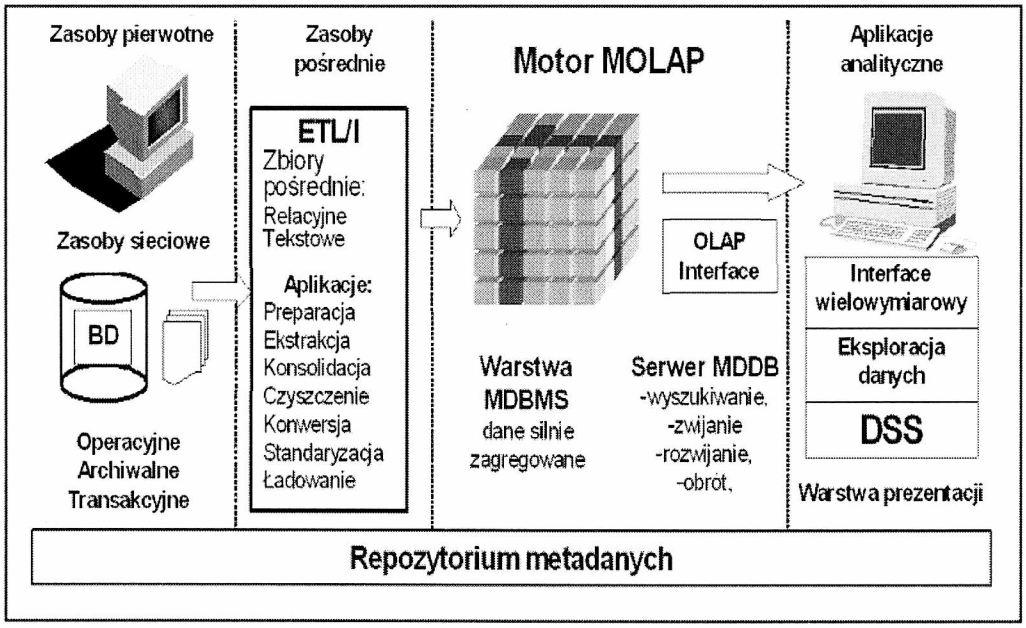
Pomiędzy poziomami w wymiarach zachodzą zależności hierarchii rodzic–dziecko. Najczęściej występującym przykładem hierarchii jest hierarchia czasu (rys. 8).

Rys. 8. Przykładowa hierarchia czasu

Każda zaprojektowana hurtownia danych ma swoją logiczną strukturę. Może nią być ROLAP (OLAP relacyjny; ang. *Relational On-Line Analytic Processing*). Dostęp do danych realizowany jest za pomocą zapytań SQL'owych. Najczęściej stosowanymi modelami przetwarzania analitycznego są wspomniany już ROLAP i MOLAP (OLAP wielowymiarowy, ang. *Multidimensional On-Line Analytic Processing*). Koncepcja pierwszego z nich oparta jest o relacyjną bazę danych przez co do jego budowy stosuje się schemat gwiazdy (rys. 14) lub płatka śniegu (rys. 15). Elementy składowe modelu ROLAP zostały przedstawione na rysunku 9.



Rys. 9. Elementy ROLAP [Gorawski 2008]



Rys. 10. Elementy MOLAP [Gorawski 2008]

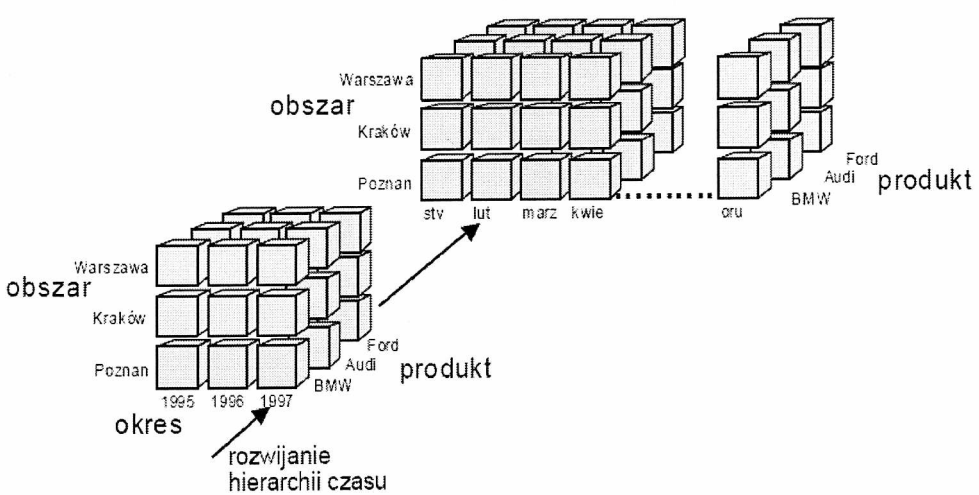
Model ROLAP posiada następujące cechy:

- wykorzystanie relacyjnych baz do organizacji struktur danych,
- agregacja danych,
- odpowiedzią na zapytania jest wykonanie transformowanych operacji,
- dostęp do dowolnych danych *ad hoc* lub za pomocą złożonych zapytań,
- niska wydajność,
- długi czas przetwarzania.

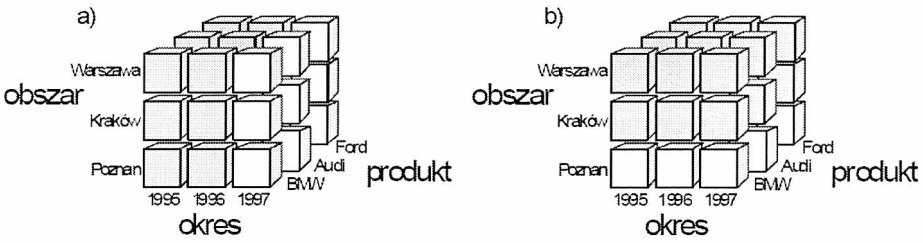
Kolejnym modelem przetwarzania analitycznego jest MOLAP. W modelu tym zastosowana jest baza wielowymiarowa MDDB (ang. *Multidimensional Database*). Dane są przechowywane wielowymiarowo (rys. 7). Występuje ich pełna agregacja. Elementy składowe MOLAP zostały przedstawione na rysunku 10.

Analiza danych w modelu MOLAP wykonywana jest za pomocą następujących operacji:

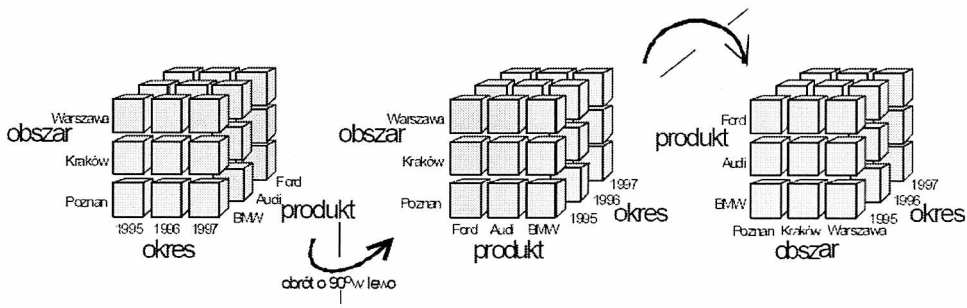
- wyznaczenie głównego punktu – wskazanie miary i wymiaru dla reprezentowanej miary,
- rozwijanie – stosowane w celu dokonania głębszej analizy hierarchii wymiaru, polega na zagłębianiu się w konkretny wymiar (rys. 11),
- zwijanie – operacja odwrotna do poprzedniej. Nie występuje zagłębianie się w wymiar, a jedynie koncentracja na danych wyższego poziomu,
- wycinanie – zawężenie analizy poprzez wybór konkretnych wartości (rys. 12),
- obracanie – różne układy, w których przedstawiane są dane (rys. 13),
- obliczanie rankingu – segregowanie danych w wymiarze według konkretnych wartości miar.



Rys. 11. Rozwijanie hierarchii wymiarów [Wrembel 2000]

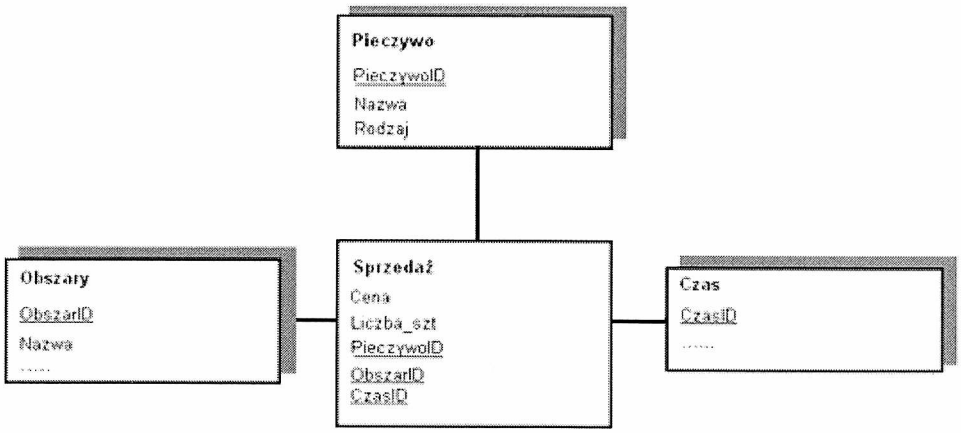


Rys. 12. Dane wycinane w różnych wymiarach [Wrembel 2000]



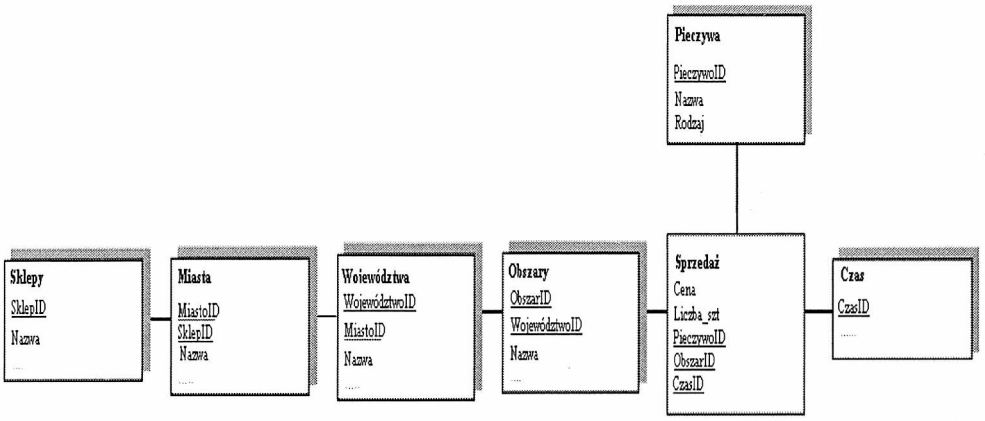
Rys. 13. Operacja obracania [Wrembel 2000]

Poza modelami ROLAP i MOLAP można także wyróżnić model DOLAP (*OLAP Desktop*) inaczej nazywany motorem MOLAP. W modelu tym nie jest stosowany serwer a dane są przechowywane lokalnie. Kolejnym modelem jest HOLAP (OLAP hybrydowy, ang. *Hybrid OLAP*) posiadający mieszaną architekturę. Jest to nic innego jak OLAP rozbudowany o mechanizmy występujące w ROLAP i MOLAP.



Rys. 14. Schemat gwiazdy

Model gwiazdy zbudowany jest z tabeli faktów wraz z tabelami wymiarów. W przykładzie przedstawionym na rysunku 14, faktem jest tabela Sprzedaży. Jest ona tabelą centralną, której klucz główny jest kluczem złożonym z kluczy głównych tabel wymiarów (Czas, Obszary, Pieczywo). Cechą charakterystyczną tego schematu jest krótki czas oczekiwania na wynik zapytania oraz możliwość występowania tabel poniżej trzeciej postaci normalnej (3PN).



Rys. 15. Schemat płatka śniegu

Drugim modelem jest płatek śniegu. Na rysunku 15 przedstawiono bardzo prosty schemat zgodny z modelem płatka śniegu. Zbudowany jest on z jednej tabeli faktów i z wielu tabel wymiarów. Tabele wymiarów zawierają hierarchię Obszaru. Cechą wyróżniającą ten model jest występowanie danych wymiarów w trzeciej postaci normalnej przy niezmienionej postaci danych tabeli faktu.

4. PROCESY ETL

Różnorodność źródeł danych wpływa na jakość informacji - im większa ich liczba tym bardziej szczegółowe i poprawne są analizy. Zasilanie hurtowni danymi z heterogennych źródeł stwarza konkretne problemy związane z procesem ich pobierania, transformacji i ładowaniem. Aby skrócić czas zasilania i umożliwić przetworzenie danych przed ich załadowaniem do hurtowni, zaimplementowano mechanizmy wspomagające te procesy [Kimball 2004], [Eckerson 2003]. Mechanizmy te nazwano procesami ETL od trzech kolejnych realizowanych w ramach ich działania faz, odczytu danych ze źródeł (ang. *Extract*), transformacji danych (ang. *Transform*) i wczytania danych do hurtowni (ang. *Load*).

Pierwszą fazą procesu ETL jest odczytanie danych z dostępnych zidentyfikowanych źródeł. Jak pokazują przeprowadzone przez *The Datawarehouse Institute* (na zlecenie *DataMirror Corporation*, *Business Objects*, *Informatica Corporation* i *Humingbird Ltd*)

badania, ich liczba w średnim podmiocie gospodarczym wynosi średnio dwanaście. Dane źródłowe w zależności od systemu zaimplementowanego w firmie mogą pochodzić z baz danych dowolnego rodzaju, dokumentów o różnym formacie, stron internetowych czy też systemów wspomagających zarządzanie przedsiębiorstwem. Najczęściej wykorzystywanym źródłem do zasilenia hurtowni danych (tabela 1) są relacyjne bazy danych.

Drugą fazą procesu ETL jest transformacja danych. Etap ten należy do najbardziej złożonych a w jego wyniku pobrane z heterogenicznych źródeł dane przetwarzane są do spójnego, jednolitego modelu. Dane są czyszczone, tzn. następuje ich selekcja ze źródeł przez wybór ich konkretnych wartości, wartości zdublowane zostają usunięte, generowane są nowe klucze, zakodowane wartości zostają ujednolicone do formatu zastosowanego w hurtowni danych, ujednolicony zostaje sposób prezentacji danych. Niekiedy następuje zastąpienie wartości z różnych źródeł jedną obliczoną wartością. Na tym etapie wykonywane są wymienione wcześniej operacje (model MOLAP) takie jak wyznaczenie głównego punktu, zwijanie, rozwijanie, obracanie, wycinanie i obliczanie rankingu [ETL 2003].

Tabela 1. Częstotliwość wykorzystywania źródeł podczas procesu ekstrakcji [Błasiński 2005]

Rodzaj źródła danych	Częstotliwość
Relacyjne bazy danych	89%
Płaskie pliki tekstowe	81%
Systemy *mainframe*	65%
Systemy klasy ERP, CRM	39%
Systemy replikacji danych	15%
Strony WWW	15%
Dokumenty XML	15%
Komunikaty systemów integracji aplikacji (EAI)	12%
Inne	4%

Po przetworzeniu danych następuje trzecia faza procesu ETL tzn. ładowanie danych do hurtowni lub składnicy danych.

W zależności od systemu bazy danych liczba mechanizmów i narzędzi wspomagających procesy ETL jest różna. W niniejszej pracy przedstawione zostały porównawcze wyniki badań dotyczące implementacji mechanizmów ETL w dwóch środowiskach: Oracle 10g w wersji 10.2.0.1.0 oraz Microsoft SQL Server 2005. Głównym kryterium wyboru tych wersji były ich zbliżone wymagania sprzętowe.

4.1. PROCESY ETL W SYSTEMIE ORACLE 10g

System Oracle 10g powstał jako uniwersalny system bazodanowy do zastosowań w małych, średnich i dużych przedsiębiorstwach. W systemie tym można wyróżnić trzy edycje: Standard, Personal i Enterprise. Niniejsza praca dotyczy edycji Enterprise,

która charakteryzuje się największą liczbą zaimplementowanych funkcji przy zachowaniu stabilności systemu dla różnych konfiguracji sprzętowych.

Każdy z SZBD Oracle'a prócz elementów „odziedziczonych" z poprzedniej wersji wprowadza nowe mechanizmy wspomagające procesy ETL. Dodatkowo każdemu ze środowisk dedykowane były oddzielne narzędzia Oracle'a. Przeprowadzone rozpoznanie wersji SZBD Oracle 10g (edycja Enterprise) pozwoliło na identyfikację przedstawionych poniżej mechanizmów i narzędzi wspomagających procesy ETL.

- Export/Import,
- Transportable Tablespaces (przenaszalne przestrzenie tabel),
- MERGE,
- SQL*Loader,
- pakiet systemowy UTL_FILE,
- Change Data Capture,
- Oracle Streams (mechanizm strumieni),
- Oracle Data Pump,
- Table Functions (funkcje tablicowe),
- Multi-Table Insert (wstawianie wielotablicowe),
- External Tables (tabele zewnętrzne),
- Materialized View (perspektywy materializowane),
- Parallel DML (równoległy DML),
- łącza bazodanowe,
- Oracle Warehouse Builder.

Powyższe narzędzia i mechanizmy będą obiektem eksperymentów przedstawionych w dalszej części pracy.

4.2. PROCESY ETL W SYSTEMIE MS SQL SERVER 2005

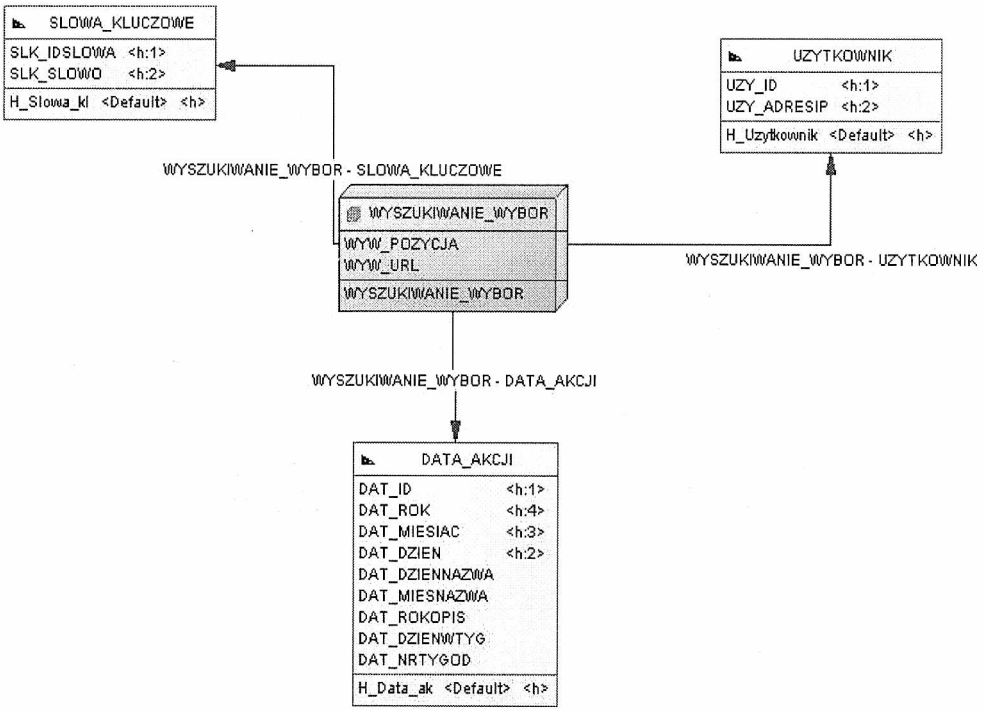
SZBD Microsoft SQL Server 2005 charakteryzuje się odmiennym, w stosunku do SZBD Oracle 10g, sposobem dostarczania mechanizmów ETL. W niniejszej pracy badanym serwerem jest MS SQL Server Enterprise Edition 2005. Poniżej wyróżnione zostały zidentyfikowane mechanizmy oraz moduły dotyczące ETL, zdefiniowane w MS SQL Server 2005.

- mechanizm Bulk Copy (BCP),
- BULK INSERT,
- OPENROWSET,
- przetwarzanie równoległe,
- perspektywy zmaterializowane,
- SQL Server Integration Services (SSIS).

Podobnie jak dla SZBD Oracle, wyróżnione powyżej mechanizmy będą obiektem przedstawionych dalej eksperymentów.

5. TESTOWA HURTOWNIA DANYCH

W niniejszej pracy wykorzystany zostanie wycinek rzeczywistości opisujący słowa kluczowe wykorzystywane podczas wyszukiwania w Internecie oraz pozycje wybranych przez użytkownika stron internetowych. Dane wykorzystane zostaną do eksperymentów badających możliwości procesów ETL w hurtowni danych oferowanych przez Oracle i Microsoft. Testowa hurtownia zrealizowana została w obu badanych środowiskach w oparciu o model ROLAP z wykorzystaniem modelowania punktowego [Todman 2003]. Na rysunku 16 przedstawiono diagram powstałej hurtowni testowej.



Rys. 16. Diagram testowej hurtowni danych
wygenerowany za pomocą PowerDesigner Studio Enterprise Trial

Dla tak zdefiniowanej hurtowni danych przeprowadzone zostały następujące eksperymenty:

Oracle 10g

- wywołanie SQL*Loader'a dla pliku płaskiego txt z wykorzystaniem pliku kontrolnego definiującego parametry,

- wywołanie SQL*Loader'a dla pliku xls z wykorzystaniem pliku kontrolnego definiującego parametry,
- wywołanie SQL*Loader'a za pomocą Oracle Enterprise Manager dla pliku xls metodą ścieżki konwencjonalnej,
- wywołanie SQL*Loader'a za pomocą Oracle Enterprise Manager dla pliku xls metodą ścieżki bezpośredniej,
- wywołanie SQL*Loader'a za pomocą Oracle Enterprise Manager dla pliku xls metodą równoległej bezpośredniej,
- dostęp do pliku płaskiego txt za pomocą External Tables,
- dostęp do pliku płaskiego txt z nagłówkami za pomocą External Tables,
- dostęp do pliku płaskiego txt o dużym rozmiarze za pomocą External Tables ze zdefiniowaną maską dla pola daty,
- wywołanie eksportu (EXP) w trybie interaktywnego dialogu,
- wywołanie eksportu (EXP) z przypisanymi wartościami do poszczególnych parametrów,
- wywołanie eksportu (EXP) z wykorzystaniem pliku z wartościami parametrów,
- wywołanie importu (IMP) w trybie interaktywnego dialogu,
- wywołanie importu (IMP) z przypisanymi wartościami do poszczególnych parametrów,
- wywołanie importu (IMP) z wykorzystaniem pliku z wartościami parametrów,
- wywołanie Transportable Tablespaces z lini komend dla pojedynczej tabeli przy zdefiniowanym warunku,
- wywołanie Transportable Tablespaces z Oracle Enterprise Manager,
- zaimplementowanie łącza bazodanowego z MySQL za pomocą NetManager,
- zaimplementowanie łącza bazodanowego z MySQL za pomocą Oracle Enterprise Manager,
- zaimplementowanie łącza bazodanowego z MySQL za pomocą Oracle Heterogeneous Services,
- wywołanie EXPDP i IMPDP z linii komend dla tabeli,
- wywołanie EXPDP i IMPDP z linii komend dla schematu,
- wywołanie EXPDP i IMPDP z linii komend dla bazy,
- wywołanie EXPDP i IMPDP z Oracle Enterprise Manager dla tabel,
- wywołanie mechanizmu MERGE dla importu danych z External Tables do hurtowni,
- wywołanie pakietu systemowego UTL_FILE dla pobrania pierwszego wiersza z pliku płaskiego,
- wywołanie pakietu systemowego UTL_FILE do zapisu danych znajdujących się w buforze do pliku,
- wywołanie pakietu systemowego UTL_FILE dla dodania nowych danych,
- wykonanie INSERT ALL ładujących dane z tabeli zewnętrznej,

- wykonanie INSERT FIRST ładujących dane z tabeli zewnętrznej,
- wykonanie modyfikacji danych przy wyłączonym i włączonym równoległym DML'em dla tabeli Użytkownik,
- wykonanie modyfikacji danych przy wyłączonym i włączonym równoległym DML'em dla tabeli Slowa_kluczowe,
- wykonanie perspektywy zmaterializowanej opartej na kluczach głównych dla tabeli bez procesu odświeżania w SQL*Plus,
- wykonanie perspektywy zmaterializowanej opartej na ROWID dla tabeli za pomocą Oracle Enterprise Manager,
- wykonanie Oracle Data Capture dla wybranej tabeli,
- wykonanie mechanizmu strumieni dla wybranej tabeli,
- import pliku płaskiego za pomocą Oracle Warehouse Builder,
- tworzenie tabel zewnętrznych za pomocą Oracle Warehouse Builder,
- import plików typu xls za pomocą Oracle Warehouse Builder,
- export/import metadanych w Oracle Warehouse Builder,
- wykonanie perspektywy zmaterializowanej w Oracle Warehouse Builder,
- wykonanie łącza bazodanowego w Oracle Warehouse Builder,
- wykonanie funkcji w Oracle Warehouse Builder,
- wykonanie procesu mapowanie z zastosowanie operatora tabel w Oracle Warehouse Builder,
- wykonanie procesu mapowanie z zastosowanie operatora transformacji w Oracle Warehouse Builder,
- wykonaniu definicji procesu przepływu danych w Oracle Warehouse Builder,
- zaimportowanie danych za pomocą Oracle Warehouse Builder.

MS SQL Server 2005

- wykonanie transformacji i eksportu z tabeli tymczasowej do zdefiniowanej hurtowni za pomocą BCP i SQL'a,
- wykonanie mechanizmu BCP z bezpośrednim wykorzystaniem SQL'a,
- wykonanie importu pliku typu csv za pomocą BULK INSERT,
- wykonanie importu pliku typu txt za pomocą BULK INSERT,
- wykonanie dodania danych do tabeli za pomocą OPENROWSET,
- wykonanie perspektywy zmaterializowanej dla wybranej tabeli,
- wykonanie dla pliku płaskiego Import and Export Wizards SSIS,
- wykonanie BULK INSERT w SSIS,
- wykorzystanie transformacji Conditional Split,
- wykorzystanie transformacji Derived Kolumn,
- wykorzystanie transformacji Merge i Sort,
- wykorzystanie transformacji Audit,
- wykorzystanie Data Conversion.

6. PORÓWNANIE MOŻLIWOŚCI PROCESÓW ETL W HURTOWNIACH ORACLE I MICROSOFT

W rozdziale przedstawimy wnioski oraz spostrzeżenia dotyczące realizacji procesów ETL wynikające z wykonanych eksperymentów. Eksperymenty te dotyczyły testowej hurtowni danych zaimplementowanej zarówno w SZBD Oracle 10g, jak i w SZBD Microsoft SQL Server 2005. Całość wykonanych badań podsumowana została w tabeli 2. W tabeli tej przyjęto za wzorcowe, mechanizmy zaimplementowane w SZBD Oracle'a 10g i to do nich porównywano implementacje w SZBD MS SQL Server. Środowisko Oracle'a zostało rozdzielone na SZBD, gdzie mechanizmy definiowane są z poziomu linii komend oraz Oracle Enterprise Manager będący web'ową nakładką do zarządzania. Powodem tego były różnice przy definicji procesów ETL. MS Server SQL z SSIS potraktowany został jako jednolita całość.

W tabeli 2a przedstawiono dwa najbardziej uniwersalne mechanizmy procesów ETL. Pierwszym z nich jest External Tables. Jest to interfejs dla pobierania danych z plików płaskich, wykorzystywany w połączeniu z innymi mechanizmami, np. MERGE. Pozwala on na dowolne transformacje danych. Drugim mechanizmem, także szeroko stosowanym, jest Export/Import. W odróżnieniu od poprzedniego pozwala on na załadowanie danych do tabeli docelowej bez użycia dodatkowych procesów transformujących. Oba mechanizmy mają jednak ograniczenie jakim jest rozmiar importowanych danych. Aby możliwe było użycie plików o dużych rozmiarach należy zdefiniować typy CLOB (dla danych znakowych).

W tabeli 2b zawarte są mechanizmy odpowiadające mechanizmowi EXP/IMP (np. Oracle Data Pump), jednak szybsze od niego. W przypadku transformacji i mapowań mechanizmy te definiowane są tylko i wyłącznie w narzędziach obu środowisk. Stosowane w obu przypadkach edytory graficzne do modelowania procesów pozwalają na łatwiejsze definiowanie procesów ETL w porównaniu z definiowaniem ich z linii komend.

W tabeli 2c pokazano, że nie wszystkie narzędzia ETL pozwalają na stworzenie łącza z innym SZBD. Bardzo niewygodną funkcjonalność w tym zakresie posiada Oracle Enterprise Manager. Pozwala ona jedynie na połączenie się z bazami Oracle. Brakującym mechanizmem, który jest pomocny w definiowaniu ładowania danych przy określonych warunkach jest MERGE. Niestety w SQL Server 2005 funkcjonalność ta nie istnieje. Minusem mechanizmu SQL*Loader jest brak wspomagania plików typu xls przy ładowaniu ich do źródeł docelowych. Szczególnie widoczne to było podczas ładowania danych za pomocą Oracle Enterprise Manager. Podgląd źródeł xls powodował wyświetlenie tzw. „krzaczków", przez co nie można było jednoznacznie ustawić formatu wczytywanych do hurtowni danych.

W tabeli 2d przedstawiono ciekawy mechanizm jakim jest równoległy DML. Jego zastosowanie w środowisku Oracle pozwalało na zauważenie czasowych różnić podczas wykonywania modyfikacji na danych w tabelach. W niektórych przypadkach jednak implementacja mechanizmu była bardzo pracochłonna (Oracle Streams).

Tabela 2a. Zidentyfikowane mechanizmy dla procesów ETL w poszczególnych środowiskach i narzędziach

	Oracle 10g R2	Oracle Enterprise Manager 10g	OWB 10g	MsSQL Server 2005
External Tables	Istnieje. Importowane pliki za pomocą SQL* Loadera o max. rozmiarze do 20 MB. Przemapowanie danych źródłowych z tabeli zewnętrznej do tabeli hurtowni bardzo proste, wystarczy zastosować polecenie INSERT.	Nie istnieje	Istnieje. Bardzo prosty sposób zaimportowania plików płaskich (brak możliwości importowania innego rodzaju plików). Definicja w dwóch etapach za pomocą wizard'a. Pierwszy etap to zdefiniowanie lokalizacji źródła i struktury metadanych. Dodatkowo możliwość określenia wartości domyślnych, masek dla konkretnych pól. Drugi etap to zlilnkowanie stworzonej struktury z plikami źródłowymi.	Nie istnieje
Export/Import	Istnieje. W przypadku wywołania z zdefiniowanymi wartościami poszczególnych parametrów, umożliwia za pomocą parametru query określenie warunków dla eksportowanych danych. W przypadku importu danych nie ma możliwości zastosowania warunków dla danych. Mechanizm optymalny dla nieskomplikowanych transformacji.	Brak typowego mechanizmu EXP/IMP. Odpowiednikiem mechanizmu jest Metadata Loader. Znajduje zastosowanie przy tworzeniu kopii zapasowych metadanych, przenoszeniu danych do nowego repozytorium, migracji pomiędzy wersjami OWB. Podczas eksportu możliwy wybór tylko języka dla danych i obiektów. Dla Importu możliwy wybór konkretnych obiektów, możliwość wyboru tworzenia nowego repozytorium, modyfikacje istniejącego, dodanie do istniejącego. Dodatkowo istnieje możliwość export/import metadanych z poziomu menu DESIGN.	W OWB istnieje mechanizm pozwalający na export/import. Nie jest to mechanizm analogiczny do zastosowanych w Oracle i Enterprise Manager. Pozwala on na import/eksport metadanych obiektów zdefiniowanych w OWB do/z pliku. Istnieje możliwość ich wyboru oraz sposobu załadowania do projektu – nadpisywania, tworzenie nowych itp.	Istnieje. W SSIS zaimplementowane jest narzędzie Import and Export Wizards. Przy imporcie z poziomu SQL'a brak możliwości zdefiniowania warunków ładowania danych do istniejących tabel.

Tabela 2b. Zidentyfikowane mechanizmy dla procesów ETL w poszczególnych środowiskach i narzędziach

	Oracle 10g R2	Oracle Enterprise Manager 10g	OWB 10g	MsSQL Server 2005
Transportable Tablespaces	Istnieje. Wykorzystuje mechanizm EXP/IMP. Przenosi same metadane, bez wartości. Nie można wykorzystać dla przestrzeni systemowych.	Istnieje. Możliwość wyboru systemu operacyjnego, przestrzeni tabel, strony kodowej. Brak możliwości dodania warunków dla danych importowanych i eksportowanych. Jedyna możliwość wyboru dotyczy konkretnych obiektów, dlatego też jest dedykowana dla przenoszenia konkretnych obiektów.	Istnieje. Możliwy export i import całego projektu.	Nie istnieje
Oracle Data Pump	Istnieje. Możliwość wyboru tabel, schematów oraz baz danych. Działanie szybsze od mechanizmu EXP/IMP.	Istnieje (opcje Load and Export Type). Możliwość eksportowania tabel, przestrzeni tabel, schematów, baz danych. Dodatkowo istnieje możliwość wyboru konkretnych wartości za pomocą zapytań. Import pozwala na wybór zachowania mechanizmu, gdy napotka na takie same (już istniejące) obiekty, zastępowanie istniejących itp. Mechanizm efektywniejszy od IMP/EXP.	Nie istnieje	Nie istnieje
Mapowania Transformacje	Nie istnieje	Nie istnieje	Istnieje. Jeden z głównych mechanizmów, na którym zostało oparte przetwarzania danych w procesach ETL. Bardzo duża funkcjonalność. Łatwość tworzenia. Narzędzie pozwalające na dowolne transformowanie danych tworzonych za pomocą narzędzi graficznych, także umożliwia ich ręczną modyfikacje za pomocą kodu PL/SQL'a. Możliwość tworzenia diagramów przepływu danych.	Istnieje. Duża liczba transformacji. Prosty interfejs graficzny . Możliwość tworzenia diagramów kontroli.

Tabela 2c. Zidentyfikowane mechanizmy dla procesów ETL w poszczególnych środowiskach i narzędziach

	Oracle 10g R2	Oracle Enterprise Manager 10g	OWB 10g	MsSQL Server 2005
SQL*Loader	Istnieje. Możliwe zdefiniowanie plików o rozmiarze większym niż 20 MB. Import plików płaskich, możliwość definiowania warunków, narzucenia maski dla eksportowanych danych. Brak wsparcia plików xls.	Istnieje. Odpowiednik Data Load. Maksymalny rozmiar źródeł to 10 MB, brak wsparcia dla plików xls.	Istnieje. Definiowany jest w edytorze funkcji i pakietów.	Istnieją mechanizmy zastępujące BULK INSERT, OPENROWSET. Wymagają one w niektórych przypadkach podania formatu pliku wejściowego, dlatego też niezbędnym narzędziem staje się BCP.
Łącza bazodanowe	Istnieje. Standardowo umożliwia połączenie się jedynie z bazami Oracle'a. Dodanie Oracle Heterogeneous Services umożliwia połączenie się z heterogenicznym SZBD – możliwe użycie poleceń SELECT, UPDATE, INSERT, DELETE.	Istnieje. Umożliwia łączenie się jedynie z SZBD Oracle. Możliwość importu obiektów.	Istnieje. Możliwość łączenia z najczęściej używanymi SZBD. Dodatkowo pozwala na programowe zdefiniowanie interfejsu łączącego.	Istnieje. Jedna z opcji SSIS Import and Export Wizards pozwala na połączenie z innymi bazami danych.
MERGE	Istnieje. Możliwość wykonywania poleceń INSERT/UPDATE/DELETE w zależności od spełnionego zdefiniowanego warunku.	Nie istnieje	Istnieje. Wykorzystywany podczas tworzenia mapowań jako jeden z operatorów transformacji danych.	Mechanizm zaimplementowany dopiero w MS SQL Server 2008. W SSIS zastosowany jest JOIN pod nazwą MERGE.
Pakiet UTL_FILE	Istnieje. Umożliwia odczyt linii z pliku, zapis danych z bufora do pliku, odczyt z płaskiego pliku danych oraz dodawanie danych, modyfikację oraz kasowanie.	Nie ma oddzielnej opcji wywołania tego pakietu. Możliwe wywołania z edytora do SQL'a oraz przy okazji definiowania innych mechanizmów i podczas ręcznej edycji.	Nie istnieje	Nie istnieje

Tabela 2d. Zidentyfikowane mechanizmy dla procesów ETL w poszczególnych środowiskach i narzędziach

	Oracle 10g R2	Oracle Enterprise Manager 10g	OWB 10g	MsSQL Server 2005
Multi Table Insert	Istnieje. INSERT ALL pozwala na dodanie do wielu tabel rekordów przy spełnionych określonych warunkach. INSERT FIRST pozwala na dodanie rekordu przy pierwszym spełnionym warunku.	Nie istnieje	Nie istnieje możliwość jego wywołania	Nie istnieje
Równoległy DML	Istnieje. Pozwala na szybsze od 30% do 80% przetworzenie danych. Pozwala na ręczne „zrównoleglenia" operacji.	Istnieje. Możliwy do wykorzystania podczas definicji innych mechanizmów np. perspektyw zmaterializowanych.	Istnieje. Nie jest wywoływany bezpośrednio, jednak jest opcją, jaka może być wywołana podczas definiowania, np. funkcji.	Istnieje. SQL Server automatycznie definiuje lub umożliwia ręczną konfigurację parametrów odpowiadających za przetwarzanie równoległe. Brak możliwości zbadania dla komputerów jedno-procesorowych.
Oracle Streams	Istnieje. Duża czasochłonność wykonania, skomplikowana definicja. Kolejkowanie komunikatów zapewnia dostarczenie ich do odbiorcy. Możliwość transformacji danych na każdym etapie wykorzystywania mechanizmu. Bardzo szybki mechanizm. Wymaga uruchomienia procesów Apply w obu systemach. Duże zużycie zasobów systemowych.	Istnieje. W porównaniu z konfiguracją z poziomu SQL*Plus'a łatwiejsza konfiguracja – interfejs intuicyjny.	Nie istnieje	Nie istnieje

Tabela 2e. Zidentyfikowane mechanizmy dla procesów ETL w poszczególnych środowiskach i narzędziach

	Oracle 10g R2	Oracle Enterprise Manager 10g	OWB 10g	MsSQL Server 2005
Perspektywy zmaterializowane	Istnieje. Pozwala na agregacje danych, tworzenie warunków nałożonych na dane (dowolna transformacja za pomocą składni SQL). W przypadku „ręcznego" odświeżania pozwala na zachowanie danych z tabel, dla których zdefiniowana została perspektywa przed usunięciem lub modyfikacją danych.	Istnieje. W prosty sposób można dodać perspektywę z dziennikiem. Pozwala na zdefiniowanie dowolnej klauzuli pobierania danych za pomocą języka zapytań SQL. Możliwe także włączenie mechanizmu zrównoleglenia.	Istnieje. Prosty interfejs do tworzenia perspektyw zmaterializowanych. Brak możliwości wyboru ich odświeżania.	Istnieje. Budowane za pomocą perspektywy i zdefiniowanego dla indeksu klastrowego. Nie odpowiada funkcjonalności Oracle'a.
Table functions	Istnieje. Pozwala na redukcję pamięci, a także czasu przetwarzania danych. Dodatkowo pozwala na zredukowanie przestrzeni dyskowej. Wykorzystuje do transformowania danych funkcje napisane w PL/SQL.	Nie istnieje	Istnieje. Tworzony jako jeden z operatorów transformacji w malowaniach.	Nie istnieje

W tabeli 2e zawarte są mechanizmy perspektyw zmaterializowanych. W przeciwieństwie do środowiska SQL Server 2005, w środowisku Oracle odgrywa on istotną rolę podczas transformacji danych.

7. PODSUMOWANIE

Z przeprowadzonych badań wynika, że SZBD Oracle w wersji 10g posiada więcej zaawansowanych, realizujących procesy ETL, mechanizmów niż MS SQL Server 2005. Porównanie narzędzi OWB (Oracle) i SSIS (Microsoft) pozwala stwierdzić, że posiadają one w porównywanych środowiskach zbliżoną do siebie funkcjonalność. Najczęściej wykorzystywane i brakujące mechanizmy ETL w wersji MS SQL Server 2005 zostały zaimplementowane w wersji 2008 serwera stąd wniosek, że Microsoft przejmuje pomysły swojego konkurenta i wdraża je w swoich serwerach. W Oracle'u bardzo dużą rolę odgrywa język PL/SQL, za pomocą którego możliwe jest ręczne dostrajanie procesu ETL, natomiast w MS SQL jego odpowiednikiem jest Transact-SQL.

Reasumując, Oracle 10 g jest trafniejszą platformą pod względem zastosowań ETL niż jego konkurent. Z drugiej zaś strony SSIS firmy Microsoft jest narzędziem bardziej przyjaznym dla użytkownika niż OWB. Jego interfejs jest bardziej czytelny i prostszy w obsłudze.

LITERATURA

[Bembenik 2002] Bembenik R., Jędrzejec B., *Technologie hurtowni danych, transformacja i integracjadanych w systemach sieciowych*, http://www.zsi.pwr.wroc.pl/zsi/missi2002/pdf/ s207.pdf.

[Błasiński 2005] Błasiński J., Wrembel R., *Oracle10g, Sybase IQ, SQL Server 2000 – ocena funkcjonalności serwerów baz danych w zastosowaniach hurtowni danych*, XI Konferencja PLOUG Kościelisko, Październik 2005.

[Connolly 2004] Connolly T, Begg C., *Systemy baz danych. Praktyczne metody projektowania, implementacji i zarządzania.* Tom 2, Wydawnictwo RM , 2004.

[Eckerson 2003] Eckerson W., White C. *Evaluating ETL and Data Integration Platforms*, The Data Warehousing Institute, 2003.

[ETL 2003] *ETL Processing within Oracle9i*, An Oracle White Paper, January 2003.

[Gorawski 2008] Gorawski M., *Ocena efektywności architektur OLAP*, http://www.e-informatyka.pl/article/show-bw/430, 2008.

[Gorawski 2004] Gorawski M., Maloczok R., *Systemy rozproszone*, Gliwice 2004.

[Hurtownia 2008] *Hurtownia danych – ODL*, http://www.odl.com.pl/odl_olap.htm, 2008.

[Inmon 1994] Inmon W.H., Hackathorn R.D., *Using a Data Warehouse*, Hardcover, 1994.

[Inmon 2005] Inmon W.H., *Building the Data Warehouses*, Whilley Publishing, 2005.

[Jarke 2000] Jarke M., Lenzerini M., Vassiliadis P., Vassiliou Y., *Fundamentals of Data Warehouses*, Springer-Verlag Berlin Heidelberg, 2000.

[Kimball 2002] Kimball R., Ross M., *The Data Warehouse Toolkit, Second Edition*, Jon Wiley & Sons, 2002.

[Kimball 2004] Kimball R, Caserta J., *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*, John Wiley & Sons, 2004.

[Poe 2000] Poe V., Klauer P., Brobst S., *Tworzenie hurtowni danych*, Wydawnictwa Naukowo-Techniczne, Warszawa 2000.

[PLOUG 2008] Szkoła IV PLOUG *danych*, http://www.ploug.org.pl/szkola/, 2008.

[Todman 2003] Todman Ch., *Projektowanie hurtowni danych :zarządzanie kontami z klientami (CRM)*, WNT, 2003.

[Walker 2006] Walker D.M, *Overview Architecture for Enterprise Data Warehouses*, Data Management & Warehousing version: 1.0 date: 01/03/2006.

[Wolski 2008] Wolski M., *Projektowanie hurtowni danych w oparciu o język UML*, http://www.wolski.waw.pl/2008/03/projektowanie-hurtowni-danych-w-oparciu-o-jezyk-uml/, 2008.

[Wrembel 2000] Wrembel R., Królikowski Z., Morzy M., *Magazyny danych – stan obecny i kierunki rozwoju*, 2000, Pro Dialog 10, s. 75–93.

CAPABILITIES COMPARISON OF ETL PROCESSES IN ORACLE AND MICROSOFT WAREHOUSES

This paper describes the issues of data warehouse and associated with them ETL processes. The first part of the work describes the theoretical issues such as warehouse architecture and data representation. The second part of the job is related to the practical issues associated with the warehouses and ETL processes. The identified ETL mechanisms and dedicated them programming tools in tested SZBD Oracle 10g and Microsoft SQL Server 2005 were there presented. All the mechanisms have been implemented in both environments for test data warehouse. They possibilities and operation were examined. The work is completed with the conclusions of the tests.

Prace Naukowe Instytutu Informatyki
Politechniki Wrocławskiej

Zeszyt 10 2009

procedury składowane,
baza danych,
odwzorowanie

Zbigniew SZPUNAR*,
Agata WOJCIECHOWSKA

PORÓWNANIE RZUTOWANIA RELACYJNO-OBIEKTOWEGO Z INTERFEJSEM PROCEDURALNYM

W pracy opisano i porównano dwa sposoby wytwarzania aplikacji bazodanowych. Pierwszy z nich opiera się na technologii Hibernate. Jest to podejście obiektowo-relacyjne. Drugi oparty jest na technologii SimpleJdbcTemplate z interfejsem proceduralnym. Podejścia te różnią się położeniem logiki biznesowej. W pierwszym przypadku logika umiejscowiona jest po stronie aplikacji, natomiast w przypadku drugim mieści się ona po stronie bazy danych.

1. WPROWADZENIE

Celem pracy jest zbadanie efektywności dwóch typów połączeń aplikacji napisanych w języku Java z bazą danych Oracle. Pierwszy z nich polega na użyciu odwzorowania obiektowo-relacyjnego; wykorzystano tu technologię Hibernate. Drugi sposób polega na zastosowaniu aplikacji z interfejsem proceduralnym, czyli środowiskiem zapewniającym swobodny dostęp do procedur składowanych, przechowywanych po stronie bazy danych, z użyciem technologii SimpleJdbcTemplate. W obu przypadkach zastosowano bazę danych opartą na tym samym schemacie tabel. Wybrane technologie nie są oczywiście jedynym możliwym rozwiązaniem.

Efektywność połączeń rozumiana jest tu nie tylko jako wydajność ale również jako możliwość szybkiego i łatwego procesu wytwarzania aplikacji.

Słowo interfejs pochodzi od angielskiej nazwy i rozumiane jest tu jako połączenie pomiędzy aplikacją a językiem proceduralnym, co podkreśla fakt, że aplikacja tylko określa sposób dostępu do procedur składowanych. Oba programy zostały utworzone w taki sposób, aby jeden z nich posiadał całą logikę biznesową po stronie aplikacji,

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

a drugi po stronie bazy danych. Dlatego program posługujący się procedurami został nazwany aplikacją z interfejsem proceduralnym.

Praca może służyć jako poradnik podczas procesu projektowania aplikacji bazodanowej. Najważniejszą cechą tego poradnika jest zestawienie wad i zalet każdego ze sposobów wytwarzania aplikacji oraz porównanie ich efektywności pod względem czasowym i pamięciowym. Testy obu programów przeprowadzono zarówno po stronie bazy danych jak i z poziomu aplikacji. Wyniki testów i ich analizy mogą służyć jako rady dla przyszłych twórców tego typu projektów.

2. ARCHITEKTURA APLIKACJI

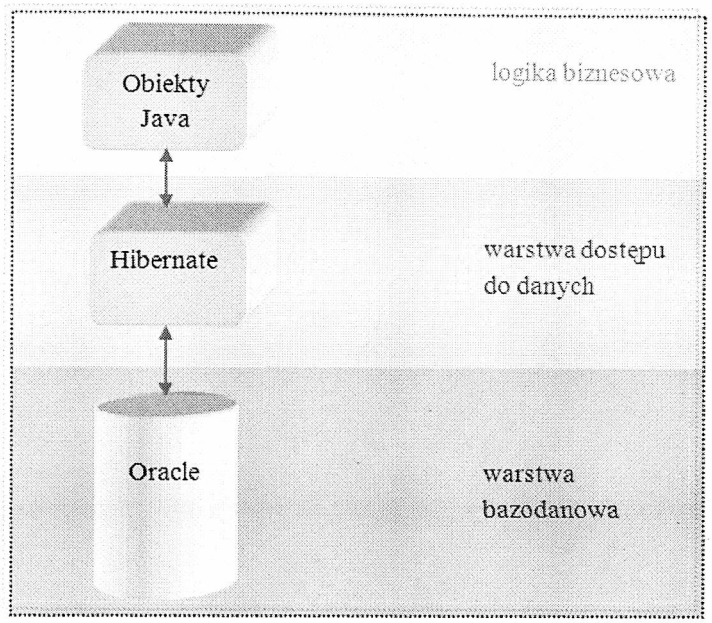
Najważniejszą różnicą dwóch omawianych architektur jest rozmieszczenie logiki biznesowej. W pierwszym przypadku znajduje się ona po stronie bazy danych, w procedurach składowanych. W drugim przypadku jest ona przeniesiona do kodu aplikacji.

2.1. APLIKACJA Z ODWZOROWANIEM OBIEKTOWO-RELACYJNYM

Działanie aplikacji korzystającej z odwzorowania obiektowo-relacyjnego przebiega w kilku etapach. Najważniejszym zadaniem tego typu aplikacji jest utrwalenie obiektów Javy w bazie danych. Na początku procesu implementacyjnego programista tworzy tak zwane obiekty POJO (ang. *Plain Old Java Object*), czyli „zwykłe obiekty Javy". Ważne jest, aby obiekty te były odseparowane od logiki wykorzystywanej do ich utrwalenia i pobrania z bazy danych. Za takie rozdzielenie odpowiada wzorzec projektowy DAO. Hibernate jest więc narzędziem pozwalającym na przechowywanie w bazie danych obiektów POJO [Red Hat 2007].

Cały silnik aplikacji, czyli wszystkie procesy wykonywane w programie, mieszczą się w klasach Javy. Są w nich zaimplementowane metody pobierania, edytowania, usuwania lub dodawania danych. Wywołanie tych metod wraz z odpowiednimi parametrami zwraca wynik, który przetwarzany jest przez Hibernate na kod SQL zrozumiały dla konkretnej bazy danych. Hibernate ustanawia połączenie z bazą i jeżeli wszystkie polecenia z jednej transakcji nie zwracają żadnych wyjątków, dane są utrwalane w bazie danych bądź z niej pobierane.

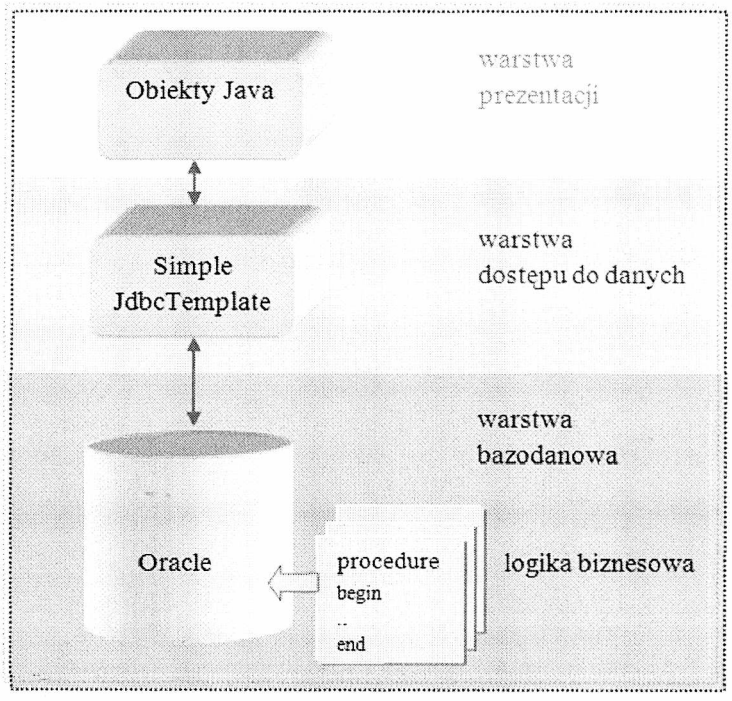
Na rysunku 1 przedstawiono schemat architektury omawianej aplikacji. Aplikacja dzieli się na trzy warstwy: warstwę logiki biznesowej, warstwę dostępu do danych oraz warstwę bazodanową, którym kolejno przyporządkowane są: obiekty Javy, Hibernate oraz baza danych Oracle. Hiberante uzyskuje dostęp do bazy danych używając pliku konfiguracyjnego, w którym znajdują się informacje o sposobie uzyskania połączenia. Dzięki temu można w prosty sposób wskazać na dowolną, inną bazę danych.



Rys. 1. Schemat architektury aplikacji z odwzorowaniem obiektowo-relacyjnym

2.2. APLIKACJA Z INTERFEJSEM PROCEDURALNYM

Cykl wytwarzania aplikacji z interfejsem proceduralnym znacząco różni się od cyklu wytwarzania aplikacji z odwzorowaniem obiektowo-relacyjnym (rys. 2). W tym przypadku najważniejszą częścią systemu są procedury składowane utworzone w bazie danych. To one decydują o tym, jakie zadania wykonuje aplikacja. Można w nich zaimplementować wszystkie podstawowe usługi bazy danych oraz zaawansowane funkcje wyliczeniowe. Żeby mieć możliwość tworzenia takich procedur, trzeba zastosować konkretną bazę danych z odpowiednim językiem proceduralnym. Jednym z najbardziej znanych języków tego typu jest język PL/SQL – narzędzie zaprojektowane specjalnie dla bazy danych Oracle [Urman 2008]. W ten sposób warstwa logiki biznesowej jest przeniesiona do warstwy bazodanowej. Mogłoby się wydawać, że w tym przypadku część kodu Javy znacznie się uprości, jednak jest tak tylko z pozoru. W momencie, gdy nie zostało użyte żadne narzędzie do odwzorowywania, należy to odwzorowanie oprogramować samemu. Jak więc komunikować się z bazą danych z poziomu aplikacji? Połączenie to musi być również skonfigurowane w pliku. Tym razem powinien być to plik konfiguracyjny Spring Framework [Johnson 2008], który również obejmuje wzorzec projektowy DAO. Jest on też odpowiedzialny za oddzielenie części związanej z obsługą bazy danych od pozostałych warstw aplikacji. Metody obsługujące bazę danych obejmują tylko wywołanie procedur składowanych.



Rys. 2. Schemat architektury aplikacji z interfejsem proceduralnym

Podsumowując, cykl dodawania danych do bazy danych wygląda następująco:

- do obiektów reprezentujących odpowiednie typy bazodanowe przypisywane są wartości,
- wywoływana jest metoda odpowiedzialna za dodanie danych,
- w parametrze tej metody przekazywany jest obiekt,
- metoda ta, przy pomocy specjalnej klasy, odwzorowuje obiekt na konkretny typ z bazy danych,
- ustanawiane jest połączenie, w którym dochodzi do przekazania nazwy wywołanej procedury wraz z parametrami,
- tak utworzony program pozwala na dodanie oraz (po odpowiedniej modyfikacji) pobranie danych z bazy danych.

3. ANALIZA PORÓWNAWCZA

Podczas każdego procesu wytwarzania aplikacji programista napotyka wiele problemów, które musi rozwiązać. Dobór odpowiedniej technologii jest bardzo istotnym elementem procesu wytwarzania aplikacji. Czy tak naprawdę można wybrać idealne

rozwiązanie? Możliwości jest wiele. Praca ta zwiera porównanie dwóch z nich. Tego typu porównania można dokonać na wiele sposobów. Jedną z możliwości porównania ze sobą dwóch odmiennych aplikacji jest zestawienie wspólnych własności, czyli opracowanie metryk.

Obie aplikacje zostały porównane przy pomocy utworzonych metryk. Wyniki zostały przedstawione w tabeli 1.

Tabela 1. Zestawienie metryk

Metryka	Aplikacja z odwzorowaniem obiektowo-relacyjnym	Aplikacja z interfejsem proceduralnym
liczba wymaganych schematów do utworzenia aplikacji (np. schemat relacyjny, obiektowy)	wymaga utworzenia tylko jednego schematu obiektowego lub relacyjnego	wymaga utworzenia schematu relacyjnego, schematu obiektowego w bazie danych, schematu obiektowego w aplikacji
liczba linii kodu źródłowego	mała liczba linii kodu	duża liczba linii kodu
stopień automatyzacji procesu tworzenia odwzorowań	gotowe rozwiązania	samodzielna implementacja klas
czas poświęcony na implementację	krótki	długi
czas poświęcony na naukę nowej technologii	długi – ciężko zacząć pracę z Hibernate	długi – błędy występują na różnych poziomach
czas poświęcony na implementację dla osób znających technologii	bardzo krótki	bardzo długi
stopień trudności zamiany bazy danych	niewielkie zmiany w kodzie	wymaga edycji dużej ilości kodu
niezależność od platformy	kod w pełni przenośny	kod w pełni przenośny
stopień trudności debugowania kodu	proste, łatwość wykrywania błędu	trudne, debuguje się tylko kod Javy, nie kod procedur składowanych
stopień znajomości systemu zarządzania bazą danych potrzebny do implementacji aplikacji	minimalny	duży (szczególnie języka PL/SQL)
ilość i jakość dostępnych materiałów	szeroko dostępne, dokumentacja, strony internetowe, książki (także w języku polskim)	dokumentacja w języku angielskim, mały zasób artykułów w Internecie, wzmianki w książkach w postaci jednego podrozdziału (głównie w języku angielskim)
stopień trudności migracji bazy do innej wersji	niewielkie zmiany	zmiana dużej części kodu, w tym procedur składowanych
liczba różnych technologii wymaganych do utworzenia projektu	• Java – zaawansowany • SQL – podstawy	• Java – zaawansowany • PL/SQL – zaawansowany

Proces wytwarzania aplikacji można podzielić na kilka głównych etapów. Na początku należy ściśle zdefiniować wymagania klienta odnośnie do systemu. Etap ten nie został tu opisany, gdyż obie aplikacje implementują ten sam przykładowy wycinek

rzeczywistości. Po zdefiniowaniu celu pracy należy zaprojektować system. Z dokładnie wykonanym projektem znacznie łatwiej przejść do części implementacyjnej. Jednak cały proces tworzenia aplikacji nie kończy się na tym etapie. Po wdrożeniu aplikacja wymaga ciągłego nadzoru i modyfikacji pod względem potrzeb użytkowników.

Część projektowa obu aplikacji jest bardzo zbliżona, gdyż mają one realizować te same procesy biznesowe. Główna różnica występująca w tej części projektu to konieczność tworzenia różnych schematów. Hibernate daje możliwość wykonania tylko jednego schematu. Można zatem utworzyć schemat np. relacyjny a Hibernate wygeneruje odpowiedni schemat obiektowy (bądź na odwrót). W przypadku SimpleJdbcTemplate należy oba te schematy utworzyć oddzielnie. Dodatkowym utrudnieniem jest konieczność utworzenia typów obiektowych po stronie bazy danych.

Proces implementacji aplikacji używającej Hibernate jest bardzo dogodny dla programisty. Wymaga pewnej znajomości Frameworka, jednak wiedza na ten temat jest szeroko dostępna. Dodatkowym atutem użycia tej technologii jest możliwość skorzystania z generatorów kodu. Pozwala to ominąć wprowadzanie dużej ilości kodu i użycie odpowiedniego schematu. Hibernate posiada wbudowane narzędzia do odwzorowywania kodu. Wystarczy tylko dobrze zdefiniować, w jaki sposób mają być odwzorowane poszczególne elementy. Na podstawie tych informacji Hibernate sam dokona odwzorowania. Ogromnym udogodnieniem jest możliwość szybkiej zmiany bazy danych. Ogranicza się to tylko do podmiany sterownika decydującego o tym, jaką bazę ma obsługiwać Hibernate. Jeżeli chodzi o migrację bazy danych sprawa wygląda identycznie. Wybiera się sterownik do nowszej wersji np. Oracle.

Debugowanie kodu aplikacji pisanej z użyciem Hibernate jest dosyć proste i odbywa się z poziomu aplikacji. Do wytworzenia tego typu aplikacji wymagany jest tylko programista znający język programowania Java, posiadający podstawową wiedzę na temat języka SQL.

Przyglądając się natomiast procesowi wytwarzania aplikacji z użyciem procedur składowanych, w odniesieniu do procesu wytwarzania aplikacji z odwzorowaniem obiektowo-relacyjnym, można dostrzec znaczne różnice. Pierwszą obserwacja dotyczy tego, że procedury składowane tworzone są dla konkretnej bazy danych. Nie wszystkie systemy zarządzania bazą danych posiadają wbudowany język proceduralny, a jeśli już posiadają, to ich składnie znacząco różnią się między sobą. W momencie zmiany bazy danych połączonej z aplikacją zachodzi potrzeba zmiany wszystkich procedur składowanych. Z opisu architektury wiadomo, że w procedurach składowanych zaimplementowana jest cała logika biznesowa aplikacji, czyli jej najważniejsza część. Łatwo wywnioskować, że taka zmiana nie jest prosta do wykonania i wymaga znacznej ilości czasu.

Przy porównaniu technologii SimpleJdbcTemplate z technologią Hibernete warto zwrócić uwagę na ilość kodu potrzebną do zaimplementowania tych samych zadań. SimpleJdbcTemplate nie zapewnia tylu możliwości co Hibernate. Porównując chociażby proces odwzorowywania tabel i relacji pomiędzy nimi dochodzi się do wniosku, że w SimpleJdbcTemplate całe odwzorowanie należy oprogramować samemu a wytwarzanie kodu aplikacji jest często bardzo schematyczne i czasochłonne.

Nie jest to jednak jedyny problem, który wynika z faktu zwiększonej ilości kodu. Kolejnym, dość istotnym problemem jest fakt, że wraz ze wzrostem ilości kodu wzrasta prawdopodobieństwo popełnienia błędu. W celu szybkiego wykrycia błędu można zaproponować debugowanie kodu. Jednak pojawia tu się następny problem: używając środowisk programistycznych języka Java, mamy możliwość prześledzenia tylko i wyłącznie kodu Javy. Żaden kompilator nie zapewnia nam śledzenia procedur składowanych, które są wywoływane w kodzie Javy.

Mogłoby się wydawać, że aplikacja, która ma za zadanie wywoływać odpowiednie procedury, nie jest trudna do zaimplementowania. Okazuje się, że w tym przypadku intuicja może być bardzo zawodna. Praca z SimpleJdbcTemplate jest dosyć trudna, szczególnie dla osób nie znających tej technologii. Istnieje stosunkowo niewiele informacji (w porównaniu np. z technologią Hibernate) dotyczących technologii a także sposobu jej użycia. Wszystkie materiały napisane są w języku angielskim, co czyni je mniej dostępnymi dla programistów nie posługujących się tym językiem.

Przy wyborze aplikacji z interfejsem proceduralnym zachodzi potrzeba zatrudnienia specjalisty od PL/SQL oraz programisty języka Java. Zwiększa to koszty utrzymania pracowników, a przy dużych projektach stwarza konieczność zatrudnienia większej liczby specjalistów. Niesie jednak pewne zalety. Istnieje bowiem prosty sposób podziału projektu na części tak, aby odpowiednie osoby były odpowiedzialne tylko za te części projektu, które powiązane są ściśle z dziedziną, w której te osoby się specjalizują.

Czy omówione wady tej technologii są na tyle ważne by jednoznacznie stwierdzić, że jej użycie nie jest odpowiednie? Być może w niektórych przypadkach warto przebrnąć przez proces nieprzyjemnej implementacji, aby uzyskać pożądany efekt.

4. BADANIA ILOŚCIOWE

Poza analizą porównawczą przeprowadzono testy wydajnościowe obu aplikacji. Miały one na celu wskazać, która z aplikacji wykonuje szybciej to samo zadanie. Posłużono się w tym celu odpowiednimi narzędziami. Poniżej przedstawiono zarys wykonanych testów oraz interpretacje otrzymanych wyników.

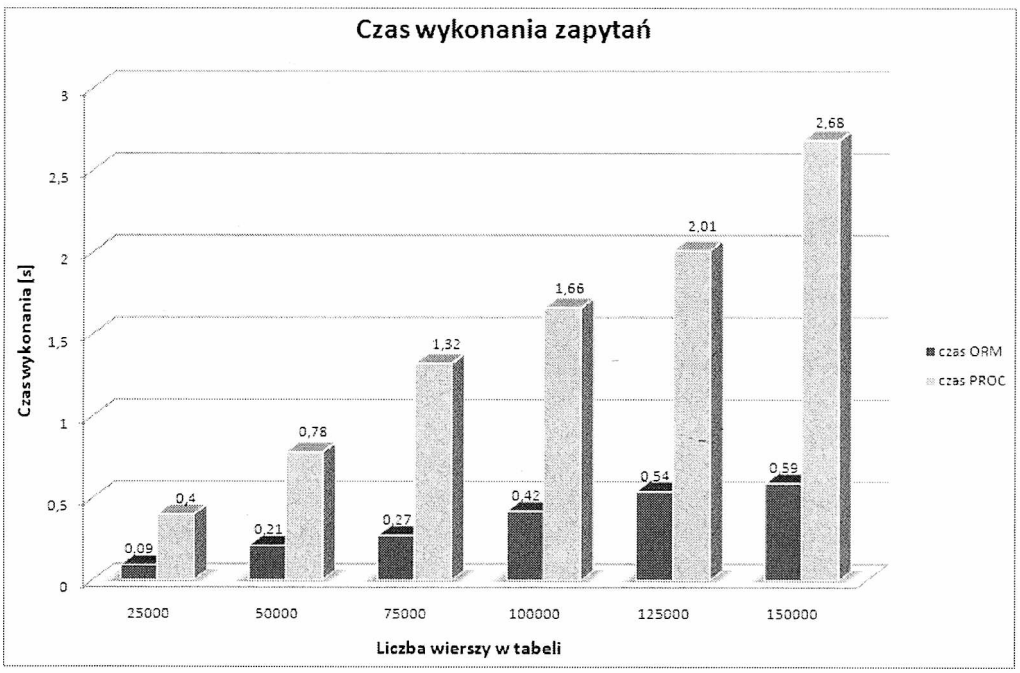
4.1. SPOSOBY TESTOWANIA

Badania ilościowe przeprowadzono przy pomocy narzędzia TKPROF. Jest to jedno z narzędzi Oracle'a, umożliwiających śledzenie poleceń SQL lub PL/SQL. Służy on głównie do oceny polecenia pod względem wydajności. Pobieranie danych w języku SQL można zrealizować na wiele sposobów. Bardzo często dwa różne pod względem składni zapytania służą do zwracania tych samych wyników. Przy pomocy programu TKPROF można łatwo stwierdzić, który wariant zapytania wykonuje się najszybciej

i dokonać odpowiedniego wyboru. Narzędzie to pomocne jest nie tylko przy optymalizacji zapytań. Śledzenie zapytań zapewnia bardzo dobry sposób porównywania dwóch aplikacji wykonujących to samo polecenie. W tej pracy TKPROF posłuży ocenie szybkości działania obu aplikacji, ale również pozwoli w szczegółowy sposób przyjrzeć się działaniu obu aplikacji po stronie bazy danych.

4.2. OTRZYMANE WYNIKI

Na rysunku 3 przedstawiono czas wykonania zapytań. Czas wykonania dla aplikacji z odwzorowaniem obiektowo-relacyjnym zaznaczony jest kolorem jasnoszarym, natomiast czas dla aplikacji z interfejsem proceduralnym kolorem ciemnoszarym.



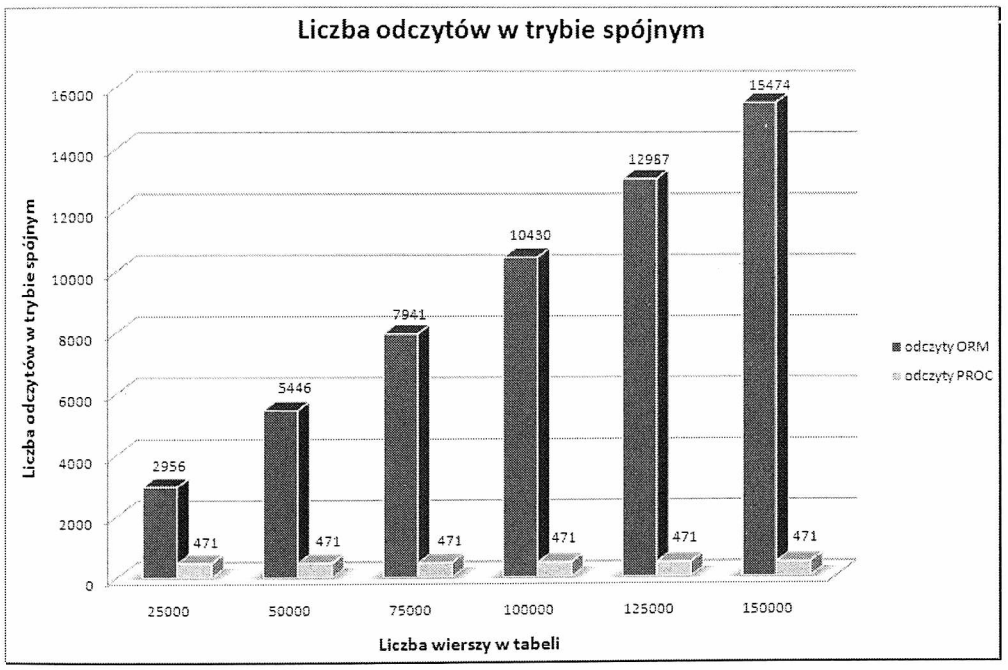
Rys. 3. Czas wykonania zapytań

Ponieważ testy przeprowadzone były dla liniowo wzrastającej ilości danych (wyrażanej liczbą rekordów w bazie danych), można zastanowić się czy czasy wykonania zapytania wzrastały także liniowo. Na wykresie widać, że zależność czasu wykonania od liczby wierszy w tabeli jest dla odwzorowania obiektowo-relacyjnego i dla interfejsu proceduralnego w przybliżeniu liniowa. Po wyznaczeniu przyrostów czasowych (przyrost czasowy jest tu rozumiany jako różnica pomiędzy czasami zapytania operującego na dwóch różnych liczbach rekordów) można wnioskować, że przypuszczenia

o liniowości rozpatrywanej zależności są słuszne zarówno dla technologii z odwzorowaniem obiektowo-relacyjnym jak i dla interfejsu proceduralnego.

Warto zaznaczyć, że wykres przedstawiony na rysunku 3 obrazuje wynik tylko jednego testu wykonanego dla różnej ilości danych. Przy powtórzeniu testu i zbadaniu czasu średniego otrzymuje się podobne wyniki. Nawet w przypadku pomiarów czasu średniego, na podstawie tak niewielkiej liczby obserwacji trudno wyciągać jakiekolwiek wnioski. Jednak celem tej pracy nie jest szczegółowa analiza danych eksperymentalnych i niewielka liczba obserwacji jest w tym przypadku uzasadniona.

Na rysunku 4 przedstawiono liczbę odczytów w trybie spójnym. Liczba odczytów w trybie spójnym dla aplikacji z odwzorowaniem obiektowo-relacyjnym zaznaczona jest kolorem jasnoszarym, natomiast dla aplikacji z interfejsem proceduralnym kolorem ciemnoszarym [Wholen 2003].



Rys. 4. Liczba odczytów w trybie spójnym

Rozpatrując liczbę odczytów w trybie spójnym można zauważyć, że liczba ta jest stała w przypadku interfejsu proceduralnego i nie zmienia się wraz ze wzrostem liczby rekordów. Natomiast w przypadku odwzorowania obiektowo-relacyjnego liczba odczytów wzrasta liniowo wraz ze wzrostem liczby wierszy w tabeli. Można przypuszczać, że dla większej ilości danych zależność ta będzie również liniowa. Ta cecha świadczy na niekorzyść technologii z odwzorowaniem obiektowo-relacyjnym. Przy wytwarzaniu systemu przeznaczonego dla dużej liczby użytkowników, wśród których

każdy będzie operował na dużej ilości danych, warto zastanowić się, czy nie lepiej skorzystać z podejścia proceduralnego.

5. PODSUMOWANIE

Celem tej pracy było porównanie dwóch sposobów wytwarzania aplikacji bazodanowych. Jako pierwszą omówiono technologię Hibernate. Drugie zaprezentowane podejście dotyczyło wytwarzania aplikacji z interfejsem proceduralnym korzystającej z SimpleJdbcTemplate. Utworzono specjalne metryki, które miały na celu opisanie między innymi: liczby linii kodu źródłowego, liczby wymaganych schematów do utworzenia aplikacji (np. schemat relacyjny, obiektowy), stopnia automatyzacji procesu tworzenia odwzorowań, czasu poświęconego na implementację, stopnia trudności zamiany bazy danych na inną oraz migracji bazy, stopnia znajomości bazy danych potrzebnego do implementacji aplikacji, ilości i jakość dostępnych materiałów, stopnia trudności debugowania kodu oraz liczby technologii wymaganych do utworzenia projektu. Przykłady dobrano w taki sposób, aby każdy z nich dotyczył osobnej metryki i podkreślał słabe i mocne strony obu rozwiązań.

Przy pomocy narzędzia TKPROF porównywano czasy działania obu aplikacji. Jak pokazały wyniki testów, wydajniejsza okazała się aplikacja korzystająca z rzutowania obiektowo relacyjnego i technologii Hibernate. Technologia ta wypadła korzystniej także przy porównaniu szybkości procesu wytwarzania aplikacji jak również pod względem łatwości debugowania kodu aplikacji i procesu jego rozwijania. Należy mieć również na uwadze fakt, że Hibernate Framework został utworzony do celów innych niż język PL/SQL. Ważną cechą Hibernate jest posiadanie pamięci podręcznej (tzw. *cache*) [Minter 2007], która umożliwia przechowywanie dużej ilości danych. Poprawia to znacznie szybkość działania aplikacji w przypadku, gdy wielokrotnie korzystamy z tych samych wyników. Kolejną cechą Hibernate, świadczącą na jego korzyść, jest tzw. opóźnione ładowanie (ang. *lazy loading*) umożliwiające pobieranie niektórych atrybutów dopiero w momencie, gdy są one potrzebne do użycia w programie.

Wybranie technologii powinno być podyktowane względami czysto praktycznymi. Okazuje się, że dla osób, które chcą szybko utworzyć wydajną aplikację bazodanową lepszym rozwiązaniem jest technologia Hibernate. Jest to więc najlepsze rozwiązanie dla programisty, dla którego najważniejszy jest proces łatwego rozwijania kodu aplikacji i czas jej działania. Dobrym przykładem jest tutaj aplikacja, która wykonuje za jednym razem wiele prostych zapytań do bazy danych i nie operuje na zaawansowanych funkcjach bazodanowych, które mogą być po prostu niedostępne z poziomu Frameworka.

Wynik ten nie jest zaskakujący. Bardziej zaskakująca okazuje się przewaga SimpleJdbcTemplate pod względem liczby odczytów w trybie spójnym. Okazuje się bowiem, że programy korzystające z tej technologii potrzebują stałej liczby odczytów

i liczba ta nie zależy od ilości danych do przetworzenia. Interfejs proceduralny jest zatem najlepszym rozwiązaniem, jeżeli w grę wchodzą ograniczenia pamięciowe. Należy też zwrócić uwagę, że przy tworzeniu dużych baz danych, obsługujących dużą liczbę użytkowników, podejście proceduralne może okazać się jedynym poprawnym.

LITERATURA

[Johnson 2008] Johnson R. i in., *The Spring Framework*, Reference Documentation, 2008. http://static.springframework.org/spring/docs/.

[Minter 2007] Minter J., Linwood D., *Hibernate od Nowicjusza do Profesjonalisty*, Lublin, Apress, 2007.

[Red Hat 2007] Red Hat. Hibernate Reference Documentation. 3.3.1 edition, 2007. http://docs.jboss.org/hibernate/stable/core/reference/en/html/index.html.

[Urman 2008] Urman S., Hardman R., McLaughlin M., *Oracle Database 10g Programowanie w języku PL/SQL*, Gliwice, Helion, 2008.

[Wholen 2003] Wholen E., Schroeter M., *Oracle. Optymalizacja wydajności*, Helion, Gliwice 2003.

COMPARISON OF OBJECT RELATIONAL MAPPING AND PROCEDURAL INTERFACE

The work presents a comparative description of two database application development methodologies, one based on Hibernate object-relational mappings and another, which applies the SimplJdbcTemplate technology with its procedural interfaces. The presented approaches differ with respect to the placement of business logic. In the former case, business logic resides within the application, while in the latter it is part of the database server.

Prace Naukowe Instytutu Informatyki
Politechniki Wrocławskiej

Zeszyt 10 2009

bezpieczeństwo danych,
wady oprogramowania,
ataki sieciowe

Teresa MENDYK-KRAJEWSKA*,
Zygmunt MAZUR*

ZAGROŻENIA BAZ DANYCH W ASPEKCIE WAD OPROGRAMOWANIA SYSTEMÓW KOMPUTEROWYCH

Bezpieczeństwo systemów baz danych, często rozproszonych, zależy od poziomu ochrony środowiska, w którym są eksploatowane. Praca dotyczy problemu zagrożenia bezpieczeństwa systemów informatycznych, w tym systemów bazodanowych, z powodu występowania błędów w oprogramowaniu (w systemach operacyjnych, programach i aplikacjach użytkowych). Skala zjawiska istnienia wad, które mogą być wykorzystane w celu bezprawnego działania w sieciach komputerowych jest ogromna. Jedyną możliwą reakcją na ujawnione błędy jest instalacja dostarczanych przez producenta nakładek systemowych, jednak proces ten nie rozwiązuje w pełni istniejącego problemu. Nieustannie bowiem wykrywane są nowe wady, a cała rzesza użytkowników nie dba wystarczająco o aktualizację wykorzystywanego oprogramowania.

1. PROBLEM LUK W OPROGRAMOWANIU SYSTEMÓW INFORMATYCZNYCH

Luka to stan systemu komputerowego, który umożliwia dostęp do danych i wykonywanie poleceń w imieniu uprawnionego użytkownika, a także podszywanie się pod inną jednostkę pozwalając między innymi na dokonanie ataku DoS. Natomiast podatność to taki stan systemu, w którym atak jest możliwy jedynie z powodu stosowania niewłaściwej polityki bezpieczeństwa [CVE].

Łatwość naruszania bezpieczeństwa systemów informatycznych wynika z problemu osiągania pełnej kontroli nad poprawnością implementacji i konfiguracji złożonych systemów oprogramowania współczesnych komputerów i ich sieci. Najwięcej luk wykrywa się w powszechnie stosowanym oprogramowaniu, zaś szczególnie waż-

* Instytut Informatyki, Wydział Informatyki i Zarządzania Politechniki Wrocławskiej, 50-370 Wrocław, Wybrzeże Wyspiańskiego 27, {teresa.mendyk-krajewska, zygmunt.mazur}@pwr.wroc.pl

ną rolę odgrywają luki znajdujące się w systemach operacyjnych i ich komponentach. Najbardziej niebezpieczne są te, które umożliwiają atakującemu przejęcie pełnej kontroli nad systemem, tzw. luki krytyczne. Z badań wynika, że największa liczba udanych ataków sieciowych jest możliwa ze względu na wady w kilku popularnych usługach sieciowych zaimplementowanych w systemach operacyjnych, oraz błędy w podstawowym oprogramowaniu użytkowym: edytorach tekstu, arkuszach kalkulacyjnych, programach prezentacyjnych itp. Ostatnio rejestruje się znaczny wzrost ataków na popularne odtwarzacze plików multimedialnych (Apple Quick Time, Real Player, Windows Media Player, Realtek Media Player).

Wady w oprogramowaniu systemów informatycznych wykrywane są nieustannie. W październiku 2009 roku informowano o największej liczbie poprawek (13 uaktualnień dla 34 błędów w różnym oprogramowaniu) przygotowanych przez Microsoft od czasu, gdy firma zaczęła publikować je w stałym, miesięcznym cyklu.

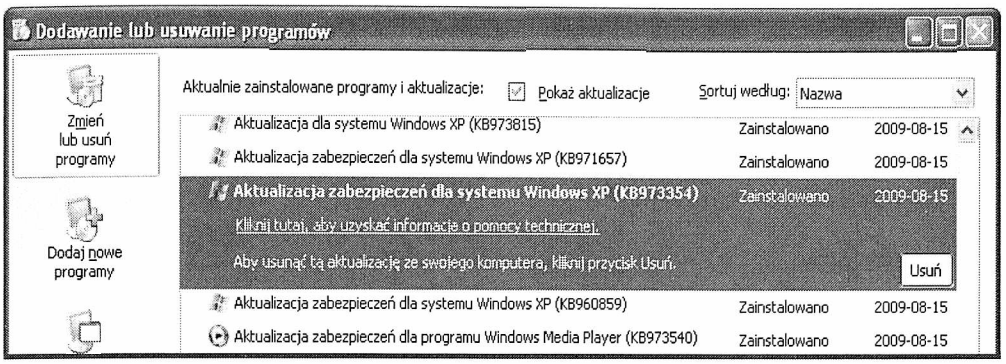
Po zidentyfikowaniu luki hakerzy opracowują i upubliczniają narzędzia (tzw. exploity) pozwalające zaatakować system zanim zostaną udostępnione i zainstalowane odpowiednie nakładki. W procesie tym można wyróżnić kilka etapów. Po odkryciu błędu umożliwiającego przeprowadzenie ataku pojawiają się pierwsze exploity, z których zaczynają korzystać początkujący włamywacze. Następnie zostają stworzone automatyczne narzędzia wykorzystujące daną lukę i następuje ich używanie na szeroką skalę. Do tego czasu liczba udanych ataków proporcjonalnie rośnie. Po ukazaniu się pierwszych exploitów zostaje opublikowana nakładka systemowa – przy czym czas od momentu wykrycia luki do jej „załatania" może być różny. Z badań wynika, że jeszcze długo po ukazaniu się poprawek, liczba przeprowadzanych ataków wykorzystujących daną lukę rośnie. Po pewnym czasie zainteresowanie nią maleje, a włamywacze zaczynają korzystać z innych możliwości [Kurek 2009].

Różne luki wykorzystywane są przez wiele szkodliwych kodów, jak na przykład przez robaki – Blaster (przepełnienie bufora stosu w interfejsie RPC[1]) czy Sircam i Nimda (luka w Usłudze stacji roboczej systemu Windows).

Wysoki poziom ochrony jest pojęciem względnym, a żaden system nie osiąga całkowitego bezpieczeństwa. Nawet, jeśli system zostanie „uszczelniony", może zawierać nieujawnione dotąd wady. Ponadto zawsze istnieje możliwość usunięcia (przypadkowego lub celowego, ręcznego lub automatycznego) wprowadzonej już poprawki (rys. 1).

Okazuje się, że włamywacze na ogół stosują proste metody, używając do przeprowadzania ataków gotowe narzędzia wykorzystujące znane luki. Potwierdzają to między innymi opublikowane w 2008 roku wyniki badań należącej do FBI agencji National Infrastructure Protection Center zajmującej się problematyką bezpieczeństwa sieciowego, która przeprowadziła analizę włamań do serwerów firm e-biznesowych, portali i sklepów internetowych.

[1] *Remote Procedure Call* – protokół zdalnego wywoływania procedur.



Rys. 1. Możliwość usunięcia zainstalowanej poprawki systemowej

Według analiz firmy Forrester Research, w ciągu 2008 roku ponad 62% firm doświadczyło naruszenia bezpieczeństwa z powodu luk oprogramowania. Badania wykazują jednak, iż większość organizacji nie zachowuje prawidłowego cyklu projektowania oprogramowania, pozwalającego na gruntowne jego przetestowanie przed wdrożeniem.

2. SKALA ZAGROŻENIA

Systemy sieciowe są automatycznie skanowane przy pomocy dostępnych narzędzi w poszukiwaniu tych, które są niezabezpieczone, podatne na włamania. Skala zagrożenia bezpieczeństwa systemów informatycznych z powodu występowania luk w oprogramowaniu jest ogromna. Świadczy o tym choćby kilka poniżej przytoczonych przykładów.

Dużym zagrożeniem jest bezpłatne oprogramowanie Adobe Reader (umożliwiające odczytywanie dokumentów w formacie PDF), w którym już wielokrotnie (np. w listopadzie 2008 roku, w lutym i w kwietniu 2009 roku) wykrywano błędną implementację obsługi JavaScript umożliwiającą zdalne uruchomienie w systemie szkodliwego kodu. O kolejnych krytycznych lukach w oprogramowaniu firmy Adobe informowano również w październiku 2009 roku; znaleziono je w wielu wersjach programów Acrobat i Reader (7.x, 8.x, 9.x). Reklamowane alternatywne (również bezpłatne) oprogramowanie – Foxit Reader, także okazało się wadliwe. Typowy atak przeprowadzany z wykorzystaniem błędów w Adobe Reader lub Foxit Reader polega na wysłaniu do użytkownika pliku w formacie PDF, którego otwarcie (lub podgląd zawartości) powoduje błąd przepełnienia bufora, co pozwala na uruchomienie szkodliwego kodu i pracę w systemie na prawach aktualnie zalogowanego użytkownika.

W lipcu 2009 roku firma Symantec poinformowała o wykryciu nowego konia trojańskiego o nazwie Trojan.Pidief.G w dokumencie typu PDF, który po otwarciu ładuje i wykonuje szkodliwy kod na zaatakowanym komputerze. W tym przypadku wyko-

rzystywana jest luka w oprogramowaniu Adobe Flash, bowiem Adobe Acrobat v.9.1 pozwala na umieszczanie filmów w plikach z rozszerzeniem .pdf. Luki w oprogramowaniu Adobe Flash stwarzają duże zagrożenie ze względu na jego powszechne wykorzystywanie w aplikacjach internetowych.

Luki umożliwiające zainfekowanie systemu znaleziono w wielu innych programach do obsługi dokumentów PDF (np. w programie Xpdf oraz bazujących na jego kodzie czytnikach CUPS, KPDF, poppler i PDF). Dotychczas nie wydano jeszcze aktualizacji usuwających znaleziony błąd.

W maju 2009 roku firma n.runs odkryła błąd w antywirusowym programie Avast firmy Alwil. Wykorzystując specjalnie spreparowane pliki instalacyjne CAB i SIS można wprowadzić szkodliwy kod do atakowanego systemu.

Specjaliści z firmy Symantec wielokrotnie przytaczali przykłady popularności kontrolek ActiveX[2] i zalecali ostrożność w ich używaniu, komentując wady oprogramowania innych firm. Tymczasem w kwietniu 2008 roku znaleziono lukę dotyczącą kontrolki ActiveX w popularnym oprogramowaniu ochronnym firmy Symantec. Luka ta może być wykorzystana do przejęcia kontroli nad komputerem pracującym pod kontrolą systemu Windows.

Krytyczny błąd związany z kontrolką ActiveX został również wykryty w zabezpieczeniach przeglądarki Internet Explorer (lipiec 2009). Panda Security wykryła około stu witryn internetowych, które zmodyfikowano w taki sposób, aby infekowały użytkowników poprzez wykorzystanie nie naprawionej luki w oprogramowaniu firmy Microsoft, między innymi przeglądarki Internet Explorer 7 pracującej w systemie Windows. Trzy krytyczne błędy zawierała również przeglądarka Google Chrome. Luki ujawniane są we wszystkich przeglądarkach internetowych. Liczbę luk znalezionych w nich w 2008 roku przedstawiono na rysunku 2.

W sierpniu 2009 roku firma Microsoft poinformowała producentów innych przeglądarek o wykryciu luki umożliwiającej (poprzez odpowiednio spreparowany serwer proxy) podstawienie użytkownikowi dowolnego kodu HTML i skryptów, przekierowywanie go na fałszywą stronę WWW, modyfikację wyświetlanych danych i kradzież cookies (co może być wykorzystane do zaatakowania użytkowników bankowości elektronicznej).

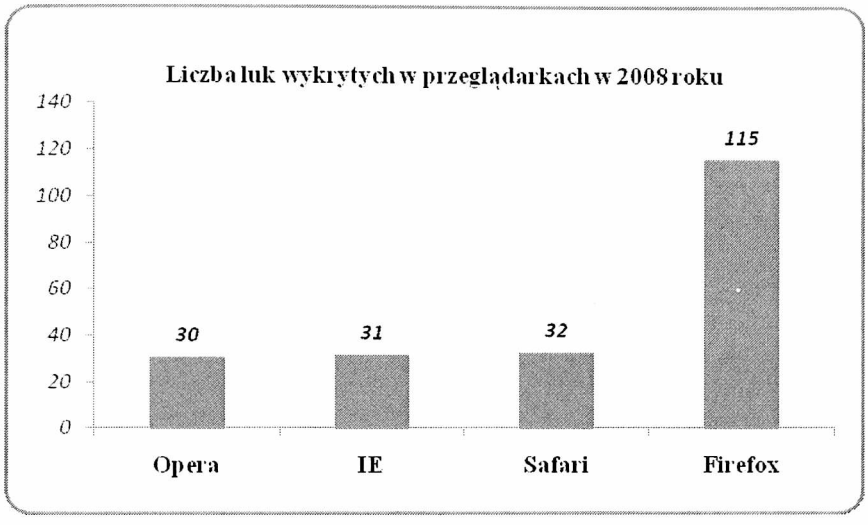
W 2008 roku specjalista ds. bezpieczeństwa poinformował o poważnym błędzie w protokole DNS (ang. *Dynamic Network Service*), umożliwiającym przejęcie kontroli nad serwerami nazw, phishing[3] oraz infekowanie komputerów szkodliwym oprogramowaniem. Skutecznym rozwiązaniem tego problemu jest instalowanie nowszej wersji DNSSEC (ang. DNS *Security Extensions*) zapewniającej autoryzację źródeł danych za pomocą metod kryptografii asymetrycznej.

---

[2] Technologia umożliwiająca przekazywanie danych pomiędzy różnymi aplikacjami działającymi w systemie Windows.

[3] Wyłudzanie poufnych danych osobowych przy wykorzystaniu dołączanych do listów elektronicznych odnośników do fałszywych stron internetowych (głównie instytucji finansowych).

W sierpniu 2009 r. fińska grupa Codenomicon poinformowała o odkryciu poważnego błędu w większości otwartych parserów XML (programów do analizowania i przekształcania struktury dokumentu XML).



Rys. 2. Liczba luk wykrytych w przeglądarkach internetowych w 2008 roku (na podstawie [Secunia 2008])

Z kolei we wrześniu 2009 r. pojawił się w sieci exploit dla serwera aplikacji IIS v.5.0 (ang. *Internet Information Services*), opublikowany przez specjalistę ds. bezpieczeństwa. Już po kilku dniach pojawiła się jego udoskonalona wersja, umożliwiająca przeprowadzenie ataku DDoS, jeśli w IIS jest włączona obsługa protokołu FTP (ang. *File Transfer Protocol*). Nowe wersje IIS eliminują wiele wad wcześniejszych rozwiązań, jednak rzesze użytkowników korzystają z dawnych implementacji usługi.

Statystyki pokazują, że na przykład wiele chińskich aplikacji zawiera w zabezpieczeniach proste błędy, dzięki którym bez problemu można zaatakować system Windows. Z tego powodu firma Microsoft chce nawiązać kontakty z hakerami z Chin oraz przeprowadzić tam szkolenia z zakresu technik tworzenia bezpiecznego kodu źródłowego.

Dobry przykład współczesnych zagrożeń sieciowych stanowi niedawno dokonane włamanie do systemu komputerowego firmy PayChoice prowadzącej obsługę płac tysięcy małych i średnich przedsiębiorstw w Stanach Zjednoczonych. W wyniku przeprowadzonego ataku hakerzy zdobyli adresy e-mailowe klientów tej firmy, oraz hasła i nazwy logujących się do jej portalu użytkowników. Przejęte dane zostały wykorzystane do rozesłania informacji o konieczności zainstalowania wtyczki do przeglądarki. W rzeczywistości rzekoma wtyczka wykorzystuje luki w oprogra-

mowaniu (Internet Explorer, Adobe Flash, Adobe Reader) i pobiera konia trojańskiego (Trojan.Downloader.Bredolab.X), który dokonuje kradzieży poufnych danych dotyczących kont bankowych [Bezp 2009].

O tym, jak niebezpieczne mogą być konsekwencje występowania luk w systemach oprogramowania świadczy działanie wykorzystujących je najnowszych szkodliwych kodów. W październiku 2009 roku pojawiła się wiadomość o nowym typie konia trojańskiego (Trojan.URLzone), sterowanego z serwera znajdującego się na Ukrainie, który zaatakował użytkowników jednego z niemieckich banków. Liczbę zarażonych nim komputerów szacuje się na kilka tysięcy. Do infekcji dochodzi po otworzeniu załącznika do e-maila lub wejściu na witrynę ze szkodliwym kodem. Instalacja konia trojańskiego jest możliwa z powodu luk w przeglądarkach internetowych. Szkodliwy kod uaktywnia się po wejściu użytkownika na stronę banku – i kolejno ustala wysokość możliwej do pobrania kwoty, dokonuje przelewu, usuwa dane o nielegalnie dokonanej operacji i wyświetla fałszywy stan konta (sprzed jej realizacji). Użytkownicy korzystający z tak zainfekowanych komputerów nie są świadomi przeprowadzanych transakcji. Ponadto koń trojański dokonuje kradzieży haseł i nazw użytkowników z witryn bankowych.

W sytuacji, gdy tak wiele programów zawiera luki, i gdy zagrożenie bezpieczeństwa systemów jest nieustanne – niezwykle istotne staje się wykrywanie ataków, do czego służą specjalistyczne narzędzia, takie jak Autopsy, EnCase, ProDiscover, F-Response czy RogueScanner. Stale opracowywane są nowe technologie pozwalające podnosić poziom ochrony systemów informatycznych. W lipcu 2009 roku pracownicy firmy Madiant poinformowali o opracowaniu nowego sposobu wykrywania ataków, które polegają na wprowadzaniu kodu do istniejących procesów i nie pozostawiają śladów ingerencji na twardym dysku. W proponowanej metodzie wykorzystywane jest zaadaptowane oprogramowanie Memoryze firmy Madiant, które przechwytuje pakiety przesyłane pomiędzy Meterpreterem (kodem stanowiącym środowisko wyposażone we własną kompletną powłokę systemową) a zarządzającym nim serwerem. Analiza tych pakietów pozwala określić rodzaj ataku i jego zakres. Niestety, prezentowane narzędzie nie potrafi przechwycić całej aktywności szkodliwego procesu, jednak pokłada się nadzieje w możliwości jego udoskonalenia.

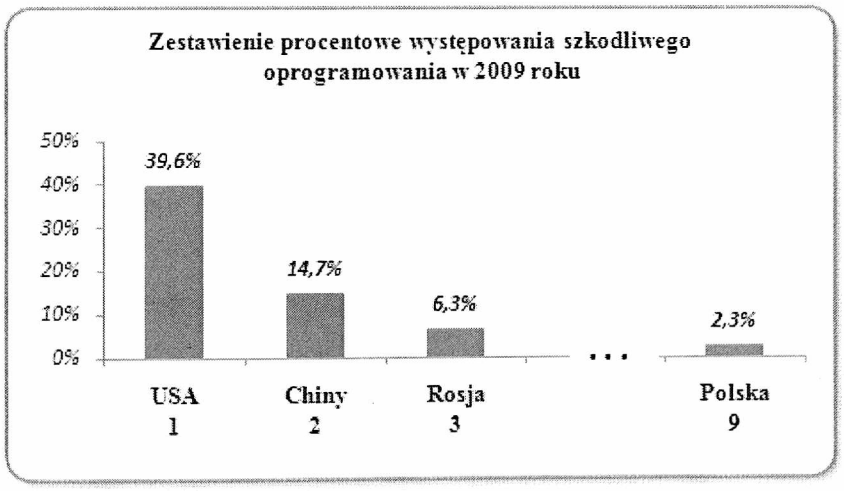
Stosunkowo nową metodą identyfikacji zagrożeń jest koncepcja zwana honeypot. Polega ona na wystawienia na atak wybranego zasobu systemu (np. oprogramowania, usługi czy fikcyjnego konta) w celu wyśledzenia i zniszczenia źródła tego ataku. Honypoty (pułapki na hakerów) mogą przyjmować wiele form i mieć różne zastosowanie. Metodę tę wykorzystuje na przykład narzędzie Glastopf umożliwiające znalezienie przejętych serwerów, na których działają boty[4] napisane w języku PHP [Dark 2009].

[4] Zadaniem botów jest utrzymywanie porządku w kanale IRC; boty atakowane są przez szkodliwe oprogramowanie (np. robaki), które najczęściej kontrolowane jest przy pomocy przesyłanych przez sieć IRC komunikatów.

3. ATAKI NA STRONY INTERNETOWE

Rozwój technologii internetowych wpłynął na zmianę sposobu przekazywania danych. Wraz z integracją baz danych z serwerami WWW i zastosowaniem dynamicznych metod generowania treści – portale i strony internetowe przerodziły się z prostych, statycznych stron w aplikacje WWW. Z kolei ze wzrostem interakcji użytkowników z aplikacjami internetowymi rośnie zagrożenie dla zasobów sieciowych. Coraz częściej ataki na systemy informatyczne prowadzone są nie na poziomie dostępowym (chronionym np. przez zapory sieciowe), lecz bezpośrednio poprzez warstwę aplikacji. Także protokół HTTP wykorzystywany przez aplikacje internetowe do komunikacji z użytkownikiem wykazuje dużą podatność na ataki, gdyż był stworzony dla statycznych stron WWW. Ataki na aplikacje internetowe skupiają się wokół takich obszarów, jak: uwierzytelnianie, zarządzanie sesją, interakcja z bazą danych oraz akceptacja danych wejściowych.

Z raportu *Security Threat Report: July 2009 update* firmy Sophos wynika, że w 2009 roku najwięcej szkodliwego oprogramowania było na amerykańskich (prawie 40%), chińskich i rosyjskich stronach WWW [CERT]. Polska w tym rankingu zajmuje dziewiąte miejsce (rys. 3).



Rys. 3. Procentowe zestawienie występowania szkodliwego oprogramowania na stronach WWW w 2009 roku (na podstawie [CERT])

Rozpowszechnianie szkodliwego oprogramowania przez strony internetowe jest możliwe głównie z powodu zabezpieczania serwerów stron WWW słabymi hasłami, których złamanie nie stanowi problemu (np. metodą słownikową). W Internecie można znaleźć strony instruujące, w jaki sposób łamać hasła administratora, oferujące tzw.

tęczowe tablice (ang. *rainbow tables*) zawierające wszystkie kombinacje znaków do łamania haseł, oraz strony udostępniające sumaryczną moc kilkudziesięciu komputerów, które można wykorzystać do szybkiego łamania haseł (w ciągu kilkudziesięciu sekund). Tablice te pozwalają na złamanie większości popularnych funkcji skrótu (np. MD5[5], SH-1, NTLM[6], Cisco Pix). Proponowaną metodą zabezpieczenia systemu przed takim działaniem jest dodawanie, jako argumentu funkcji skrótu, losowo wygenerowanej wartości. Większość aplikacji internetowych napisanych w języku PHP szyfruje hasła użytkowników przy wykorzystaniu funkcji MD5, natomiast większość dystrybucji GNU/Linuksa i systemów BSD wykorzystuje dodatkowo generowane wartości losowe do zapisywania haseł, dlatego też są one trudniejsze do złamania.

Wiele internetowych aplikacji bazodanowych pisanych przez mało doświadczonych programistów, nie posiada odpowiedniej walidacji danych wprowadzanych przez formularze. Dane mogą podlegać walidacji na serwerze, z wykorzystaniem mechanizmu sesji, i po stronie klienta. Język PHP oferuje programistom implementację sesji, niestety, nie jest ona doskonała – aby móc ją bezpiecznie użyć w systemach autoryzacji mechanizm wymaga modyfikacji. Wielu doświadczonych programistów opracowuje własny system sesji.

Aby zapewnić bezpieczeństwo systemu zarządzania bazą danych MySQL w zastosowaniach internetowych, należy między innymi zwrócić uwagę na zarządzanie uprawnieniami przez system bazodanowy, kwestie związane z zapewnieniem integralności danych, oraz ochronę systemu przed wprowadzaniem szkodliwego kodu do zapytań SQL (ang. *SQL Injection*). Są to główne zagrożenia dla danych, jednak zagrożenie to nie wynika ze słabości systemu bazodanowego, lecz sposobu przetwarzania danych przez aplikację kliencką. Dlatego też zabezpieczenia powinny być implementowane właśnie na poziomie aplikacji, na przykład poprzez weryfikację danych wejściowych, stosowanie funkcji wyjścia, unikanie korzystania z konta o maksymalnych uprawnieniach, nie ujawniania użytkownikom schematu bazy danych ani komunikatów o błędach.

Z raportu firmy IBM za pierwsze półrocze 2009 roku wynika, że najwięcej luk, na które nie opracowano nakładek systemowych, odnotowano w oprogramowaniu Joomla! [IBM 2009]. W systemach tego sprzedawcy „nie załatano" aż 80% z 40 odkrytych luk (z oprogramowania tego korzysta między innymi portal tygodnika Przekrój i serwis motoryzacyjny Porsche) [Joomla 2009]. Joomla! to niezwykle popularny (11 milionów pobrań), oparty na języku PHP, bezpłatny system open-source do zarządzania treścią (CMS, ang. *Content Management System*). Umożliwia on zakładanie i modyfikowanie witryn internetowych.

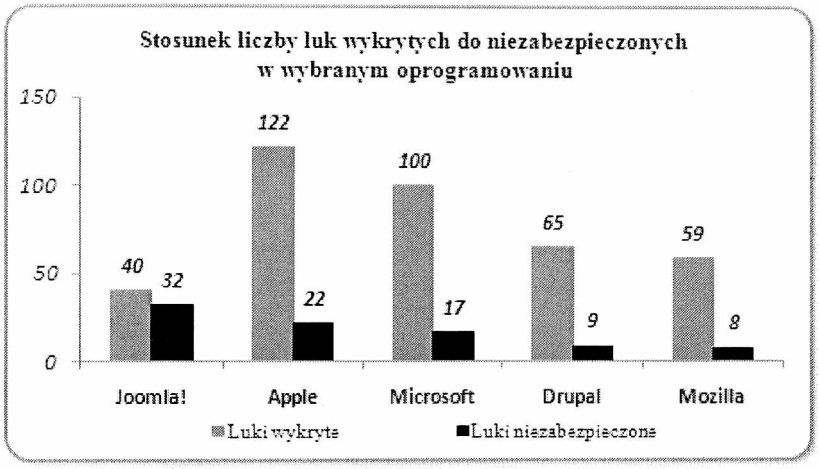
Popularnym systemem CMS wykorzystującym język PHP jest też Drupal (wykorzystywany np. przez Biały Dom w USA), współpracujący z bazami danych MySQL i PostreSQL, oraz serwerami Apache i IIS. Zestawienie liczby wykrytych i pozostają-

[5] *Message Digest Algorithm* – funkcja mieszająca opracowana przez Rona Rivesta dająca w wyniku 128-bitowy skrót.

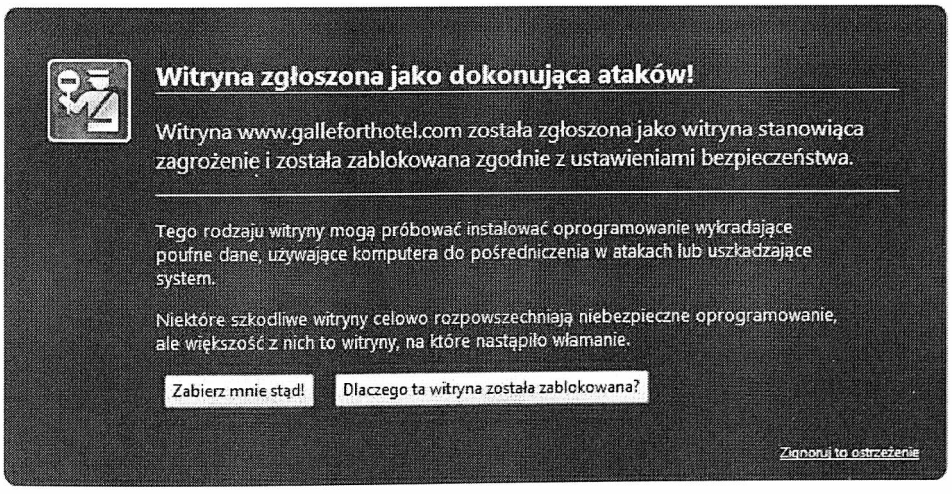
[6] *NT LAN Manager* – kryptograficzny protokół sieciowy opracowany przez firmę Microsoft.

cych bez nakładek systemowych luk w oprogramowaniu różnych dostawców przedstawiono na rysunku 4.

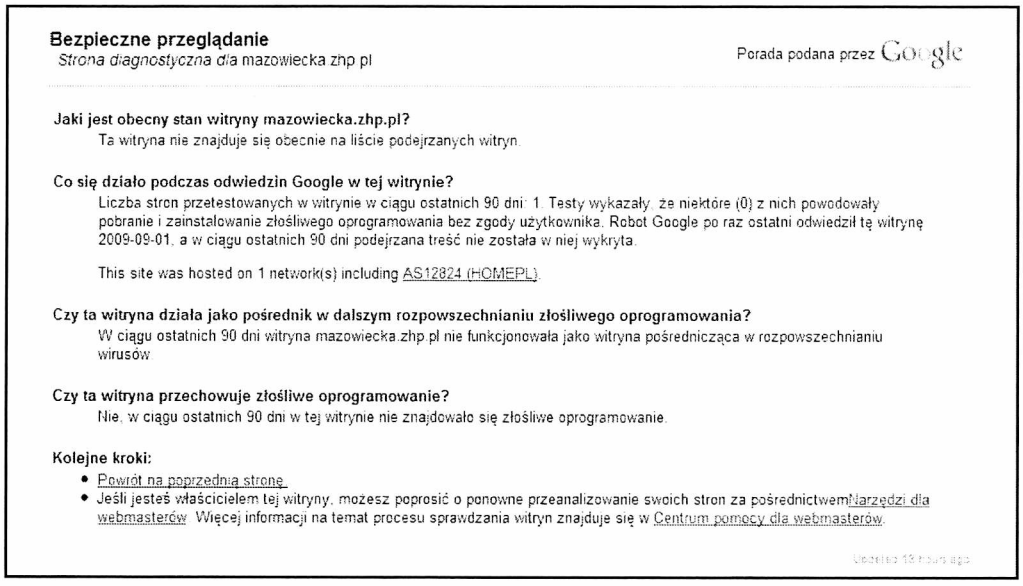
Zdarza się, że przy próbie przejścia na wybraną stronę WWW otrzymuje się ostrzeżenie o niebezpieczeństwie lub że dana strona jest niedostępna (rys. 5 i 6). Wyszukiwarka Google ostrzega użytkowników, jeśli wybrana strona WWW może nieść zagrożenie, jednak system ten nie usuwa jej ze swojego indeksu dopuszczając do przejścia na nią.



Rys. 4. Liczba luk wykrytych i nie zabezpieczonych w pierwszym kwartale 2009 roku różnych sprzedawców (na podstawie [IBM 2009])



Rys. 5. Blokowanie dostępu do zagrożonej witryny



Bezpieczne przeglądanie
Strona diagnostyczna dla mazowiecka.zhp.pl

Porada podana przez Google

Jaki jest obecny stan witryny mazowiecka.zhp.pl?
Ta witryna nie znajduje się obecnie na liście podejrzanych witryn.

Co się działo podczas odwiedzin Google w tej witrynie?
Liczba stron przetestowanych w witrynie w ciągu ostatnich 90 dni: 1. Testy wykazały, że niektóre (0) z nich powodowały pobranie i zainstalowanie złośliwego oprogramowania bez zgody użytkownika. Robot Google po raz ostatni odwiedził tę witrynę 2009-09-01, a w ciągu ostatnich 90 dni podejrzana treść nie została w niej wykryta.

This site was hosted on 1 network(s) including AS12824 (HOMEPL).

Czy ta witryna działa jako pośrednik w dalszym rozpowszechnianiu złośliwego oprogramowania?
W ciągu ostatnich 90 dni witryna mazowiecka.zhp.pl nie funkcjonowała jako witryna pośrednicząca w rozpowszechnianiu wirusów.

Czy ta witryna przechowuje złośliwe oprogramowanie?
Nie, w ciągu ostatnich 90 dni w tej witrynie nie znajdowało się złośliwe oprogramowanie.

Kolejne kroki:
- Powrót na poprzednią stronę.
- Jeśli jesteś właścicielem tej witryny, możesz poprosić o ponowne przeanalizowanie swoich stron za pośrednictwem Narzędzi dla webmasterów. Więcej informacji na temat procesu sprawdzania witryn znajduje się w Centrum pomocy dla webmasterów.

Rys. 6. Raport dla strony zidentyfikowanej jako zagrożona

W celu wyeliminowania zautomatyzowanego działania botów[7] (np. zbierania informacji, pobierania plików, wysyłania spamu) w wielu aplikacjach internetowych wykorzystywany jest mechanizm CAPTCHA (ang. *Completely Automated Public Turing test to tell Computers and Humans Apart*), czyli zautomatyzowany test do odróżniania działań człowieka od maszyny. System ten, w celu sprawdzenia, kto podejmuje daną akcję, wymaga wprowadzenia z klawiatury ciągu przedstawionych graficznie, zdeformowanych znaków (Rys. 7). Wymagana czynność jest łatwa do wykonania przez człowieka, zaś dla automatów odczyt staje się niemożliwy.

Rys. 7. Przykład kodu CAPTCHA

[7] Bot – komputer zainfekowany ukrytym szkodliwym kodem działający w ramach całej grupy (botnetu), nad którym przejęto zdalnie kontrolę.

Technikę tę zakupiła firma Google w celu wykorzystania jej do przepisywania książek i starodruków (projekt reCAPTCHA), w przypadku problemów z odczytywaniem znaków przez programy OCR (ang. *Optical Character Recognition*).

W celu ochrony użytkowników Internetu przed atakami typu XSS[8] (ang. *Cross-Site Scripting*) firma Mozilla zaproponowała nową technologię CSP (ang. *Content Security Policy*), która będzie zastosowana w przeglądarce Firefox v3.6 umożliwiając autorom stron i aplikacji internetowych selektywny wybór treści do wyświetlania w przeglądarce. Technologia CSP dzięki możliwości dynamicznego wyłączania niechcianych treści, blokowania skryptów i programów o szkodliwym działaniu (dodawanych do stron lub aplikacji), umożliwia ochronę przed tego typu atakami. W styczniu w 2008 roku luka non-persistent XSS w internetowym serwisie włoskiego banku Banca Fideuram umożliwiała podmianę fragmentu strony logowania, co mogło prowadzić do phishingu i nieświadomego udostępnienia danych klientów nieuprawnionym osobom [Italy 2008].

Z opublikowanych przez firmę ESET danych wynika, że we wrześniu 2009 roku najczęstszym powodem infekcji komputerów polskich użytkowników były kolejno:

- zagrożenia typu Win32/PSW.OnLineGames (konie trojańskie działające jak keyloggery),
- szkodliwe programy INF/Autorun ukrywające się w plikach autostartu różnego typu nośników, infekujące m.in. za pośrednictwem pamięci USB (7,1% wszystkich infekcji),
- robak Conficker (4,8% infekcji),
- koń trojański Win32/TrojanDownloader.Bredolab (2,44% infekcji), który po przedostaniu się na komputer za pośrednictwem plików PDF lub SWF[9] łączy się ze zdalnym serwerem z wykorzystaniem protokołu HTTP i pobiera kolejne zagrożenia (np. spyware czy keylogger).

4. ATAKI NA ALGORYTMY I PROTOKOŁY KRYPTOGRAFICZNE

Wykorzystywane do ochrony danych (ich poufności, integralności i autentyczności) protokoły kryptograficzne implementowane w standardach sieci bezprzewodowych WEP i WPA oraz protokołach transmisyjnych (SSH, SSL, IPSec), a także mechanizmy uwierzytelniania też nie są pozbawione wad. Wszystkie odkrywane w nich luki mogą być wykorzystane do przeprowadzenia ataku przy pomocy dostępnych narzędzi.

---

[8] Polegają na umieszczeniu szkodliwego kodu na stronie WWW lub w hiperłączu, a następnie zainfekowaniu nim komputera każdego odwiedzającego ją użytkownika.

[9] Format grafiki wektorowej.

Problemy z protokołem WEP, który ma chronić przed podsłuchem, tkwią w samym protokole, jego implementacji oraz sposobie wykorzystania dostępnych mechanizmów. Ataki na WEP wykorzystują fakt, iż nie są w nim szyfrowane nagłówki pakietów, wektory inicjalizujące oraz pola zawierające identyfikator. Protokół WEP jest podatny na ataki man-in-the-middle z powodu braku uwierzytelniania punktu dostępowego, a także ataki przejmowania sesji – w wyniku braku mechanizmów zapewniania poufności komunikatów i niski poziom uwierzytelniania, skutkiem czego atakujący może anulować połączenie użytkownika z punktem dostępowym i podszyć się pod niego. Niestety, mimo licznych wad, ten standard bezpieczeństwa sieci bezprzewodowych jest nadal powszechnie używany. Mocniejsze zabezpieczenia udostępnia WPA, w którym wprowadzono zmiany w algorytmie szyfrowania RC4, zaimplementowano protokół zarządzania kluczami TKIP (ang. *Temporal Key Integrity Protocol*), do integralności danych zastosowano mechanizm MIC (ang. *Message Integrity Code*), wprowadzono obowiązkowe uwierzytelnianie przy pomocy protokołu EAP (ang. *Extensible Authentication Protocol*), jednak i ten standard zawiera poważne luki w zabezpieczeniach. Umożliwiają one podjęcie ataku na protokół TKIP oraz ominięcie procesu uwierzytelniania (na skutek wad protokołu RADIUS[10]). Przedmiotem ataku może być też protokół LEAP (ang. *Lightweight Extensible Authentication Protocol*) Cisco wykorzystujący zdalny serwer RADIUS. Do przeprowadzenia ataku na urządzenia z włączoną obsługą LEAP można użyć dostępnych narzędzi, których zadaniem jest przechwytywanie i deszyfrowanie słabych haseł protokołu z punktów dostępowych Cisco oraz kart bezprzewodowych.

W sierpniu 2008 roku pojawiła się wiadomość o opracowaniu przez grupę japońskich naukowców nowej metody łamania zabezpieczeń WPA (wykorzystującej wykrytą przez nich wadę w protokole TKIP), która pozwala uzyskać nieautoryzowany dostęp do sieci bezprzewodowej w czasie krótszym od 1 minuty (do tej pory można było osiągnąć cel w co najmniej kwadrans). Na szczęście standard WPA2 używający algorytmu szyfrowania AES okazał się odporny na ten rodzaj ataku.

Atakowany jest również popularny protokół uwierzytelniania użytkowników sieci lokalnych Kerberos. Jego słabym punktem jest moment uwierzytelniania użytkownika w serwerze uwierzytelniającym – istnieje bowiem możliwość przechwycenia komunikatu przesyłanego między serwerem a użytkownikiem.

Na ataki typu man-in-the-middle podatny jest też protokół SSH (ang. *Secure Shell*), gdyż dopuszczalne jest jego stosowanie bez wcześniejszej weryfikacji klucza i serwera. System można zaatakować, jeśli na przykład użytkownik zaakceptuje klucz podany przez atakującego jako klucz serwera. Protokół SSH wykazuje też podatność na atak DoS, ponieważ atakujący może wymusić na serwerze wykonywanie obciążających operacji ustanawiania połączenia i wymiany kluczy.

---

[10] *Remote Authentication Dial-In User Service* – standardowy wielofunkcyjny protokół i usługa uwierzytelniania sieciowego o wielu zastosowaniach.

Z końcem 2008 roku pojawiła się informacja o odkryciu słabości w internetowym systemie certyfikatów protokołu SSL[11] (ang. *Secure Socket Layer*) wykorzystywanym do tzw. bezpiecznych połączeń, m.in. w rozproszonych systemach baz danych. Zostało udowodnione, że można (wykorzystując słabość używanej funkcji haszującej MD5) utworzyć certyfikat dla dowolnej strony WWW – i mimo iż został sfałszowany, jest on akceptowany przez większość popularnych przeglądarek internetowych. Zatem metodę tę można wykorzystać do tworzenia certyfikatów fałszywym stronom internetowym (np. stronom podszywającym się pod banki). Firmy posługujące się algorytmem MD5 przy tworzeniu certyfikatów powinny umożliwić użytkownikom bezpłatne zastąpienie go innymi mocniejszymi mechanizmami (SHA-1[12], SHA-2, SHA-3). Jedna z firm wydających certyfikaty SSL – VeriSign – już w styczniu 2009 roku zapowiedziała wprowadzenie certyfikatów RapidSSL SHA-1 [Bezp 2009].

5. WADY SYSTEMÓW BAZ DANYCH NA WYBRANYCH PRZYKŁADACH

Na bezpieczeństwo systemu bazodanowego jako całości wpływa bezpieczeństwo jego poszczególnych elementów. Odpowiednie zabezpieczanie systemu stwarza coraz to nowe problemy, co wiąże się z otwartością systemów i ich dostępnością poprzez Internet i wprowadzaniem nowych technologii. Bazy danych (często rozproszone) wymagają odpowiednich zabezpieczeń ze strony systemu zarządzania bazą danych, serwerów, protokołów komunikacyjnych – zarówno podczas gromadzenia, przechowywania i transmisji danych, jak również podczas tworzenia i przechowywania kopii zapasowych. Poziom ochrony systemu bazodanowego zależy przede wszystkim od poprawności jego implementacji i środowiska eksploatacji (w tym działania aplikacji po stronie użytkownika), ale również od wiedzy i rzetelności administratorów.

Do najczęściej stosowanych mechanizmów ochrony danych zalicza się kontrolę dostępu, kontrolę spójności danych, ich szyfrowanie, audyt bezpieczeństwa systemu, tworzenie kopii zapasowych. Mechanizmy ochrony danych są zawarte zarówno w systemie operacyjnym, systemie zarządzania bazą danych, oprogramowaniu aplikacyjnym, jak i wszelkim innym oprogramowaniu wykorzystywanym przez dany system. Podstawowe mechanizmy ochrony dostępu do danych to:

- uznaniowa kontrola dostępu (ang. *Discretionary Access Control*), oparta na identyfikacji użytkownika lub grupy użytkowników i obiektów,
- obowiązkowa kontrola dostępu (ang. *Mandatory Access Control*), oparta na kontroli uprawnień użytkowników (klasach poufności).

---

[11] Standard bezpiecznej transmisji danych w Internecie; zapewnia ich poufność (szyfry symetryczne), integralność (funkcje skrótu) i autentyczność (system certyfikatów).

[12] *Secure Hash Algorithm* – SHA-1 tworzy skrót długości 160 bitów; nowe funkcje dają w wyniku odpowiednio 256 i 512 bitów.

Zagrożenia dla baz danych wynikają zarówno z przypadkowych, nieświadomych działań ich administratorów i użytkowników, jak i z celowych działań nieuprawnionych i destrukcyjnych. Komunikacja pomiędzy warstwami systemu zarządzania bazą danych realizowana jest za pomocą protokołu IP, który ma wiele wad. Komunikacja z wykorzystaniem IP jest narażona na podsłuchiwanie, przechwytywanie i modyfikowanie pakietów przesyłanego strumienia danych. Dlatego dodatkowo wykorzystuje się inne protokoły transmisyjne, takie jak SSH, SSL, TLS (ang. *Transport Layer Security*; SSL v.3) i IPSec (ang. *Internet Protocol Security*).

Wiele powszechnie wykorzystywanych systemów zarządzania bazami danych, pomimo wielu zastosowanych mechanizmów ochrony, nie jest niestety bez wad. Przykładowo, popularny system Microsoft Access (z pakietu Microsoft Office) umożliwia tworzenie grup roboczych, zakładanie kont indywidualnych i nadawanie uprawnień użytkownikom, szyfrowanie danych, rozdzielenie warstwy danych od warstwy prezentacji, ukrywanie obiektów, zabezpieczanie bazy hasłem. Wcześniejsze wersje pakietu Microsoft Office do szyfrowania wykorzystywały słabe algorytmy szyfrujące, stąd odczyt hasła zabezpieczającego plik nie stanowił dużej przeszkody. W Microsoft Office 2007 do szyfrowania haseł wykorzystywany jest silniejszy, 128-bitowy algorytm szyfrujący. W lipcu 2008 roku firma Microsoft opublikowała komunikat na temat luki w zabezpieczeniach formantu ActiveX przeglądarki Snapshot Viewer dla programu Microsoft Access wersji 2000, 2002, 2003 (wersja 2007 nie zawiera tej kontrolki) [Microsoft 2008]. Luka umożliwiała atakującemu uzyskanie takich samych uprawnień, jakie posiadał zalogowany użytkownik (tak więc największe zagrożenie dla systemu jest podczas pracy użytkownika na prawach administratora).

Niezwykle popularnym systemem zarządzania bazą danych jest MySQL (open-source). Hasła MySQL są zapisywane w tabeli mysql.user w postaci zaszyfrowanej przy użyciu algorytmu MD5. Poprzez specyficzne dane wpisywane przez użytkownika może dojść do błędów zagrażających nie tylko danym w bazie, ale i całemu serwerowi. Manipulując żądaniami można uzyskać nielegalny dostęp do zasobów danych, na przykład poprzez domyślne odwoływanie się do plików szablonów czy podstron danego serwisu. Informacje na temat możliwości przeprowadzenia tego typu ataku są dostępne w Internecie (!).

Rzadkim, ale niebezpiecznym atakiem na bazę MySQL jest atak typu Arbitrary File Download (pobieranie dowolnego pliku). Przy braku właściwej kontroli, po odpowiednim spreparowaniu adresu pliku (przeznaczonego do pobierania z serwera), możliwe jest skopiowanie plików przez osoby nieupoważnione.

Pomimo dużego przywiązywania wagi do spraw bezpieczeństwa przez firmę Oracle, system zarządzania bazą danych tej firmy też nie jest całkowicie zabezpieczony. Najbardziej narażony na atak jest komponent Oracle Listener, odpowiedzialny za komunikację klient–serwer oraz serwer–serwer. Na przykład, za pomocą skryptu tnscmd (dostępnego w sieci Internet) napisanego w języku Perl można przeprowadzić atak na Oracle Listener i zebrać wiele informacji na temat systemu bazy danych (m.in. ścieżkę dostępu do plików z logami czy zmienne systemowe komponentu Oracle Li-

stener). Atak będzie nieskuteczny, jeśli Listener będzie zabezpieczony hasłem, ale w standardowych instalacjach bazy Oracle Listener nie jest to domyślnie stosowane.

Ochrona danych w systemach informatycznych dużych firm, w których wdrożono systemy ERP (ang. *Enterprise Resource Planning*) umożliwiające zaawansowane zarządzanie zasobami, dotyczy zarówno ochrony danych zasobów materiałowych, jak i danych osobowych. Systemy ERP mają budowę modułową, czyli składają się z wielu niezależnych, ale współpracujących ze sobą aplikacji związanych z różnymi obszarami działalności firmy (np. dostawami, sprzedażą, działem osobowym, księgowością i finansami, planowaniem produkcji i gospodarką materiałową). Systemy takie muszą spełniać wymagania określone przez Ministra Spraw Wewnętrznych i Administracji z 29 kwietnia 2004 r. w sprawie dokumentacji przetwarzania danych osobowych oraz warunków technicznych i organizacyjnych, jakim powinny odpowiadać urządzenia i systemy informatyczne służące do przetwarzania danych osobowych (Dz. U. Nr 100 Poz. 1024). Administrator systemu powinien opracować odpowiednią dokumentację zawierającą opis przyjętej polityki bezpieczeństwa oraz instrukcję zarządzania systemem informatycznym. W dokumentacji tej między innymi muszą być zawarte procedury dostępu do danych, tworzenia i przechowywania kopii zapasowych oraz zarządzania kontami użytkowników. Pomimo spełnienia wielopłaszczyznowych, wysokich wymagań określonych w ustawie, zdarzają się luki w wykorzystywanych systemach spowodowane na przykład:

- przechowywaniem haseł niezaszyfrowanych lub słabo zaszyfrowanych,
- dostępem do danych poprzez różne interfejsy (np. ODBC, Web Services),
- lokalnym przechowywaniem danych w pamięci cache w postaci niezaszyfrowanej lub słabo zaszyfrowanej,
- udostępnianiem danych fragmentarycznych w różnych aplikacjach systemu, na podstawie których można dotrzeć do innych danych.

Zagrożeniem dla bezpieczeństwa danych może być również zbyt duża liczba użytkowników z uprawnieniami administratora, pozostawianie kont i uprawnień osobom zwolnionym z pracy, brak kontroli nad wydrukami, kopiowanie danych na nośniki zewnętrzne, zbędne (niekontrolowane) kopiowanie danych, zbędne powielanie wydruków itp.

Dla każdego wyróżnionego obszaru, który powinien podlegać zabezpieczeniu i wzmożonej kontroli, dostępne są różne rozwiązania programowe i sprzętowe. Wśród nich znajdują się narzędzia do zarządzania hasłami, uprawnieniami, kopiami bezpieczeństwa, do szyfrowania dysków oraz programy ochronne (antywirusowe, antyspamowe, antyszpiegujące, zapory sieciowe). Ponadto oferowane są wysoko specjalizowane narzędzia do przeprowadzania audytu bezpieczeństwa systemów i raportowania wyników. Stosowanie dostępnych środków zabezpieczeń wymaga właściwego ich skonfigurowania, ciągłej aktualizacji, obciąża znacząco system spowalniając jego działanie, a także wiąże się z dodatkowymi kosztami. Ponadto wymaga stałego dokształcania się osób, od których zależy poziom ochrony systemów informatycznych.

6. PODSUMOWANIE

Jednym z największych problemów bezpiecznego użytkowania systemów informatycznych jest brak systematycznego aktualizowania oprogramowania. Automatyczna aktualizacja systemów operacyjnych jest niewystarczająca, bowiem luki, także o znaczeniu krytycznym, wykrywane są również w ich komponentach oraz oprogramowaniu użytkowym. Obowiązkiem administratorów jest okresowe dokonywanie audytu bezpieczeństwa systemu przy pomocy specjalizowanych narzędzi, jakimi są skanery, jednak i one nie są pozbawione wad. Narzędzia te są w stanie wykryć znane zagrożenia, jednak nie potrafią łączyć istniejących podatności, by wskazać poważniejsze luki w poziomie zabezpieczeń systemu. Ponadto są one na tyle skuteczne, na ile aktualne są ich bazy zagrożeń. Jeśli narzędzia nie mają zapewnionej bieżącej aktualizacji, wyniki nie mogą być wiarygodne. Niektóre z bardziej rozbudowanych funkcjonalnie skanerów mogą okazać się trudne w prawidłowej obsłudze, a niewłaściwie użytkowane same stanowić zagrożenie (jak np. Nessus pozwalający na edycję dostępnych testów i pisanie własnych skryptów).

Sporządzane na podstawie wykonanych testów raporty niosą podpowiedzi na usunięcie zidentyfikowanych zagrożeń, jednak do rozwiązania problemów niezbędna jest wiedza i doświadczenie audytora. I tu niebagatelną przeszkodę stanowią koszty przeprowadzenia profesjonalnego audytu bezpieczeństwa.

Dystrybutorzy sieciowi podejmują różnego typu starania, by zapewnić maksymalne bezpieczeństwo oferowanych usług sieciowych. Przykładowo, w celu podniesienia poziomu bezpieczeństwa korzystania z poczty elektronicznej (a tym samym systemów komputerowych), Telekomunikacja Polska dnia 1 grudnia 2009 roku doda do obecnie blokowanych portów TCP/UDP (wykorzystywanych do atakowania i przejmowania kontroli nad komputerami) port 25 TCP. Głównym celem proponowanych zmian jest ograniczenie wysyłania i otrzymywania przez internautów spamu. Zabieg ten wiąże się z koniecznością wykorzystywania przez programy pocztowe (np. Outlook, Outlook Express) innego portu niż 25 (np. 587, 465).

Nie należy jednak zapominać, iż istnieje pewien procent ataków nie bazujących na wykorzystywaniu znanych luk i podatności – i te stanowią jeszcze większe zagrożenie, gdyż są trudne do przewidzenia.

LITERATURA

[Bezp 2009] www.bezpieczenstwo.onet.pl

[CERT] www.cert.pl

[CVE] www.cve.mitre.org/about/terminology.html

[Dark 2009] http://darkreading.com/database_security/security/app-security/showArticle.jhtml?articleID=221300001

[IBM 2009] *IBM Internet Security Systems 2009 Mid-Year Trend & Risk Report*, IBM Global Technology Services, August 2009.

[Italy 2008] http://news.netcraft.com/archives/2008/01/08/italian_banks_xss_opportunity_seized_by_fraudsters.html
[Joomla 2009] www.joomla.pl
[Kenan 2007] Kenan K., *Kryptografia w bazach danych. Ostatnia linia obrony*, Wydawnictwo Naukowe PWN, Warszawa, 2007.
[Kurek 2009] Kurek M., Lutynia A., *Zagrożenia związane z udostępnianiem aplikacji w sieci Internet*, [w:]Materiały konferencyjne I Ogólnopolska Konferencja Informatyki Śledczej, Katowice 2009.
[McCScaKur 2006] McClure S., Scambray J., Kurtz G., *Hacking zdemaskowany. Bezpieczeństwo sieci – sekrety i rozwiązania*, Wydawnictwo Naukowe PWN, Warszawa, 2006.
[Microsoft 2008] www.microsoft.com/poland/technet/security/advisory/955179.mspx
[RusMul 2004] Rusell R., Mullen T., Dan F.X. i inni, *Hakerzy atakują. Jak przejąć kontrolę nad siecią*, Helion, Gliwice, 2004.
[SchMen 2004] Scheliga M., Mendrala D., *Bezpieczeństwo Twojego komputera*, Helion, Gliwice 2004.
[Secunia 2008] Secunia Stay Secure.2008 Report.

DATABASE HAZARDS IN THE CONTEXT OF SOFTWARE DEFECTS

The security of database systems, often distributed, depends on the level of protection of the environment in which they operate. This paper talks about the problem of safety hazards in information systems, including database systems, caused by errors in software (including operating systems, applications and utility programs). The scale of the phenomenon of the existence of defects, which can be used for unlawful activities in computer networks, is enormous. The only possible action that can be taken in response to discovered vulnerabilities is an installation of system patches released by system manufacturers, but this process does not fully solve the existing problem. New defects are constantly detected, and many users do not update their software often enough.

Prace Naukowe Instytutu Informatyki
Politechniki Wrocławskiej

Zeszyt 10 2009

archiwum elektroniczne,
archiwizacja danych,
backup, digitalizacja

Hanna MAZUR*,
Zygmunt MAZUR*

ELEKTRONICZNE ARCHIWA DANYCH

Przyrost danych, w różnych postaciach, jest coraz większy. Gromadzone i przechowywane są coraz większe zasoby dokumentów, danych graficznych, tekstowych, multimedialnych. W pracy przedstawiono zagadnienia i problemy związane z archiwizacją danych, a szczególności z digitalizacją zasobów i elektronicznymi archiwami danych.

1. WPROWADZENIE

Gromadzenie danych, dokumentów oraz różnych innych zasobów (np. map, fotografii, filmów, słuchowisk itd.) nastręcza coraz więcej problemów. Zasoby te powiększają się lawinowo, wiele spośród nich ma wartość ponadczasową i należy je pieczołowicie przechowywać ale są problemy z ich magazynowaniem, konserwowaniem, porządkowaniem, szybkim wyszukiwaniem i udostępnianiem. Na przykład zasoby Biblioteki Narodowej (BN) na koniec 2008 roku (wraz z dubletami) liczyły 8 706 532 woluminów/jednostek. Obecnie BN gromadzi już tylko po jednym egzemplarzu każdego druku i zrezygnowała z niektórych rodzajów druków (np. z przechowywania plakatów reklamowych). Oprócz tradycyjnych zasobów w postaci papierowej (druków) gromadzone są również zasoby w postaci cyfrowej [BiblNar 2009].

Podmioty, które posiadają dokumentację finansową, osobową (wynikającą z *Kodeksu pracy*) czy materiałową, są zobowiązane do jej przechowywania przez określony ustawowo czas w ściśle określonych warunkach. W ten sposób każda jednostka, instytucja i firma tworzy swoje archiwum, które musi być odpowiednio zorganizowane. O ile w przypadku małych jednostek nie jest to większym problemem, to w przypadku dużych przedsiębiorstw, czy w skali kraju – już jest.

Archiwizowanie danych zaczyna stanowić ogromny problem z wielu względów (przestrzennych, technologicznych, prawnych). Należy przede wszystkim zadecydować:

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

- co archiwizować,
- jak długo przechowywać,
- w jakiej postaci archiwizować (źródłowej, papierowej, cyfrowej, oryginalnej, skompresowanej, zaszyfrowanej itd.),
- komu udostępniać zgromadzone zasoby i na jakich zasadach,
- gdzie magazynować,
- w ilu kopiach,
- jak konserwować,
- jak zapewnić bezpieczeństwo zgromadzonym zasobom.

Podstawowe zadanie związane z archiwizacją danych to zapewnienie odpowiedniej jakości zgromadzonym zasobom i umożliwienie korzystania z nich (czyli ich odczytywania, wyszukiwania, aktualizowania, przetwarzania) obecnie i w przyszłości.

Istnieją uzasadnione obawy, że za kilka lat wiele danych zapisanych w postaci plików nie będzie można odczytać (lub będzie to bardzo utrudnione) na przykład ze względu na zmianę oprogramowania, szybko rozwijające (zmieniające się) technologie, które zmieniają sposób zapisu danych elektronicznych, sprzęt, nośniki danych itd. Wiele osób miałoby dzisiaj problem z odsłuchaniem płyt winylowych ze względu na brak odpowiednich urządzeń (adapterów). Problemem jest również odczyt danych sprzed kilku lat zapisanych na dyskietkach w formacie 5,25" czy 3,5", które nie można już wykorzystać ze względu na brak odpowiednich napędów. Są problemy z odczytaniem treści zapisanych w starych formatach. Zapewne wiele firm i osób ma pliki archiwalne utworzone za pomocą oprogramowania dziś już nie wykorzystywanego, na przykład edytora TAG, ChiWriter lub AmiPro, arkusza kalkulacyjnego QuatroPro czy Works. Odczytanie zawartości tych plików jest więc utrudnione pomimo, że od ich zapisu upłynęło zaledwie kilkanaście lat.

Wiele materiałów czy opisów wydarzeń istnieje tylko w postaci stron WWW w Internecie (jako bieżące informacje, blogi, aktualności), które nie są archiwizowane. Często są usuwane bezpowrotnie, tracone są do nich łącza (powiązania). Na potrzeby różnych okazjonalnych przedsięwzięć zakładane są portale internetowe, często aktualizowane (bez zachowania poprzednich treści) i żyją one na ogół tak długo jak dane przedsięwzięcie. Na przykład witryny firm czy konferencji tworzone są na serwerach, które po pewnym czasie są zmieniane (zastępowane) przez inne, utworzone adresy internetowe stają się nieaktualne a treści tam zawarte zazwyczaj nie są utrwalane w formie papierowej. W ten sposób ginie wiele danych, które po pewnym czasie są już nie do odtworzenia. Z kolei ogrom materiału dostępnego w Internecie może utrudnić archiwistom ustalenie autorów danego zasobu (dzieła) czy historii (różne źródła mogą ją różnie przedstawiać).

2. ARCHIWIZACJA I BACKUP DANYCH

W języku polskim pojęcia archiwizacja (ang. *archiving*) i backup (ang. *backup*) często traktuje się jak synonimy i stosuje się zamiennie. Tymczasem w języku angielskim są to dwa różne pojęcia.

Backup danych to proces związany z tworzeniem kopii zapasowych danych a więc z kopiowaniem danych i ich przechowywaniem w bezpiecznym miejscu, umożliwiającym ich odtworzenie.

Archiwizacja związana jest z umieszczaniem danych w różnych obszarach przechowywania w zależności od uznania ich za aktywne, nieaktywne lub referencyjne (dane podstawowe, muszą być wciąż dostępne, ale czas dostępu do nich nie jest krytyczny). Przydział do jednej z grup jest głównie uzależniony od czasu dostępu do danych. Problemem jest więc zarówno przeprowadzenie odpowiedniej klasyfikacji danych jak i wszystko co się wiąże z odpowiednią organizacja archiwum.

Oprócz archiwizowania danych w celu ich długotrwałego przechowywania, niezwykle ważne jest bieżące sporządzanie ich kopii zapasowych.

Jak wynika z raportu [OGICOM 2009] opracowanego na podstawie badań przeprowadzonych w miesiącach VIII/IX 2008 roku przez firmę OGICOM przy współpracy pracowników Zakładu Innowacji i Przedsiębiorczości Instytutu Organizacji i Zarządzania na Wydziale Informatyki i Zarządzania Politechniki Wrocławskiej, aż 50% firm (małych i średnich) utraciło dane w wyniku awarii sprzętu komputerowego, a prawie 20% z powodu ich nieumyślnego skasowania przez pracownika. Pomimo tego prawie 30% firm (z 470 badanych) nie sporządza kopii zapasowych, podając jako główną przyczynę zbyt dużą czasochłonność tej czynności (47%), 25% przechowuje kopie zapasowe na tym samym komputerze co dane źródłowe. Wykonywanie kopii zapasowych powinno być proste, a najlepiej aby było zautomatyzowane.

Niezwykle ważne jest przechowywanie kopii danych nie tylko na innym nośniku ale jeszcze powinien być to nośnik odległy terytorialnie. Na przykład, w celu zapewnienia ciągłości działania, Giełda Papierów Wartościowych ma system o wartości kilkunastu milinów złotych, zapewniający multibackup online (czyli kopiowanie danych w czasie rzeczywistym w kilku miejscach jednocześnie), a kopie danych są przechowywane na serwerach w dużym oddaleniu geograficznym, tak by pożar czy awaria sieci elektrycznej (nawet w całej okolicy) nie spowodowała braku dostępu do danych i przerwy w funkcjonowaniu giełdy. Oczywiście do zapewnienia ciągłości pracy są jeszcze potrzebne inne rozwiązania sprzętowe (np. macierze dyskowe, taśmy magnetyczne, zabezpieczenia antyprzepięciowe, zasilacze awaryjne, generatory prądu itd.).

Zagadnienia związane z tworzeniem kopii zapasowych i odtwarzania danych po awarii są ściśle związane z zarządzaniem cyklem życia informacji ILM (ang. *Information Lifecycle Management*).

3. ARCHIWA CYFROWE W POLSCE

Aby właściwie tworzyć archiwa należy przede wszystkim zadecydować co ma podlegać przechowywaniu.

Decyzją Ministra Kultury i Dziedzictwa Narodowego dnia 8 marca 2008 roku z Archiwum Dokumentacji Mechanicznej (ADM) utworzono Narodowe Archiwum

Cyfrowe (NAC), którego zadaniem jest archiwizacja zasobów cyfrowych (dokumentów, fotografii, nagrań, filmów), digitalizacja istniejących zasobów papierowych oraz udostępnianie zgromadzonych materiałów (m.in. w sieci Internet). NAC sprawuje również nadzór nad archiwami telewizji, radia, prasy, wytwórni filmowych [NAC 2009]. Obecnie zbiory NAC liczą 2 400 filmów, około 14 milionów zdjęć (z czego ponad 115 tysięcy zostało już zeskanowanych), około 30 tysięcy nagrań dźwiękowych (z czego ponad 15 tys. jest już opisanych), 8 milionów klatek mikrofilmów zabezpieczających zasoby Archiwów Państwowych (AP). Dla AP pracuje 17 pracowni mikrofilmowych oraz jedna pracownia digitalizacji mikrofilmów [AP 2009]. Przykładowe ceny za skorzystanie z zasobów archiwum, to 10 zł za minutę nagrania, 100 zł za zdjęcie formatu A4 w rozdzielczości 600 dpi. Te przykładowe dane obrazują skalę wielkości zasobów i problemów, z jakimi należy się zmierzyć przy archiwizacji.

Od 2008 roku opracowywany jest Zintegrowany System Informacji Archiwalnej ZoSIA, który ma być systemem ogólnodostępnym i jedynym wykorzystywanym przez Archiwa Państwowe, którego zbiory obecnie liczą 34 miliony jednostek materiałowych (ok. 253 km). Do przechowywania i zarządzania tak potężnymi zbiorami danych potrzeba odpowiednich narzędzi i systemów informatycznych. System ZoSIA zbudowany jest na otwartych standardach (ISAD(G), EAD). Obecnie system jest na etapie testowym (zawiera ok. 1 milion rekordów danych).

Z systemem ZoSIA będzie współpracował z opracowywanym obecnie Systemem Digitalizacji Archiwalnej SeDAn. Jego zadaniem będzie zarządzanie digitalizacją zasobów archiwalnych (selekcją materiałów do archiwizowania, skanowaniem, składowaniem na macierzach dyskowych, udostępnianiem kopii cyfrowych użytkownikom w Internecie). Zbudowany będzie również z wykorzystaniem otwartych standardów.

W celu efektywnego gromadzenia, przechowywania i udostępniania publikacji tworzone są między innymi biblioteki cyfrowe (BC), które są tak projektowane by umożliwiały długotrwałe przechowywanie danych cyfrowych w dowolnym formacie, opisanych przy pomocy zróżnicowanych schematów metadanych. Ich architektura jest coraz częściej zorientowana na usługi (ang. *Service Oriented Architecture,* SOA). Takie podejście umożliwia integrowanie ich z innymi systemami i pełne wykorzystywanie ich funkcjonalności. Na świecie opracowano wiele modeli bibliotek cyfrowych, np. opracowany w USA otwarty model OAIS (ang. *Open Archival Information System*), referencyjny model DLRM (ang. *Digital Library Reference Model*) europejskiej organizacji DELOS. Ponadto dostępnych jest wiele systemów do budowy BC, np. Fedora, DSpace, Open DLib. W Polsce wiele bibliotek wykorzystało oprogramowanie dLibra (ang. *Digital Library Framework*). Jest to pierwszy polski system do budowy bibliotek cyfrowych, rozwijany przez Poznańskie Centrum Superkomputerowo-Sieciowe od 1999 roku. Pełna lista wdrożeń tego systemu znajduje się na stronie projektu [dLibra 2009]. Aktualnie wykonano 40 wdrożeń dla bibliotek regionalnych (19), instytucjonalnych (18), dydaktycznych (1) i dwa wdrożenia dla bibliotek wewnętrznych czyli niedostępnych przez Internet (dla Komisji Ścigania Zbrodni przeciwko Narodowi Polskiemu IPN oraz dla Biblioteki Cyfrowej Akademii Obrony Narodowej).

W 2006 roku w Instytucie Sztuki Polskiej Akademii Nauk (ISPAN) rozpoczęto digitalizację nagrań polskiej muzyki tradycyjnej (ludowej), w wyniku czego zarchiwizowano ok. 130 tys. nagrań. W 2009 roku w Instytucie rozpoczęto opracowanie systemu informatycznego w ramach projektu CADIS (ang. *Centre for Archivisation and Digitisation of Image and Sound* – Centrum Archiwizacji i Digitalizacji Materiałów Foto- i Fonograficznych).

W ramach europejskiego projektu DISMARC (ang. *DIScovering Music ARChives*) utworzono portal internetowy udostępniający europejskie archiwa dźwiękowe a wśród nich zasoby ISPAN.

Dnia 1 kwietnia 2009 roku został powołany Narodowy Instytut Audiowizualny (NInA), którego zadaniem jest digitalizacja, upowszechnianie oraz koordynacja programu zachowania, rozbudowy i udostępniania cyfrowego dziedzictwa kulturowego. Obecnie trwa zgłaszanie wniosków do projektu „Zasoby cyfrowe", którego celem jest rozbudowa infrastruktury digitalizacyjnej w archiwach, digitalizacja materiałów archiwalnych, zapewnienie bezpiecznego i długookresowego przechowywanie materiałów archiwalnych w postaci cyfrowej, upowszechnianie materiałów archiwalnych w wersji cyfrowej oraz edukacja w zakresie digitalizacji archiwaliów i długookresowego przechowywania zasobów cyfrowych [NInA 2009]. Na projekt przeznaczono 5 mln zł.

W wielu dziedzinach papier coraz częściej wypierany jest przez zapis elektroniczny. Coraz więcej instytucji, w tym również administracja państwowa, zapisuje dane w postaci cyfrowej. Tak samo, jak i wersje papierowe, podlegają one ustawowemu obowiązkowi archiwizacji i muszą być w uporządkowanej formie przekazywane archiwom państwowym, których zadaniem jest ich przechowywanie i udostępnianie obywatelom. Na stronie [ADE 2009] dostępna jest wersja testowa systemu ADE (Archiwum Dokumentacji Elektronicznej) do zarządzania archiwalnymi dokumentami elektronicznymi wytworzonymi przez administrację publiczną.

Przytoczone przykłady pokazują kilka różnych obszarów potrzeby archiwizacji danych, prace jakie wykonano lub są podejmowane w tym kierunku oraz koszty, jakie się z tymi przedsięwzięciami łączą. Do budowy wielu systemów archiwizujących w administracji publicznej jest wykorzystywane oprogramowanie open source i otwarte standardy, co zapewnia przejrzystość i pełną jawność informacji publicznej, otwartość informacji publicznej dla wszystkich użytkowników niezależnie od posiadanego oprogramowania, umożliwia kontrolę nad tworzonymi systemami i dostęp do kodu źródłowego, pozwala na modyfikację istniejących systemów zgodnie z zapotrzebowaniem instytucji a także obniża koszty budowy i eksploatacji systemów.

4. PRZYKŁADY PROBLEMÓW ZWIĄZANYCH Z ARCHIWIZACJĄ DANYCH

Z archiwizacją danych związanych jest wiele zagadnień i problemów. Rośnie liczba systemów informatycznych, wytwarzanych dokumentów, gromadzonych i przesy-

łanych danych. Wolumeny danych gromadzą ich coraz większe ilości, przy czym zazwyczaj niewielka ich część jest potrzebna i wykorzystywana, a jednak wszystkim trzeba zapewnić bezpieczne przechowywanie, szybkie wyszukiwanie i efektywne zarządzanie nimi.

Problemy związane z archiwizacją danych można podzielić na:

- merytoryczne – dotyczące sprecyzowania kryteriów dotyczących określenia materiałów, które należy archiwizować,
- organizacyjne – związane ze sprawną organizacją procesu archiwizowania i korzystania z materiałów archiwalnych,
- ekonomiczne –związane z określeniem i akceptacją rzeczywistego kosztu (finansowego, czasowego) archiwizacji,
- techniczne – związane z określeniem i wyborem infrastruktury związanej z archiwizacją (potrzebnym sprzętem, miejscem do przeprowadzania prac archiwizujących, miejscem przechowywania archiwaliów itd.).

Aby ułatwić archiwizację danych i dokumentów (tekstowych, multimedialnych) powstają systemy, których zadaniem jest składowanie, zarządzanie oraz przeglądnie bardzo dużych ilości danych przez praktycznie nieograniczoną ilość użytkowników oraz integracja z innymi systemami wykorzystywanymi w przedsiębiorstwie.

W celu ograniczenia zasobów archiwalnych nie powinno się, na wszelki wypadek, archiwizować wszystkiego. Ustalenie zasobów podlegających archiwizacji nie jest łatwe. Zależy od decydentów, od jakości materiałów oryginalnych (np. audiowizualnych, fotograficznych), od zasobów finansowych, jakie można przeznaczyć na ten cel, od liczby pracowników przydzielonych do tego typu prac itd. Przykładowo, aby ocenić, które z nagrań audiowizualnych należy przeznaczyć do archiwizacji, należy je najpierw przesłuchać/obejrzeć, a następnie dokonać odpowiedniego wyboru. W celu oceny, które bazy danych należy zarchiwizować, trzeba dysponować odpowiednim oprogramowaniem i sprzętem do ich odczytu. Potrzebni są również specjaliści z poszczególnych dziedzin do właściwej oceny zasobów przeznaczonych do archiwizacji (ich wartości materialnej, historycznej, kulturowej).

Obecnie z archiwizacją wiążę się ściśle digitalizacja, czyli zapisywanie wszelkich zasobów (np. druków, rękopisów, zapisów na skórze czy płótnie) w postaci cyfrowej. Dzięki digitalizacji możliwa jest nie tylko archiwizacja, ale również ochrona oryginału, łatwiejsze przechowywanie, wyszukiwanie, ewidencjonowanie i udostępnianie zdigitalizowanych zasobów.

Często udostępniane archiwalne dane – w celu ich ochrony – można by przechowywać dodatkowo w postaci kopii zapasowych o gorszej jakości w stosunku do oryginału. Pojawia się wówczas pytanie o koszt przeprowadzenia tego zabiegu i o możliwe granice pogorszenia jakości kopii.

Do digitalizacji potrzebne są dobrej jakości skanery, umożliwiające zapisywanie obrazów dokumentów oraz systemy umożliwiające masowe skanowanie, klasyfikację i archiwizację.

Aby ograniczyć ilość miejsca potrzebnego do przechowywania cyfrowych zbiorów archiwalnych potrzebne są dobre metody kompresji i ocena, do których zasobów można zastosować kompresję stratną a do których bezstratną.

Z kolei zapewnienie bezpieczeństwa zgromadzonym zasobom, często wymaga szyfrowania zasobów archiwalnych i kontrolowania dostępu do nich.

Archiwizacja danych elektronicznych wiąże się z ich odpowiednim odświeżaniem, czyli z ich przenoszeniem (migracją) ze starych formatów i nośników danych na nowe. Dużym wyzwaniem jest zapewnienie ponadczasowego przechowywania danych. Obecne nośniki danych ulegają uszkodzeniom i nie gwarantują wieczystego użytkowania umożliwiającego nieskończoną liczbę razy zapis i odczyt. Konieczna jest więc, po pewnym czasie, regeneracja nośników danych i odświeżanie danych.

Z archiwizacją wiąże się wiele innych problemów, chociażby ustalenie norm ogólnotechnicznych związanych z formatowaniem i wymianą danych, stanu prawnego i praw autorskich do zasobów, zasad udostępniania i okresu przechowywania danych itd.

5. SYSTEMY CIĄGŁEJ ARCHIWIZACJI

Standardowe metody archiwizacji i backupu danych (ręczne i półautomatyczne) wymagają dobrej jakości nośników danych i miejsca do ich przechowywania. Wiążą się z wysokimi kosztami i są czasochłonne. W dużym stopniu zależą od systematyczności i rzetelności pracownika. Wykonywane w pewnych odstępach czasu nie zawsze gwarantują możliwości odtworzenia zmian (jeśli w danym czasie nastąpiły wielokrotne zmiany).

Obecnie wiele firm oferuje usługi archiwizacji i backupu danych online (np. ibackup.com, Auto-backup.pl). Dane są szyfrowane (np. algorytmami szyfrowania TLS i AES) i kompresowane po stronie klienta, a następnie przesyłane (z wykorzystaniem protokołu SSL) na serwer archiwizujący. Wszystko odbywa się w tle pracy użytkownika, bez jego ingerencji, zgodnie z ustalonym harmonogramem. Problemem jest jednak ilość przesyłanych danych, gdyż ich przesyłanie w tle znacznie obciąża system spowalniając jego pracę, i jest w zasadzie efektywne tylko dla plików o małych rozmiarach.

Według firmy EMC w okresie styczeń –październik 2008 roku na świecie przybyło 394 eksabajtów[1] danych (prawie dwa razy więcej niż w roku 2006). Szacunkowy przyrost informacji generowanych przez firmy wynosi 60% rocznie. Dlatego potrzebne są nowe rozwiązania np. w zakresie pamięci masowych. Firma Intel zaproponowała ustalenie maksymalnego czasu dostępu do poszczególnych typów danych oraz jak długo dane powinny być przechowywane i kiedy można, czy też należy, je kasować.

[1] Eksabajt (EB) – 2^{60} bajtów (w przybliżeniu trylion czyli 10^{18} bajtów).

Wiele nadziei na zmniejszenie kosztów przechowywania danych wiąże się z optymalizacją pamięci masowych i z wykorzystaniem dynamicznego przydzielania miejsca (ang. *thin provisioning*).

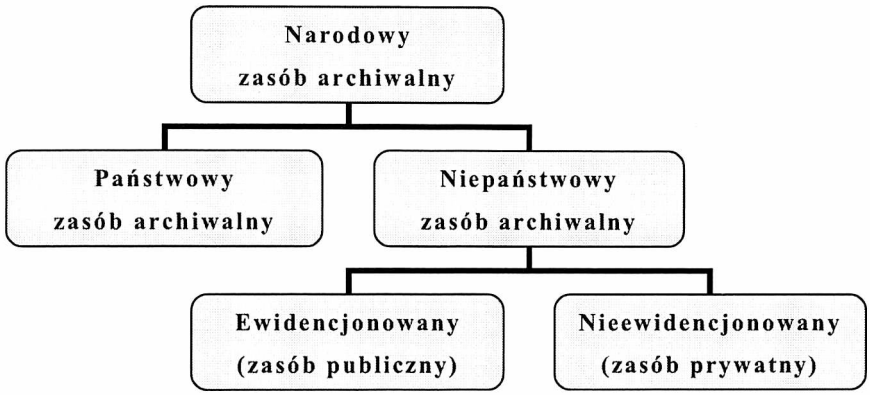
Kolejnym problemem jest wielokrotne, zbędne, zapisywanie tych samych danych. Dostawcy systemów pamięci masowych oferują narzędzia do deduplikacji danych. System przed zapisaniem danych sprawdza, czy dane były już wcześniej zapisywane.

6. REGULACJE NORMATYWNE I PRAWNE W POLSCE DOTYCZĄCE OBIEGU I ARCHIWIZACJI DOKUMENTÓW ELEKTRONICZNYCH

W Polsce obowiązuje wiele aktów prawnych regulujących zasady przechowywania, obiegu i archiwizacji dokumentów elektronicznych, z których najważniejsze to:

- Ustawa o narodowym zasobie archiwalnym i archiwach z dnia 14 lipca 1983 r. (DzU z 1983 Nr 38, poz. 173) wraz z późniejszymi zmianami (DzU z 2002 Nr 171, poz. 1396 – tekst ujednolicony),
- Ustawa z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (tekst pierwotny: DzU 1994 r. Nr 24, poz. 83).
- Ustawa o ochronie informacji niejawnych z dnia 22 stycznia 1999 r. (DzU Nr 11, poz. 95),
- Rozporządzenie Ministra Kultury i Dziedzictwa Narodowego z dnia 5 listopada 1999 r. w sprawie zasad ewidencji materiałów bibliotecznych (DzU z dnia 20 listopada 1999 r.),
- Ustawa o ochronie baz danych z 27 lipca 2001 r. (DzU Nr 128, poz. 1402),
- Ustawa o informatyzacji działalności podmiotów realizujących zadania publiczne z dnia 17 lutego 2005 r. (DzU Nr 64, poz. 565),
- Rozporządzenie Ministra Kultury i Dziedzictwa Narodowego z dnia 22 maja 2006 r. w sprawie szczegółowych zasad i trybu gromadzenia, ewidencjonowania, kwalifikowania, klasyfikacji oraz udostępniania materiałów archiwalnych tworzących zasób jednostek publicznej radiofonii i telewizji (DzU z dnia 9 czerwca 2006 r. Nr 98, poz. 680),
- Rozporządzenie Ministra Nauki, Szkolnictwa Wyższego i Techniki z 25 lipca 1984 w sprawie wcześniejszego udostępniania materiałów archiwalnych (DzU Nr 41, poz. 217).
- Rozporządzenie Ministra Kultury i Dziedzictwa Narodowego z 29 lipca 2008 r. w sprawie określenia szczególnych wypadków i trybu wcześniejszego udostępniania materiałów archiwalnych (DzU Nr 156, poz. 970).

Z *Ustawy o narodowym zasobie archiwalnym i archiwach* wynika, że na narodowy zasób archiwalny składają się zasoby archiwalne państwowe i niepaństwowe (rys. 1). Natomiast do zasobów niepaństwowych zalicza się ewidencjonowane zasoby publiczne (np. kościelne) oraz nieewidencjonowane zasoby prywatne. Nadzór nad narodowym zasobem archiwalnym sprawuje minister do spraw kultury i ochrony dziedzictwa narodowego za pośrednictwem naczelnego dyrektora Archiwów Państwowych.



Rys. 1. Struktura narodowego zasobu archiwalnego

W myśl ustawy materiałami archiwalnymi są:

wszelkiego rodzaju akta i dokumenty, korespondencja, dokumentacja finansowa, techniczna i statystyczna, mapy i plany, fotografie, filmy i mikrofilmy, nagrania dźwiękowe i wideofonowe, dokumenty elektroniczne w rozumieniu przepisów ustawy z dnia 17 lutego 2005 r. o informatyzacji działalności podmiotów realizujących zadania publiczne (DzU Nr 64, poz. 565 oraz z 2006 r. Nr 12, poz. 65 i Nr 73, poz. 501) *oraz inna dokumentacja, bez względu na sposób jej wytworzenia, mająca znaczenie jako źródło informacji o wartości historycznej o działalności Państwa Polskiego, jego poszczególnych organów i innych państwowych jednostek organizacyjnych oraz o jego stosunkach z innymi państwami, o rozwoju życia społecznego i gospodarczego, o działalności organizacji o charakterze politycznym, społecznym i gospodarczym, zawodowym i wyznaniowym, o organizacji i rozwoju nauki, kultury i sztuki, a także o działalności jednostek samorządu terytorialnego i innych samorządowych jednostek organizacyjnych – powstała w przeszłości i powstająca współcześnie.*

Dokumentacja stanowiąca materiały archiwalne lub dokumentację niearchiwalną może być zapisywana na różnego rodzaju nośnikach. Podstawowym nośnikiem był i jest papier. Dokumentację zarejestrowaną za pomocą pisma (niezależnie od techniki wykonania) nazywa się aktową [Kudło 2006]. Dokumentacja nieaktowa to na przykład dokumentacja techniczna, geodezyjna, kartograficzna, geologiczna, audiowizualna a więc plany, mapy, fotografie, filmy, nagrania – zapisane na różnych nośnikach obrazu i dźwięku.

7. OKRES PRZECHOWYWANIA DANYCH ARCHIWALNYCH

Oprócz wymienionych w rozdziale 6 ustaw obowiązuje jeszcze wiele innych aktów regulujących okres i warunki przechowywania zasobów archiwalnych. Na przykład okres przechowywania dokumentacji szkolnej (placówki oświatowej) regulują:

- Rozporządzenie Ministra Kultury z dnia 16 września 2002 r. w sprawie postępowania z dokumentacją, zasad jej klasyfikowania i kwalifikowania oraz zasad i trybu przekazywania materiałów archiwalnych do archiwów państwowych (DzU z 2002 r. Nr 167, poz. 1375),
- Statut szkoły,
- Regulamin organizacyjny szkoły.

Niezwykle ważne są zasady przechowywania danych medycznych pacjentów, co reguluje

- Rozporządzenie Ministra Zdrowia z dnia 21 grudnia 2006 r. w sprawie rodzajów i zakresu dokumentacji medycznej w zakładach opieki zdrowotnej oraz sposobu jej przetwarzania (DzU z dnia 28 grudnia 2006 r.).

Dokumentacja medyczna, która nie jest przekazywana do narodowego zasobu archiwalnego, musi być zniszczona w sposób uniemożliwiający identyfikację pacjenta, którego dotyczyła.

W celu ułatwienia zarządzaniem dokumentacją wprowadza się podział na kategorie, z których wynika okres przechowywania dokumentacji:

- kategoria A – obejmuje materiały archiwalne, mające znaczenie jako źródło informacji o wartości historycznej dotyczącej działalności państwa polskiego, jego stosunkach z innymi państwami, o rozwoju życia społecznego i gospodarczego; po okresie przechowywania w archiwum zakładowym są przekazywane do Archiwum Państwowego (na wieczyste przechowywanie),
- kategoria B*n* – oznacza *n*-letni okres przechowywania i obejmuje dokumentację o czasowym znaczeniu praktycznym (po upływie okresu przechowywania materiały podlegają brakowaniu czyli niszczeniu),
- kategoria B*c* – oznacza dokumentację posiadającą krótkotrwałe znaczenie praktyczne, która po pełnym jej wykorzystaniu przekazywana jest na makulaturę (okres przechowywania tych materiałów nie przekracza 1 roku po uzgodnieniu z Archiwum Państwowym),
- kategoria BE – obejmuje materiały, które po upływie obowiązującego okresu przechowywania podlegają ekspertyzie archiwalnej, przeprowadzanej przez właściwe Archiwum Państwowe, w wyniku której można dokonać zmiany kategorii tej dokumentacji.

Okres przechowywania akt liczy się w pełnych latach kalendarzowych, poczynając od 1 stycznia roku następnego, w którym to roku akta powstały.

Okres przechowywania dokumentacji finansowej określa Ordynacja podatkowa oraz zapisy ustawy o rachunkowości. Zgodnie z brzmieniem art. 74 ustawy o rachunkowości, księgi rachunkowe przechowuje się co najmniej przez okres 5 lat a zatwierdzone roczne sprawozdania finansowe podlegają trwałemu przechowywaniu.

Po upływie obowiązującego okresu przechowywania w instytucji określone materiały są przekazywane do właściwego archiwum państwowego na wieczyste przechowywanie. Pozostałe akta stanowią dokumentację niearchiwalną i po upływie obowiązującego okresu przechowywania można je zniszczyć.

Jeśli w zakładzie gromadzone są materiały archiwalne kategorii A i dokumentacja niearchiwalna kat. B, to zakład zobowiązany jest do prowadzenia archiwum zakładowego. Natomiast jeśli zakład wytwarza tylko materiały niearchiwalne, to jest zobowiązany prowadzić składnicę akt.

Materiały archiwalne stanowiące narodowy zasób archiwalny przechowuje się wieczyście [Ustawa 1983].

8. PODSUMOWANIE

W Polsce, tak jak i na świecie, wiele uwagi przywiązuje się do tworzenia archiwów cyfrowych.

Elektroniczna archiwizacja danych jest niezwykle kosztownym i organizacyjnie obszernym procesem. Wymaga zmierzenia się z wieloma decyzjami dotyczącymi organizacji, opracowania, nadzoru i udostępniania archiwaliów. Muszą być określone cele archiwizacji i ich priorytety. Potrzebne są precyzyjne kryteria służące określeniu zakresu danych przeznaczonych do archiwizacji, ich wartościowania, standardów budowania archiwów.

Archiwizacja danych cyfrowych wymaga odpowiednich uregulowań prawnych, narzędzi sprzętowych i programistycznych, zbudowania odpowiedniej infrastruktury.

LITERATURA

[ADE 2009] Archiwum Dokumentacji Elektronicznej, ade.ap.gov.pl

[AP 2009] Archiwa Państwowe, www.archiwa.gov.pl

[BiblNar 2009] Biblioteka Narodowa, www.bn.org.pl

[dLibra 2009] Digital Library Framework, dlibra.psnc.pl

[Kudło 2006] Kudło H., *Dokumentacja aktowa*, Archiwum Państwowe we Wrocławiu, Legnica, 2006.

[NAC 2009] Narodowe Archiwum Cyfrowe, www.nac.gov.pl

[NInA 2009] Narodowy Instytut Audiowizualny, www.nina.gov.pl

[OGICOM 2009] OGICOM, www.ogicom.pl

[Ustawa 1983] Ustawa o narodowym zasobie archiwalnym i archiwach z dnia 14 lipca 1983 r. (DzU 1983 Nr 38, poz. 173).

ELECTRONIC DATA ARCHIVES

Increase of data of different forms is getting more and more greater. Gathered and archived collections of text documents, images and multimedia data are constantly growing in size. This paper presents various issues related to digital archives.

Prace Naukowe Instytutu Informatyki
Politechniki Wrocławskiej

Zeszyt 10 2009

integracja danych,
jakość danych,
zaufanie

Przemysław PAWLUK*

Ipsa scientia potestas est[1]

Sir Francis Bacon

JAKOŚĆ DANYCH W KONTEKŚCIE DANYCH ZINTEGROWANYCH

Integracja danych to zadanie jakie stawiane jest przed aplikacjami, dla których konieczne jest pozyskiwanie informacji z wielu autonomicznych i heterogenicznych źródeł. Potrzeba integracji szczególnie widoczna jest w dynamicznie rozwijających się instytucjach, jakimi są banki, ale także np. w dużych projektach związanych z informatyzacją instytucji rządowych i wielu innych miejscach.

Jakość danych jest w literaturze definiowana w różny sposób. Potrzeba pomiaru jakości danych pojawia się wraz z napływem coraz większej ich ilości. W niniejszej pracy przedstawimy podstawowe zagadnienia związane z jakością danych, w szczególności zaś skoncentrujemy się na jakości danych zintegrowanych.

1. WSTĘP

Wiedza na temat jakości, historii i innych aspektów przetwarzanych danych daje nam zdolność do lepszej oceny przydatności danych wykorzystywanych do podejmowania decyzji. Celem niniejszej pracy jest przybliżenie czytelnikowi zagadnień związanych z jakością i integracją danych oraz wskazanie „niejakościowych" czynników wpływających na zaufanie użytkownika do danych i ich przydatność w procesie podejmowania decyzji. Eksploracja tych zagadnień ma w przyszłości doprowadzić do wytworzenia narzędzia wspomagającego użytkowników w podejmowaniu decyzji przez ocenę dostarczanych danych.

* Department of Computer Science and Engineering, York University, 4700 Keele Street, Toronto, Ontario M3J 1P3, Canada, pawluk@cse.yorku.ca

[1] *Wiedza jest siłą* – Sir Francis Bacon, Meditationes Sacra, De Haresibus (1597).

Postęp poczyniony w ciągu ostatnich lat w dziedzinie baz danych, pozwolił na wytworzenie systemów oferujących dostęp do wielkich zbiorów danych, często rozproszonych pomiędzy heterogenicznymi źródłami danych i integrowanych za pomocą najróżniejszych narzędzi. Systemy te zapewniają zunifikowany dostęp do danych, nie gwarantując jednak utrzymania ich wysokiej jakości. Nie dostarczają również narzędzi do pomiaru jakości dla tak dostarczanych danych.

Problem jakości danych, w kontekście pojedynczego źródła danych, rozważany był przez wielu autorów ([BaPa 1985], [BaWaPaTa 1998], [BaSc 2006], [Crosby 1979], [DrHeMaObS 2007], [English 1996], [FoHe 2007], [Olson 2002], [PaSaJa 2002], [PaSaJa 2004], [Redman 2001], [TaBa 1998], [Tupek 2006], [WaSt 1996], [PiLeWa 2002]). Część autorów poruszała także ten problem w kontekście integracji danych ([BoKe 2002], [CuWiWi 2000], [GeSch 2007], [GuWi 1993], [Naumann 2002], [ReWa 1995]) podkreślając istotność tego problemu w tym kontekście. Jednakże mimo tak wielu publikacji i badań poczynionych w dziedzinie jakości, do dziś nie została ustalona jej wspólna definicja, a dodatkowo wymiary jakości (ang. *dimensions*)[2] posiadają niejednoznaczne, a czasami mylące definicje.

Problem integracji danych od kilku dekad pozostaje interesującym zagadnieniem badawczym. Badacze eksplorowali różne zagadnienia i aspekty procesu integracji poczynając od podstaw przedstawionych przez Brazhnika, Hassa, Halevy'a czy Ullmana ([BraJo 2007], [Hass 2007], [HaRaOr 2006], [Ullman 1997]), do bardziej złożonych i szczegółowych zagadnień jak *data fusion* (problem fuzji dwóch lub więcej rekordów odnoszących się do tego samego obiektu świata rzeczywistego) [BlNa 2008], *schema mapping* lub *matching* (problem mapowania schematów bazodanowych przy integracji lub wymianie danych) [Fagin 2006], [FaKoTaP 2004], [Kolaitis 2005], [Ra Be 2001]. Interesującym zagadnieniem w tym kontekście jest także problem wykrywania i usuwania duplikatów [ElIpVe 2007] oraz tzw. oczyszczania danych (ang. *data cleaning problem*) [DaJo 2003].

Innym ciekawym zagadnieniem zwiazanym zarówno z integracją jak i z jakością danych jest problem spójnej odpowiedzi (ang. *consistent query answering*) – prace [ArBeCho 1999], [ArBeCHR 2003], [Bertossi 2006], [BraBer 2004], oraz utrzymywanie zależności i więzów w bazach dla zapewnienia spójności ([CaCaGiLe 2002], [CaCaGiLe 2004], [Fan 2008], [FaGeJiK 2008]). Wspomniane więzy integralności stanowią istotny zasób wiedzy na temat semantyki danych i są niezbędne do oceny jakości danych, a w szczególności wymiarów związanych z semantyką.

Wszystkie wspomniane powyżej czynniki mają znaczny wpływ na zaufanie użytkownika do danych i ich przydatność. W literaturze zaufanie (ang. *trust*) definiowane jest jako relacja pomiędzy dwoma aktorami (ludźmi, organizacjami). Niektórzy badacze wykorzystują to pojęcie w odniesieniu do baz danych, jednakże raczej w kontekście bezpieczeństwa niż zaufania użytkownika do danych.

---

[2] Zwane też czasami faktorami (ang. *factors*) lub kryteriami (ang. *criteria*) w zależności od źródła.

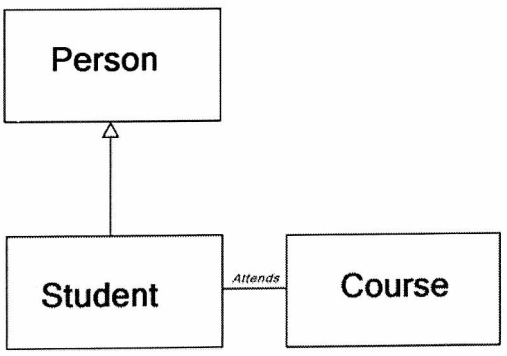
Nasze podejście w tym temacie wygląda nieco inaczej. Naszym celem jest znalezienie odpowiedzi na pytanie: Czy użytkownik może zaufać dostarczonym danym? Chcemy to uczynić badając ich jakość, a także pewne cechy nie należące do wymiarów jakości danych.

2. JAKOŚĆ DANYCH

Problem jakości danych (ang. *data quality*) ma swój początek w problemie zalewania użytkownika danymi. Użytkownik taki paradoksalnie potrzebuje dodatkowych informacji, aby był zdolny do podjęcia decyzji. Te dodatkowe informacje mają służyć selekcji i dotyczyć jakości i innych aspektów. Z tego powodu są to raczej meta-dane gdyż stanowią opis danych.

2.1. DANE

Dane – od łacińskiego słowa *datum* – znaczy tyle, co „coś, co jest dane". W kontekście klasycznej informatyki słowo to zaczęło oznaczać nie więcej niż numeryczną lub inną zrozumiałą dla komputera reprezentację informacji. Obecnie jednakże oznacza ono raczej reprezentację faktu, rozumianego jako „coś co się wydarzyło lub istnieje" [SiWe 2009]. Dane mogą mieć także swoją reprezentację, jako atrybuty lub encje oraz relacje między nimi. Encja jest rozumiana jako "coś co istnieje niezależnie i jest unikalne w realnej lub w konceptualnej rzeczywistości" (według słownika Webster's Dictionary). Atrybut możemy tu rozumieć jako reprezentację faktu dotyczącego encji np. `First_name="John"` będące atrybutem encji Person.



Rys. 1. Przykład encji i związku

Na rysunku 1 przedstawiono prosty model z trzema encjami oraz powiązaniami pomiędzy nimi. Związek `Attend` może być rozumiany jako reprezentacja faktu, że

student uczęszcza na kurs. W tej pracy rozpatrujemy jednak dane jako reprezentację faktów lub obiektów świata rzeczywistego składowaną w relacyjnej bazie danych lub rozproszonych kilku bazach danych.

2.2. ISTOTNOŚĆ JAKOŚCI DANYCH

W literaturze odnaleźć możemy wiele różnych definicji jakości danych. Badacze podają wiele znacząco różnych definicji jakości. Jedne z nich koncentrują się na tzw. czynnikach wewnętrznych lub niezależnych (ang. *instrinsic*) takich jak dokładność (ang. *accuracy*) [Olson 2002] lub kompletność (ang. *completeness*) [LePiStWa 2004], inne stanowią próby obliczania jakości bazujące na pewnym kontekście użycia [TaBa 1998]. Ten brak powszechnie uznanej definicji prowadzi do prób definiowania jakości przez wymienienie czynników jakie się na nią składają, co jednak wprowadza kolejne nieścisłości ponieważ definicje czynników nie są także spójne i powszechnie uznane. Problem ten analizują Wang i Strong [WaSt 1996] oraz Foley i Helfert [FoHe 2007].

Istotność zagadnień związanych z jakością danych była omawiana w wielu pracach. W [BaSc 2006] znajdujemy trzy przykłady procesów zachodzących w organizacjach, dla których jakość danych jest niezmiernie istotna:

- synchronizacja danych – problem znany w organizacjach posiadających więcej niż jeden system z „nakładającymi" się bazami danych,
- identyfikacja osób prowadzących wspólne gospodarstwo domowe lub związanych przez uczestnictwo w określonej grupie [WaChe 2005] ,
- fuzja organizacji – wydarzenie wymagające połączenia zbiorów danych obu organizacji a często także współdziałania z zastanymi systemami.

Te trzy przykłady świetnie dowodzą, jak istotne są zagadnienia związane z jakością oraz zrozumienie problemów z nią związanych. Pokazują one także, że każdy proces wymaga nieznacznie innego podejścia do jakości oraz nieznacznie innej interpretacji wyników.

2.3. DEFINICJA JAKOŚCI DANYCH

Kiedy większość ludzi myśli o jakości, myślą jedynie o poprawności czy też dokładności danych (ang. *accuracy*) rozumianej jako poprawność wartości czyli np. dane zawierające tzw. literówki są uważane za dane niższej jakości. Jednakże jakość jest pojęciem bardziej złożonym i nie ogranicza się jedynie do poprawności danych. W tej części pracy pokażemy jak wiele różnych czynników składa się na jakość danych.

Definicja jakości najczęściej spotykana w literaturze przedmiotu to definicja podana przez Redmana mówiąca, że „dane są wysokiej jakości, jeśli są zdatne do ich planowego wykorzystania przez klienta w operacjach, podejmowaniu decyzji oraz planowaniu" [Redman 2001]. Podstawowym problemem jest tu wyznaczenie klienta jako

jedynego arbitra decydującego o jakości danych. Szeroki wachlarz potrzeb, wymagań i oczekiwań klientów wobec danych komplikuje pomiar i możliwości zarządzania jakością. Oczywistym natomiast jest, że wysoka jakość znaczy co innego dla użytkownika zamawiającego ogólny raport, który ma być łatwy do przeczytania i zrozumienia, a co innego dla użytkownika wymagającego bardzo szczegółowych i jak „najświeższych" danych.

Definicji zbliżonych do zaproponowanej przez Redmana można znaleźć jeszcze kilka w literaturze. English określa jakość jako „ciągłe zaspakajanie oczekiwań klientów" [English 1996]. Crosby definiuje ją natomiast jako „zgodność z wymaganiami" [Crosby 1979] nie wskazując jednoznacznie na pisemną specyfikację. Norma ISO 8402 definiuje jakość jako „charakterystykę określającą zdolność do zaspokajania wyrażonych oraz ukrytych potrzeb" [ISO 1994].

Ciekawą i ważną obserwację czyni English mówiąc, że jakość służyć ma zaspokajaniu potrzeb klienta ale niekoniecznie ich przewyższaniu [English 1996]. To proste przesłanie świetnie ilustruje, co prawda nie informatyczny, ale wiele mówiący przykład firmy Rolls Royce. Firma ta, stanęła w obliczu bankructwa we wczesnych latach osiemdziesiątych ubiegłego stulecia, dzięki inwestycjom mającym na celu ciągłe ulepszanie elementów, które klienci uważali za nieistotne, co spowodowało podnoszenie ceny ich luksusowych samochodów znacznie powyżej tej, którą klienci mogli zaakceptować jako odpowiadającą wartości auta. Wniosek jaki można wysnuć z tego przykładu świetnie pasuje także do informatycznego kontekstu. Zarządzając jakością zwracać uwagę powinno się przede wszystkim na elementy istotne dla użytkownika i przez niego zauważane. W innym razie tracimy czas i pieniądze na usprawnienia czegoś, co z punktu widzenia użytkownika nie istnieje i za co nie będzie skłonny zapłacić.

Interesująca definicja jakości podana została przez Nuamanna [Naumann 2002], który określił jakość jako „zagregowaną wartość wielu kryteriów jakościowych (IQ-criteria)". Definicja ta nie przywiązuje nas bezpośrednio do oczekiwań użytkownika, jednakże wskazuje na kryteria, a te mogą być wyliczane w sposób automatyczny lub oceniane przez użytkownika czy też eksperta.

Jakość nie jest prostą metryką. Składa się na nią wiele czynników i może ona być oceniana w różnym kontekście. Przedstawimy teraz przykład obrazujący niewielki wycinek złożoności tego konceptu.

Tabela 1. Relacja FamousBuildings. Przykład różnych problemów z jakością danych

Id	Name	City	Country	Height	
				m	ft
1	CN-Tower	Toronto	Canada	553,3	1 315,0
2	Sers Tower	Chicago IL	USA	527,0	1 730,0
3	Petronas Towers	New York NY	USA	451,9	1 482,6
4	Empire State Building	Kuala Lumpur	Malaysia	381,0	1 250,0

Analizując zawartość tabeli 1 możemy w niej odnaleźć szereg problemów. Najłatwiejszy do odnalezienia jest zapewne problem z „literówką" w wierszu nr 2, gdzie *Sears Tower* zamienione zostało na *Sers Tower*. Jednakże relacja ta zawiera znacznie więcej błędów. Zamienione lokalizacje w wierszach 3 i 4, błąd często spotykany przy ręcznym wprowadzaniu danych, to kolejny przykład braku dokładności. W wierszu pierwszym mamy do czynienia natomiast z problemem niespójności danych. Wysokość podana w stopach (ft) i w metrach (m) nie są sobie równe (553,3 m = 1815 ft). Bazując na tym prostym przykładzie zauważyć możemy tylko niewielki wycinek problematyki związanej z jakością danych. Dodatkowo, jeśli rozważymy bardziej skomplikowane modele danych, wzrosnąć może liczba różnego rodzaju błędów i problemów z jakością.

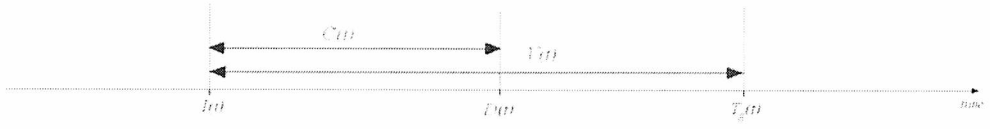
2.3.1. CZYNNIKI ZALEŻNE OD CZASU

Problem świeżości danych (ang. *data freshness*) wydaje się intuicyjny, jednakże w literaturze występuje wiele definicji czynników, co może prowadzić do nieporozumień. Dla przykładu, Naumann definiuje aktualność danych (ang. *timeliness*) jako średni wiek danych w źródle [Naumann 2002]. Inne źródła definiują aktualność jako czas upływający od utworzenia danych do momentu dostarczenia do klienta [Tupek 2006].

Segev i Fang w [SeFa 1990] definiują „bieżącość" (ang. *currency*) jako okres od ekstrakcji do dostarczenia danych do użytkownika, jednakże w tej pracy zakładamy, że dane dostarczane są bez opóźnień i wykorzystujemy inne znaczenie *currency* jako wieku danych.

Data Currency – wiek danych

Currency $C(t)$ jest to wiek krotki t czyli czas jaki upłynął od jej utworzenia w bazie danych. Jest on podstawową informacją na temat tego jak „świeże" są dane. Jednakże interpretacja tej wartości może przysporzyć kilku trudności. Bez dokładnej wiedzy na temat domeny nie możemy bowiem w prosty sposób określić, czy dane przechowywane w naszej bazie od 2 czy 3 lat są „świeże", czy może te otrzymane dwa dni temu są już zbyt stare, aby być wykorzystane. Semantyka danych jest tu kluczowa, a przykład jaki zostanie przedstawiony później zilustruje ten problem.



Rys. 2. Wiek danych oraz ulotność zaznaczone na osi czasu

Ulotność danych

Ulotność (ang. *volatility*) to okres czasu przez jaki dane mogą być uznane za „świeże" lub „zdatne". Daje ona nam pewną część wiedzy koniecznej do interpretacji wieku danych. Jest to czynnik silnie zależny od kontekstu użycia oraz domeny. Niektóre dane mogą mieć bardzo długi okres ulotności (np. data urodzenia, nazwisko, imię itp.), inne są wysoce ulotne i szybko się dezaktualizują (stan konta, wyniki częstych pomiarów itp.). Ulotność może być zdefiniowana na dwa sposoby: przez podanie punku w czasie (T_E) lub okresu (Δ_E), po którym dane ulegają przedawnieniu. Oba sposoby są sobie równoważne i mogą być wzajemnie transformowane w łatwy sposób. Mając punkt T_E możemy wyrazić ulotność jako

$$V(t) = T_E - I(t) \tag{1}$$

Aktualność danych

Aktualność danych (ang. *timeliness*) ($T(t)$) określa, jak aktualne są dane dla aktualnie wykonywanego zadania. Łączy ona dwa poprzednio zdefiniowane czynniki (wiek i ulotność), w celu uzyskania wartości umożliwiającej jej prostą interpretację. Zależności pomiędzy wiekiem a ulotnością przedstawione na rysunku 2 i mogą być wskazówką do interpretacji aktualności. Batini i Scannapieco [BaSc 2006] definiują aktualność jako:

$$T(t) = \max(0,1 - C(t)/V(t)) \tag{2}$$

Ballou i inni [BaWaPaTa 1998] modyfikują powyższy wzór dodając czynnik czułości (ang. *sensibility factor*)

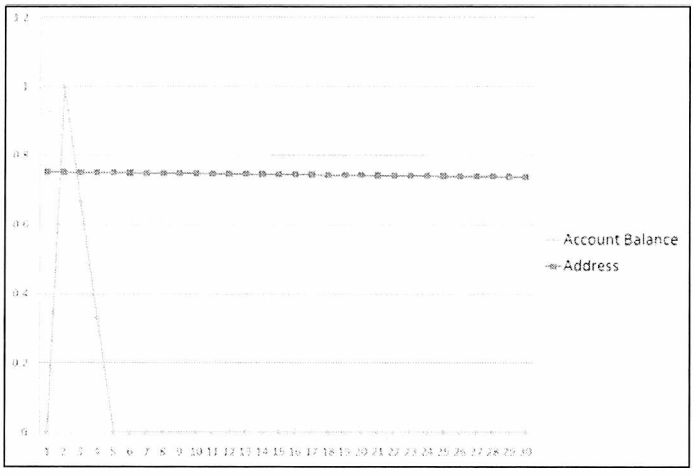
$$T(t) = \max(0,1 - C(t)/V(\mathrm{t}))^s \tag{3}$$

Przykład

Rozważmy dwa zapisy w bazie danych:

1. Address (adres)
   a. Ulotność $V(t) = 2000$ dni
   b. Wiek $C(t) = 500$ dni
   c. Aktualność $T(t) = \max(0,1 - 500/2000) = 0,75$
2. Account Balance (Stan konta)
   a. Ulotność $V(t) = 3$ dni
   b. Wiek $C(t) = 2$ dni
   c. Aktualność $T(t) = \max(0,1 - 2/3) = 0,33(3)$

Przy założeniu, że im wyższa wartość aktualności tym „świeższe" są dane, łatwo jest zauważyć, że stan konta, mimo, że teoretycznie młodszy, jest uznany za mniej godny zaufania, czyli mniej „świeży" niż adres, mimo, że ten jest 250 razy starszy. Zależności obu wartości przedstawione zostały na wykresie na rysunku 3.



Rys. 3. Zależność pomiędzy wartościami aktualności dla stanu konta (Account Balance) oraz adresu (Address)

Iloraz wieku i ulotności daje nam informację opisującą jak duża część ulotności została już wykorzystana.

2.3.2. CZYNNIKI ZWIĄZANE Z DANYMI

Najpopularniejsze z czynników jakościowych związane są bezpośrednio z danymi oraz ich poprawnością, kompletnością itd.

Dokładność danych

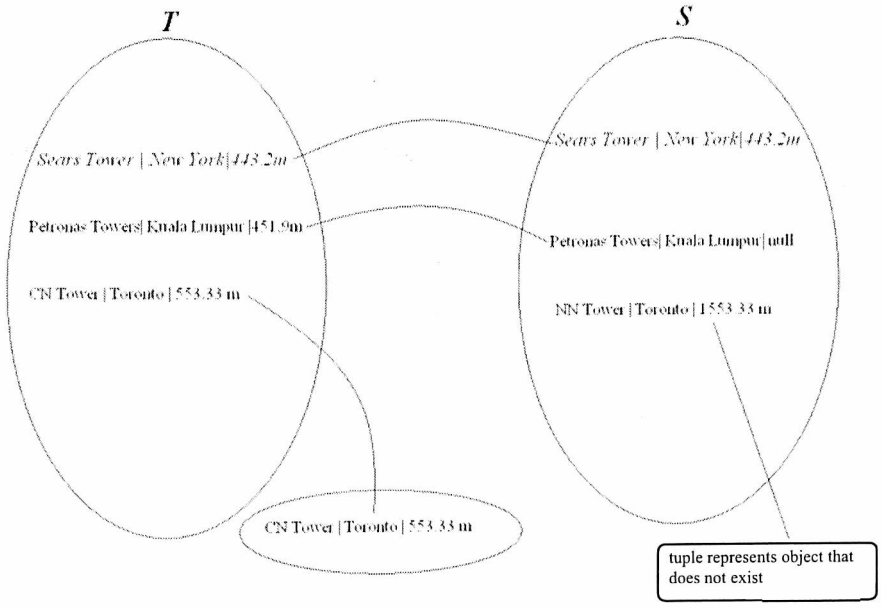
Dokładność danych jest to czynnik rozważany przez większość autorów prac dotyczących jakości danych [BaPa 1985], [BaSc 2006], [GeSch 2007], [KriMo 1982], [PaSaJa 2002], [PaSaJa 2004], [Shin 2003], [WaWa 1996]. Mimo, że ma on intuicyjny wydźwięk, to jednak nie ma w literaturze powszechnie uznanej jego definicji [WaWa 1996]. Ballou i Pazer [BaPa 1985] opisują dokładność jako zgodność wartości zarejestrowanej z rzeczywistą. Kriebel i Moore [KriMo 1982] charakteryzują dokładność jako poprawność informacji, co jednak wprowadza wrażenie jednoznaczności czynników dokładności i poprawności.

W pracy [BaSc 2006] dokładność jest zdefiniowana jako „odległość pomiędzy wartością v, a pewną wartością v' rozumiana jako poprawna reprezentacja rzeczywistego fragmentu jaki reprezentuje także v." Prostym przykładem jest nazwa miasta. Załóżmy, że v = *Tronto*, natomiast v' = *Toronto*. Wartość v w takim przypadku jest niepoprawna (niedokładna). Kolejnym krokiem może być tutaj próba obliczenia odległości pomiędzy v, a v'. Batini i Scannapieca podają dwa rodzaje dokładności:

- Syntaktyczna – określa czy wartość należy do pewnego zbioru wartości czyli do dziedziny. Tak rozważana dokładność może prowadzić do paradoksalnych sytuacji, w których wartość mimo, że nie jest poprawna w potocznym tego słowa znaczeniu, jest jednak uznana za dokładną. Na przykład, jeśli rozważamy wcześniej przytoczony przykład z budynkami i położenie wieży *CN-Tower*. Jeśli domeną są nazwy miast kanadyjskich, to wprowadzenie *Hamilton* jako wartości lokalizacji *CN-Tower* będzie uznane za dokładne, gdyż takie miasto istnieje w Kanadzie mimo, że poprawną wartością jest *Toronto*. Do pomiaru odległości w tym przypadku musi być wykorzystana funkcja działająca na wartości i zbiorze reprezentującym dziedzinę, obliczająca odległość elementu od zbioru.
- Semantyczna – określa odległość wartości od rzeczywistej (prawidłowej) wartości [BaSc 2006]. W tym wypadku wartość *Hamilton* dla lokalizacji *CN-Tower* będzie uważana za niepoprawną (niedokładną) ponieważ prawidłowa wartość to *Toronto*.

Czasami do tego zestawu dodawany jest też czynnik precyzji (ang. *precision factor*) uwzględniający precyzję z jaką mają zostać zachowane dane [Redman 1997] oraz stopień szczegółowości [BoMaYa 1998], [BoKe 2002].

Prace nad sformalizowaniem pojęcia dokładności prowadzili tez Parssian i inni [PaSaJa 2002], [PaSaJa 2004], podając dla dokładności odpowiedzi oraz złączeń, oraz Wang [ReWa 1995], który analizował proces propagacji błędów. Jednakże ich prace nie wyczerpują wszystkich możliwych przypadków.



Rys. 4. Przykład mapowania pomiędzy rzeczywistymi wartościami a wpisami w bazie danych

Na rysunku 4 przedstawiono zestaw możliwych sytuacji, jakie możemy spotkać w bazach danych. Zbiór *T* reprezentuje rzeczywiste wartości, zbiór *S* wartości zapisane w bazie danych. Przeanalizujmy zatem zawartość obu zbiorów. *Sears Tower* jest poprawnie reprezentowane w zbiorze *S*, natomiast *Petronas Towers* reprezentowane są w sposób niekompletny (brak wysokości). *CN-Tower* nie ma swojej reprezentacji w zbiorze *S*, co zaznaczone jest przez referencję do pewnego nienazwanego zbioru. Co ciekawe w zbiorze *S* występuje też element nie mający rzeczywistego odpowiednika (*NN Tower*).

Kompletność danych

Pojęcie zdefiniowane przez Naumanna [Naumann 2002] korespondujące z angielskim *nullability*, a rozumiane jako iloraz liczby pól o wartościach NULL do liczebności relacji (liczby wierszy). Innymi słowy, im mniej pól o wartości Null (niewypełnionych, nieokreślonych, nieznanych), tym lepsza jakość danych, wyższa ich kompletność.

Wang [WaSt 1996] definiuje dane kompletne, jako takie, których szczegółowość i zakres są wystarczające dla wykonywanego zadania. W pracy [PiLeWa 2002] Wang, Pipino i Yang precyzują pojęcie kompletności podając jej trzy typy (schematu, kolumny, populacji). Gertz określa kompletność jako stopień, w jakim dane istotne dla aplikacji zostały zapisane w systemie [GeSch 2007]. Bobrowski i inni [BoMaYa 1998] twierdzą natomiast, że kompletność oznacza, iż każdy fakt z realnego świata odzwierciedlony został w systemie informacyjnym. Kompletność można także rozpatrywać na poziomie encji (czy wszystkie konieczne encje zostały zarejestrowane) oraz na poziomie atrybutu (czy wszystkie konieczne atrybuty zostały uwzględnione) [Redman 1997].

Ocena kompletności dla całego źródła może być niezmiernie kosztowna. Inspekcja każdego wiersza może zabierać olbrzymią ilość czasu i często wymaga eksperta do wykonania tego zadania. Jednak koszty takiej inspekcji można ograniczyć wykorzystując metody statystyczne np. *sampling* opisany przez Olkena [Olken 1993] oraz Gryza [GrGLZ 2004].

Inne czynniki związane z danymi

W literaturze anglojęzycznej występuje szereg innych czynników jakościowych, jednakże często są one tożsame z już opisanymi. *Nullability* utożsamiana jest z kompletnością, *validity* z poprawnością lub dokładnością (*accuracy* lub *consistency*), *correctness* utożsamia się z dokładnością (*accuracy*) [BaSc 2006].

2.3.3. CZYNNIKI ZWIĄZANE Z SEMANTYKĄ DANYCH

Czynniki zależne od semantyki wymagają od oceniającego wiedzy na temat ich znaczenia, a często także kontekstu użycia.

Spójność danych

Spójność (ang. *consistency*) mierzy poziom łamania reguł spójności w zbiorze danych [BaSc 2006]. Więzy integralności w relacyjnych bazach danych mogą być widziane jako pewien podzbiór takich właśnie reguł. Więzy musza być zachowane przez wszystkie instancje w bazie danych. Definiowane są one na poziomie schematu, ale w trakcie wykonania sprawdzane są na poziomie instancji. Reguły poprawności mogą być interrelacyjne (zależności funkcyjne, klucze itp.) oraz intrarelacyjne definiowane na poziomie instancji (wiersza) i obejmujące jeden lub kilka atrybutów. Przykładem reguły interrelacyjnej może być zależność klucza obcego. Poniższa reguła natomiast jest przykładem reguły intrarelacyjnej wyrażającej ograniczenie wiekowe (skończone 16 lat) dla posiadacza prawa jazdy.

```
if driver_license_no <> NULL then age>15
```

Fan ([Fan 2008], [FaGeJiK 2008]) proponuje wprowadzenie nowych odmian reguł zawierających wyrażenia warunkowe aby usprawnić zarządzanie jakością danych.

2.4. KONTEKST UŻYCIA A JAKOŚĆ DANYCH

Poprzednio opisane czynniki jakościowe są zdefiniowane (poza zależnymi od semantyki) jako bezkontekstowe. Jednakże uwzględniając definicje przytoczone wcześniej (Redman, English, ISO) jakość powinna być rozpatrywana przy uwzględnieniu wymagań użytkownika i w kontekście użycia danych.

Strong rozważa ten problem w pracy [StLeWa 1997] proponując zestaw czynników zależnych od kontekstu oraz identyfikując wzorce powodujące problemy z jakością w systemach informacyjnych. Słabością rozważań Stringa jest brak propozycji narzędzi do pomiaru jakości danych w kontekście użycia.

2.5. PODSUMOWANIE ZAGADNIEŃ ZWIĄZANYCH Z JAKOŚCIĄ

W rozdziale drugim przedstawiono podstawowe zagadnienia związane z jakością danych oraz wskazano jeden z podstawowych problemów, jakim jest brak spójnej definicji jakości w bazach danych. W tej sytuacji przyjęto jedną ze znanych definicji (definicję Naumanna) jako obowiązującą w dalszej części pracy.

Przedstawiono także wagę zagadnienia jakości danych w systemach informacyjnych w szczególności dla systemów integrujących dane oraz brak narzędzi do pomiarów jakości takich danych co stanowi motywację prowadzonych prac badawczych.

3. INTEGRACJA DANYCH

Integracja danych jest tematem eksplorowanym przez kilka ostatnich dekad przez wielu badaczy (np. [BoKe 2002], [BraJo 2007], [CuWiWi 2000], [FaKoTaP 2004], [GeSch 2007], [GuWi 1993], [Hass 2007], [HaRaOr 2006], [Kolaitis 2005], [Lenz 2002], [Pankowski 2007], [ReWa 1995], [ShSh 2006], [Ullman 1997]). W tym czasie zostało zaproponowanych wiele różnych podejść: od hurtowni danych po bazy rozproszone i systemy mediujące oraz P2P. W rozdziale przedstawimy krótki opis podstawowych rozwiązań z zakresu integracji danych oraz wpływ integracji na jakość danych.

3.1. OPIS PROBLEMU

Integracja danych to problem pozyskiwania i łączenia danych pochodzących z różnych źródeł oraz przedstawienia ich użytkownikowi w sposób jednorodny [Lenz 2002]. Problem projektowania systemów integracji danych staje się bardzo istotnym zagadnieniem w projektach informatycznych. Haas [Hass 2007] wskazuje cztery główne zadania stojące przed integratorem jakimi są zrozumienie (ang. *understanding*), standaryzacja (ang. *standardization*), specyfikacja (ang. *specification*) oraz wykonanie (ang. *execution*).

Bleiholder i Naumann w pracy [BlNa 2008] wskazują na trzy etapy integracji: mapowanie schematów, wykrywanie duplikatów oraz fuzję. Mimo pominięcia standaryzacji oraz zrozumienia proponowanego przez Haasa, są one niejawnie włączone w poszczególne kroki integracji.

3.2. ARCHITEKTURY ROZWIĄZAŃ INTEGRUJĄCYCH

Rozważając problem integracji danych spotkać możemy wiele rozwiązań opartych o różne architektury, które rozwinęły się w ciągu ostatnich dekad rozwoju w tej dziedzinie:

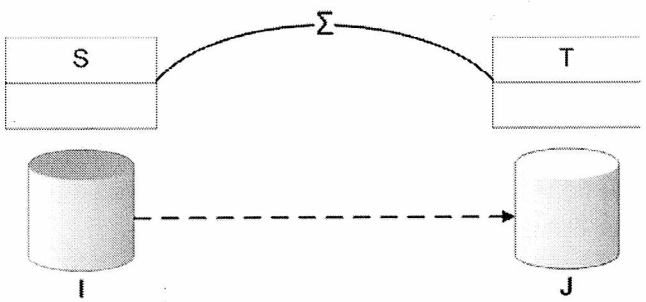
- bazy rozproszone – homogeniczne źródła danych pod kontrolą jednego systemu zarządzania rozproszona bazą danych;
- federacja baz danych, multi-bazy – autonomiczne, heterogeniczne źródła danych, dostęp opisany proceduralnie;
- systemy z mediatorem – dostęp do danych przez globalny schemat mapowany do autonomicznych, heterogenicznych źródeł, opisane deklaratywnie;
- hurtownie danych – wykorzystujące materializację (oraz agregację) jako metodę integracji danych pochodzących z różnych źródeł;
- P2P – niezależne systemy, każdy posiada mapowanie innych systemów do swojego schematu, brak globalnego schematu oraz deklaratywnych opisów.

3.3. MOŻLIWE PROBLEMY

W dziedzinie integracji danych przez ostatnie dziesięciolecia poczyniono olbrzymie postępy, jednakże ciągle możemy tam znaleźć problemy czekające na rozwiązanie. Część z zaobserwowanych problemów ma rozwiązania teoretyczne, inne mają jedynie praktyczne rozwiązania bez podstaw teoretycznych. Jedne i drugie mogą natomiast mieć wpływ na jakość danych, który powinien zostać rozważony.

Mapowanie schematów nie dotyka bezpośrednio danych, jednakże w czasie tego procesu pewne więzy i reguły mogą zostać utracone lub mogą być wprowadzone nowe; ze względu na zmiany w schemacie interpretacja niektórych danych może zostać zmieniona, a to z kolei może wpływać na postrzeganie jakości zintegrowanych danych.

Problemy z duplikatami oraz standaryzacją wartości wymagają już ingerencji w zapisy. Ingerencja taka może natomiast istotnie wpłynąć na jakość danych. Mimo, że intencją wykonującego takie operacje jest poprawa jakości, skutki mogą być różne i dlatego konieczny jest monitoring.



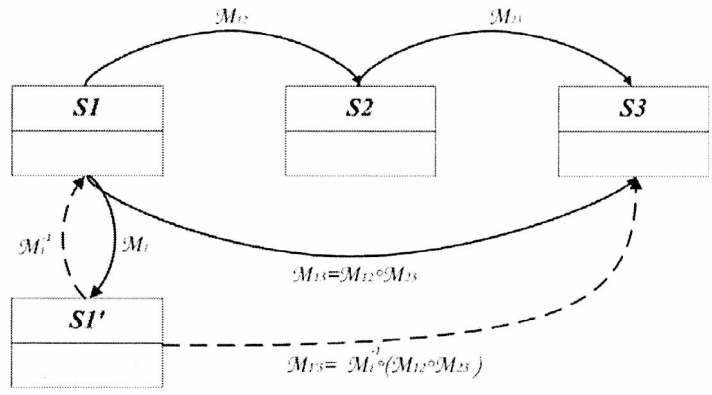
Rys. 5. Mapowanie schematów

3.3.1. MAPOWANIE SCHEMATÓW

Mapowania (ang. *schema mappings*) są to wysokopoziomowe opisy relacji pomiędzy schematami danych [Kolaitis 2005]. Używane są w systemach zarządzania danymi wykorzystującymi integrację danych, współdzielenie lub wymianę danych. W niektórych przypadkach owo mapowanie nie jest jednoznaczne i jeden schemat źródłowy może mapować się do wielu schematów docelowych. Taki stan powoduje, że pojawia się szereg pytań: Które z mapowań jest lepsze? Jakie jest znaczenie pytań do schematów źródłowych i jaka jest złożoność ewaluacji zapytania ponad wybranym schematem? Jak połączyć wiele transformacji?

Fagin, Kolaitis i inni badali to zagadnienie, a rozważany przez nich schemat integracji danych przedstawiony został na rysunku 5. Rozważali oni różne operatory konieczne do operacji na schematach danych takie jak inwersja [Fagin 2006] czy kom-

pozycja [FaKoTaP 2004]. Zaproponowany przez nich przykład ewolucji schematów możemy zobaczyć na rysunku 6, gdzie odnaleźć możemy kompozycję oraz inwersję. Prace w tej dziedzinie prowadzone były także przez grupę kierowaną przez Bernsteina ([BeMe 2007], [Ra Be 2001]). Zaproponowała ona model zarządzania schematami zwany *Model Management* oraz wymienili i nazwali konieczne operatory.



Rys. 6. Ewolucja schematów danych

Mapowanie schematów odgrywa kluczową rolę w integracji danych. Pozwala na opis zależności pomiędzy schematami. Mapowanie takie jednakże może istotnie wpłynąć na jakość integrowanych danych.

3.3.2. DUPLIKATY

Wykrywanie duplikatów oraz fuzja, czyli łączenie nie jest prostym zadaniem. Jednakże są to niezbędne kroki do otrzymania poprawnie zintegrowanych danych. Integrowane rekordy mogą nie współdzielić kluczy lub zawierać błędy, które czynią cały proces jeszcze trudniejszym ([ElIpVe 2007], [ShSh 2006]). Pojawiające się błędy mogą być różnorakiego typu, od błędów w literowaniu, przez niekompletne informacje aż po braki w standaryzacji pól. Mogą one występować pojedynczo lub jako kombinacje wielu błędów.

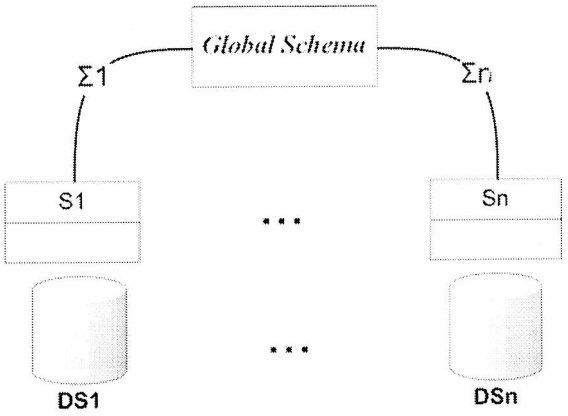
Fuzja następuje po wykryciu duplikatów. Ma za zadanie połączyć dane dotyczące tego samego obiektu świata rzeczywistego rozwiązując przy tym zaistniałe konflikty [BlNa 2008]. W czasie tego procesu mogą wystąpić dwa typy konfliktów: *niepewność* będąca konfliktem pomiędzy wartością nienullową oraz jedną lub więcej wartościami nullowymi opisującymi tę samą własność obiektu świata rzeczywistego, oraz *sprzeczność* będąca konfliktem pomiędzy dwiema lub więcej wartościami nienullowymi. Niepewność jest spowodowana brakiem informacji w źródle, sprzeczność różnicami pomiędzy źródłami.

Bleiholder i Naumann wymieniają szereg nierozwiązanych problemów w tym zagadnieniu takich jak złożone funkcje łączące (ang. *complex fusion functions*) biorące pod uwagę więcej niż jeden atrybut w czasie łączenia danych, fuzja inkrementacyjna, fuzja wykonywana on-line, czyli w trakcie wykonania zapytania itd.

Łączenie danych oraz eliminacja duplikatów silnie wpływa na jakość danych. Nasze zaufanie do tego procesu silnie rzutuje na zaufanie do wyprodukowanych w takim procesie danych. Poprawnie przeprowadzony proces może znacznie poprawić jakość przetworzonych danych jednakże błędy w tym procesie mogą zniszczyć jakość nawet najlepszych danych.

3.4. FEDERACJA DANYCH I INTEGRACJA W OPARCIU O MEDIATORA

Jednym z najważniejszych aspektów w tworzeniu mechanizmów integracji danych w systemach informatycznych jest specyfikacja zależności pomiędzy źródłami danych a schematem globalnym. Zależności te modelowane są w podobny sposób jak w wymianie danych za pomocą mechanizmu mapowania schematów. Dwa podstawowe podejścia do tego zagadnienia zostały opisane w tym podrozdziale. Są to *Global As View* (GAV) oraz *Local As View* (LAV).



Rys. 7. Schemat integracji danych wykorzystującej globalny schemat danych

3.4.1. GLOBAL AS VIEW

Opisany w [Halevy 2001] GAV jest podejściem opartym o proste założenie: schemat globalny musi zostać wyrażony w terminach schematów źródłowych. Daje to prosty przepis na pozyskanie danych. Innymi słowy, z każdym elementem schematu globalnego

związany jest pewien widok (zapytanie) ponad schematami źródłowymi. Zapytanie to definiuje bezpośrednio sposób pozyskania danych z systemów źródłowych.

Należy podkreślić, że podejście to sprawdza się jedynie w systemach, w których schematy źródłowe są stabilne. Stabilność ta jest konieczna ze względu na duży koszt zmian – zmiana schematu źródłowego wymaga nie tylko lokalnych modyfikacji, ale także zmian na poziomie globalnym (schemat globalny oraz związane z nim systemy).

Przetwarzanie zapytań w systemach opartych o model GAV, w większości przypadków, oparte jest o proste rozwijanie zapytań. To znaczy, mając zapytanie q ponad schematem globalnym, podstawia się za każdy element tego schematu, odpowiadające mu zapytanie, odnoszące się do źródła danych. Opisana tu prosta strategia działa przy założeniach, że system nie pozwala na więzy integralności (ang. *integrity constraints*) w schemacie globalnym oraz że widoki są dokładne (ang. *exact*). Jeśli natomiast dopuszcza się możliwość definiowania więzów integralności na poziomie globalnym, uzyskanie odpowiedzi na zapytania w modelu GAV staje się trudniejsze [Lenz 2002].

3.4.2. LOCAL AS VIEW

Podejście LAV [Lenz 2002] oparte jest o założenie, że schematy źródłowe powinny być opisane w terminach schematu globalnego. Innymi słowy, z każdym elementem schematu źródłowego związany jest widok ponad schematem globalnym. Otrzymuje się w ten sposób opis zawartości źródła danych w terminach schematu globalnego, jednak takie podejście nie definiuje w jaki sposób dane powinny zostać wydobyte z poszczególnych źródeł danych.

Podejście to wydawać się może niezrozumiałe lub nieintuicyjne, jednak niesie ze sobą olbrzymie korzyści dla systemów o dynamicznej strukturze źródeł danych. Dzięki „odwróceniu" spojrzenia na schematy i ich zależności, zmiany wprowadzane na poziomie schematów źródłowych mają jedynie lokalne konsekwencje. Modyfikacje schematu globalnego wymagają jednak w takiej sytuacji dużego nakładu pracy, ponieważ pociągają za sobą przedefiniowanie wszystkich schematów źródłowych.

Przetwarzanie zapytań w systemach opartych o LAV jest rozszerzeniem wnioskowania przy niepełnej informacji. Dwa podstawowe podejścia do przetwarzania zapytań w oparciu o widoki to przepisywanie zapytań oparte o widoki (ang. *view-based query rewriting*) oraz odpowiadanie na zapytania w oparciu o widoki (ang. *view-based query answering*).

Przepisywanie zapytania q oparte o widoki ma na celu przekształcenie zapytania do pewnego wyrażenia w pewnym ustalonym języku. Wyrażenie to ma odnosić się jedynie do źródeł danych i odpowiadać na zadane zapytanie q. Jeśli nie istnieje ekwiwalent zapytania q w języku docelowym, wtedy poszukuje się wyrażenia, które pokrywa zapytanie q w najlepszy możliwy sposób (ang. *maximally contained rewriting*).

W odpowiadaniu na zapytanie q opartym o widoki, poza zapytaniem i definicją widoków, dysponuje się także pewnym rozszerzeniem widoków. Celem jest tu znale-

zienie zbioru krotek t takich, że wiedza zawarta w rozszerzeniu widoków logicznie implikuje, że t jest odpowiedzią na zapytanie q.

Zauważyć należy różnicę pomiędzy opisanymi powyżej podejściami do przetwarzania zapytań w modelu LAV. Przepisywanie zapytań podzielone jest na dwie fazy. W pierwszej zapytanie jest tłumaczone, w drugiej ewaluowane. W podejściu drugim nie ma żadnych ograniczeń na to, w jaki sposób zapytanie będzie przetworzone, a jedynym celem jest uzyskanie całej dostępnej informacji na podstawie rozszerzenia widoków, aby odpowiedzieć na zapytanie.

Podejście LAV proponowane jest do tworzenia systemów o stabilnej strukturze globalnej oraz dla tych, w których źródła danych mają dynamiczne struktury lub się często zmieniają (dodawane są nowe, usuwane stare).

3.4.3. GLOBAL AND LOCAL AS VIEW (GLAV)

GLAV jest kombinacją poprzednio opisanych podejść [AlHaSh 2005]. Xu i Embley [XuEm 2003] proponują podobne podejście nazwane przez nich BGLaV. Motywacją do prac nad połączeniem obu podejść były trudności i ograniczenia, jakie stwarzało każde z osobna. W GAV były to koszty utrzymywania mapowania pomiędzy schematami lokalnymi a globalnym, przez co dynamiczne zmiany w lokalnych schematach wykluczały to podejście, mimo, że odpowiedzi na zapytania są w nim bardzo proste. W podejściu LAV złożoność odpowiedzi zniechęcała do zastosowania tego podejścia, mimo, że jest ono odporne na dynamiczne zmiany na poziomie lokalnym.

Model BGLaV zaproponowany przez Xu i Embley składa się z następujących warstw:

- Schemat globalny zdefiniowany niezależnie,
- Wrapping (widok) zdefiniowany dla lokalnych źródeł zdefiniowany niezależnie od schematu globalnego,
- Mapowanie pomiędzy schematem globalnym a „opakowanymi" elementami ze źródeł danych.

W takim podejściu przetwarzanie zapytań odbywa się jak w GAV z wykorzystaniem rozwijania. Zmiany na poziomie lokalnym, np. dodawanie nowych źródeł może być wykonane z taką łatwością jak w LAV i wymaga jedynie uwzględnienia w mapowaniach. Zmiany na poziomie globalnym wymagają natomiast przedefiniowania mapowań bez konieczności ingerencji w lokalne widoki.

3.5. MATERIALIZACJA – HURTOWNIE DANYCH

Celem materializacji danych jest utworzenie hurtowni danych zawierającej dane ze źródeł w jednolitym schemacie. Podstawowa i najbardziej powszechna technika materializacji danych zwana jest ETL (od angielskich słów *Extract, Transform, Load* czyli

ekstrakcja, transformacja i ładowanie). W tym podejściu dane są pozyskiwane ze źródeł i transformowane według wcześniej zdefiniowanych reguł. Następnie są składowane w odpowiednio do tego przygotowanej bazie danych.

Często wykorzystywanym mechanizmem, służącym do zmniejszenia objętości danych składowanych w hurtowniach jest agregacja. Dzięki jej wykorzystaniu oszczędzane jest miejsce a często także czas odpowiedzi (przy często obliczanych agregatach).

Wpływ tak przeprowadzonej integracji na jakość danych jest znaczący. Materializując dane musimy przeprowadzić większość kroków wykonywanych we wcześniej opisanej wirtualnej integracji (oczyszczenie, deduplikacja, fuzja), a jak wiemy, ma to istotny wpływ na jakość danych. Dodatkowo stosowana tu agregacja oraz częstotliwość odświeżania tak zeskładowanych danych ma wpływ na jakość odpowiedzi.

3.6. JAKOŚĆ DANYCH ZINTEGROWANYCH

W tej części pracy skoncentrujemy się bardziej na zagadnieniu jakości danych w procesie integracji. Część badań i raportów wskazuje bowiem, że aplikacje zbudowane ponad systemami integrującymi dane ma poważne problemy z jakością danych [GeSch 2007], [Gertz 1996], [JaVa 1997].

3.6.1. JAKOŚĆ DANYCH W SYSTEMACH FEDERACYJNYCH

Gertz w pracy [Gertz 1996] zaprezentował ciekawy punkt widzenia w zakresie integracji i jakości danych. Czyni on starania, aby zbudować formalny model jakości oraz wprowadza *ziarnistość* (ang. *granularity*) jako pewien częściowy porządek jednostek danych względem jednego z jakościowych aspektów.

Jednakże praca Gertza jest bardzo ogólna i nie znajdujemy tam wskazówek dotyczących transformacji wyników dla poszczególnych poziomów ziarnistości. Jedynie pewne wskazówki odnośnie estymacji wartości dla wyższych poziomów podane zostały w publikacji.

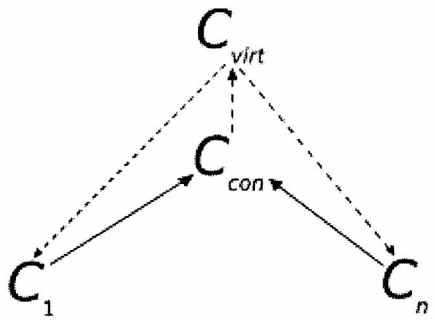
Naumann w pracy [Naumann 2002] rozważa jakość jako czynnik przy wyborze źródeł oraz planu wykonania zapytania ponad zintegrowaną bazą danych. Eliminacja źródeł o niskiej jakości może w takiej sytuacji przynieść oszczędności w czasie wykonania zapytania, jednakże pociąga za sobą ryzyko utraty istotnych danych. Dlatego też Naumann proponuje uznanie źródeł autorytatywnych (jedynych posiadających określoną informację np. atrybut lub określone rekordy) jako wysokiej jakości, nawet jeśli z obliczeń wynika inaczej. Wysiłki w celu formalizacji pojęcia jakości w kontekście integracji czyniła też Peralta [Peralta 2006] rozważając świeżość[3] danych oraz dokładność i podając estymatory dla jakości odpowiedzi.

---

[3] Pod tym pojęciem ukrywają się zarówno wiek danych jak i ich aktualność (*currency* i *timeliness*).

3.6.2. JAKOŚĆ W HURTOWNIACH DANYCH

Gertz i Schmitt w pracy [GeSch 2007] zaprezentowali podejście do jakości oparte o idee klas konceptualnych i wirtualnych.



Rys. 8. Zależności pomiędzy klasami wirtualnymi, konceptualnymi i lokalnymi

Na rysunku 8 przedstawiono zależności pomiędzy klasami wirtualnymi, konceptualnymi oraz lokalnymi. C_{con}, klasa konceptualna, jest reprezentacją pewnej globalnej klasy (schematu globalnego). Klasa wirtualna C_{virt} jest reprezentacją obiektu świata rzeczywistego, do którego referuje klasa konceptualna. Klasy lokalne $C_1 ... C_n$ są semantycznie równoważne i referują do tego samego obiektu świata rzeczywistego.

W oparciu o tak zbudowany model Gertz i Schmitt definiują aktualność, kompletność oraz dokładność jako pewien częściowy porządek pomiędzy klasami co pozwala na określenie, która klasa jest wyższej jakości (bardziej dokładna, kompletna lub aktualna).

Problem jakości w hurtowniach poruszany był także przez Jarke i Vasilliou [JaVa 1997] oraz Wanga [ReWa 1995], podających podział czynników jakościowych dla hurtowni danych, a także szerzej w pracy [JaLeVa 2001].

4. KIERUNKI DALSZYCH BADAŃ

Rozważając zagadnienia związane z integracją danych zauważamy, że nie istnieją rozwiązania pozwalające na obliczanie jakości danych zintegrowanych w oparciu o wyliczenia poczynione dla źródeł danych. Brak w literaturze jednolitego spojrzenia na zagadnienie oraz narzędzi wspomagających takie obliczenia.

Przyszłe prace będą miały na celu uzupełnienie tej luki i dostarczenie teoretycznego modelu oraz praktycznych narzędzi pozwalających na pomiary jakości danych zintegrowanych, które bazować będą na informacji o jakości danych w poszczególnych źródłach. Istotnym problemem będzie tu wydajność algorytmów obliczających poszczególne czynniki jakościowe, w szczególności w sytuacji, kiedy czynniki te będą

wyliczane on-line, czyli w trakcie wykonania zapytania. Chcemy także zbadać możliwości optymalizacji zapytań do systemu zarządzania danymi zintegrowanymi w oparciu o czynniki jakościowe.

Ciekawym zagadnieniem wynikającym częściowo z rozważań na temat jakości jest identyfikacja innych czynników (niejakościowych) wpływających na zaufanie użytkownika do danych. Zaufanie definiowane jest zazwyczaj, jako relacja pomiędzy dwoma aktorami (osobami lub organizacjami) [DViJaPaSa 2007], [DiBe 2004], [JoIsBo 2007], [ZuPa 2005]. Brak w literaturze odniesień do zaufania w kontekście *użytkownik-dane*.

Nasze badania będą miały na celu umożliwienie odpowiedzi na pytanie stawiane często przez użytkownika „*Na ile mogę ufać tym danym?*". W tym celu konieczne będzie zidentyfikowanie czynników wpływających na zaufanie zarówno jakościowych jak niejakościowych, oraz zbudowanie modelu pozwalającego na obliczenia.

5. PODSUMOWANIE

W powyższym rozdziale przedstawione zostały podstawowe zagadnienia związane z jakością danych oraz z integracją. Wskazaliśmy na istotny wpływ integracji danych na ich jakość oraz istotność jakości dla użytkowników.

Integracja danych dla wielu instytucji jest koniecznością, niestety wykonanie poszczególnych kroków integracji nie gwarantuje otrzymania wysokiej jakości produktu końcowego. Istotny wpływ na wynik ma nie tylko jakość danych wejściowych, ale także wiedza przeprowadzających integrację.

Dla osób podejmujących decyzje przestało być problemem pozyskanie danych. Integracja dostępnych źródeł pozwala bowiem na szybki dostęp do zasobów. Problemem stało się natomiast ocenienie przydatności czy też zaufania do danych pozyskanych w ten sposób. Użytkownik często nie posiada wiedzy na temat pochodzenia danego wycinka informacji, a wiedza taka stanowi istotny element procesu decyzyjnego. Zaufanie do źródła danych skutkuje w większym lub mniejszym stopniu uwzględnieniem danych informacji w czasie podejmowania decyzji. Obecnie użytkownik zalany jest danymi, których przydatności nie jest w stanie ocenić bez dodatkowej wiedzy.

W rozdziale tym wskazaliśmy na braki w dotychczasowych modelach jakości, nieuwzględniających integracji danych, a także na brak spójnej definicji jakości. Wskazujemy też możliwe kierunki badań w tej dziedzinie, co być może zaowocuje nowym spojrzeniem na zagadnienia zaufania do danych oraz jakości danych.

LITERATURA

[AlHaSh 2005] Alasoud A., Haarslev V., Shiri N., *A hybrid approach for ontology integration*. In: Proceedings of the 31st VLDB Conference, 2005.

[ArBeCho 1999] Arenas M., Bertossi L., Chomicki J., *Consistent query answers in inconsistent databases.* In: PODS '99: Proceedings of the eighteenth ACM SIGMODSIGACT-SIGART symposium on Principles of database systems, ACM, New York, NY, USA, 1999, 68–79.

[ArBeCHR 2003] Arenas M., Bertossi L., Chomicki J., He X., Raghavan V, Spinrad J., *Scalar aggregation in inconsistent databases*, Theor. Comput. Sci., 296(3), 2003, 405–434.

[BaPa 1985] Ballou D., Pazer H., *Modeling data and process quality in multi-input, multioutput information systems*, Management Science, 31(2), 1985, 150–162.

[BaWaPaTa 1998] Ballou D., Wang R., Pazer H., Tayi G.K., *Modeling information manufacturing systems to determine information product quality*, Manage. Sci., 44(4), 1998, 462–484.

[BaSc 2006] Batini C., Scannapieco M., *Data Quality: Concepts, Methodologies and Techniques. Data-Centric Systems and Applications*, Springer-Verlag, New York, Inc., Secaucus, NJ, USA, 2006.

[BeMe 2007] Bernstein P.A., Melnik S., *Model management 2.0: manipulating richer mappings.* In: Chee Yong Chan, Beng Chin Ooi, and Aoying Zhou, editors, SIGMOD Conference, ACM, 2007, 1–12.

[Bertossi 2006] Bertossi L., *Consistent query answering in databases*, SIGMOD Rec., 35(2), 2006, 68–76.

[BlNa 2008] Bleiholder J., Naumann F., *Data fusion*, ACM Comput. Surv., 41(1), 2008.

[BoMaYa 1998] Bobrowski M., Marr M., Yankelevich D., *A software engineering view of data quality.* In: European Quality Week Conference, 1998.

[BoKe 2002] Bouzeghoub M., Kedad Z., *Quality in Data Warehousing*, Kluwer Academic Publisher, 2002.

[BraBer 2004] Bravo L., Bertossi L., *Consistent query answering under inclusion dependencies.* In: CASCON '04. Proceedings of the 2004 conference of the Centre for Advanced Studies on Collaborative research, IBM Press, 2004, 202–216.

[BraJo 2007] Brazhnik O., Jones J.F., *Anatomy of data integration*, J. of Biomedical Informatics, 40(3), 2007, 252–269.

[CaCaGiLe 2002] Cali A., Calvanese D., Giacomo G, Lenzerini M., *On the role of integrity constraints in data integration*, IEEE Data Eng. Bull., 25(3), 2002, 39–45.

[CaCaGiLe 2004] Cali A., Calvanese D., Giacomo G, Lenzerini M., *Data integration under integrity constraints*, Inf. Syst., 29(2), 2004, 147–163.

[Crosby 1979] Crosby P.B., *Quality is free: the art of making quality certain*, McGraw-Hill, New York 1979.

[CuWiWi 2000] Cui Y., Widom J., Wiener J.L., *Tracing the lineage of view data in a warehousing environment*, ACM Trans. Database Syst., 25(2), 2000, 179–227.

[DaJo 2003] Dasu T., Johnson T., *Exploratory Data Mining and Data Cleaning*, Wiley-Interscience, May 2003.

[DViJaPaSa 2007] De Capitani di Vimercati S., Jajodia S., Paraboschi S., Samarati P., *Trust management services in relational databases.* In: ASIACCS '07. Proceedings of the 2nd ACM symposium on Information, computer and communications security, ACM, New York, NY, USA, 2007, 149–160.

[DiBe 2004] Dimmock N., Belokosztolszki A., Eyers D., Bacon J., Moody K., *Using trust and risk in role-based access control policies.* In: SACMAT '04. Proceedings of the ninth ACM symposium on Access control models and technologies, ACM, New York, NY, USA, 2004, 156–162.

[DrHeMaObS 2007] Dreibelbis A., Hechler E., Mathews B., Oberhofer M., Sauter G., *Master data management architecture patterns*, 2007.

[ElIpVe 2007] Elmagarmid A.K., Ipeirotis P.G., Verykios V.S., *Duplicate record detection: A survey*, Knowledge and Data Engineering, IEEE Transactions on, 19(1), 2007, 1–16.

[English 1996] English L., *Information quality improvement: Principles, methods, and management*, Seminar, 1996.

[Fagin 2006] Fagin R., *Inverting schema mappings*. In: PODS '06. Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, ACM, New York, NY, USA, 2006, 50–59.

[FaKoTaP 2004] Fagin R., Kolaitis P.G., Tan W.C, Popa L., *Composing schema mappings: second-order dependencies to the rescue*. In: PODS '04. Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, ACM, New York, NY, USA, 2004, 83–94.

[Fan 2008] Fan W., *Dependencies revisited for improving data quality*. In: PODS '08. Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, ACM, New York, NY, USA, 2008, 159–170.

[FaGeJiK 2008] Fan W., Geerts F., Jia X., Kementsietsidis A., *Conditional functional dependencies for capturing data inconsistencies*, ACM Trans. Database Syst., 33(2), 2008, 1–48.

[FoHe 2007] Foley O., Helfert M., *The development of an objective metric for the accessibility dimension of data quality*. In: Proceedings of International Conference on Innovations in Information Technology, IEEE, Dublin 2007, 11–15.

[GeSch 2007] Gertz M., Schmitt I., *Data Integration Techniques based on Data Quality Aspects*. In I. Schmitt, C. Türker, E. Hildebrandt, M. Höding, editors, Proceedings 3. Workshop „Föderierte Datenbanken", Magdeburg, 10–11 Dezember 1998, Shaker Verlag, Aachen, 1–19.

[Gertz 1996] Gertz M., *Managing data quality and integrity in federated databases*. In: 2nd Annual IFIP TC-11 WG 11.5 Working Conf. on Integrity and Internal Control in Information Systems, 1996, 211–230.

[GrGLZ 2004] Gryz J., Guo J., Liu L., Zuzarte C., *Query sampling in db2 universal database*. In: SIGMOD '04. Proceedings of the 2004 ACM SIGMOD international conference on Management of data, ACM, New York, NY, USA, 2004, 839–843.

[GuWi 1993] Gupta A., Widom J., *Local verification of global integrity constraints in distributed databases*. In: P. Buneman, S. Jajodia (Eds.), Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 26–28, 1993, ACM Press, 1993, 49–58.

[Hass 2007] Haas L.M., *Beauty and the beast: The theory and practice of information integration*. In: Th. Schwentick, D. Suciu (Eds.), ICDT, Vol. 4353 of Lecture Notes in Computer Science, Springer, 2007, 28–43.

[HaRaOr 2006] Halevy A., Rajaraman A., Ordille J., *Data integration: the teenage years*. In: VLDB '06. Proceedings of the 32nd international conference on Very large data bases, VLDB Endowment, 2006, 9–16.

[Halevy 2001] Halevy A.Y., *Answering queries using views: A survey*, The VLDB Journal, 10(4), 2001, 270–294.

[ISO 1994] ISO8402. *Quality management and quality assurance: Vocabulary*. Published standard, 1994.

[JaVa 1997] Jarke M., Vassiliou Y., *Data warehouse quality design: A review of the DWQ project*. In: Proc. 2nd Conference on Information Quality, 1997.

[JaLeVa 2001] Jarke M., Lenzerini M., Vassiliou Y., Vassiliadis P., *Fundamentals of Data Warehouses*, Springer-Verlag, New York, Inc., Secaucus, NJ, USA, 2001.

[JoIsBo 2007] Jøsang A., Ismail R., Boyd C., *A survey of trust and reputation systems for online service provision*, Decision Support Systems, 43(2), March 2007, 618–644.

[Kolaitis 2005] Kolaitis P.G., *Schema mappings, data exchange, and metadata management*. In: PODS '05. Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, ACM, New York, NY, USA, 2005, 61–75.

[KriMo 1982] Kriebel C.H., Moore J.H., *Economics and management information systems*, SIGMIS Database, 14(1), 1982, 30–40.

[LePiStWa 2004] Lee Y.W., Pipino L., Strong D.M., Wang R.Y., *Process embedded data integrity*, J. Database Manag., 15(1), 2004, 87–103.

[Lenz 2002] Lenzerini M., *Data integration: a theoretical perspective*. In: PODS '02. Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, ACM, New York, NY, USA, 2002, 233–246.

[Naumann 2002] Naumann F., *Quality-driven query answering for integrated information systems*, Springer-Verlag, New York, Inc., New York, NY, USA, 2002.

[Olken 1993] Olken F., *Random Sampling from Databases*, PhD thesis, University of California at Berkeley, 1993.

[Olson 2002] Olson J., *Data Quality: The Accuracy Dimension*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[Pankowski 2007] Pankowski T., *Integracja danych w teorii i praktyce-przegląd problemów i rozwiązań*. W: Bazy Danych: Nowe Technologie, WK, 2007, 45–55.

[PaSaJa 2002] Parssian A., Sarkar S., Jacob V.S., *Assessing information quality for the composite relational operation join*. In: IQ, 2002, 225–237.

[PaSaJa 2004] Parssian A., Sarkar S., Jacob V.S., *Assessing data quality for information products: Impact of selection, projection, and cartesian product*, Manage. Sci., 50(7), 2004, 967–982.

[Peralta 2006] Peralta V., *Data Quality Evaluation in Data Integration Systems*, PhD thesis, Universite de Versailles Saint-Quentin-en-Yvelines (France) and Univeridad de la Republica (Uruguay), 2006.

[PiLeWa 2002] Pipino L.L., Lee Y.W, Wang R.Y., *Data quality assessment*. Communications of the ACM, 45, 2002, 211–218.

[Ra Be 2001] Rahm E., Bernstein P.A., *A survey of approaches to automatic schema matching*, VLDB Journal, Very Large Data Bases, 10(4), 2001, 334–350.

[ReWa 1995] Reddy M.P., Wang R.Y., *Estimating data accuracy in a federated database environment*. In: CISMOD, 1995, 115–134.

[Redman 1997] Redman T.C., *Data Quality for the Information Age*, Foreword by Godfrey, A. Blanton, Artech House, Inc., Norwood, MA, USA, 1997.

[Redman 2001] Redman T.C., *Data quality: the field guide*, Digital Pr. [u.a.], Boston 2001.

[SeFa 1990] Segev A., Fang W., *Currency-based updates to distributed materialized views*. In: Proceedings of the Sixth International Conference on Data EngineeringWashington, IEEE Computer Society, DC, USA, 1990, 512–520.

[ShSh 2006] Shahri H.H., Shahri S.H., *Eliminating duplicates in information integration: An adaptive, extensible framework*, IEEE Intelligent Systems, 21(5), 2006, 63–71.

[Shin 2003] Shin B., *An exploratory investigation of system success factors in data warehousing*, J. AIS, 4, 2003.

[SiWe 2009] Simpson J., Weiner E., *Oxford english dictionary*, 2009.

[StLeWa 1997] Strong D.M., Lee Y.W., Wang R.Y., *Data quality in context*, Commun. ACM, 40(5), 1997, 103–110.

[TaBa 1998] Tayi G.K., Ballou D.P., *Examining data quality*, Commun. ACM, 41(2), 1998, 54–57.

[Tupek 2006] Tupek A.R., *Definition of data quality*, 2006.

[Ullman 1997] Ullman J.D., *Information integration using logical views*. In: ICDT '97. Proceedings of the 6th International Conference on Database Theory, Springer-Verlag, London, UK, 1997, 19–40.

[WaWa 1996] Wand Y., Wang R.Y., *Anchoring data quality dimensions in ontological foundations*, Commun. ACM, 39(11), 1996, 86–95.

[WaChe 2005] Wang, R.Y., Chettayar K., Dravis F., Funk J., Katz-Haas R., Lee C., Lee Y., Xian X., Bhansali S., *Exemplifying business oppurtunities for improving data quality from corporate household research*. In: Advances in Management Information Systems – Information Quality (AMIS-IQ) Monograph, April 2005.

[WaSt 1996] Wang R.Y., Strong D.M., *Beyond accuracy: what data quality means to data consumers*, J. Manage. Inf. Syst., 12(4), 1996, 5–33.

[XuEm 2003] Xu L., Embley D.W., *Combining the best of global-as-view and local-as-view for data integration*. In: Information Systems Technologies and its Applications – ISTA, 2003, 123–136.

[ZuPa 2005] Zuo Y., Panda B., *Component based trust management in the context of a virtual organization*. In: SAC '05. Proceedings of the 2005 ACM symposium on Applied computing, ACM, New York, NY, USA, 2005, 1582–1588.

DATA IN THE CONTEXT OF INTEGRATED DATA

The data integration is a problem rises in the areas where applications require information from multiple autonomous and heterogeneous sources. The need of integration is seen especially in dynamically growing organizations such as banks, but also in large projects done for governmental institutions.

The data quality is in the literature defined in many different ways. The need on measurement of the quality rises with the increasing number of incoming data. In this chapter we have shown basic concepts of data integration and data quality emphasising the problem of the quality of integrated data.

Prace Naukowe Instytutu Informatyki
Politechniki Wrocławskiej

Zeszyt 10 2009

protokół TCP,
niedostrzegalne kanały komunikacyjne,
podpisy cyfrowe

Arkadiusz LIBER*,
Jarosław DYBOWSKI

MOŻLIWOŚCI STOSOWANIA PROTOKOŁU TCP DO KONSTRUKCJI NIEDOSTRZEGALNYCH KANAŁÓW TRANSMISYJNYCH DLA CELÓW KRYPTOGRAFICZNYCH

W pracy przedstawiono wyniki badań autorów w zakresie konstrukcji niedostrzegalnych kanałów informacyjnych osadzonych w znormalizowanych protokołach komunikacyjnych. Kanały takie mogą być wykorzystywane nie tylko do wytworzenia dodatkowych niezależnych kanałów przesyłowych, lecz również mogą być wykorzystane do realizacji protokołów związanych z ochroną danych lub weryfikacją użytkownika. W ramach pracy przedstawiono analizę szczegółową protokołu TCP pod kątem konstrukcji niedostrzegalnych kanałów. Przeprowadzono badania niedostrzegalnych kanałów osadzonych w protokole TCP pod kątem umieszczania w nich podpisów cyfrowych. W ramach przeprowadzonych badań wykonano eksperymentalną dystrybucję systemu LINUX ze zmodyfikowanym jądrem. W jądrze zaimplementowano niedostrzegalne warstwy oparte na protokole TCP. Przeprowadzono badania ukrytych warstw zawierających podpisy cyfrowe danych. Praca stanowi kontynuację i rozszerzenie wcześniejszych badań autorów nad protokołami komunikacyjnymi w tym nad protokołem IP [Liber 2004], [Liber 2006].

1. WPROWADZENIE

Problematyka przesyłania dodatkowych informacji w protokołach komunikacyjnych pojawiła się w drugiej połowie lat dziewięćdziesiątych. Znalazła ona swój wyraz w pracach Handela, Sandforda i Rowlanda [Handel 1996], [Rowland 1997]. Propozycje zaawansowanych zastosowań kryptograficznych pojawiły się w pracach Ashana [Ashan 2002], Szczypiorskiego [Szczyp 2003] oraz Libera i Kosmulskiej-Bochenek [Liber 2004]. Szczegółowa analiza protokołu IP poparta skonstruowaniem

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

eksperymentalnej platformy badawczej znalazła swój wyraz w pracy Libera i Dybowskiego [Liber 2006]. Przedmiotem niniejszej pracy jest opis rezultatów badań autorów nad możliwościami konstrukcji niedostrzegalnych warstw informacyjnych osadzonych na protokole TCP. Szczególny nacisk położono na możliwości stosowania ukrytych warstw do osadzania w protokole TCP schematów kryptograficznych ze szczególnym uwzględnieniem podpisów cyfrowych.

2. PROTOKÓŁ TCP JAKO NOŚNIK DODATKOWEJ INFORMACJI

Zadaniem protokołu TCP jest dostarczenie możliwości nawiązania niezawodnego połączenia pomiędzy nadawcą a odbiorcą. Niezawodne połączenie zapewnia dotarcie wszystkich wysłanych danych z zachowaniem ich kolejności.

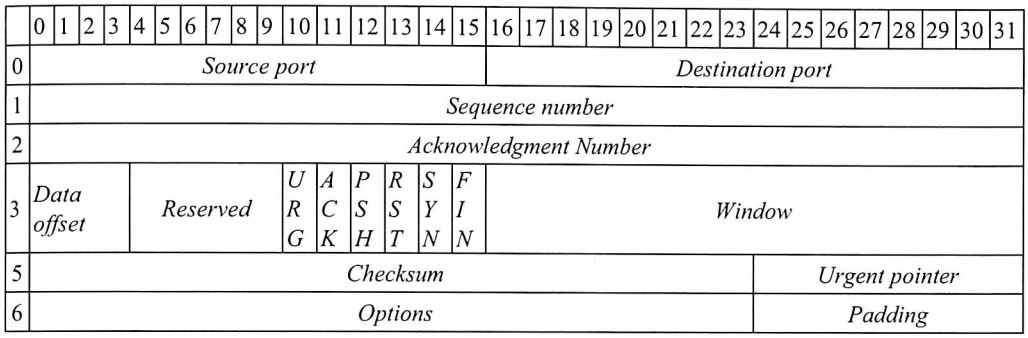
W celu uszeregowania u odbiorcy pakietów TCP w prawidłowej kolejności, protokół TCP wprowadza numery sekwencyjne dla każdej strony połączenia, których wymiana następuje w trakcie nawiązywania połączenia [Postel 1981]. Od chwili otrzymania pakietu synchronizującego (pakietu złożonego tylko z nagłówka TCP z ustawioną flagą SYN), strona odbierająca jest zobowiązana do potwierdzania stronie nadającej odbioru pakietu z danymi za pomocą pakietu potwierdzającego (złożonego tylko z nagłówka TCP z ustawioną flagą ACK oraz numerem potwierdzającym wskazującym jednoznacznie pakiet, którego przybycie potwierdza). Każdy wychodzący pakiet z danymi również powinien potwierdzać odbiór któregoś z odebranych pakietów. W specyfikacji protokołu TCP nie definiuje się, jak często powinny być wysyłane pakiety potwierdzające, wobec czego w celu zwiększenia wydajności często stosuje się zbiorcze potwierdzenie odebrania kilku pakietów jednym pakietem potwierdzającym.

W celu zapewnienia integralności wszystkich przesyłanych danych, suma kontrolna przesyłana w pakiecie TCP to nie tylko suma kontrolna nagłówka, ale całego pakietu, łącznie z niektórymi polami nagłówka ramki IP. Na dotarcie do odbiorcy pakietu TCP mają wpływ zapory ogniowe, które mogą uniemożliwiać komunikację poprzez określone porty lub zatrzymywać pakiety o nieprawidłowej zawartości. Rolę filtrującą często pełnią również pełnomocnicy połączeń (ang. *proxy*), którzy nawiązują połączenie z odbiorcą w imieniu nadawcy. Zazwyczaj żadna ze stron połączenia nie ma wpływu na obecność tych urządzeń na trasie połączenia.

2.1. NAGŁÓWEK PROTOKOŁU TCP

W celu przeanalizowania możliwości konstrukcji niedostrzegalnych kanałów komunikacyjnych na bazie protokołu TCP należy dokładnie zbadać postać ramek TCP oraz sposób ich propagacji w sieci. Na rysunku 1 przedstawiono widok na-

główka ramki TCP. Na rysunku zaznaczono nazwy poszczególnych pól oraz ich wzajemne położenie. W kolejnych rozdziałach przeprowadzono analizę poszczególnych pól i flag pod kątem przydatności do tworzenia niedostrzegalnych kanałów kryptograficznych.



Rys. 1. Format nagłówka protokołu TCP: *Source Port* – pole *Port źródłowy*; *Destination Port* – pole *Port docelowy*; *Sequence Number* – pole *Numer kolejny; Acknowledgment Number* – pole *Numer potwierdzający*; *Data Offset* – pole *Przesunięcie danych*; *Reserved* – pole *Miejsce zarezerwowane*; *URG* – flaga *Wskaźnik na pilne dane obecny*; *ACK* – flaga *Numer potwierdzający obecny*; *PSH* – flaga *Przekaż dane aplikacji*; *RST* – flaga *Przerywam połączenie*; *SYN* – flaga *Synchronizuj numery sekwencji*; *FIN* – flaga *Żądanie zakończenia połączenia*; *Window* – pole *Rozmiar okna*; *Checksum* – pole *Suma kontrolna*; *Urgent Pointer* – pole *Wskaźnik na pilne dane*; *Options* – pole *Opcje*; *Padding* – pole *Wypełnienie*

2.2. POLE *PORT ŹRÓDŁOWY*

Wartość pola *Port źródłowy* oznacza port nadawcy, na który odbiorca pakietu powinien wysłać odpowiedź po jego przetworzeniu, tj. wartość, którą powinien wpisać odbiorca w polu *Port docelowy* przygotowując pakiet będący odpowiedzią na pakiet właśnie otrzymany. Pole *Port źródłowy* może być nośnikiem ukrytego podpisu, gdy nadawca uzgodni wcześniej z odbiorcą, że będzie oczekiwał na odpowiedzi od niego tylko pod jednym, konkretnym portem (port ten nie powinien być wykorzystywany przez nadawcę do żadnej innej komunikacji z odbiorcą). Wtedy odbiorca powinien wysyłać pakiety zwrotne tylko na ten jeden port nadawcy, niezależnie od faktycznej wartości pola *Port źródłowy* w odbieranych pakietach. W takim przypadku możliwe jest wykorzystanie 16 bitów na ukrycie niejawnej informacji. Rozwiązanie takie oznacza zaangażowanie przez nadawcę wszystkich portów w komunikację z odbiorcą. Skutkiem tego jest możliwość utrzymywania tylko jednego połączenia nadawcy z konkretnym odbiorcą, czyli ograniczenie ilości aplikacji nadawcy podłączonych do maszyny odbiorcy do jednej. W rzeczywistych warunkach często nie jest to przeszkodą w stosowaniu tego typu rozwiązania.

W przypadku nawiązania przez nadawcę jednocześnie kilku połączeń z odbiorcą, można przeznaczyć część z 16 oferowanych bitów na przechowanie informacji pozwalającej odróżnić od siebie poszczególne połączenia. Należy wtedy wcześniej ustalić z odbiorcą sposób przyporządkowania poszczególnych zakodowanych identyfikatorów połączeń do konkretnych, fizycznych portów nadawcy. Zakładając, że nadawca chce nawiązać jednocześnie k różnych połączeń, ilość bitów informacji dostępnych dla niewidocznego podpisu równa jest największej liczbie całkowitej nie większej od $16 - \log_2(k)$.

Dużą przeszkodą w zastosowaniu tego rozwiązania może być obecność urządzenia przekierowującego porty lub zapory ogniowej na brzegu sieci nadawcy lub odbiorcy. Wystąpienie tych urządzeń może znacznie ograniczyć liczbę dostępnych portów możliwych do zastosowania przez nadawcę w polu *Port źródłowy*, osłabiając w ten sposób siłę podpisu. Zapory ogniowe mające możliwość śledzenia nawiązanych połączeń mogą odmówić przekazania pakietu widząc, że nadawca używa innego portu do wysyłania i innego do odbierania pakietów podczas komunikacji z tym samym portem odbiorcy. Również z tego samego powodu podsłuchujący może nabrać podejrzeń, co do ukrytego znaczenia pola *Port źródłowy*, jeśli tylko przez podsłuchiwany segment sieci przechodzą pakiety wysyłane przez obydwie strony połączenia. Rozwiązanie to charakteryzuje się dużą złożonością oprogramowania po stronie nadawcy, który jest zmuszony do przetwarzania danych z wielu portów niemalże jednocześnie.

2.3. POLE *PORT DOCELOWY*

Wartość pola *Port docelowy* oznacza aplikację, do której powinny zostać skierowane dane zawarte w odebranym pakiecie. Pole *Port docelowy* może być nośnikiem ukrytego podpisu, gdy odbiorca przeznaczy łącznie wszystkie swoje porty na jedno połączenie z nadawcą. W takim przypadku możliwe jest wykorzystanie 16 bitów na ukrycie niejawnej informacji. Rozwiązanie takie oznacza zaangażowanie przez odbiorcę wszystkich portów w komunikację z nadawcą. Skutkiem tego jest możliwość utrzymywania tylko jednego połączenia odbiorcy z konkretnym nadawcą, czyli ograniczenie ilości aplikacji odbiorcy udostępnianych dla nadawcy do jednej. W rzeczywistych warunkach często nie jest to przeszkodą w stosowaniu tego typu rozwiązania.

W przypadku udostępnienia przez odbiorcę jednocześnie kilku połączeń z nadawcą, można przeznaczyć część z 16 oferowanych bitów na przechowanie informacji pozwalającej odróżnić od siebie poszczególne połączenia. Należy wtedy wcześniej ustalić z nadawcą sposób przyporządkowania poszczególnych zakodowanych identyfikatorów połączeń do zbioru fizycznych portów odbiorcy. Zakładając, że odbiorca chce udostępnić jednocześnie k różnych połączeń, ilość bitów informacji dostępnych dla niewidocznego podpisu równa jest największej liczbie całkowitej nie większej od $16 - \log_2(k)$.

Dużą przeszkodą w zastosowaniu tego rozwiązania może być obecność urządzenia przekierowującego porty lub zapory ogniowej na brzegu sieci nadawcy lub odbiorcy. Wystąpienie tych urządzeń może znacznie ograniczyć ilość dostępnych portów możliwych do zastosowania przez nadawcę w polu *Port docelowy*, osłabiając w ten sposób siłę podpisu. Zapory ogniowe mające możliwość śledzenia nawiązanych połączeń mogą odmówić przekazania pakietu widząc, że nadawca wysyła pakiety pod porty odbiorcy, z którymi nie nawiązał właściwego połączenia. Również z tego samego powodu podsłuchujący może nabrać podejrzeń, co do ukrytego znaczenia pola *Port docelowy*. Rozwiązanie to charakteryzuje się dużą złożonością oprogramowania po stronie odbiorcy, który jest zmuszony do przetwarzania danych z wielu portów niemalże jednocześnie.

2.4. POLE *NUMER KOLEJNY*

Ponieważ podczas wysyłania danych cały pakiet protokołu TCP musi zmieścić się w pojedynczej ramce protokołu IP, a aplikacje końcowe oparte na protokole TCP często wysyłają dane o rozmiarze przekraczającym maksymalny dozwolony rozmiar ramki IP (65535 bajtów), na poziomie warstwy transportowej zachodzi konieczność podzielenia danych wysyłanych przez aplikację na porcje mieszczące się w pojedynczych ramkach protokołu IP. W celu zagwarantowania, że podzielone dane będą mogły zostać przetworzone przez odbiorcę w dobrej kolejności (protokół IP nie dostarcza takiej pewności), TCP jako niezawodny protokół połączeniowy powinien dostarczyć mechanizmów pozwalających na poprawne złączenie danych przez odbiorcę. Wartość pola *Numer kolejny* dostarcza odbiorcy informacji o miejscu odebranego pakietu w sekwencji pakietów odbieranych w trakcie trwania całego połączenia (jednostką porządkową jest bajt). Pozwala to na bezbłędne scalenie dużych danych przed przekazaniem ich do aplikacji końcowej. Pole to umożliwia też identyfikację pakietu w sieci i dlatego możliwe jest potwierdzenie przez odbiorcę poprawnego odebrania konkretnego pakietu.

Pole *Numer kolejny* może być nośnikiem ukrytej informacji, jeśli tylko nie przeszkodzi to w poprawnym umiejscowieniu odebranego pakietu sekwencji oraz, jeśli w dalszym ciągu zapewniona będzie różna wartość tego pola dla pakietów o różnej zawartości, niezaakceptowanych jeszcze przez odbiorcę. Miejscem, gdzie zachodzi porządkowanie odebranych pakietów od nadawcy jest bufor odbiorczy, który w ogólności może mieć dowolnie duży rozmiar. Minimalizując liczbę pakietów znajdujących się w dowolnej chwili w buforze odbiorczym można zminimalizować prawdopodobieństwo kolizji numerów sekwencyjnych podczas przeprowadzania procesu porządkowania pakietów. W większości systemów operacyjnych (m. in. w systemie operacyjnym Linux, Microsoft Windows 2000 i Microsoft Windows XP), maksymalny rozmiar bufora odbiorczego można ustawić na poziomie interfejsu sieciowego. Odbiorca może również sterować ilością wysyłanych do niego pakietów informując in-

dywidualnie nadawcę o rozmiarze danych, jaki jest w stanie w danej chwili przetworzyć, za pomocą pola *Rozmiar okna* (pole to zostało opisane dokładniej w dalszej części pracy). Wartość tego pola odbiorca wysyła w każdym pakiecie potwierdzającym przyjęcie danych (w każdym pakiecie z włączoną flagą ACK, również podczas fazy nawiązywania połączenia), a więc bardzo często. Zgodnie ze standardem, pole *Rozmiar okna* z każdym nadejściem takiego pakietu musi być respektowane przez nadawcę, tzn. nadawca do czasu otrzymania kolejnego pakietu potwierdzającego (z nową wartością pola *Rozmiar okna*) nie może wysłać do odbiorcy więcej danych (w bajtach), niż wynosi aktualna wartość tego pola.

Prawdopodobieństwo kolizji numerów sekwencyjnych podczas przeprowadzania procesu porządkowania pakietów można by zminimalizować do zera, gdyby odbiorca miał pewność, że w dowolnej chwili w jego buforze odbiorczym znajdować się będzie co najwyżej jeden pakiet. Wtedy możliwe byłoby przeznaczenie na niewidoczny podpis nawet całego rozmiaru pola *Numer kolejny*, czyli czterech bajtów, a odbiorca odzyskałby prawidłowy następny numer sekwencyjny oraz numer potwierdzenia na podstawie informacji o początkowym numerze sekwencyjnym (ustalanym podczas nawiązywania połączenia, przy założeniu że podczas nawiązywania połączenia pole *Numer kolejny* zawierało prawidłową, oryginalną informację) i długości odebranego pakietu. Następnie odesłałby nadawcy odzyskany numer potwierdzenia w pakiecie potwierdzającym, umożliwiając w ten sposób wysłanie przez nadawcę kolejnego, jednego pakietu. Rozwiązanie takie wymagałoby jednak ustalenia wartości rozmiaru okna odbiorczego na równą wielkości najmniejszego poprawnego pakietu protokołu TCP (bez nagłówka TCP), która wynosi 1 bajt. Spowodowałoby to ogromne nakłady czasowe potrzebne na wielokrotną fragmentację większych pakietów na maszynie nadawcy oraz drastyczne spowolnienie przebiegu samej transmisji lub nawet jej przerwanie. Dla celów praktycznych lepiej pozwolić na odbiór kilku pakietów jednocześnie i ustalić większy rozmiar okna odbiorczego, który utrzyma szybkość transmisji na przyzwoitym poziomie kosztem kilku nadmiarowych bitów informacji. Wtedy trzeba zadbać jeszcze o to, by odbiorca nie otrzymywał od nadawcy żadnych pakietów do czasu przetworzenia wszystkich pakietów, które nadeszły w wyniku przyzwolenia na ich emisję wysłanego w ostatnim pakiecie potwierdzającym (w liczbie bajtów nie większej od wartości pola *Rozmiar okna* w tym pakiecie potwierdzającym). Można to osiągnąć, jeśli odbiorca będzie wysyłał jedno zbiorcze potwierdzenie odbioru dla wszystkich odebranych z ostatniego okna pakietów dopiero po zakończeniu przetwarzania ostatniego z nich (opóźnione potwierdzanie jest dozwolone przez specyfikację protokołu TCP). W ten sposób uzyskuje się „sekwencyjność oknami" i wyeliminowanie możliwości przetwarzania przez odbiorcę pakietów z różnych okien w tym samym czasie. Natomiast bezbłędne uporządkowanie pakietów w ramach jednego okna zapewni się dzięki rezygnacji z traktowania najmniej znaczących bitów pola *Numer kolejny* jako nadmiarowych oraz poprzez zastosowanie arytmetyki „modulo rozmiar okna" podczas operacji na tym polu. W takim przypadku, przy założeniu, że odbiorca będzie każdorazowo informował nadawcę o możliwości przyjęcia nie więcej niż

k bajtów, liczba nadmiarowych bitów informacji w polu *Numer kolejny* jest równa największej liczbie całkowitej nie przekraczającej 32 – $\log_2(k)$. Ponieważ typowe rozmiary okien gwarantujące wydajną transmisję bardzo rzadko przekraczają 65.536 bajtów, w polu *Numer kolejny* możliwe jest ukrycie podpisu o długości nawet dwóch bajtów. Można minimalnie zmniejszyć liczbę nadmiarowych bitów w celu zwiększenia rozmiaru okna i szybkości transmisji, gdyż rozwiązanie wymagające „sekwencyjności oknami" po stronie odbiorcy w sposób znaczący spowalnia szybkość transmisji w stosunku do transmisji z oknem przesuwanym „płynnie", zachodzącej w normalnych warunkach (tj. wtedy, gdy odbiorca pozwala nadawcy na wysyłanie kolejnych pakietów po przetworzeniu prawie każdego pakietu, a nie całego okna odebranych pakietów). Można zrezygnować z wymogu „sekwencyjności oknami" kosztem kilku nadmiarowych bitów informacji oraz zmniejszając rozmiar okna odbiorczego, jednak w zależności od niezawodności sieci i jej zdolności dostarczania do odbiorcy pakietów w oryginalnej kolejności, koszt takiego rozwiązania może okazać się zbyt duży, by można je było stosować w praktyce.

W polu *Numer kolejny* można też ukryć dodatkową informację niezależnie od wielkości okna odbiorczego, jeśli tylko nadawca zapewni odbiorcę, że w trakcie transmisji nie przekroczy pewnego progu liczby wysłanych bajtów. Pole *Numer kolejny* zapewnia niepowtarzalność numeru sekwencyjnego podczas wysyłania danych o wielkości do 4 GB. W praktyce wielkość wysyłanych danych w trakcie jednego połączenia jest dużo mniejsza. Można na przykład przyjąć, że liczba wysłanych bajtów nie przekroczy 128 MB w trakcie połączeń innych niż połączenia FTP i połączenia przy użyciu protokołów zaprojektowanych z myślą o wymianie dużych plików. W ogólności, jeśli nadawca zobowiąże się do ograniczenia liczby wysyłanych danych do k bajtów, na niewidoczny podpis można przeznaczyć 32 – $\log_2(k)$ najbardziej znaczących bitów pola *Numer kolejny*. Prawdopodobieństwo kolizji numerów sekwencyjnych jest w dalszym ciągu bardzo niskie nawet po przekroczeniu ustalonego progu liczby wysłanych bajtów, jeśli tylko bufory odbiorcze nie osiągają rozmiarów bliskich wielkości progu i transmisja nie odbywa się bardzo szybko.

W każdym przypadku modyfikacji pola *Numer kolejny* w pakiecie wychodzącym należy mieć na uwadze, że projektanci protokołu TCP nie bez powodu ustalili tak duży zakres wartości, jakie może przyjmować to pole. Chodzi nie tyle o zapobieganie kolizji numerów sekwencyjnych w buforach odbiorczych o dużych rozmiarach podczas bardzo szybkich transmisji (chociaż w tej chwili protokół wspiera już informowanie nadawcy o oknach wielkości nawet 1GB), co o duży rozrzut losowo wybieranych początkowych numerów sekwencyjnych. Duży rozrzut początkowych numerów sekwencyjnych pozwala na przykład odróżnić odbiorcy krążące jeszcze po sieci, niepotrzebnie zduplikowane przez nadawcę pakiety z poprzedniego, zamkniętego już połączenia od pakietów należących do nowego, właśnie otwartego połączenia z tym samym nadawcą. Nieprzewidywalne początkowe numery sekwencyjne zapobiegają też skutecznym atakom przejęcia sesji. Należy więc ostrożnie zmieniać znaczenie bitów tego pola, by w dalszym ciągu transmisja pozostała odporna na błędy.

Próbę ukrycia dodatkowych danych w polu *Numer kolejny* mogą skutecznie udaremnić występujące na brzegu sieci nadawcy lub odbiorcy zapory ogniowe, które mają możliwość śledzenia nawiązanych połączeń. Mogą one odmówić przekazania pakietu widząc duże rozbieżności pomiędzy numerami sekwencyjnymi, które w rzeczywistości podczas całej transmisji powinny być ciągłe.

2.5. POLE *NUMER POTWIERDZAJĄCY*

Wartość pola *Numer potwierdzający* jest potwierdzeniem dla odbiorcy pomyślnego odebrania przez nadawcę wszystkich pakietów zawierających bajty o numerach sekwencyjnych od niej mniejszych. Pozwala to odbiorcy na zwolnienie pamięci, w której przechowuje dane wysłanych pakietów na wypadek ich retransmisji. Ponowna transmisja pakietu ma miejsce wtedy, gdy przez ustalony czas od jego ostatniego wysłania nie nadejdzie potwierdzenie jego odbioru. Specyfikacja protokołu TCP nie narzuca konieczności potwierdzania odbioru każdego pakietu, a nawet, w związku z dużą niezawodnością sieci, zachęca do potwierdzania odbioru całej sekwencji pakietów jednym pakietem potwierdzającym. W sieciach o pewnej niezawodności działania niższych warstw, tj. warstwy fizycznej, łącza danych oraz sieciowej, można całkowicie zrezygnować z mechanizmu potwierdzania odbioru pakietów, zakładając że pakiet na pewno dojdzie do odbiorcy i każdy pakiet usuwać z bufora retransmisji zaraz po jego wysłaniu. Wtedy na przechowanie dodatkowej informacji można wykorzystać całą przestrzeń udostępnianą przez pole *Numer potwierdzający*, czyli 4 bajty. Ponieważ nigdy nie można przewidzieć, jaką niezawodnością wykaże się sieć podczas rozpoczynanej transmisji, pole *Numer potwierdzający* może mieścić ukryte dane tylko, jeśli odbiorca będzie w stanie odzyskać informację, potwierdzeniem odebrania którego pakietu jest ten numer. Podobnie, jak w przypadku pola *Numer kolejny*, w celu ukrycia informacji można wykorzystać kilka najbardziej znaczących bitów pola. Ich ilość może jednak zostać znacznie ograniczona przez konieczność zachowania pewności, czy potwierdzany pakiet znajduje się aktualnie w buforze retransmisji (i należy go z niego usunąć), czy też jego odebranie zostało już potwierdzone wcześniej odebranym pakietem potwierdzającym i pakiet ten został już usunięty z bufora retransmisji. Druga możliwość może zajść w przypadku, gdy pierwsze potwierdzenie odbioru dotarło do odbiorcy zbyt późno i nim przybyło, wysłał on pakiet do nadawcy ponownie, akceptując jednocześnie pierwsze potwierdzenie i usuwając pakiet z bufora retransmisji. Ta możliwość jest konsekwencją braku wymogu potwierdzania odbioru pakietów potwierdzających przez protokół TCP. Ze względu na konieczność takiego poszerzenia obszaru poszukiwań, maksymalna ilość bitów mogących być bezpiecznie modyfikowanymi w polu *Numer potwierdzający* jest trudna do określenia. Zależy ona bowiem od tego, jak szybko odbiorca może wysyłać pakiety oraz jak szybko otrzymuje potwierdzenia o ich dostarczeniu. Ma na to wpływ szereg czynników, takich jak:

a) rozmiar okna odbiorczego nadawcy,
b) szybkość transmisji w sieci w kierunku odbiorca–nadawca,
c) szybkość przetwarzania pakietów odebranych przez nadawcę i wysyłania przez niego potwierdzeń,
d) szybkość transmisji w sieci w kierunku nadawca–odbiorca.

Nadawca może manipulować wartościami tylko pierwszego z tych parametrów. Na pozostałe nie mają wpływu ani odbiorca, ani nadawca i dlatego są one źródłem trudności w oszacowaniu nadmiarowej liczby bitów. Ponadto nadawca, nawet nieznacznie minimalizując rozmiar okna odbiorczego, może doprowadzić do zmniejszenia szybkości połączenia do nieakceptowalnej wartości. W celu ukrycia dodatkowej informacji w polu *Numer potwierdzający* można też wykorzystać fakt, że każdy pakiet wychodzący zawierający dane również posiada prawidłowo wypełnione to pole. Często powoduje to zdublowanie potwierdzenia, bowiem otrzymywane pakiety są zawsze potwierdzane również zwykłym pakietem potwierdzającym. Zatem w przypadku wysyłania własnych danych na ukryty podpis w pakietach wychodzących można przeznaczyć 4 bajty. Podpisywanie takie ma sens, gdy nadawca jest stroną, która wysyła dużo danych, np. jest serwerem FTP. Taki sposób podpisywania nie obejmuje pakietów potwierdzających. W polu *Numer potwierdzający* (podobnie jak w polu *Numer kolejny*) można też ukryć dodatkową informację, jeśli tylko odbiorca zapewni nadawcę, że w trakcie transmisji nie przekroczy pewnego progu liczby wysłanych bajtów. Jeśli odbiorca zobowiąże się do ograniczenia ilości wysyłanych danych do k bajtów, nadawca na niewidoczny podpis może przeznaczyć $32 - \log_2(k)$ najbardziej znaczących bitów pola *Numer potwierdzający*. Prawdopodobieństwo kolizji numerów potwierdzenia jest w dalszym ciągu bardzo niskie nawet po przekroczeniu ustalonego progu liczby wysłanych bajtów, jeśli tylko bufory odbiorcze nie osiągają rozmiarów bliskich wielkości progu i transmisja nie odbywa się bardzo szybko. W próbie ukrycia dodatkowych danych w polu *Numer potwierdzający* mogą przeszkodzić występujące na brzegu sieci nadawcy lub odbiorcy zapory ogniowe, które mają możliwość śledzenia nawiązanych połączeń. Mogą one odmówić przekazania pakietu widząc duże rozbieżności między odbieranymi numerami sekwencyjnymi a wysyłanymi numerami potwierdzającymi, które w rzeczywistości podczas całej transmisji powinny być ciągłe.

2.6. POLE *PRZESUNIĘCIE DANYCH*

Wartość pola *Przesunięcie danych* oznacza długość nagłówka TCP w słowach czterobajtowych. Minimalna dozwolona wartość tego pola wynosi 5; obecność pola *Opcje* w nagłówku objawia się wartością większą od 5. Pole może być nośnikiem dodatkowej informacji, przy założeniu, że nadawca ograniczy zbiór wartości faktycznych długości wysyłanych nagłówków do zbioru o liczności mniejszej niż 16 i uzgodni z odbiorcą zmienione znaczenie pola. Jeśli nadawca ograniczy liczność tego zbioru do k, nośnikiem podpisu może być liczba bitów równa największej liczbie całkowitej

nie przekraczającej $4 - \log_2(k)$. Biorąc pod uwagę, że pole *Opcje* pojawia się w nagłówku TCP niezwykle rzadko, jeśli tylko nadawca poinformuje odbiorcę, że będzie zawsze wysyłał nagłówki TCP o długości 20 bajtów, na umieszczenie podpisu możliwe jest przeznaczenie całej długości pola *Przesunięcie danych*. Danych przesyłanych w tym polu nie można jednak traktować jako niewidocznych. W rzeczywistości długość nagłówka TCP bardzo rzadko jest inna niż 20 bajtów, dlatego podsłuchujący może nabrać podejrzeń co do obecności niewidocznego podpisu obserwując wartości pola różne od 5. Również wystąpienie zapór ogniowych na brzegu sieci nadawcy bądź odbiorcy może uniemożliwić stosowanie tego typu niewidocznego podpisu. W przypadku wartości pola różnej od 5, zapory ogniowe mogą analizować pakiet pod kątem faktycznej obecności poprawnie sformatowanych opcji w nagłówku i odrzucać pakiety o niepoprawnej zawartości.

2.7. POLE *MIEJSCE ZAREZERWOWANE*

Pole *Miejsce zarezerwowane* to sześciobitowe pole o niezdefiniowanej nazwie i nieprzypisanym znaczeniu – zarezerwowane do wykorzystania w przyszłości, gdy zajdzie potrzeba rozszerzenia funkcjonalności protokołu TCP. Specyfikacja protokołu wymaga, by było one wypełnione zerami. W polu tym można umieścić dodatkową informację, ale nie można traktować jej jako niedostrzegalną dla podsłuchującego, gdyż podsłuchujący może nabrać podejrzeń co do istnienia ukrytego podpisu, obserwując wartość tego pola różną od zera. Również zapory sieciowe mogą nie przekazywać pakietów z niezgodną ze specyfikacją wartością pola *Miejsce zarezerwowane*.

2.8. POLE *ZNACZNIKI*

Pole *Znaczniki* zawiera flagi sterujące stanem połączenia oraz flagi wskazujące na obecność niezerowych wartości w innych polach nagłówka. Chcąc wykorzystać pole

Tabela 1. Niesprzeczne kombinacje wartości flag w nagłówku protokołu TCP

URG	ACK	PSH	RST	SYN	FIN
0	0	0	0	1	0
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	0	0	1
0	1	1	0	0	1
1	1	1	0	0	1
0	1	1	0	0	0
1	1	1	0	0	0
1	1	0	0	0	0
0	1	0	0	0	0

Znaczniki do przesłania nadmiarowej informacji należy pamiętać, że zapory ogniowe obecne na trasie połączenia mogą odmówić przekazania pakietu zawierającego niedozwoloną kombinację wartości flag (najczęściej kombinację wprowadzającą sprzeczność). W tabeli 1 zestawiono kombinacje wartości flag traktowanych jako poprawne i wystarczające do poinformowania odbiorcy o zamiarach nadawcy.

2.8.1. FLAGA *WSKAŹNIK NA PILNE DANE OBECNY*

Ustawiona flaga *Wskaźnik na pilne dane obecny* oznacza, że pakiet zawiera pilne dane, a ich koniec wskazuje wartość pola *Wskaźnik na pilne dane*. Flaga ta może być nośnikiem nadmiarowej informacji, ponieważ bardzo rzadko zdarza się, by aplikacja nadawcy wymagała potraktowania przez odbiorcę jakichkolwiek danych jako pilne. Z tego samego powodu umieszczona nadmiarowa informacja nie może być traktowana jako niedostrzegalna dla podsłuchującego, ponieważ może on nabrać podejrzeń co do istnienia niewidocznego podpisu obserwując w tym polu nietypową wartość 1.

2.8.2. FLAGA *NUMER POTWIERDZAJĄCY OBECNY*

Ustawiona flaga *Numer potwierdzający obecny* oznacza, że nagłówek pakietu zawiera numer potwierdzenia, a jego wartość wskazuje na potwierdzenie odebrania wszystkich bajtów połączenia o numerach od niej mniejszych. Flaga ta może być nośnikiem nadmiarowej informacji, ponieważ tylko w momencie nawiązywania połączenia flaga ta nie jest ustawiona. Z tego samego powodu umieszczona nadmiarowa informacja nie może być traktowana jako niedostrzegalna dla podsłuchującego, ponieważ może on nabrać podejrzeń co do istnienia niewidocznego podpisu obserwując w tym polu nietypową wartość 0.

2.8.3. FLAGA *PRZEKAŻ DANE APLIKACJI*

Ustawiona flaga *Przekaż dane aplikacji* wskazuje na konieczność obudzenia procesu odbiorczego warstwy aplikacji na skutek zgromadzenia przez moduł odbiorczy protokołu TCP wszystkich danych, które chciał przesłać nadawca w bieżącej fazie wymiany informacji. Ponieważ aplikacja odbiorcza powinna w poszczególnych fazach połączenia odbierać dane w takiej samej ilości, w jakiej wysłała do niej dane aplikacja nadawcza, nie można umieścić we fladze *Przekaż dane aplikacji* żadnej dodatkowej informacji, gdyż mogłoby to skutkować w utracie danych.

2.8.4. FLAGA *PRZERYWAM POŁĄCZENIE*

Ustawiona flaga *Przerywam połączenie* oznacza, że nadawca natychmiast przerywa połączenie i nie będzie już odbierał od odbiorcy żadnych pakietów należących do zakończonego właśnie połączenia. Przerwanie połączenia następuje najczęściej podczas jego nawiązywania (gdy port odbiorczy jest zamknięty) lub podczas nawiązanego połączenia na skutek poważnego błędu po stronie odbiorcy (jednakże następuje to bardzo rzadko). Zatem w trakcie trwania połączenia flaga ta może być nośnikiem nadmiarowej informacji, jednak nie może być ona traktowana jako niedostrzegalna dla podsłuchującego, ponieważ może on nabrać podejrzeń co do istnienia niewidocznego podpisu obserwując w tym polu w trakcie trwania połączenia nietypową wartość 1.

2.8.5. FLAGA *SYNCHRONIZUJ NUMERY SEKWENCJI*

Ustawiona flaga *Synchronizuj numery sekwencji* oznacza, że pakiet jest pakietem synchronizującym rozpoczynającym połączenie. Flaga ta może być nośnikiem nadmiarowej informacji, ponieważ jest ona ustawiona tylko w momencie nawiązywania połączenia, a nawiązanie połączenia można również zasygnalizować wyzerowaniem pola *Numer kolejny* bądź flagi *Numer potwierdzający obecny*. Umieszczona nadmiarowa informacja nie może być traktowana jako niedostrzegalna dla podsłuchującego, ponieważ może on nabrać podejrzeń co do istnienia niewidocznego podpisu obserwując w tym polu w trakcie trwania połączenia nietypową wartość 1.

2.8.6. FLAGA *ŻĄDANIE ZAKOŃCZENIA POŁĄCZENIA*

Ustawiona flaga *Żądanie zakończenia połączenia* wskazuje na chęć zakończenia połączenia przez nadawcę. Ponieważ trudno przewidzieć, kiedy w rzeczywistości aplikacja nadawcza będzie chciała zakończyć połączenie, nie powinno stosować się tej flagi jako miejsca przechowującego dodatkowe dane, gdyż grozi to przedwczesnym rozłączeniem z odbiorcą.

2.9. POLE *ROZMIAR OKNA*

Wartość pola *Rozmiar okna* dostarcza odbiorcy informacji, jak wiele bajtów jest w stanie aktualnie przyjąć nadawca. Pole to zostało wprowadzone w celu możliwości zoptymalizowania szybkości połączenia, co uzyskuje się dzięki dostosowywaniu wartości tego pola w trakcie połączenia do faktycznej szybkości transmisji w sieci oraz do zasobów pamięciowych i obliczeniowych obydwóch stron połączenia. Skutkuje to dopasowaniem tempa wysyłania pakietów przez odbiorcę do szybkości ich odbierania i przetwarzania przez nadawcę.

Manipulowanie wartością pola *Rozmiar okna* tak, by mogło ono przechować dowolną informację, bardzo rzadko prowadzi do całkowitego przerwania połączenia. Wartość tego pola większa od aktualnego rozmiaru okna odbiorczego nadawcy spowoduje zmniejszenie szybkości połączenia na skutek konieczności retransmisji przez odbiorcę pakietów, które nie mogły zostać przyjęte przez nadawcę. Wartość mniejsza od aktualnego rozmiaru okna odbiorczego spowoduje zmniejszenie szybkości połączenia na skutek niewykorzystania całej szerokości pasma transmisji. Modyfikacja wartości pola *Rozmiar okna* może spowodować przerwanie połączenia tylko, gdy odbiorca chce wysłać dane, ale przez długi czas jest mu ogłaszana zerowa lub inna zbyt mała wartość rozmiaru okna odbiorczego nadawcy. Dla całkowitej pewności wyeliminowania takiej sytuacji, nadawca może przeznaczyć na podpis 15 bitów pola *Rozmiar okna*. Podpis taki można traktować jako niewidoczny, ponieważ ani podsłuchujący, ani zapory ogniowe nie są w stanie przewidzieć faktycznej ilości bajtów, jaką jest w stanie odebrać nadawca, a co za tym idzie, nie są w stanie wykryć obecności ukrytych danych w polu *Rozmiar okna*.

2.10. POLE *SUMA KONTROLNA*

Wartość pola *Suma kontrolna* równa jest sumie kontrolnej bitów całego pakietu TCP (zarówno nagłówka, jak i właściwych danych), co umożliwia kontrolę poprawności przesłanych danych. W przypadku stwierdzenia przez odbiorcę niezgodności wyliczonej przez niego sumy kontrolnej odebranego pakietu z wartością przechowywaną w tym polu, odbiorca nie wysyła potwierdzenia odebrania pakietu, co po pewnym ustalonym czasie skutkuje ponownym jego nadaniem przez nadawcę.

Pole *Suma kontrolna* nie powinno być nośnikiem dowolnej ukrytej informacji, jeśli nadawca i odbiorca chcą, by transmisja w dalszym ciągu była odporna na błędy. Jednak mogą oni zdecydować się na przeznaczenie wszystkich 16 bitów pola na ukrycie dowolnych danych w przypadku wysokiej niezawodności sieci, w której zachodzi transmisja. W przeciwnym wypadku pole *Suma kontrolna* powinno być nośnikiem takiego rodzaju ukrytej informacji, który zapewniałby integralność przesyłanych danych na poziomie co najmniej takim, jak obecnie stosowany algorytm sumy kontrolnej. Dlatego pole to może być nośnikiem podpisu, a także dodatkowo dowolnych innych danych, jeśli rozmiar podpisu będzie mniejszy od rozmiaru pola *Suma kontrolna*. Danych przesyłanych w tym polu nie można jednak traktować jako niewidocznych. Algorytm wyliczania sumy kontrolnej jest prosty i powszechnie dostępny, dlatego podsłuchujący może nabrać podejrzeń co do obecności niewidocznego podpisu obserwując stale wartości pola *Suma kontrolna* dalece odbiegające od poprawnych oraz widząc odpowiedzi odbiorcy na pakiety z niepoprawną sumą kontrolną. Wystąpienie zapór ogniowych na brzegu sieci nadawcy bądź odbiorcy może uniemożliwić stosowanie tego typu niewidocznego podpisu. Zapory ogniowe mogą odrzucać pakiety o niepoprawnej sumie kontrolnej.

2.11. POLE *WSKAŹNIK NA PILNE DANE*

Wartość pola *Wskaźnik na pilne dane* wyznacza koniec danych, które powinny być potraktowane przez odbiorcę jako „pilne", tj. obsłużone szybciej niż pozostałe dane pakietu. W przypadku ustawionej flagi *Wskaźnik na pilne dane obecny*, „pilne" dane rozpoczynają się zaraz za nagłówkiem TCP, a pierwszy bajt, który już nie powinien być traktowany jako „pilny", wskazywany jest przez wartość pola *Wskaźnik na pilne dane*. Pomysł stosowania „pilnych" danych został zarzucony, wobec czego w praktyce flaga *Wskaźnik na pilne dane obecny* nigdy nie jest ustawiona, a pole *Wskaźnik na pilne dane* jest zawsze wyzerowane. Dlatego pole to może być nośnikiem dodatkowej informacji, ale podobnie jak w przypadku poprzedniej flagi, informacja ta nie może być traktowana jako niedostrzegalna, ponieważ niezerowa wartość tego pola może wzbudzić podejrzenia podsłuchującego co do obecności niewidocznego podpisu.

2.12. POLE *OPCJE*

Pole *Opcje* jest opcjonalnym polem nagłówka dodanym głównie w celu umożliwienia nadawcy poinformowania odbiorcy o obsłudze rozszerzeń protokołu, zazwyczaj odnoszących się do zwiększenia szybkości transmisji. Przykładowo, dzięki ustawieniu odpowiednich wartości w tym polu, nadawca może poinformować o obsługiwaniu przez niego dużego okna odbiorczego (do 1 GB) lub o możliwości zbiorczego potwierdzania kilku otrzymanych pakietów jednym pakietem potwierdzającym.

Pole *Opcje* nie jest dobrym nośnikiem ukrytej informacji, głównie ze względu na swoją opcjonalność i bardzo rzadkie występowanie w praktyce. W związku z tym pojawienie się tego pola w nagłówkach pakietów, 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. Obecność tego pola można łatwo wywnioskować z niestandardowej, większej od 5 wartości pola *Przesunięcie danych* w nagłówku. Należy też mieć na uwadze, że zapory ogniowe mogą nie przekazywać pakietów protokołu TCP z nagłówkami zawierającymi pole *Opcje*.

2.13. POLE *WYPEŁNIENIE*

Pole *Wypełnienie* jest obecne tylko w przypadku użycia w nagłówku TCP opcji. Służy do wypełnienia pozostałej części nagłówka tak, by jego długość była wielo-

krotnością 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 ogniowe 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.

2.14. PODSUMOWANIE ANALIZY PROTOKOŁU TCP

W ramach podsumowania w tabeli 2 zestawiono liczbę nadmiarowych bitów informacji udostępnianych do konstrukcji niedostrzegalnego kanału przez poszczególne pola nagłówka TCP. W tabeli 2 zamieszczone zostały warunki, konieczne do spełnienia przy wykorzystaniu poszczególnych pól.

Tabela 2. Podsumowanie liczby nadmiarowych bitów udostępnianych na niewidoczny podpis przez poszczególne pola nagłówka protokołu TCP

Nazwa pola	Długość pola w bitach	Maksymalna liczba nadmiarowych bitów w jednym pakiecie[2)]	Minimalna liczba nadmiarowych bitów w jednym pakiecie[3)]
1	2	3	4
Port źródłowy	16	$16 - \log_2(k)$ [4)] ≈ 16	0
Port docelowy	16	$16 - \log_2(k)$ [5)] ≈ 16	0
Numer kolejny	32	$32 - \log_2(k)$ [6)] ≈ 16 $32 - \log_2(l)$ [8)] ≈ 4	0
Numer potwierdzający	32	$32 - \log_2(k)$ [7)] ≈ 10 $32 - \log_2(l)$ [8)] ≈ 4	0
Przesunięcie danych	4	– [1)]	– [1)]
Miejsce zarezerwowane	6	– [1)]	– [1)]
Wskaźnik na pilne dane obecny	1	– [1)]	– [1)]
Numer potwierdzający obecny	1	– [1)]	– [1)]
Przekaż dane aplikacji	1	–	–
Przerywam połączenie	1	– [1)]	– [1)]
Synchronizuj numery sekwencji	1	– [1)]	– [1)]
Żądanie zakończenia połączenia	1	–	–
Rozmiar okna	16	15	15
Suma kontrolna	16	– [1)]	– [1)]

1	2	3	4
Wskaźnik na pilne dane	16	– [1]	– [1]
Opcje	0–320	– [1]	– [1]
Wypełnienie	0–24	– [1]	– [1]
Suma[9]	–	**≈ 73**	**≈ 15**

[1] Pole umożliwia przechowanie nadmiarowej informacji, ale nie w sposób niewidoczny dla podsłuchującego.
[2] W przypadku braku występowania na trasie połączenia zapór ogniowych lub obecności tylko zapór nie śledzących poprawności przebiegu połączenia.
[3] W przypadku występowania na trasie połączenia zapór ogniowych śledzących poprawność przebiegu połączenia.
[4] Gdy odbiorca będzie traktował pakiety z *k* portów źródłowych jako należące do tego samego połączenia.
[5] Gdy odbiorca będzie traktował pakiety odebrane z *k* swoich portów jako należące do tego samego połączenia.
[6] Gdy istnieje pewność, że w buforze odbiorczym odbiorcy nie dojdzie do kolizji numerów sekwencyjnych pakietów oddalonych od siebie o *k* bajtów.
[7] Gdy istnieje pewność, że nie dojdzie do kolizji numerów potwierdzeń pakietów oddalonych od siebie o *k* bajtów.
[8] Gdy istnieje pewność, że podczas transmisji ilość wysyłanych bajtów nie przekroczy *l*
[9] Podczas wyznaczania sumy ilości nadmiarowych bitów, dla pól *Numer kolejny* i *Numer potwierdzający* przyjęto wariant najbardziej prawdopodobny, tj. możliwość pierwszą.

Łatwo zauważyć, że liczba bitów mogących być nośnikiem niewidocznego podpisu silnie zależy od obecności na trasie transmisji zapór ogniowych. W przypadku wystąpienia choć jednej zapory potrafiącej śledzić stan połączenia, na ukrycie dodatkowej informacji można przeznaczyć tylko pole *Rozmiar okna*. Ponieważ dokładne śledzenie stanu połączenia jest techniką zaawansowaną, często do ukrycia informacji można też wykorzystać pola *Numer kolejny* oraz *Numer potwierdzający*. Wykorzystanie pól *Port docelowy* oraz *Port źródłowy* kłóci się z koncepcjami leżącymi u podstaw protokołu TCP, ale jest możliwe.

Mała liczba pozostających do dyspozycji nadmiarowych bitów informacji w przypadku wystąpienia na trasie zapory ogniowej wynika z łatwości, z jaką może ona wykrywać niepoprawne wartości pól w nagłówku TCP. Wartości te są bowiem z reguły przewidywalne, gdyż protokół TCP zapewnia niezawodną, połączeniową transmisję danych i podczas trwania całego połączenia wartości pól nagłówków muszą spełniać pewne ograniczenia. Taka sytuacja powoduje, że nagłówek TCP dostarcza zbyt mało nadmiarowych bitów, by mogły posłużyć do przechowania silnego podpisu. W celu osiągnięcia silnego podpisu należy zastosować również nadmiarowe bity nagłówka protokołu IP [LIBER 2006].

Reasumując, analiza wykazała, że największy poziom niewidoczności ukrytej informacji zapewniają pola stworzone z myślą o zapobieganiu sytuacjom źle wpływającym na kondycję sieci bądź powodującym nieoptymalne wykorzystanie zasobów (*Licznik czasu*, *Rozmiar okna*) lub pola jednoznacznie identyfikujące ramkę lub pakiet w sieci w pewnym przedziale czasu (*Identyfikator*, *Numer kolejny*).

3. NIEDOSTRZEGALNE KANAŁY TCP – BADANIA EKSPERYMENTALNE

W ramach pracy przeprowadzono badania eksperymentalne w zakresie konstrukcji niedostrzegalnych kanałów informacyjnych i ich wykorzystania do generacji podpisów cyfrowych. Eksperymenty zostały wykonane w specjalnie do tego przygotowanej dystrybucji systemu operacyjnego Linux. W systemie operacyjnym zmodyfikowano oprogramowanie jądra tak aby można było swobodnie modelować różne przypadki realizacji niedostrzegalnych kanałów. W tabeli 3 zaznaczono pola nagłówka wykorzystywane do badań. Każdy eksperyment 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 oraz FTP.

Tabela 3. Pola nagłówka TCP wykorzystane do przesłania dodatkowej informacji. Kolorem szarym zaznaczono wszystkie pola, które mogą być wykorzystane do przesłania dodatkowej informacji

Nazwa pola	Długość pola w bitach	Liczba nadmiarowych bitów w jednym pakiecie
Port źródłowy	16	–
Port docelowy	16	–
Numer kolejny	32	–
Numer potwierdzający	32	–
Przesunięcie danych	4	–
Miejsce zarezerwowane	6	–
Wskaźnik na pilne dane obecny	1	–
Numer potwierdzający obecny	1	–
Przekaż dane aplikacji	1	–
Przerywam połączenie	1	–
Synchronizuj numery sekwencji	1	–
Żądanie zakończenia połączenia	1	–
Rozmiar okna	16	15
Suma kontrolna	16	–
Wskaźnik na pilne dane	16	–
Opcje	0–320	–
Wypełnienie	0–24	–
Suma	–	**15**

W dalszej części opisano przykładowe eksperymenty z wykorzystaniem protokołów HTTP i FTP w układzie sieci opisanym w pracy [LIBER 2006]. W przypadku badań z wykorzystaniem protokołu HTTP, po zalogowaniu się jako użytkownik *root* i poprawnym skonfigurowaniu urządzeń sieciowych nastąpiło wykonanie poleceń:

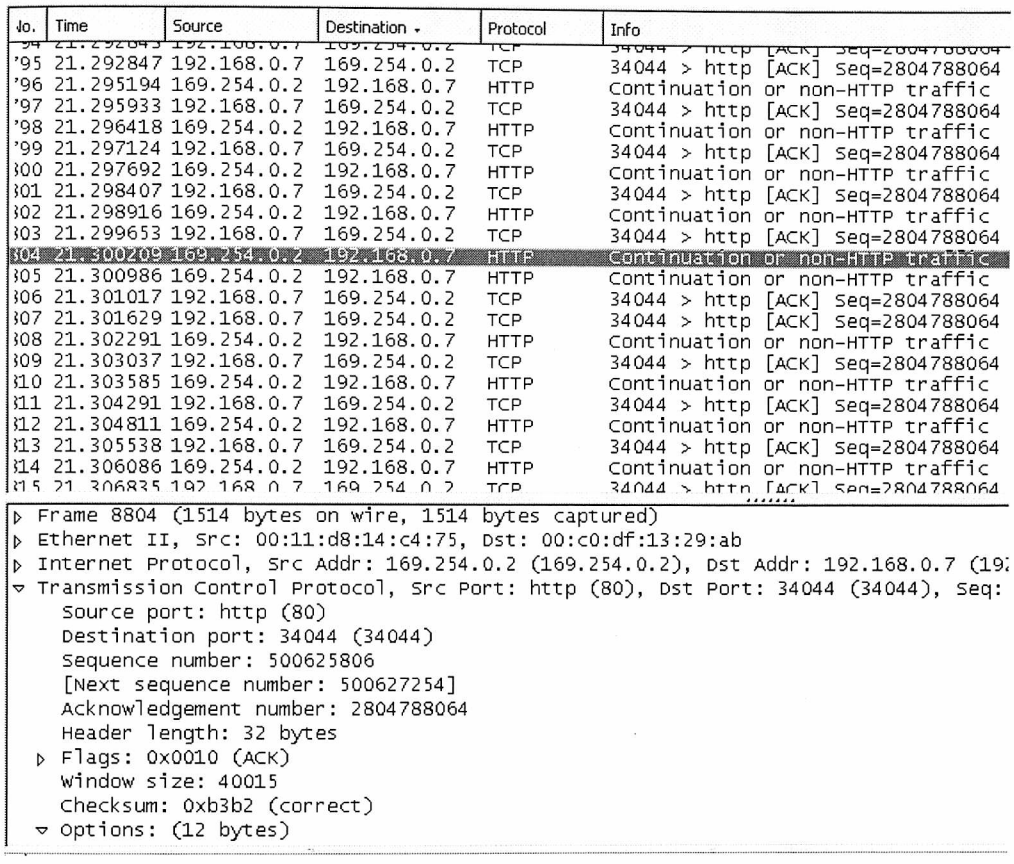
```
(na maszynie o adresie 169.254.0.2)
echo „117483712" > /proc/sys/net/ipv4/ip_inv_sign_rcv_src
(co oznacza 192.168.0.7)
echo „4" > /proc/sys/net/ipv4/ip_inv_sign_rcv_modes
echo „AAAAAAAAAAAAAAAAAAAA" >
/proc/sys/net/ipv4/ip_inv_sign_rcv_key
echo „117483712" > /proc/sys/net/ipv4/ip_inv_sign_snd_dst
(co oznacza 192.168.0.7)
echo „4" > /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 „4" > /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 „4" > /proc/sys/net/ipv4/ip_inv_sign_snd_modes
echo „AAAAAAAAAAAAAAAAAAAA" >
/proc/sys/net/ipv4/ip_inv_sign_snd_key
```

Postępowanie takie zapewniło ruch dwustronnie podpisywany przy wykorzystaniu pola *Rozmiar okna* (15 najmniej znaczących bitów). Aby zaobserwować propagację podpisu cyfrowego przy wykorzystaniu protokołu HTTP na maszynie o adresie 192.168.0.7 wykonano polecenie:

```
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. Zdarzenia zachodzące podczas eksperymentu można obserwować z komputera pośredniczącego w transmisji przy pomocy programu analizatora pakietów sieciowych *Ethereal*. Na rysunku 2 przedstawiono widok okna

roboczego programu *Ethereal* podczas obserwacji transmisji zawierającej podpisane niedostrzegalnie ramki.

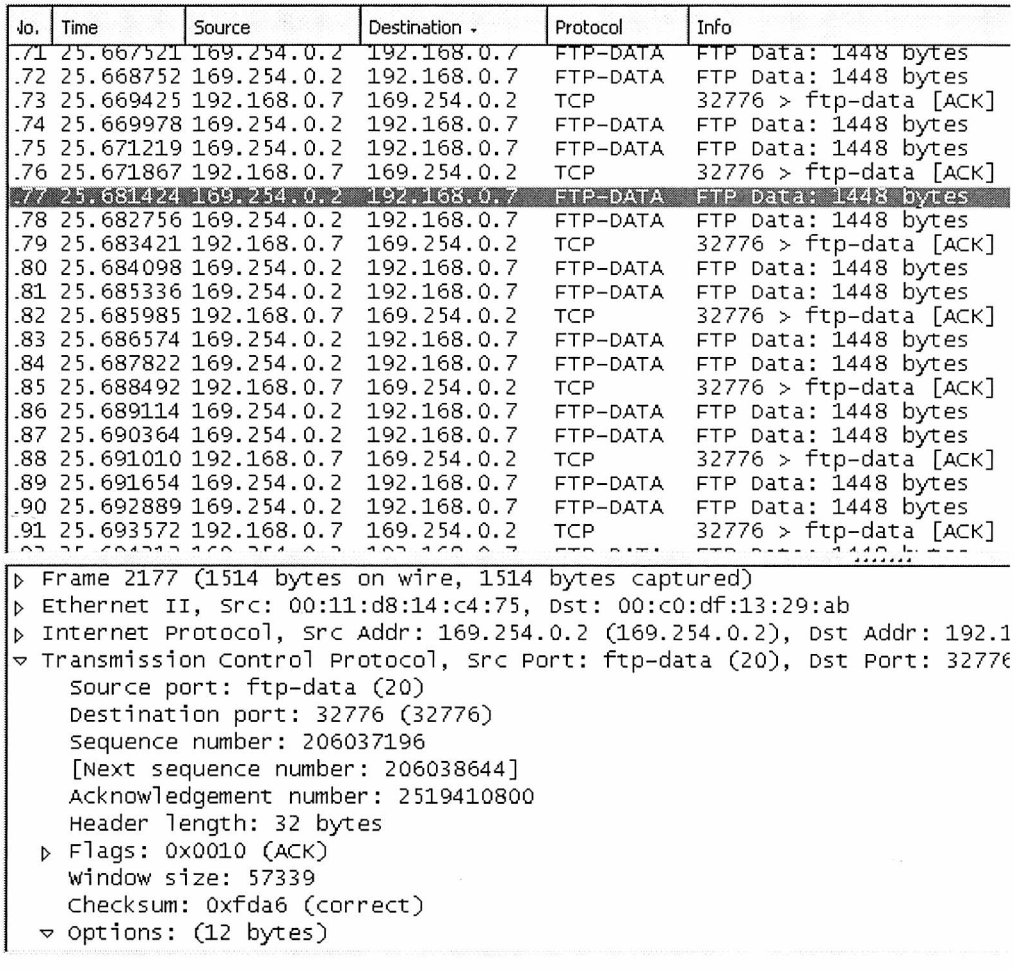


Rys. 2. Widok ekranu roboczego z programu analizatora pakietów sieciowych Ethereal podczas obserwacji podpisanej transmisji HTTP

Eksperymenty z wykorzystaniem protokołu FTP przeprowadzono podobnie do poprzednich przez wywołanie na maszynie o adresie 192.168.0.7 polecenia:

```
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 rejestrując jednocześnie przebieg transmisji przy wykorzystaniu programu *Ethereal*. Na rysunku 3 przedstawiono widok okna roboczego programu podczas obserwacji pola *Rozmiar okna* zależnego od wartości zaszyfrowanego podpisu.



Rys. 3. Widok ekranu z programu analizatora pakietów sieciowych *Ethereal* podczas podpisanej transmisji FTP

4. PODSUMOWANIE

W ramach pracy przeprowadzono analizę możliwości wykorzystania protokołu TCP do konstrukcji niedostrzegalnych kanałów komunikacyjnych mogących mieć zastosowanie w kryptograficznej ochronie danych. Do badań wykonano specjalistyczną wersję systemu LINUX ze zmodyfikowanym jądrem. Stwierdzono, że do konstrukcji niedostrzegalnych kanałów transmisyjnych nadają się pola nagłówka ramki TCP: *Port źródłowy*, *Port docelowy*, *Numer kolejny, Numer potwierdzający*, *Rozmiar okna.* Stwierdzono, 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 tutaj ewentualne wystąpienie na drodze transmisji zaawansowanych zapór ogniowych, których obecność może znacznie ograniczyć ilość dodatkowej informacji możliwej do przesłania.

LITERATURA

[Ashan 2002] 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).

[Handel 1996] Handel T. G., Sandford M. T., *Hiding data in the OSI network model*, Information Hiding, Lecture Notes in Computer Science. Springer-Verlag, 1996.

[Liber 2004] Liber A., Kosmulska-Bochenek E., *Niewidoczne sygnatury symetryczne protokołów komunikacyjnych*. W: *Współczesne problemy sieci komputerowych, zastosowanie i bezpieczeństwo*, Wydawnictwa Naukowo-Techniczne, Warszawa 2004.

[Liber 2006] Liber A., Dybowski J., *Konstrukcja niedostrzegalnych kanałów komunikacyjnych z wykorzystaniem protokołu IP i ich zastosowanie do podpisywania danych*. W: *Bazy danych. Prace naukowe Instytutu Informatyki Stosowanej*, Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław 2006.

[Postel 1981] Postel J., *Internet Protocol*, IETF, 1981.

[Rowland 1997] 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 2003] 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).

POSSIBILITIES OF THE USAGE OF THE TCP PROTOCOL TO THE CONSTRUCTION OF IMPERCEPTIBLE BROADCASTING CHANNELS FOR CRYPTOGRAPHICAL AIM

In frames of the work one introduced results of research of authors in the range of the construction of imperceptible inquiry embedded channels in standardized communication protocols. Such channels can be used to producing of additional independent transmission channels and to the realization of cryptographic protocol. In frames of the work one introduced the detailed analysis of the TCP protocol at an angle of the construction of imperceptible channels. One explored imperceptible embedded channels on the TCP protocol at an angle of placing in them of digital signatures. In frames of passed research one executed the distribution of the system LINUX with modified core. In the core {nucleus} one implemented imperceptible leaning layers on the official record TCP. One explored hidden layers containing {contracting} digital signatures given. The work is the continuation and the enlargement of earlier research of authors [Liber 2004], [Liber 2006].

Prace Naukowe Instytutu Informatyki
Politechniki Wrocławskiej

Zeszyt 10 2009

zabezpieczenia programów,
implementacja zabezpieczeń na platformie .NET,
implementacja zabezpieczeń w języku C#

Arkadiusz LIBER*,
Bartosz KITA

WŁASNOŚCI ZABEZPIECZEŃ OPROGRAMOWANIA PRZED NIELEGALNYM ROZPOWSZECHNIANIEM IMPLEMENTOWANYCH NA PLATFORMIE .NET PRZY WYKORZYSTANIU JĘZYKA C#

W ramach pracy przedstawiono wyniki badań autorów w zakresie konstrukcji zabezpieczeń programów komputerowych przed nielegalnym rozpowszechnianiem. Opisane i zbadane zostały implementacje zabezpieczeń oprogramowania oparte na algorytmach: weryfikacji numeru seryjnego, weryfikacji numeru seryjnego i nazwy, weryfikacji pliku klucza, weryfikacja plikowego licznika uruchomień, weryfikacja rejestrowego licznika uruchomień, weryfikacja daty instalacji w rejestrze systemowym, identyfikacji klucza sprzętowego. Wybór powyższych algorytmów podyktowany był jak najściślejszym porównaniem implementacji podstawowych zabezpieczeń programów w języku C# oraz w językach C, Java, Assembler opisywanych we wcześniejszych pracach autorów [Liber 2006a], [Liber 2006b], [Liber 2006c], [Liber 2006d]. Wszystkie implementacje zostały zrealizowane na platformie .NET w języku C# w postaci samodzielnych aplikacji przygotowanych do szczegółowych badań porównawczych. Ze względu na charakter pracy algorytmy przygotowane zostały w formie maksymalnie uproszczonej, w szczególności zaś do algorytmów nie wprowadzono dodatkowych zabezpieczeń kryptograficznych mogących bardzo znacznie zwiększyć ich siłę. W końcowej części pracy przedstawiono zaproponowany przez autorów algorytm obfuskacyjny wraz z opisem implementacji w języku C# oraz analizą.

1. WPROWADZENIE

Rosnąca popularność platformy Microsoft .NET dostępnej w systemach Microsoft Windows XP oraz Microsoft Windows Vista, powoduje pojawianie się coraz większej ilości aplikacji napisanych pod tę platformę implementowanych między innymi przy

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

wykorzystaniu nowego języka programowania C#. Firma Microsoft tworząc platformę .NET stworzyła wspólne środowisko uruchomieniowe (ang. *Common Language Runtime*) jako podstawę całego systemu .NET Framework. Nowością jest również kompilator JIT (ang. *Just-In-Time*). Programy napisane pod .NET kompilowane są do postaci kodu pośredniego. Podczas uruchomienia programu wymagana jego część jest ponownie kompilowana do postaci kodu maszynowego. Takie podejście skutkuje bezpośrednim dostępem do kodu programu a co za tym idzie powstaje pytanie o możliwości techniczne implementacji skutecznych zabezpieczeń oprogramowania przed nielegalnym rozpowszechnianiem na tej platformie.

Opracowanie skutecznego zabezpieczenia oprogramowania przed nielegalnym rozpowszechnianiem i użytkowaniem nie jest zadaniem łatwym. Powszechna dostępność sprzętu i oprogramowania powoduje, iż nielegalnym zwielokrotnianiem zajmują się nawet osoby z niewielką wiedzą informatyczną. Powszechnie dostępna jest również wiedza o sposobach konstrukcji zabezpieczeń [Cerven 2001], [Kaczor 2004], [Zemanem 2004].

Na proces konstrukcji skutecznego zabezpieczenia składa się teoretyczne opracowanie projektu zabezpieczenia, jego implementacja i testowanie. Nowy rodzaj skutecznych zabezpieczeń nie powinien zawierać słabych elementów ujawnionych w zabezpieczeniach istniejących do tej pory [Liber 2006a].

2. MODELOWE ZABEZPIECZENIA I ICH IMPLEMENTACJA W JĘZYKU C#

W dalszej części pracy opisano modelowe zabezpieczenia oprogramowania zaimplementowane i przebadane na platformie .NET. Opisane zostały implementacje zabezpieczeń oprogramowania oparte na algorytmach: weryfikacji numeru seryjnego, weryfikacji numeru seryjnego i nazwy, weryfikacji pliku klucza, weryfikacja plikowego licznika uruchomień, weryfikacja rejestrowego licznika uruchomień, weryfikacja daty instalacji w rejestrze systemowym, identyfikacji klucza sprzętowego. Wybór algorytmów podyktowany był koniecznością przeprowadzenia badań porównawczych z analogicznymi rozwiązaniami zaimplementowanymi wcześniej w językach: C, Java oraz Assembler.

2.1. ALGORYTM 1 – WERYFIKACJA NUMERU SERYJNEGO

Zabezpieczenie programu przez numer seryjny polega na tym, że użytkownik jest proszony o wprowadzenie dostarczanego przez producenta numeru do programu. Następnie wprowadzony numer porównywany jest w numerem zapisanym w programie. Numer zapisany w programie przechowywany jest w sposób jawny lub dla zwiększe-

nia bezpieczeństwa numer szyfrowany jest funkcją przekształcającą go z postaci jawnej do postaci zaszyfrowanej.

Numer seryjny lub jakąś jego odmianę można bezsprzecznie uznać za jedno z najczęściej stosowanych zabezpieczeń we współczesnych programach. Bardzo często bywa łączone z najrozmaitszymi ograniczeniami w programie, które zostają usunięte po podaniu właściwego numeru seryjnego [Zemanem 2004].

2.1.1. IMPLEMENTACJA ZABEZPIECZENIA

Na potrzeby pracy przygotowano dwa modelowe pogramy napisane w środowisku Microsoft Visual C# 2005 Express Edition. Po uruchomieniu programów użytkownik musi wpisać numer seryjny w celu ich odblokowania. Pierwszy program *Numer seryjny 1* przechowuje numer seryjny w postaci jawnej, natomiast drugi *Numer seryjny 2* przechowuje numer w postaci zaszyfrowanej. Dla uproszczenia do szyfrowania numeru w drugim programie użyto funkcji XOR zamiast zaawansowanego schematu szyfrowania.

Poniżej przedstawiono fragment kodu źródłowego porównujący wprowadzony numer z numerem jawnie zapisanym w programie:

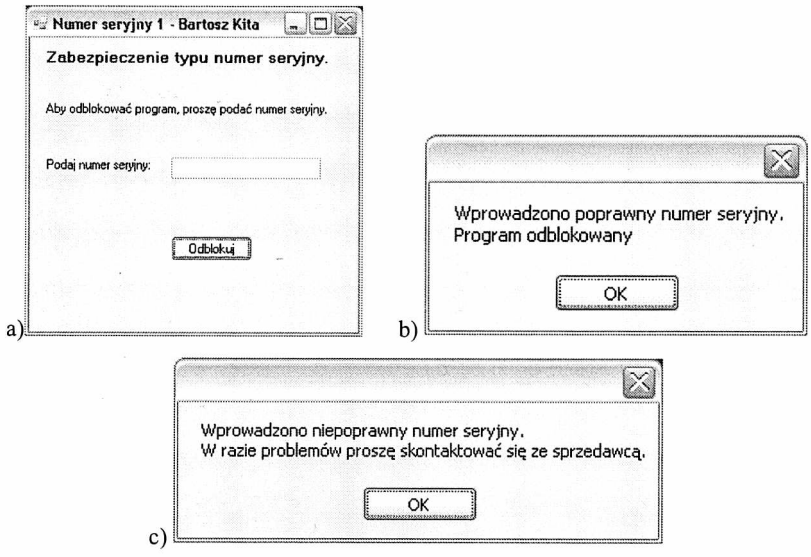
```
private void button1_Click(object sender, EventArgs e)
        {
            if(textBox1.Text == "12345")
                MessageBox.Show("Wprowadzono poprawny nu-
mer seryjny. \nProgram odblokowany");
            else
            MessageBox.Show("Wprowadzono niepoprawny nu-
mer seryjny. \nW razie problemów proszę skontaktować się
ze sprzedawcą.");
        }
```

Poniżej przedstawiono fragment kodu źródłowego porównujący wprowadzony numer z numerem zaszyfrowanym funkcją XOR:

```
private void button1_Click(object sender, EventArgs e)
        {
            string num;
            num = textBox1.Text;
            int a = 55555;
            int b = 59706;
            int suma;
            int num2;
```

```
            num2 = int.Parse(num);
            suma = num2 ^ a;
            if (suma == b)
                MessageBox.Show("Wprowadzono poprawny nu-
mer seryjny. \nProgram odblokowany");
            else
                MessageBox.Show("Wprowadzono niepoprawny
numer seryjny. \nW razie problemów proszę skontaktować
się ze sprzedawcą.");
        }.
```

Na rysunku 1 przedstawiono główne okno programu zabezpieczonego przy użyciu numeru seryjnego. Po poprawnym wprowadzeniu numeru pojawi się okno informacyjne przedstawione na rysunku 1b. W przypadku podania niepoprawnego numeru seryjnego, użytkownik zostanie poinformowany, a w razie problemów poproszony o kontakt ze sprzedawcą – rysunek 1c.



Rys. 1. Widok okna a) programu *Numer seryjny 1*, b) okna programu *Numer seryjny 1* po poprawnym wprowadzeniu numeru, c) okno programu *Numer Seryjny 1* w przypadku niepoprawnego numeru seryjnego

2.1.2. ANALIZA ZABEZPIECZENIA

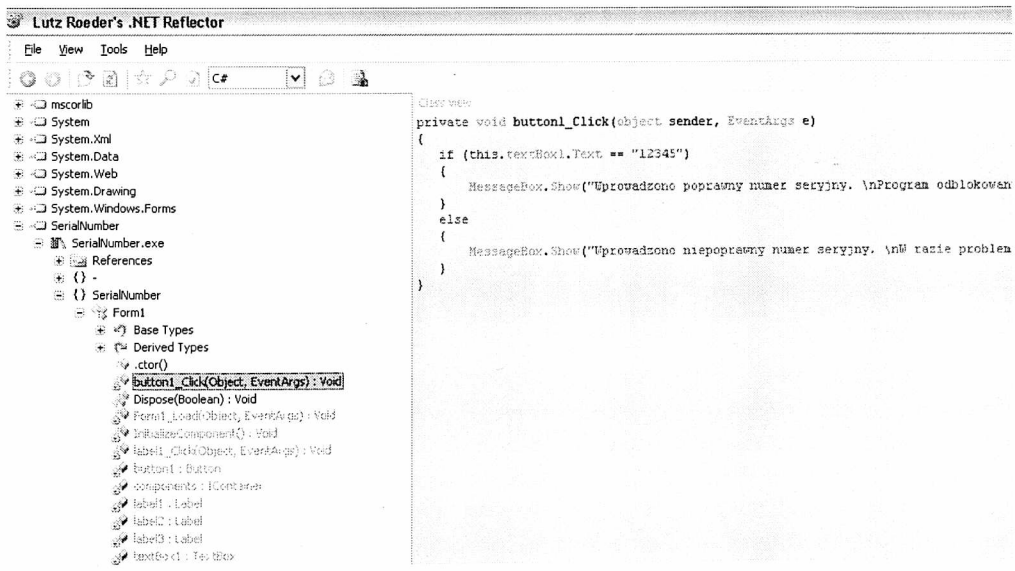
Plik wykonywalny zabezpieczony przy pomocy numeru seryjnego można dokładnie przeanalizować korzystając z programu *Reflector*. Do programu *Reflector* należy dodat-

kowo zainstalować rozszerzenie *Reflector.ClassView.dll*. Po uruchomieniu programu *Reflector.exe* z zakładki *Tools* należy wybrać opcję *Class View*. Otworzy się dodatkowe okno, w którym pokazane będą zaimplementowane klasy. Po otwarciu modelowego programu *Numer seryjny 1* należy wybrać klasę *SerialNumber.Form1.button1_Click(Object, EventArgs): Void*.

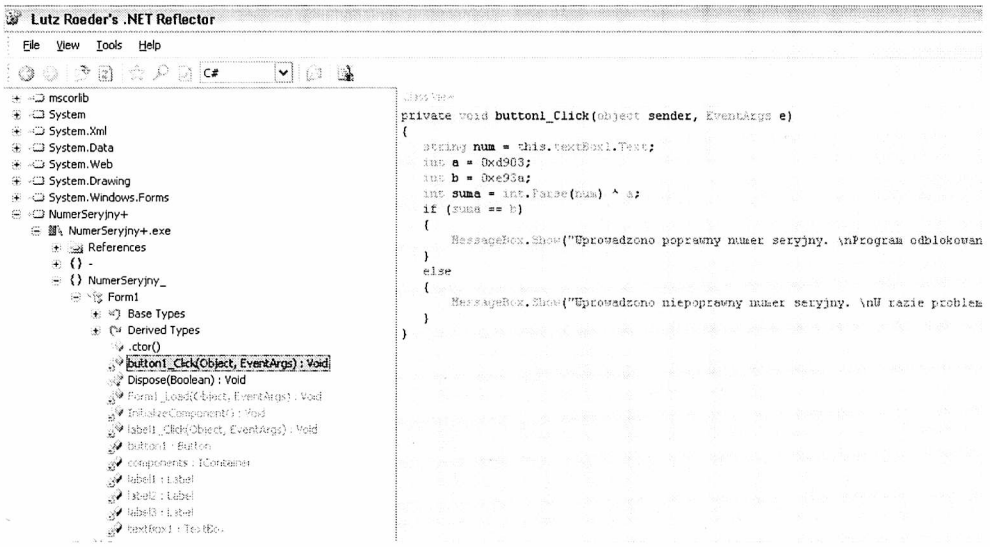
W oknie *Class View* pokaże się klasa sprawdzająca poprawność wprowadzonego numeru seryjnego. W przypadku pierwszego programu (rysunek 2) widać jawny numer seryjny, do którego porównywany jest numer wprowadzony przez użytkownika. Identycznie należy postąpić z drugim programem, w którym pokazana zostanie uproszczona funkcja szyfrująca numer seryjny.

W celu dalszej analizy można skorzystać z programu *32bit Calculator*, w którym wygenerowany zostanie szukany numer seryjny na podstawie informacji zawartych w funkcji pokazanej na rysunku 3. Wprowadzany numer poddany jest działaniu funkcji XOR z liczbą 55555 (0xd903), następnie wynik funkcji porównywany jest z liczbą 59706 (0xe93a). Wprowadzając następujące liczby do kalkulatora otrzymujemy szukany numer seryjny równy 12345.

Analizując zabezpieczenie programu przez numer seryjny można stwierdzić, że zabezpieczenie tego typu nie zapewnia wystarczającego stopnia bezpieczeństwa dla oprogramowania. Ten rodzaj zabezpieczenia dobrze jest stosować w kombinacji z innymi typami zabezpieczeń takimi jak: nazwa i numer seryjny czy numer seryjny generowany on-line na podstawie danych firmy lub danych określonego komputera.



Rys. 2. Okno programu *Reflector*, w którym pokazano porównanie numeru seryjnego wykorzystanego jako zabezpieczenie w modelowym programie *Numer seryjny*



Rys. 3. Okno programu *Reflector*, w którym pokazano porównanie numeru seryjnego zaszyfrowanego funkcją XOR wykorzystanego jako zabezpieczenie w modelowym programie *Numer seryjny 2*

2.2. ALGORYTM 2 – WERYFIKACJA NUMERU SERYJNEGO I NAZWY

Rozszerzeniem zabezpieczenia typu numer seryjny jest dodanie kolejnego parametru wymaganego w procesie rejestracji programu. Zabezpieczenie polega na tym, że użytkownik proszony jest o wpisanie do formularza swojej nazwy oraz numeru seryjnego dostarczanego przez producenta oprogramowania. Numer seryjny może być stały lub automatycznie generowany na podstawie nazwy użytkownika.

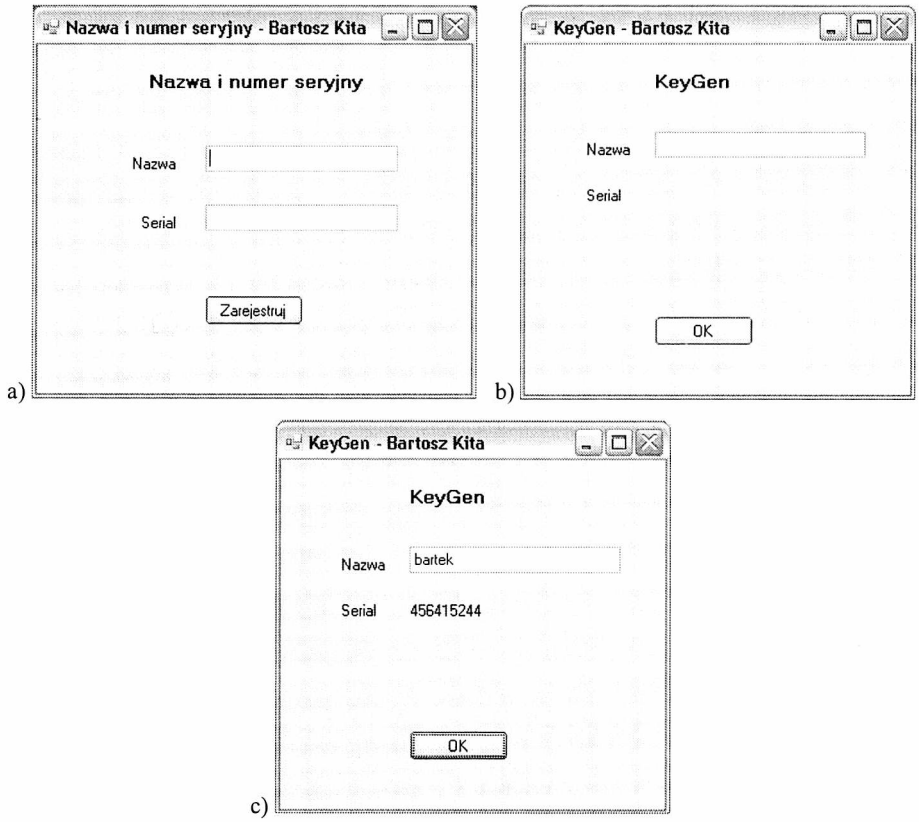
2.2.1. IMPLEMENTACJA ZABEZPIECZENIA

Na potrzeby pracy przygotowano modelowy pogram napisany w środowisku Microsoft Visual C# 2005 Express Edition. W celu zarejestrowania programu, użytkownik proszony jest o wypełnienie formularza składającego się z dwóch pól (nazwa i numer seryjny). W celu zwiększenia poziomu bezpieczeństwa numer seryjny generowany jest na podstawie nazwy użytkownika. Dla sprawdzenia poprawności działania funkcji generującej numer napisano program pomocniczy *KeyGen* generujący numery dla podanych nazw.

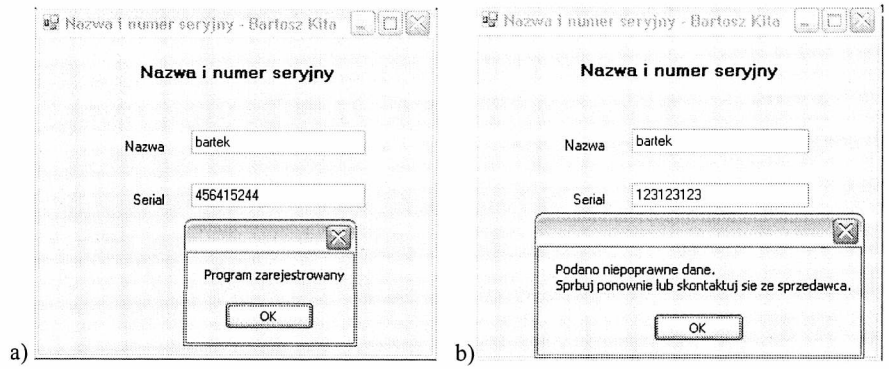
Poniżej przedstawiono fragment kodu źródłowego generującego numer seryjny na podstawie nazwy wprowadzonej przez użytkownika:

```
private void button1_Click(object sender, EventArgs e)
        {
            //---  dlugosc  ---------------
            string name = textBox1.Text;
            //---    pierwsza    -------------
            int pierwsza = (name.Length - 1);
            string p = pierwsza.ToString();
            //---    ostatnia    -----------
            int ostatnia = ((name.Length) + (name.Length-
1));
            string o = ostatnia.ToString();
            //---    srodek    -------------
            string sr = name.Remove(name.Length - 1, 1);
            string s = sr.Remove(0, 1);
            //---    a-44  k-88 ---------------
            for (int i = 0; i < s.Length; i++)
            {
                string s2;
                s2 = s.Replace("a", "44");
                for (int j = 0; j < s2.Length; j++)
                {
                    string s3;
                    s3 = s2.Replace("k", "77");
                    string s4 = p + s3 + o;
                    int wynik = s4.GetHashCode();
                    if (wynik < 0)
                    {
                        wynik = wynik * (-1);
                    }
                    if ((textBox1.Text == name) && (text-
Box2.Text == wynik.ToString()))
                        MessageBox.Show("Program zareje-
strowany");
                    else
                        MessageBox.Show("Podano niepo-
prawne dane. \nSprbuj ponownie lub skontaktuj sie ze
sprzedawca.");
                }
            }
        }
```

Na rysunku 4 przedstawiono główne okno programu zabezpieczonego przy użyciu nazwy użytkownika i numeru seryjnego.



Rys. 4. Widok okna a) programu *Name&Serial*, b) programu pomocniczego *KeyGen*, c) programu pomocniczego *KeyGen* po wygenerowaniu numeru seryjnego dla podanej nazwy



Rys. 5. Okno programu: a) *Name&Serial* po poprawnym wprowadzeniu nazwy i numeru seryjnego, b) *Name&Serial* w przypadku wprowadzenia niepoprawnego numeru seryjnego

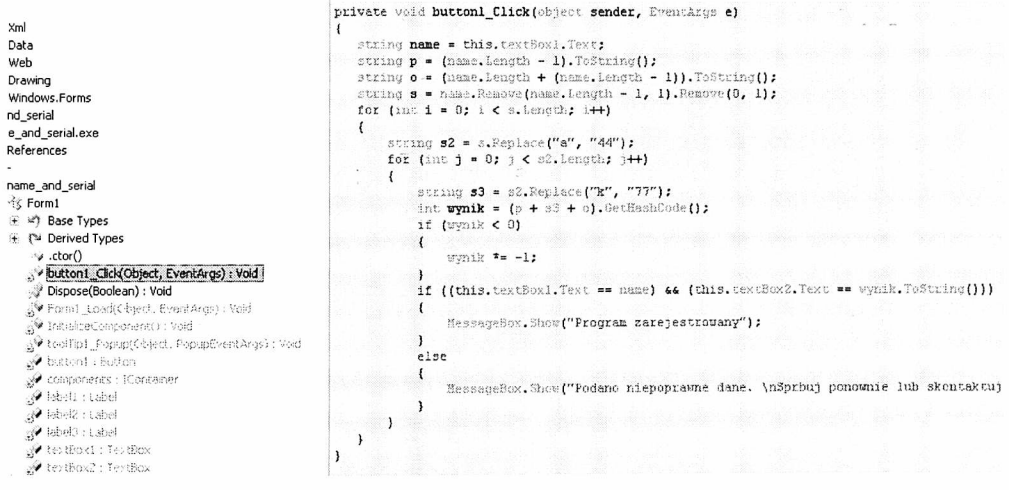
Do sprawdzenia poprawności działania programu potrzebny będzie program pomocniczy *KeyGen*, dzięki niemu możliwe jest wygenerowanie numeru seryjnego dla dowolnie wybranej nazwy użytkownika (rysunki 4b i 4c).

W przypadku podania niepoprawnego numeru seryjnego, użytkownik zostanie poinformowany oraz poproszony o kontakt ze sprzedawcą – rysunek 5b.

2.2.2. ANALIZA ZABEZPIECZENIA

Plik wykonywalny zabezpieczony przy pomocy nazwy oraz numeru seryjnego można przeanalizować korzystając z programu *Reflector*.

W programie należy wyszukać interesującą nas klasę. W oknie *Class View* widać kod funkcji generującej numer seryjny na podstawie wprowadzonego ciągu znaków. Podczas analizy kodu w programie *Reflector* widać, że wprowadzana nazwa poddawana jest różnym modyfikacjom. Dla przykładu, wszystkie litery „a" zamienione są na liczbę „44", natomiast wszystkie litery „k" – zamieniane są na liczbę „77". Po wykonaniu wszystkich modyfikacji wprowadzona nazwa poddana jest funkcji *GetHashCode*, która generuje ciąg liczbowy z zadanego łańcucha.



Rys. 6. Okno programu *Reflector*, w którym pokazano metodę generowania numeru seryjnego na podstawie wprowadzonej nazwy użytkownika

Zabezpieczenie typu nazwa i numer seryjny można bezsprzecznie uznać za jedno z najczęściej stosowanych zabezpieczeń we współczesnych programach. Bardzo często bywa łączone z najrozmaitszymi ograniczeniami w programie, które zostają usunięte po podaniu właściwego numeru seryjnego. Z analizy programu wynika, że implementacja powyższego kodu wprowadza dużo wyższy stopień zabezpieczenia

oprogramowania niż zabezpieczenie polegające na wprowadzeniu samego numeru seryjnego zapisanego w kodzie programu.

2.3. ALGORYTM 3 – WERYFIKACJA KLUCZA W PLIKU

Zabezpieczenie programu za pomocą pliku klucza polega na tym, że podczas uruchamiania program sprawdza w określonej lokalizacji czy istnieje dany plik. W przypadku, gdy plik klucza istnieje program uruchamia się w pełnej wersji, natomiast gdy plik nie istnieje program uruchamia się w wersji ograniczonej. W celu utrudnienia analizy takiego rozwiązania pliki klucza umieszcza się w nietypowych lokalizacjach, np.: w katalogach systemowe nadając im trudne do skojarzenia nazwy. W rozwiązaniach komercyjnych samo istnienie pliku klucza nie wystarcza. Dla lepszego zabezpieczenia oprogramowania wykorzystuje się informacje zapisane w takim pliku. Dla przykładu, plik klucza może zwierać informacje rejestracyjne, informacje autoryzacyjne lub kod programu, który zostanie umieszczony w programie (np. funkcje, które w wersji nie zarejestrowanej były niedostępne).

2.3.1. IMPLEMENTACJA ZABEZPIECZENIA

Na potrzeby pracy przygotowano dwa modelowe pogramy *KeyFile* i *PlikKlucza+*, napisane w środowisku Microsoft Visual C# 2005 Express Edition.

Po uruchomieniu, programy sprawdzają czy istnieje plik klucza, jeżeli tak to, można korzystać z pełnej wersji programu. W przeciwnym wypadku korzystanie zostaje ograniczone do wersji niepełnej. W pierwszym programie *KeyFile* dla utrudnienia analizy zabezpieczenia, plik klucza został ukryty w katalogu systemowym *Windows\system32\KeyFile* oraz nazwany config.log. W drugim przypadku zaimplementowano dodatkowe zabezpieczenie polegające na tym, że plik klucza zawiera dodatkowe funkcje programu.

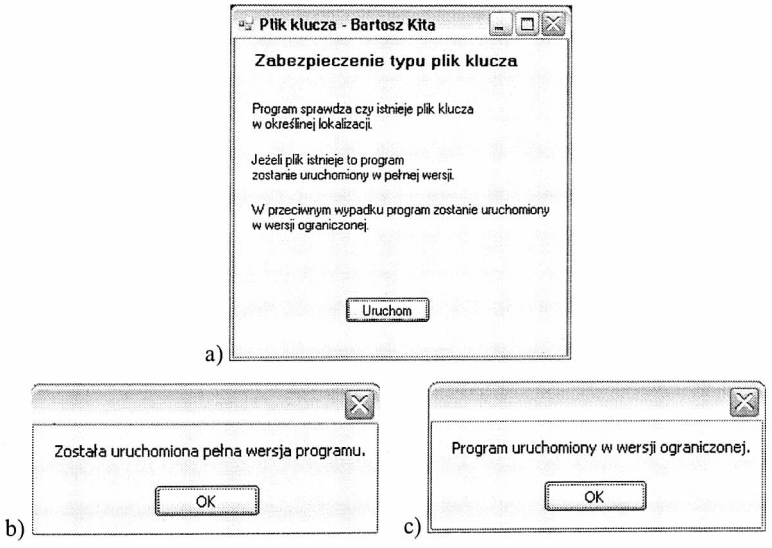
Poniżej przedstawiono fragment kodu źródłowego sprawdzającego istnienie pliku klucza zaimplementowana w programie *KeyFile*:

```
private void button1_Click(object sender, EventArgs e)
        {
            string path =
@"C:\WINDOWS\system32\KeyFile\config.log";
            if (System.IO.File.Exists(path))
            {
                MessageBox.Show("Została uruchomiona
pełna wersja programu.");
            }
```

```
            else
                MessageBox.Show("Program uruchomiony
w wersji ograniczonej.");
            }
```

Poniżej przedstawiono fragment kodu źródłowego sprawdzającego istnienie pliku klucza zaimplementowana w programie *PlikKlucza+*:

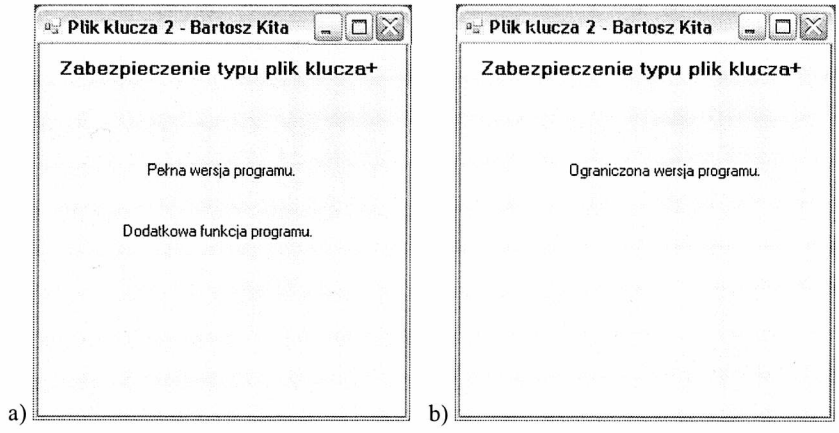
```
private void Form1_Load(object sender, EventArgs e)
{
string path = @"c:\klucz.txt";
if (System.IO.File.Exists(path))
{
string klucz = System.IO.File.ReadAllText(path);
label1.Text = ("Pełna wersja programu.");
label2.Text = (klucz);
}
else
label1.Text = ("Ograniczona wersja programu.");}
```



Rys. 7. Widok okna a) programu *KeyFile*, b) programu *KeyFile* po wykryciu pliku klucza, c) programu *KeyFile* po niewykryciu pliku klucza

Na rysunku 7a przedstawiono główne okno programu zabezpieczonego przy użyciu pliku klucza. W przypadku wykrycia istnienia przez program pliku klucza, użyt-

kownik zostaje poinformowany o uruchomieniu pełnej wersji programu. W przeciwnym razie użytkownik zostaje poinformowany o uruchomieniu programu w wersji ograniczonej.



Rys. 8. Widok okna a) programu *PlikKlcza+*,
b) programu *PlikKlucza+* po niewykryciu pliku klucza

2.3.2. ANALIZA ZABEZPIECZENIA

Plik wykonywalny zabezpieczony za pomocą pliku klucza można dokładnie przeanalizować korzystając z programu *Reflector*. W celu analizy można również skorzystać z programu *FileMon*, który sprawdza zależności programu wykonywalnego z zewnętrznymi plikami.

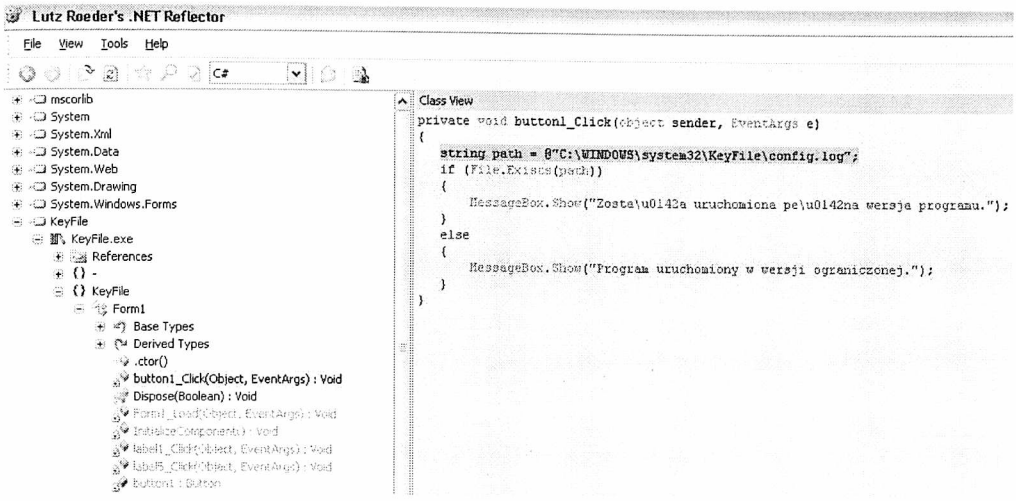
Po otwarciu testowanego modelu w programie *Reflector*, należy wybrać klasę *KeyFile.Form1.button1_Click(Object, EventArgs) : Void*. W oknie Class View można zauważyć, że program sprawdza ścieżkę:

```
string path = @"C:\WINDOWS\system32\KeyFile\config.log";
```

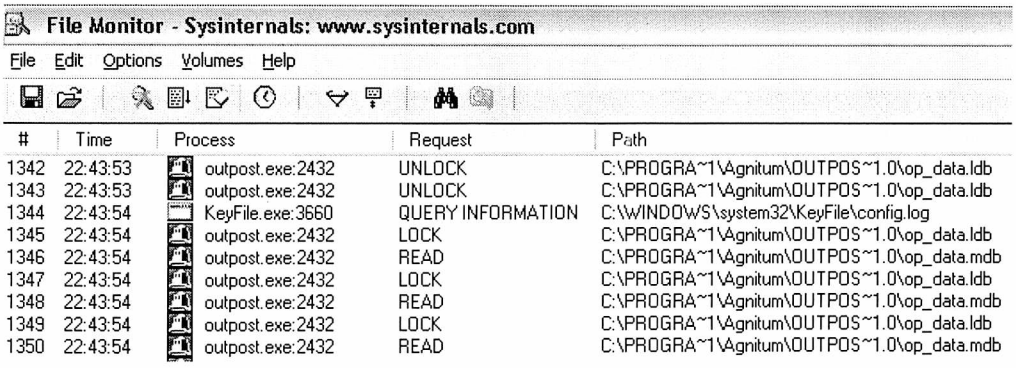
w celu uruchomienia odpowiedniej wersji programu.

Drugim programem, który umożliwia analizę zależności programu wykonywalnego z zewnętrznymi plikami jest *FileMon*. W celu analizy należy uruchomić program monitorujący, a następnie uruchomić testowany program. Kolejnym krokiem jest zapisanie logu z programu *FileMon* i prześledzenie połączeń badanego programu z zewnętrznymi plikami.

Analiza logu z programu *FileMon* może być dość uciążliwa, ponieważ programy napisane pod platformę .Net podczas uruchamiania wysyłają wiele innych zapytań do plików zewnętrznych.



Rys. 9. Okno programu *Reflector*, w którym pokazano funkcję sprawdzającą istnienie pliku klucza w programie *KeyFile*



#	Time	Process	Request	Path
1342	22:43:53	outpost.exe:2432	UNLOCK	C:\PROGRA~1\Agnitum\OUTPOS~1.0\op_data.ldb
1343	22:43:53	outpost.exe:2432	UNLOCK	C:\PROGRA~1\Agnitum\OUTPOS~1.0\op_data.ldb
1344	22:43:54	KeyFile.exe:3660	QUERY INFORMATION	C:\WINDOWS\system32\KeyFile\config.log
1345	22:43:54	outpost.exe:2432	LOCK	C:\PROGRA~1\Agnitum\OUTPOS~1.0\op_data.ldb
1346	22:43:54	outpost.exe:2432	READ	C:\PROGRA~1\Agnitum\OUTPOS~1.0\op_data.mdb
1347	22:43:54	outpost.exe:2432	LOCK	C:\PROGRA~1\Agnitum\OUTPOS~1.0\op_data.ldb
1348	22:43:54	outpost.exe:2432	READ	C:\PROGRA~1\Agnitum\OUTPOS~1.0\op_data.mdb
1349	22:43:54	outpost.exe:2432	LOCK	C:\PROGRA~1\Agnitum\OUTPOS~1.0\op_data.ldb
1350	22:43:54	outpost.exe:2432	READ	C:\PROGRA~1\Agnitum\OUTPOS~1.0\op_data.mdb

Rys. 10. Okno programu *FileMon*, w którym pokazano wysłanie zapytania przez program *KeyFile* dotyczące pliku config.log

Poniżej przedstawiono wycinek wydruku zapytań wygenerowanych przez testowany program.

```
456   22:43:52   KeyFile.exe:3660OPEN
      C:\WINDOWS\Prefetch\KEYFILE.EXE-0813DCF9.pf
      SUCCESS    Options: Open  Access: Read
457   22:43:52   KeyFile.exe:3660QUERY INFORMATION
      C:\WINDOWS\Prefetch\KEYFILE.EXE-0813DCF9.pf
      SUCCESS    Length: 41606
```

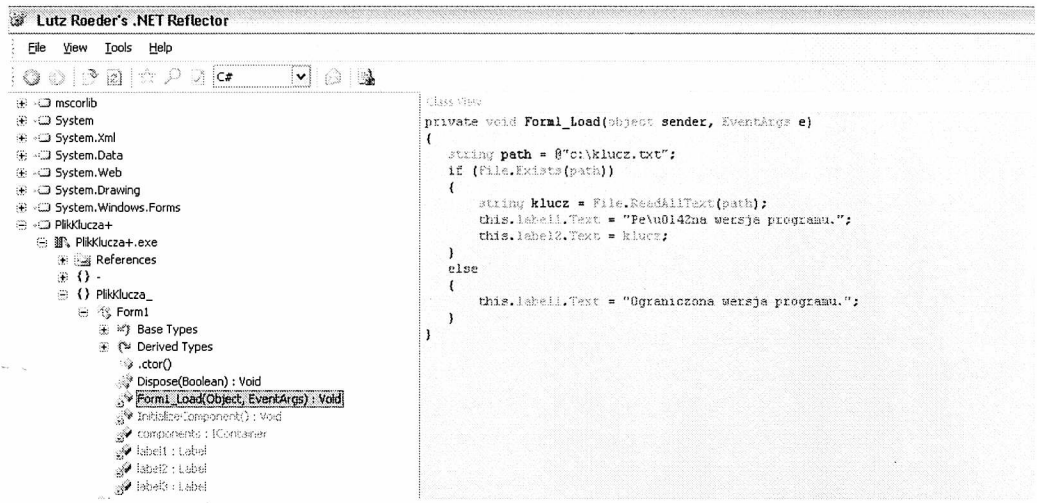
```
458   22:43:52   KeyFile.exe:3660READ
      C:\WINDOWS\Prefetch\KEYFILE.EXE-0813DCF9.pf
      SUCCESS    Offset: 0 Length: 41606
459   22:43:52   KeyFile.exe:3660OPEN C:    SUCCESS
      Options: Open  Access: 00100180
460   22:43:52   KeyFile.exe:3660QUERY INFORMATION     C:
      BUFFER OVERFLOW FileFsVolumeInformation
461   22:43:52   KeyFile.exe:3660OPEN C:\   SUCCESS
      Options: Open Directory  Access: 00100001
462   22:43:52   KeyFile.exe:3660DIRECTORY  C:\   SUCCESS
      FileNamesInformation
463   22:43:52   KeyFile.exe:3660DIRECTORY  C:\   NO MORE
FILES FileNamesInformation
464   22:43:52   KeyFile.exe:3660OPEN C:\DOCUMENTS AND
SETTINGS\  SUCCESS    Options: Open Directory  Access:
00100001
465   22:43:52   KeyFile.exe:3660DIRECTORY  C:\DOCUMENTS
AND SETTINGS\    SUCCESS    FileNamesInformation
466   22:43:52   KeyFile.exe:3660DIRECTORY  C:\DOCUMENTS
AND SETTINGS\    NO MORE FILES    FileNamesInformation
467   22:43:52   KeyFile.exe:3660OPEN C:\DOCUMENTS AND
SETTINGS\BARTEK\SUCCESS    Options: Open Directory
Access: 00100001
468   22:43:52   KeyFile.exe:3660DIRECTORY  C:\DOCUMENTS
AND SETTINGS\BARTEK\  SUCCESS    FileNamesInformation
469   22:43:52   KeyFile.exe:3660DIRECTORY  C:\DOCUMENTS
AND SETTINGS\BARTEK\  NO MORE FILES    FileNamesInformation
470   22:43:52   KeyFile.exe:3660OPEN C:\DOCUMENTS AND
SETTINGS\BARTEK\USTAWIENIA LOKALNE\   SUCCESS    Options:
Open Directory  Access: 00100001
471   22:43:52   KeyFile.exe:3660DIRECTORY  C:\DOCUMENTS
AND SETTINGS\BARTEK\USTAWIENIA LOKALNE\    SUCCESS
      FileNamesInformation
472   22:43:52   KeyFile.exe:3660DIRECTORY  C:\DOCUMENTS
AND SETTINGS\BARTEK\USTAWIENIA LOKALNE\    NO MORE FILES
      FileNamesInformation
473   22:43:52   KeyFile.exe:3660OPEN C:\DOCUMENTS AND
SETTINGS\BARTEK\USTAWIENIA LOKALNE\DANE APLIKACJI\
      SUCCESS    Options: Open Directory  Access: 00100001
474   22:43:52   KeyFile.exe:3660DIRECTORY  C:\DOCUMENTS
AND SETTINGS\BARTEK\USTAWIENIA LOKALNE\DANE APLIKACJI\
      SUCCESS    FileNamesInformation
```

```
475  22:43:52  KeyFile.exe:3660DIRECTORY  C:\DOCUMENTS
AND SETTINGS\BARTEK\USTAWIENIA LOKALNE\DANE APLIKACJI\
     NO MORE FILES   FileNamesInformation
476  22:43:52  KeyFile.exe:3660OPEN C:\PROGRAM FILES\
     SUCCESS    Options: Open Directory  Access: 00100001
477  22:43:52  KeyFile.exe:3660DIRECTORY  C:\PROGRAM
FILES\     SUCCESS    FileNamesInformation
478  22:43:52  KeyFile.exe:3660DIRECTORY  C:\PROGRAM
FILES\     NO MORE FILES   FileNamesInformation
479  22:43:52  KeyFile.exe:3660OPEN C:\PROGRAM
FILES\GADU-GADU\SUCCESS    Options: Open Directory  Ac-
cess: 00100001
480  22:43:52  KeyFile.exe:3660DIRECTORY  C:\PROGRAM
FILES\GADU-GADU\SUCCESS    FileNamesInformation
481  22:43:52  KeyFile.exe:3660DIRECTORY  C:\PROGRAM
FILES\GADU-GADU\NO MORE FILES   FileNamesInformation
482  22:43:52  KeyFile.exe:3660OPEN C:\PROGRA~1\
     SUCCESS    Options: Open Directory  Access: 00100001

483  22:43:52  KeyFile.exe:3660DIRECTORY  C:\PROGRA~1\
     SUCCESS    FileNamesInformation
484  22:43:52  KeyFile.exe:3660DIRECTORY  C:\PROGRA~1\
     NO MORE FILES   FileNamesInformation
485  22:43:52  KeyFile.exe:3660OPEN C:\PROGRA~1\AGNITUM\
     SUCCESS    Options: Open Directory  Access: 00100001
486  22:43:52  KeyFile.exe:3660DIRECTORY
     C:\PROGRA~1\AGNITUM\ SUCCESS    FileNamesInformation
487  22:43:52  KeyFile.exe:3660DIRECTORY
     C:\PROGRA~1\AGNITUM\ NO MORE FILES
     FileNamesInformation
```

W przypadku drugiego programu analiza zabezpieczenia będzie analogiczna jak w pierwszym programie. Jedyną zauważalną różnicą w programie *Reflector* będzie wykorzystanie dodatkowych informacji zawartych w pliku klucza. W tym wypadku będzie to wyświetlenie informacji zapisanej w pliku klucza.

Zabezpieczenie programu z wykorzystaniem zewnętrznego pliku klucza jest dość łatwe do wykrycia i pomimo ukrywania pliku w katalogach systemowych oraz nadawania mu specjalnej nazwy, nie stanowi bariery nie do przełamania. Po dogłębnej analizie można stwierdzić do jakiego pliku program wysłał zapytanie. Jeżeli będzie to zabezpieczenie polegające na sprawdzaniu istnienia pliku, możemy sami stworzyć taki plik. Problem może pojawić się w momencie, gdy w pliku klucza zapisane są dodatkowe informacje, np.: dodatkowe funkcje programu,

a kraker nie jest w posiadaniu pliku klucza. Ominięcie takiego zabezpieczenia jest praktycznie niemożliwe, ponieważ nie można przewidzieć jakie funkcje zostały ukryte w pliku.



Rys. 11. Okno programu *Reflector*, w którym pokazano funkcję sprawdzającą istnienie pliku klucza oraz wykorzystanie informacji zapisanej w tym pliku zastosowane programie *PlikKlucza+*

2.4. ALGORYTM 4 – WERYFIKACJA LICZNIKA W PLIKU

Zabezpieczenie programu wykorzystujące licznik uruchomień w pliku tekstowym polega na przechowywaniu liczby uruchomień programu w zewnętrznym pliku. Podczas pierwszego uruchomienia program, w określonej lokalizacji, tworzy plik tekstowy i zapisuje do niego darmową liczbę uruchomień. Podczas każdego uruchamiania program odczytuje wartość z pliku i wyświetla ją użytkownikowi. Po odczytaniu wartości licznik jest zmniejszany i ponownie zapisywany do pliku.

2.4.1. IMPLEMENTACJA ZABEZPIECZENIA

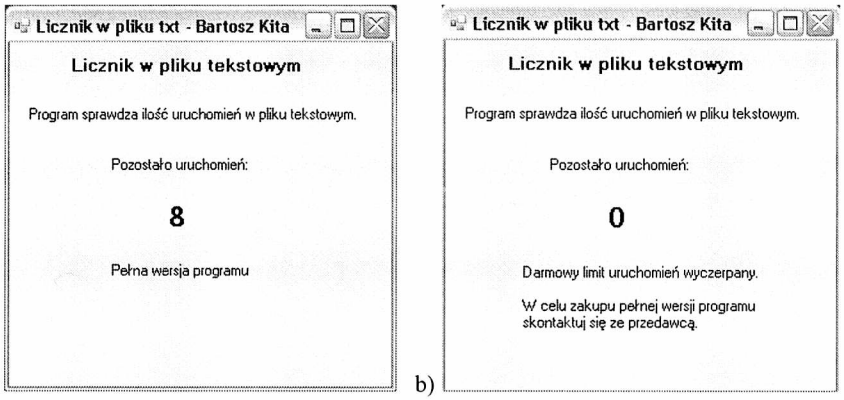
Na potrzeby pracy przygotowano modelowy pogram *Licznik w pliku txt*, napisany w środowisku Microsoft Visual C# 2005 Express Edition. Po uruchomieniu program otwiera zewnętrzny plik tekstowy i odczytuje z niego wartość licznika, następnie zmniejsza go i zapisuje z powrotem do pliku. W przypadku, gdy wartość licznika za-

pisanego w pliku równa jest zero, program informuje użytkownika, że darmowy limit uruchomień został wyczerpany.

Poniżej przedstawiono fragment kodu źródłowego sprawdzającego wartość licznika zapisanego w pliku tekstowym:

```
private void Form1_Load(object sender, EventArgs e)
        {
            string path = @"c:\licznik.txt";
            if (System.IO.File.Exists(path))
            {
                string licznik = Sys-
tem.IO.File.ReadAllText(path);
                {
                    int licznik2 = int.Parse(licznik);
                    if (licznik2 > 0)
                    {
                        label2.Text =
licznik2.ToString();
                        licznik2--;
                        string nowylicznik
= licznik2.ToString();
                        System.IO.File.WriteAllText(path,
nowylicznik);
                        label4.Text = "Pełna wersja
programu";
                    }
                    else
                    {
                        label2.Text = licz-
nik2.ToString();
                        label4.Text = "Darmowy limit
 uruchomień wyczerpany. \n\nW celu zakupu pełnej wersji
programu \nskontaktuj się ze przedawcą.";
                    }
                }
            }}
```

Na rysunku 12 przedstawiono główne okno programu zabezpieczonego przy użyciu licznika w pliku tekstowym. W przypadku wyczerpania darmowej liczby uruchomień użytkownik proszony jest o kontakt ze sprzedawcą w celu nabycia pełnej wersji programu.



Rys. 12. Widok okna: a) programu *Licznik w pliku txt*, b) programu *Licznik w pliku txt* po wyczerpaniu darmowych uruchomień

2.4.2. ANALIZA ZABEZPIECZENIA

Analiza zabezpieczenia programu przez licznik w pliku tekstowym jest bardzo podobna do analizy wykonywanej podczas zabezpieczenia programu z wykorzystaniem pliku klucza.

Dokładną analizę można przeprowadzić wykorzystując program *Reflector*, jak również skorzystać z programu *FileMon* w celu wyśledzenia powiązań pliku wykonywalnego z zewnętrznym plikiem.

W programie *Reflector* wybieramy klasę:

Licznik_w_pliku_txt.Form1.Form1_Load(Object, EventArgs) : Void

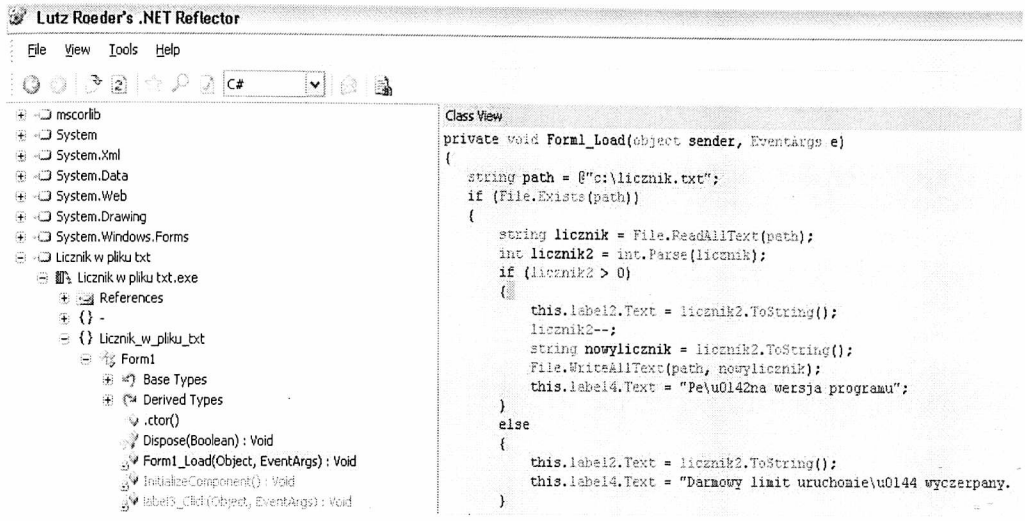
Po wybraniu klasy w oknie *Class View* widać funkcję, która otwiera plik tekstowy, odczytuje z niego wartość licznika, następnie sprawdza czy jest on większy od zera. Jeżeli tak, to licznik jest zmniejszany i zapisywany do pliku. Jeżeli nie, wyświetlana jest informacja, że pozostało zero darmowych uruchomień.

Również podczas analizy tego programu przydatny będzie program *FileMon.* Jak w poprzednim przypadku (*KeyFile*) prześledzono połączenia pliku wykonywalnego z zewnętrznymi plikami.

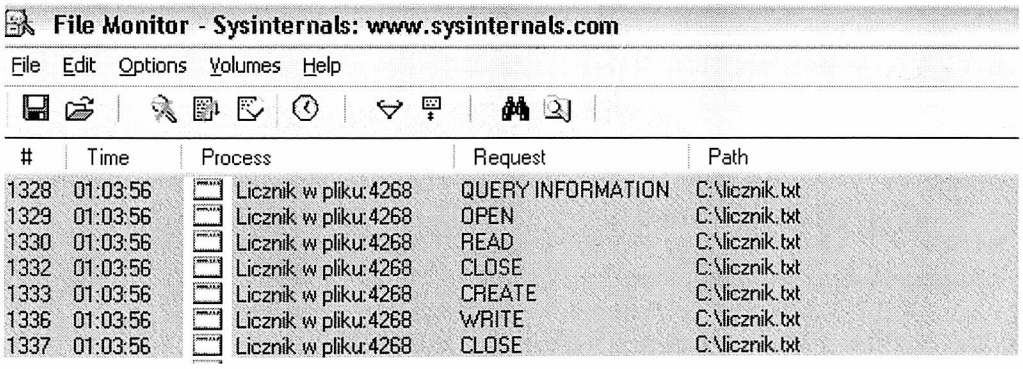
Z rysunku 13 wynika, że wykonywalny plik *Licznik w pliku txt* wysłał zapytanie odnośnie pliku tekstowego licznik.txt. Następnie plik został otwarty i program odczytał wartość z pliku. Logiczne jest, że licznik został zmniejszony co widać również na analizowanym rysunku. Plik został ponownie otwarty, a zmniejszona wartość licznika została zapisana do pliku *licznik.txt*.

Zaimplementowanie zabezpieczenia opartego na liczniku ukrytym w pliku tekstowym również nie daje wysokiej skuteczności. Wystarczy odszukać do jakiego pliku odwołuje się dany program i przeanalizować zapisane w nim informacje. Dobrą metodą zwiększającą bezpieczeństwo jest szyfrowanie zawartości pliku tekstowego, jak

również ukrywanie go w katalogach systemowych oraz nadawanie mu atrybutów „archiwalny", „tylko do odczytu", „systemowy" lub „ukryty".



Rys. 13. Okno programu *Reflector*, w którym pokazano funkcję sprawdzającą liczbę dla programu *Licznik w pliku txt*



Rys. 14. Analiza połączeń programu *Licznik w pliku* z zewnętrznym plikiem tekstowym

2.5. ALGORYTM 5 – WERYFIKACJA LICZNIKA W REJESTRZE SYSTEMOWYM

Zabezpieczenie programu wykorzystujące licznik uruchomień w rejestrze polega na przechowywaniu liczby uruchomień programu w rejestrze systemu. Podczas pierwszego uruchomienia program tworzy wpis w rejestrze, w którym ustawia określoną

liczbę uruchomień programu. Podczas każdego uruchamiania program odczytuje wartość z rejestru i wyświetla ją użytkownikowi. Po odczytaniu wartości licznik jest zmniejszany i ponownie zapisywany do rejestru.

2.5.1. IMPLEMENTACJA ZABEZPIECZENIA

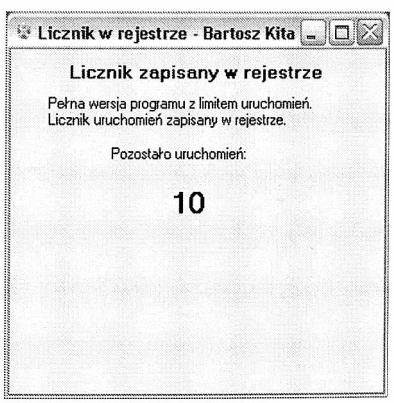
Na potrzeby pracy przygotowano modelowy pogram *Licznik w rejestrze*, napisany w środowisku Microsoft Visual C# 2005 Express Edition, umożliwiający dziesięciokrotne otwarcie pełnej jego wersji. Po uruchomieniu, program odwołuje się do określonego wpisu w rejestrze i odczytuje z niego wartość licznika, następnie zmniejsza go i ponownie zapisuje. W przypadku, gdy wartość licznika równa jest zero, program informuje użytkownika, że darmowy limit uruchomień został wyczerpany.

Poniżej przedstawiono fragment kodu źródłowego sprawdzającego wartość licznika zapisanego w rejestrze systemowym:

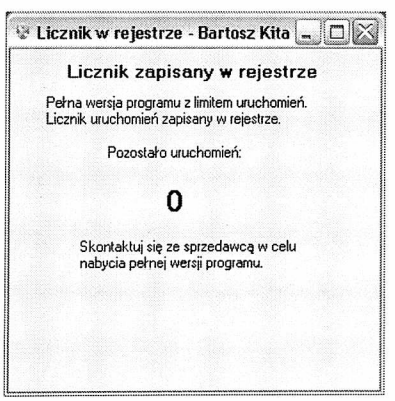
```
private void Form1_Load(object sender, EventArgs e)
        {
            label3.Text = ("Pełna wersja programu
z limitem uruchomień. \nLicznik uruchomień zapisany
w rejestrze.");
                Microsoft.Win32.RegistryKey key;
                key = Microso-
ft.Win32.Registry.CurrentUser.OpenSubKey("LicznikRejestr
Dyplom");
             if (key == null)
             {
                int counter = 10;
                key =
Microsoft.Win32.Registry.CurrentUser.CreateSubKey
("LicznikRejestrDyplom");
                key.SetValue("licznik", counter);
                foreach (string valuename in
key.GetValueNames())
                {
                    label1.Text =
(key.GetValue(valuename).ToString());
                }
            }
             else
            {
                key =
Microsoft.Win32.Registry.CurrentUser.CreateSubKey
```

```
("LicznikRejestrDyplom");
                foreach (string valuename in
key.GetValueNames())
                {
                    string zm;
                    zm =
key.GetValue(valuename).ToString();
                    int counter;
                    counter = int.Parse(zm);
                    if (counter > 0)
                    {
                        counter--;
                        key.SetValue("licznik", counter);
                        label1.Text =
(key.GetValue(valuename).ToString());
                    }
                    else
                    {
                        label2.Text = ("Skontaktuj się ze
sprzedawcą w celu \nnabycia pełnej wersji programu.");
                        label1.Text =
(key.GetValue(valuename).ToString());
                    }
                }
            }
            key.Close();
        }
```

a)



b)



Rys. 15. Widok okna: a) programu *Licznik w rejestrze*, b) programu *Licznik w rejestrze* po wyczerpaniu darmowych uruchomień

2.5.2. ANALIZA ZABEZPIECZENIA

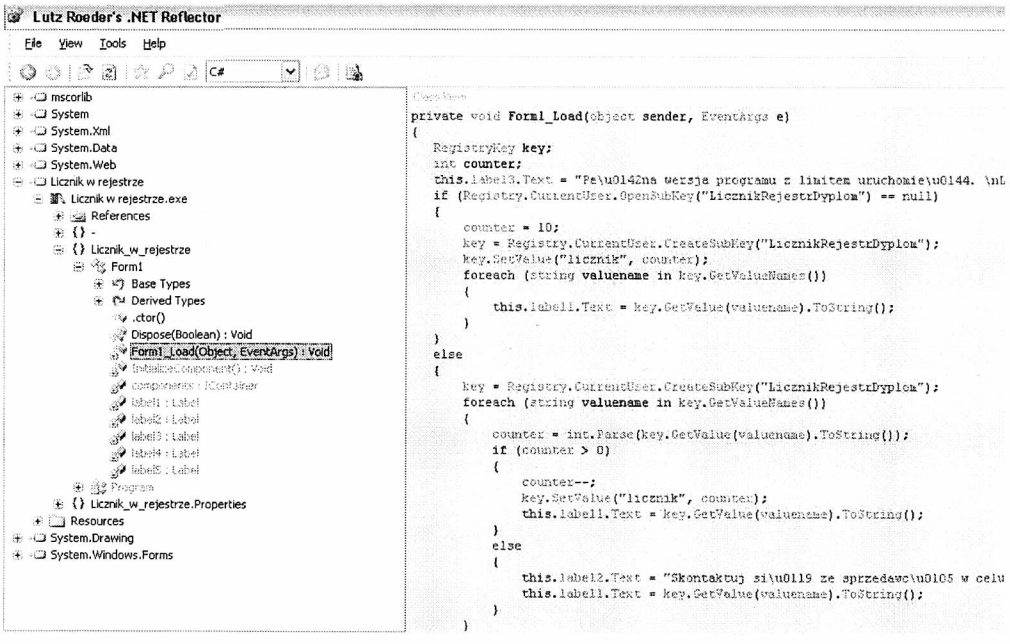
Analiza zabezpieczenia programu jest bardzo podobna do analizy wykonywanej podczas zabezpieczenia programu z wykorzystaniem licznika uruchomień zapisanego w pliku tekstowym. Dokładną analizę można przeprowadzić wykorzystując program *Reflector*, jak również skorzystać z programu *RegMon,* w celu wyśledzenia powiązań pliku wykonywalnego z rejestrem systemu.

W programie *Reflector* należy wybrać klasę:

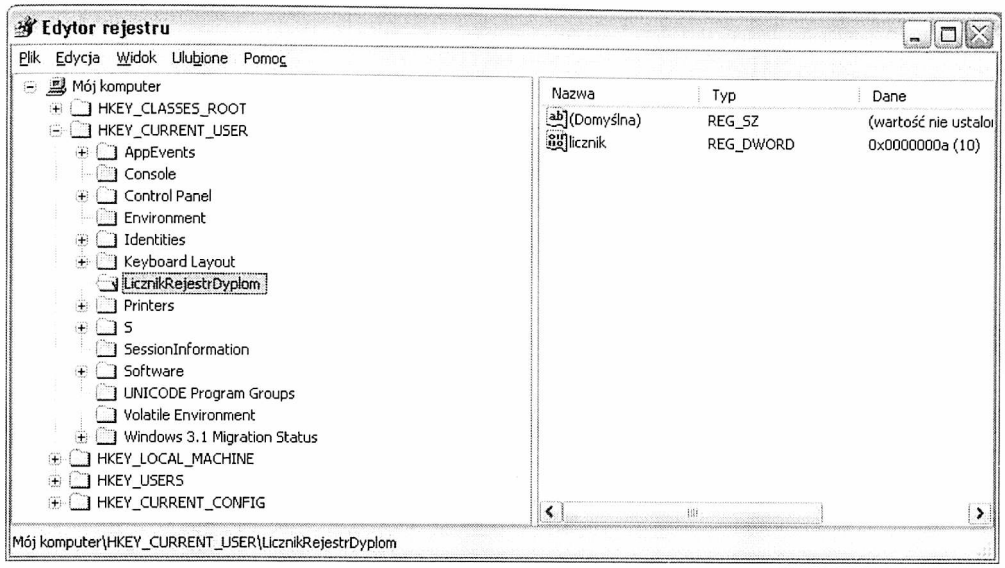
```
Licznik_w_rejestrze.Form1.Form1_Load(Object, EventArgs) : Void
```

Po wybraniu klasy w oknie Class View widać funkcję, która podczas uruchamiania programu sprawdza czy istnieje określony wpis w rejestrze. Jeżeli nie, tworzy nowy wpis w rejestrze systemowym LicznikRejestrDyplom i ustawia jego wartość na 10. Następnie odczytuje ustawioną wartość i wyświetla ją użytkownikowi. Kolejnym krokiem jest zmniejszenie wartości o jeden i zapisanie nowej wartości do rejestru. Podczas kolejnego uruchomienia program ponownie odczytuje wartość, wyświetla ją, a następnie pomniejsza.

Funkcja działa tak do momentu, gdy wartość licznika jest większa od zera. W przeciwnym wypadku użytkownik zostaje poinformowany o wyczerpaniu darmowych uruchomień.

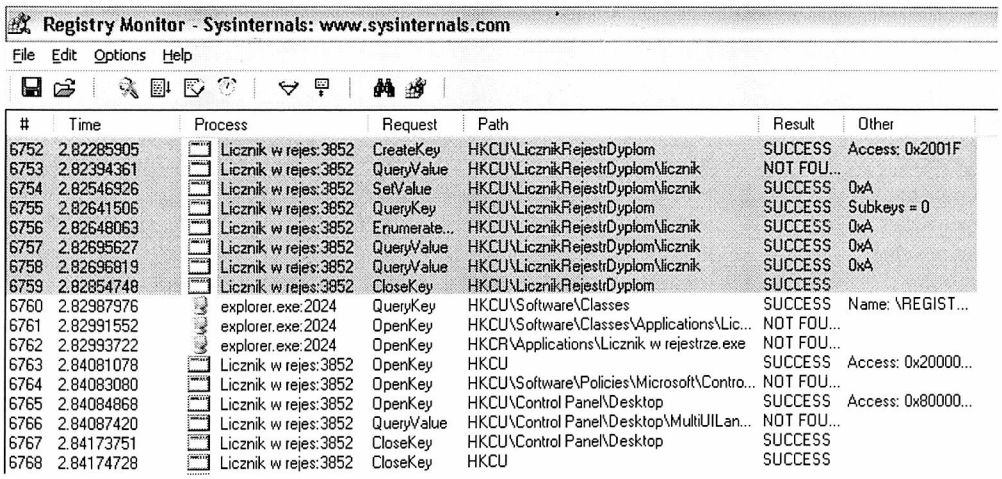


Rys. 16. Okno programu *Reflector*, w którym pokazano funkcję sprawdzającą licznik uruchomień zapisany w rejestrze dla programu *Licznik w rejestrze*



Rys. 17. Okno *Edytora rejestru*, w którym pokazano wpis oraz wartość nowego rejestru

Podczas analizy tego programu można również skorzystać z programu *RegMon*, w którym można zobaczyć połączenia analizowanego programu z rejestrem systemowym.



Rys. 18. Analiza połączeń programu *Licznik w rejestrze* z rejestrem systemowym

Analizując logi utworzone podczas kolejnych uruchomień programu, można zauważyć, że podczas pierwszego uruchomienia utworzony został rejestr: *HKCU\LicznikRejestrDyplom*, w którym został utworzony licznik z wartością 0xA.

Z każdym następnym uruchomieniem wartość licznika jest zmniejszana, aż do ustawienia licznika na 0x0.

W tabelach 1–11 przedstawiono fragmenty logów wygenerowane przez program *RegMon* podczas uruchamiania testowego programu *Licznik w rejestrze*.

Tabela 1. Fragment z logu wygenerowany przez program *RegMon* podczas pierwszego uruchomienia programu *Licznik w rejestrze*

Request	Path	Result	Other
OpenKey	HKCU\LicznikRejestrDyplom	NOT FOUND	
OpenKey	HKCU\LicznikRejestrDyplom	NOT FOUND	Access: 0x2001F
CreateKey	HKCU\LicznikRejestrDyplom	SUCCESS	0xA
SetValue	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	
CloseKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	

Tabela 2. Fragment z logu wygenerowany przez program *RegMon* podczas drugiego uruchomienia programu *Licznik w rejestrze*

Request	Path	Result	Other
OpenKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	Access: 0x2001F
CloseKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	
OpenKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	Access: 0x2001F
QueryValue	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	0xA
SetValue	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	0x9
CloseKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	

Tabela 3. Fragment z logu wygenerowany przez program *RegMon* podczas trzeciego uruchomienia programu *Licznik w rejestrze*

Request	Path	Result	Other
OpenKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	Access: 0x2001F
CloseKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	
OpenKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	Access: 0x2001F
QueryValue	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	0x9
SetValue	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	0x8
CloseKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	

Tabela 4. Fragment z logu wygenerowany przez program *RegMon* podczas czwartego uruchomienia programu *Licznik w rejestrze*

Request	Path	Result	Other
OpenKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	Access: 0x2001F
CloseKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	
OpenKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	Access: 0x2001F
QueryValue	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	0x8
SetValue	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	0x7
CloseKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	

Tabela 5. Fragment z logu wygenerowany przez program *RegMon* podczas piątego uruchomienia programu *Licznik w rejestrze*

Request	Path	Result	Other
OpenKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	Access: 0x2001F
CloseKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	
OpenKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	Access: 0x2001F
QueryValue	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	0x7
SetValue	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	0x6
CloseKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	

Tabela 6. Fragment z logu wygenerowany przez program *RegMon* podczas szóstego uruchomienia programu *Licznik w rejestrze*

Request	Path	Result	Other
OpenKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	Access: 0x2001F
CloseKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	
OpenKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	Access: 0x2001F
QueryValue	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	0x6
SetValue	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	0x5
CloseKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	

Tabela 7. Fragment z logu wygenerowany przez program *RegMon* podczas siódmego uruchomienia programu *Licznik w rejestrze*

Request	Path	Result	Other
OpenKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	Access: 0x2001F
CloseKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	
OpenKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	Access: 0x2001F
QueryValue	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	0x5
SetValue	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	0x4
CloseKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	

Tabela 8. Fragment z logu wygenerowany przez program *RegMon* podczas ósmego uruchomienia programu *Licznik w rejestrze*

Request	Path	Result	Other
OpenKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	Access: 0x2001F
CloseKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	
OpenKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	Access: 0x2001F
QueryValue	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	0x4
SetValue	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	0x3
CloseKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	

Tabela 9. Fragment z logu wygenerowany przez program *RegMon* podczas dziewiątego uruchomienia programu *Licznik w rejestrze*

Request	Path	Result	Other
OpenKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	Access: 0x2001F
CloseKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	
OpenKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	Access: 0x2001F
QueryValue	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	0x3
SetValue	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	0x2
CloseKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	

Tabela 10. Fragment z logu wygenerowany przez program *RegMon* podczas dziesiątego uruchomienia programu *Licznik w rejestrze*

Request	Path	Result	Other
OpenKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	Access: 0x2001F
CloseKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	
OpenKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	Access: 0x2001F
QueryValue	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	0x2
SetValue	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	0x1
CloseKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	

Tabela 11. Fragment z logu wygenerowany przez program *RegMon* podczas jedenastego uruchomienia programu *Licznik w rejestrze*

Request	Path	Result	Other
OpenKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	Access: 0x2001F
CloseKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	
OpenKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	Access: 0x2001F
QueryValue	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	0x1
SetValue	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	0x0
CloseKey	HKCU\LicznikRejestrDyplom\licznik	SUCCESS	

Analizując powyższe zabezpieczenie można stwierdzić, iż jest ono bardzo podobne do zabezpieczenia programu przez licznik zapisany w pliku tekstowym. Głównym problemem, który może się pojawić podczas próby obejścia tego zabezpieczenia, może być lokalizacja wpisu w rejestrze. Również w tym przypadku zalecane jest ukrywanie wykorzystywanego rejestru, jak również szyfrowanie wartości wpisu w rejestrze. Dla utrudnienia warto również zastosować metodę polegającą na utworzeniu kilku wpisów w rejestrze z jednoczesnym odczytywaniem i odszyfrowywaniem ich zawartości.

2.6. ALGORYTM 6. WERYFIKACJA DATY W REJESTRZE SYSTEMOWYM

Innym stosowanym zabezpieczeniem jest zabezpieczenie programu polegające na zapisie do rejestru systemowego daty, po przekroczeniu, której program nie będzie działał. Podczas pierwszego uruchomienia program tworzy wpis w rejestrze, w którym zapisuje datę powiększoną o określoną liczbę dni. Każde następne uruchomienie powoduje, że program odczytuje wartość z rejestru i porównuje ją z aktualną datą w systemie. Jeżeli data systemowa jest mniejsza od daty zapisanej w rejestrze program działa w pełnej wersji.

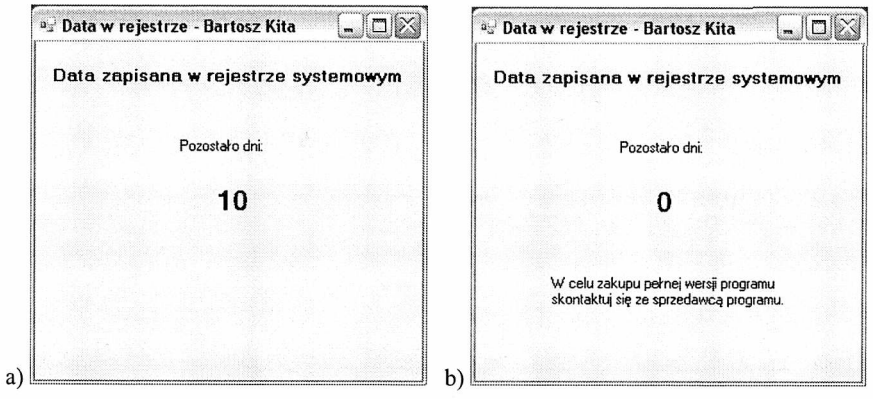
2.6.1. IMPLEMENTACJA ZABEZPIECZENIA

Na potrzeby pracy przygotowano modelowy pogram *Data w rejestrze*, napisany w środowisku Microsoft Visual C# 2005 Express Edition, umożliwiający dziesięciodniowe korzystanie z pełnej jego wersji. Po uruchomieniu, program odwołuje się do określonego wpisu w rejestrze i odczytuje z niego wartość daty porównując ją z datą aktualną. W przypadku, gdy różnica dat równa jest zero, program informuje użytkownika, że darmowy limit uruchomień został wyczerpany.

Poniżej przedstawiono fragment kodu źródłowego sprawdzającego datę zapisaną w rejestrze systemowym:

```
public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs
e)
        {
            Microsoft.Win32.RegistryKey key;
            key = Microso-
ft.Win32.Registry.CurrentUser.OpenSubKey("DataRejestr");
            DateTime dt = DateTime.Now;
            dt = dt.AddDays(10);
            DateTime dtnow = DateTime.Now;
            //DateTime dataroznica;
            if (key == null)
            {
                key =
Microsoft.Win32.Registry.CurrentUser.CreateSubKey
("DataRejestr");
```

```
                key.SetValue("dt", dt.ToString("d"));
            }
            foreach (string valuename in
key.GetValueNames())
            {
                label1.Text = ("Pozostało dni:");
                string roznica =
key.GetValue(valuename).ToString();
                DateTime DoKonca;
                DoKonca = new DateTime();
                DoKonca = DateTime.ParseExact(roznica,
"yyyy-MM-dd", null);
                TimeSpan ts = DoKonca - dtnow;
                int wynik = ts.Days;
                wynik += 1;
                if (wynik <= 0)
                {
                    label2.Text = ("0");
                    label4.Text = ("W celu zakupu pełnej
wersji programu \nskontaktuj się ze sprzedawcą programu.");
                }
                else
                {
                    label2.Text = (wynik.ToString());
                }
            }
            key.Close();
        }
```



Rys. 19. Widok okna: a) programu *Data w rejestrze*,
b) programu *Data w rejestrze* po wyczerpaniu darmowych uruchomień

Na rysunku 19 przedstawiono główne okno programu zabezpieczonego przy użyciu daty zapisanej w rejestrze systemowym.

W przypadku wyczerpania darmowej liczby uruchomień użytkownik proszony jest o kontakt ze sprzedawcą w celu nabycia pełnej wersji programu.

2.6.2. ANALIZA ZABEZPIECZENIA

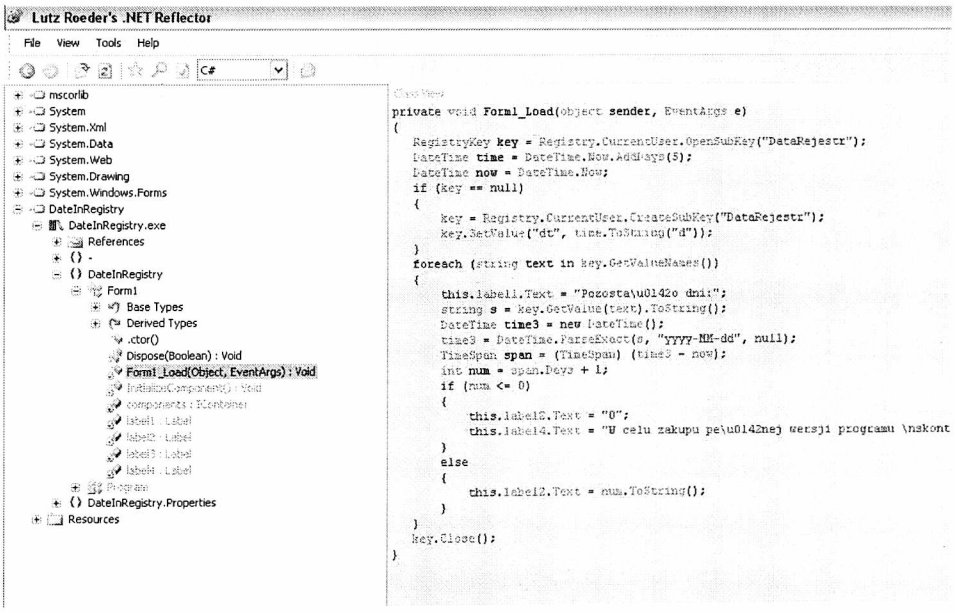
Analiza zabezpieczenia programu jest prawie identyczna do analizy wykonywanej podczas zabezpieczenia programu z wykorzystaniem licznika uruchomień zapisanego w rejestrze.

Dokładną analizę można przeprowadzić wykorzystując program *Reflector*, jak również skorzystać z programu *RegMon*, w celu wyśledzenia powiązań pliku wykonywalnego z rejestrem systemu.

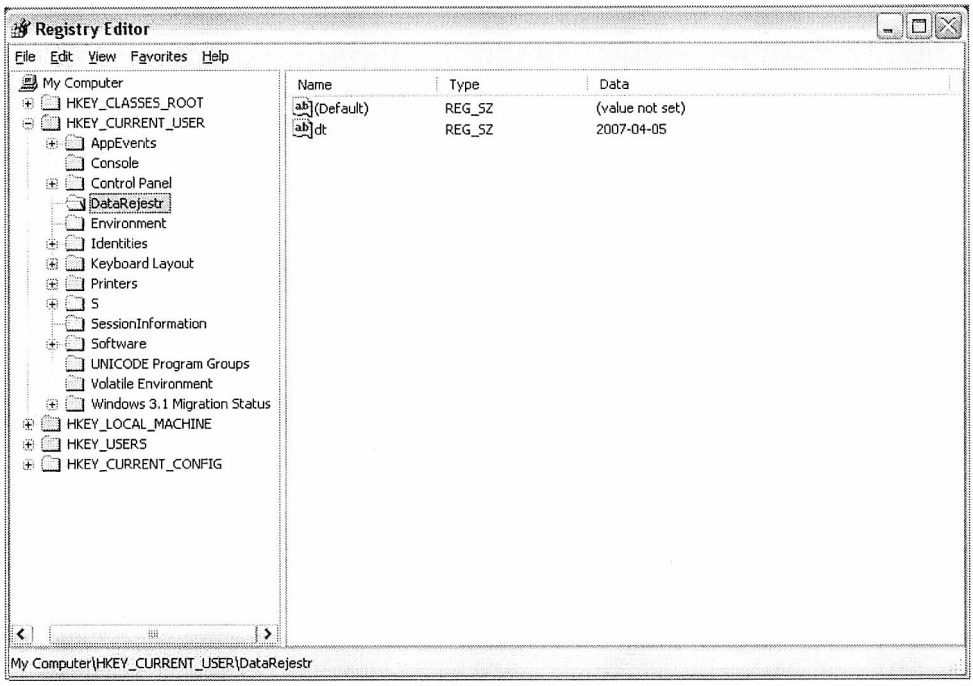
W programie *Reflector* należy wybrać klasę:

`DateInRegistry.Form1.Form1_Load(Object, EventArgs) : Void`

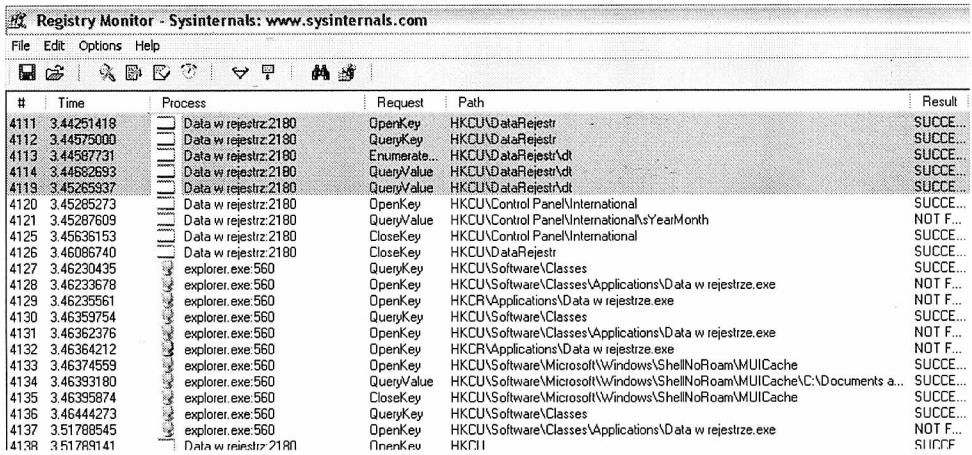
Po wybraniu klasy w oknie Class View widać funkcję, która podczas uruchamiania programu sprawdza czy, istnieje określony wpis w rejestrze. Jeżeli nie, tworzy nowy wpis w rejestrze systemowym *DataRejestr* i zapisuje do niego aktualną datę zwiększoną o określoną liczbą dni. Następnie porównuje, czy zapisana data nie jest równa aktualnej dacie. Funkcja działa do momentu, gdy różnica daty w rejestrze i daty aktualnej równa jest zero.



Rys. 20. Okno programu *Reflector*, w którym pokazano funkcję porównującą datę z rejestru z datą aktualną dla programu *Data w rejestrze*



Rys. 21. Okno *Edytora rejestru*, w którym pokazano zawartość wpisu przechowującego datę



Rys. 22. Analiza połączeń programu *Data w rejestrze* z rejestrem systemowym

Podczas analizy tego programu można również skorzystać z programu *RegMon*, w którym można zobaczyć połączenia analizowanego programu z rejestrem systemowym.

Analizując logi utworzone podczas kolejnych uruchomień modelowego programu, można zauważyć, że podczas pierwszego uruchomienia utworzony został rejestr: HKCU\DataRejestr, w którym został utworzony wpis z wartością nowej daty. Widać, że w tym przypadku wartość zapisana w rejestrze nie jest zmniejszana podczas kolejnych uruchomień programu. Z tego faktu można wywnioskować, że data zapisana w rejestrze jest porównywana z datą aktualnie ustawioną w systemie.

Poniżej przedstawiono fragmenty logów wygenerowane przez program *RegMon* podczas uruchamiania testowego programu.

Tabela 12. Fragment z logu wygenerowany przez program *RegMon* podczas pierwszego uruchomienia programu *Data w rejestrze*

Request	Path	Result	Other
OpenKey	HKCU\DataRejestr	NOT FOUND	
OpenKey	HKCU\DataRejestr	NOT FOUND	Access: 0x20019
CreateKey	HKCU\DataRejestr	SUCCESS	Access: 0x2001F
SetValue	HKCU\DataRejestr\dt	SUCCESS	2007-04-05
CloseKey	HKCU\DataRejestr\dt	SUCCESS	

Tabela 13. Fragment z logu wygenerowany przez program *RegMon* podczas drugiego uruchomienia programu *Data w rejestrze*

Request	Path	Result	Other
OpenKey	HKCU\DataRejestr\dt	SUCCESS	Access: 0x20019
EnumerateVaule	HKCU\DataRejestr\dt	SUCCESS	2007-04-05
QueryValue	HKCU\DataRejestr\dt	SUCCESS	2007-04-05
CloseKey	HKCU\DataRejestr\dt	SUCCESS	

Tabela 14. Fragment z logu wygenerowany przez program *RegMon* podczas trzeciego uruchomienia programu *Data w rejestrze*

Request	Path	Result	Other
OpenKey	HKCU\DataRejestr\dt	SUCCESS	Access: 0x20019
EnumerateVaule	HKCU\DataRejestr\dt	SUCCESS	2007-04-05
QueryValue	HKCU\DataRejestr\dt	SUCCESS	2007-04-05
CloseKey	HKCU\DataRejestr\dt	SUCCESS	

Tabela 15. Fragment z logu wygenerowany przez program *RegMon* podczas czwartego uruchomienia programu *Data w rejestrze*

Request	Path	Result	Other
OpenKey	HKCU\DataRejestr\dt	SUCCESS	Access: 0x20019
EnumerateVaule	HKCU\DataRejestr\dt	SUCCESS	2007-04-05
QueryValue	HKCU\DataRejestr\dt	SUCCESS	2007-04-05
CloseKey	HKCU\DataRejestr\dt	SUCCESS	

W przypadku tego typu zabezpieczenia, identycznie jak przy zabezpieczaniu programu przez licznik zapisany w rejestrze, zalecane jest ukrywanie wykorzystywanego rejestru, jak również szyfrowanie wartości wpisu w rejestrze. Dla utrudnienia można również zastosować metodę polegającą na utworzeniu kilku wpisów w rejestrze z jednoczesnym odczytywaniem i odszyfrowywaniem ich zawartości.

2.7. ALGORYTM 7 – KLUCZ SPRZĘTOWY

Jednymi z bardziej interesujących metod zabezpieczania programów są metody oparte na konieczności posiadania zewnętrznych urządzeń lub specjalnie przygotowanych nośników danych. Przykładem zabezpieczenia należącego do tej grupy jest zabezpieczenie programu z wykorzystaniem klucza sprzętowego.

Zabezpieczenie programu za pomocą klucza sprzętowego polega na tym, że program wysyła do portu, gdzie podłączony jest klucz sprzętowy, pewne dane i czeka na określoną odpowiedź. Jeżeli jej nie otrzyma nie zostanie uruchomiony i na ekranie zostanie wyświetlona stosowna informacja. Inną możliwością jest udostępnienie tylko niektórych funkcji programu w przypadku niepoprawnego klucza [Liber 2006b].

Klucze sprzętowe są często stosowane przez firmy rozprowadzające szczególnie drogie oprogramowanie, bardzo często uważa się, że jest to jeden z najlepszych sposobów zabezpieczania oprogramowania. Klucze sprzętowe podłącza się do dostępnych w komputerze złącz, najczęściej do: LPT, COM, USB [Liber 2006a].

2.7.1. IMPLEMENTACJA ZABEZPIECZENIA

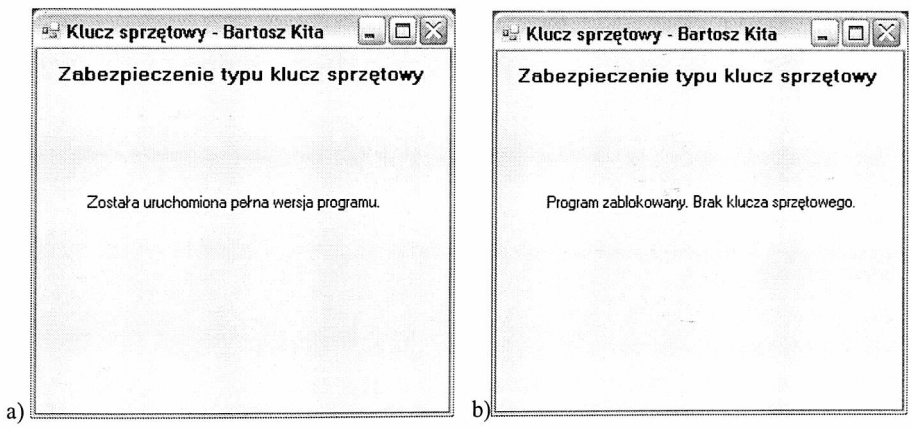
Na potrzeby pracy zaimplementowano modelowy pogram *KluczSprzetowy* napisany w środowisku Microsoft Visual C# 2005 Express Edition. Podczas uruchamiania program sprawdza obecność klucza sprzętowego podpiętego do złącza USB. W przypadku wykrycia urządzenia program weryfikuje, czy podpięte urządzenie jest typu *removable*, a dodatkowo sprawdzana jest nazwa klucza oraz informacja czy klucz posiada dodatkowy ukryty plik, który potwierdza autentyczność podłączonego urządzenia.

Poniżej przedstawiono fragment kodu źródłowego sprawdzającego obecność i autentyczność klucza sprzętowego:

```
private void Form1_Load(object sender, EventArgs e)
        {
           DriveInfo[] allDrives = DriveInfo.GetDrives();
            foreach (DriveInfo d in allDrives)
            {
                {
                    if (d.IsReady == true)
```

```
                    {
                        if (d.DriveType.ToString() ==
"Removable" && d.VolumeLabel == "SAMSUNG")
                        {
                            string path = @d.Name +
"config.txt";
                            if
(System.IO.File.Exists(path))
                            {
                                label1.Text = "Została
uruchomiona pełna wersja programu.";
                            }
                            else
                                label1.Text = "Program
zablokowany. Brak klucza sprzętowego.";
                        }
                    }
                    else
                        label1.Text = "Program
zablokowany. Brak klucza sprzętowego.";
                }}}
```

Na rysunku 23 przedstawiono główne okno programu zabezpieczonego przy użyciu klucza sprzętowego.



Rys. 23. Widok okna: a) programu *KluczSprzetowy*, b) programu *KluczSprzetowy*

W przypadku wykrycia obecności klucza sprzętowego podłączonego do portu USB oraz sprawdzenia jego autentyczności, użytkownik zostaje poinformowany o uruchomieniu pełnej wersji programu (rys. 23a). W przeciwnym wypadku, czyli przy braku

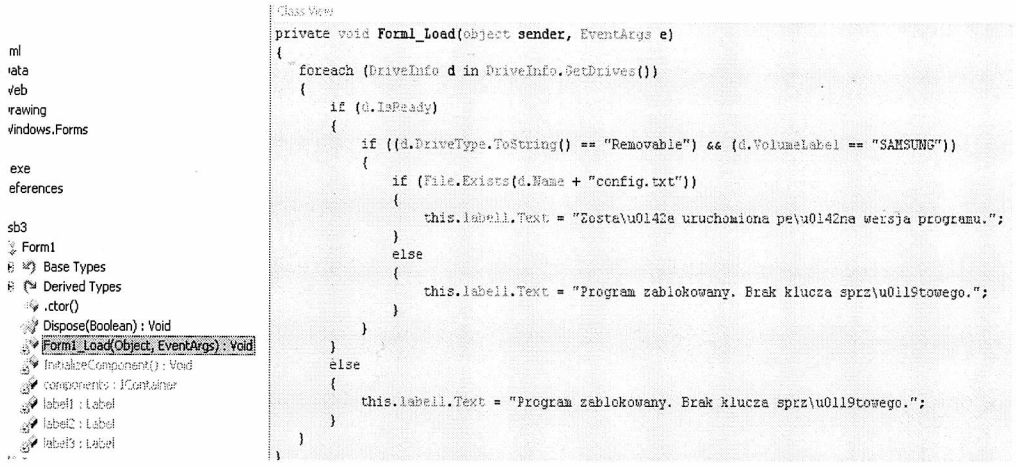
podłączenia klucza lub negatywnym wyniku testu sprawdzającego autentyczność klucza (w tym przypadku niezgodność nazwy lub brak ukrytego pliku) użytkownik otrzymuje komunikat o zablokowaniu programu z powodu braku klucza sprzętowego (rys. 23b).

2.7.2. ANALIZA ZABEZPIECZENIA

Zabezpieczenie wykorzystane w pliku wykonywalnym *KluczSprzetowy* można poddać analizie korzystając z programu *Reflector*. Po wyborze odpowiedniej klasy w drzewie programu, w oknie Class View widać kod odpowiedzialny za sprawdzanie klucza.

Jak widać program w ustalonej kolejności sprawdza autentyczność podłączonego urządzenia.

1. Czy urządzenie jest gotowe – `d.IsRedy`
2. Czy jest typu Removable – `d.DriveType.ToString() == „Removable"`
3. Czy etykieta jest zgodna z zapisaną w programie – `d.VolumeLabel == „SAMSUNG"`
4. Czy w klucz zawiera dodatkowy plik config.txt – `if (File.Exist(d.Name + „config.txt"))`



Rys. 24. Okno programu *Reflector* przedstawiające funkcje sprawdzające autentyczność podłączonego klucza sprzętowego

Większość kluczy sprzętowych jest bardzo prosta. Program wysyła do portu, gdzie podłączony jest klucz sprzętowy pewne dane i czeka na określoną odpowiedź. Jeśli jej nie otrzyma, nie zostanie uruchomiony lub uruchomi się w wersji ograniczonej.

Niektóre bardziej zaawansowane klucze sprzętowe wykorzystują różne sposoby kodowania danych przesyłanych do portu. Jedną z możliwości utrudnienia pracy krakerowi jest wprowadzenie do klucza EPROM-u z ważną częścią programu. Jeżeli do usunięcia zabezpieczenia kraker nie dysponuje kluczem sprzętowym, a jedynie samym programem, jest prawie niemożliwe, by mu się udało.

Istnieje kilka możliwości zamieszczania procedury testującej obecność klucza sprzętowego. Często wykorzystuje się API hook, który przy każdym wywołaniu danego API testuje jego obecność. Inną możliwością jest wywołanie funkcji API dla danego typu klucza. Niektóre bardziej zaawansowane klucze sprzętowe wykorzystują własne sterowniki.

3. DODATKOWA POWŁOKA OCHRONY PROGRAMÓW W JĘZYKU C#

Jak pokazano we wcześniejszych rozdziałach, w przypadku programów pisanych w języku C# pod platformę .Net 2.0 wszystkie opisane do tej pory zabezpieczenia nie dostarczają odpowiedniego poziomu bezpieczeństwa. Możliwość analizy kodu w programie *Reflector* dostarcza niemal wszystkich informacji potrzebnych do złamania zastosowanego zabezpieczenia. Ze względu na fakt, iż głównym problemem implementacyjnym przy konstrukcji skutecznych zabezpieczeń w języku C# jest łatwość odtworzenia i analizy kodu źródłowego programu, istotne staje się dodanie dodatkowej warstwy zabezpieczającej powodującej zaciemnianie kodu.

3.1. SZCZEGÓLNE WŁASNOŚCI PLATFORMY .NET I MOŻLIWOŚCI ICH WYKORRZYSTANIA PRZY KONSTRUKCJI ZABEZPIECZEŃ

Główną cechą, która wyróżnia platformę .NET wśród innych środowisk programistycznych jest wsparcie wielu języków programowania. Język C# to język programowania Microsoft, który został zaprojektowany od samego początku specjalnie dla tej platformy i wspólnego środowiska uruchomieniowego CLR. Chociaż CLR (Common Language Runtime) obsługuje wiele języków, to tylko C# był projektowany równolegle z CLR. Te dwie technologie miały na siebie duży wpływ, przez co C# świetnie nadaje się do pisania kodu zarządzanego. Kod zarządzany komponentów platformy .NET, takich jak biblioteki klas i środowisko programistyczne ASP.NET, został napisany właśnie w języku C#.

Język C# jest znacznie prostszym językiem niż C++, jednak należy do rodziny języków C. Oznacza to, że ma wiele cech wspólnych z C/C++, których nie mają języki takie jak Visual Basic. Na przykład w C# rozróżnia się wielkie i małe litery, a VB nie. C# wymaga od programistów jawnej konwersji pomiędzy typami danych, a Visual

Basic dokonuje niektórych konwersji automatycznie. Składnia języka C# jest podobna do składni języków C++ i Java. C# ma jednak w stosunku do C++ kilka dodatkowych cech obiektowych, takich jak właściwości, atrybuty, delegaty czy zdarzenia.

W języku C# można także tworzyć kod „niebezpieczny". W C# można na przykład uzyskać bezpośredni dostęp do pamięci zaalokowanej dla bufora i przeglądać tę pamięć przy użyciu wskaźników. Określenie „niebezpieczny" może wydawać się dość szokujące, jednak nie taki diabeł straszny, jak go malują. Niebezpieczny kod nie jest kodem, który jest źle napisany i ma luki. Nazywany jest „niebezpiecznym" dlatego, że niemożliwe jest sprawdzenie, czy kod bezpośrednio operujący na pamięci wykona niedozwoloną akcję.

Ważną elementem jest wspólne środowisko uruchomieniowe CLR. Wszystkie języki środowiska .NET (na przykład C# czy Visual Basic .NET), a także wszystkie biblioteki klas obecne w .NET Framework (ASP.NET, ADO.NET i inne) oparte są na CLR.

Środowisko CLR kompiluje i wykonuje zapisany w standardowym języku pośrednim Microsoft (MSIL) kod aplikacji zwany kodem zarządzanym (ang. *managed code*), zapewniając wszystkie podstawowe funkcje konieczne do działania aplikacji. Podstawowym elementem CLR jest standardowy zestaw typów danych, wykorzystywanych przez wszystkie języki oparte na CLR, a także standardowy format metadanych, służących do opisu oprogramowania wykorzystującego te typy danych. CLR zapewnia także mechanizmy umożliwiające pakowanie kodu zarządzanego w jednostki zwane podzespołami.

Just-in-time compilation to metoda wykonywania programów polegająca na kompilacji do kodu maszynowego w locie, czyli bezpośrednio przed wykonaniem fragmentu kodu. Cała procedura wygląda następująco:

- kod źródłowy jest kompilowany do kodu pośredniego (bajtowego),
- program jest rozpowszechniany w postaci kodu pośredniego,
- na maszynie, na której program zostaje uruchomiony, maszyna wirtualna przeprowadza kompilację kodu pośredniego do kodu maszynowego.

Kompilacja może się odbywać w momencie pierwszego dostępu do kodu znajdującego się w pliku lub pierwszego wywołania funkcji (stąd nazwa just-in-time). Dodatkową zaletą stosowania techniki JIT jest fakt, że w momencie startu programu nie musi istnieć cały kod pośredni programu. Kompilacja odbywa się dopiero w momencie dostępu do konkretnego kodu programu co wymaga stosowania silnych zabezpieczeń już na poziomie kodu źródłowego programu.

3.2. ZACIEMNIANIE KODU Z ZASTOSOWANIEM PODPISU PROGRAMU OPARTEGO NA RUCHOMYCH ZAPORACH PSEUDOLOSOWYCH

Aby umożliwić skuteczne przeniesienie dotychczasowych zabezpieczeń programów implementowanych w językach C, C++ i Assembler do języka C++ należy

skonstruować warstwę ochronną kodu źródłowego. Warstwę taką można skonstruować przez np. zastosowanie inwariantów kodu [LIBER 2006a]. Wydaje się jednak, że do takiej metody należy wprowadzić dodatkowe mechanizmy podpisywania i transformacji kodu na przykład metody opisane w pracy [LIBER 2006d]. Metody przedstawione w powołanej pracy bazują na tak zwanych pseudolosowych ścianach zaporowych.

W ramach pracy wykonano implementację oprogramowania zaciemniającego z pseudolosowymi ścianami zaporowymi w języku C#. Pierwszą czynnością jaka należy tu wykonać, jest dezasemblacja programu testowego do postaci kodu pośredniego. Do tego celu użyto program ILdasm.exe, udostępniony przez firmę Microsoft, wchodzący w skład pakietu Framework SDK. Kolejnym krokiem jest generacja pseudolosowych ciągów znaków, zawierających ukryty podpis. Wygenerowane ciągi zastępują nazwy zmiennych, klas i funkcji zawartych w programie testowym. Ostatnim etapem jest kompilacja zaciemnionego kodu do programu w postaci wykonywalnej.

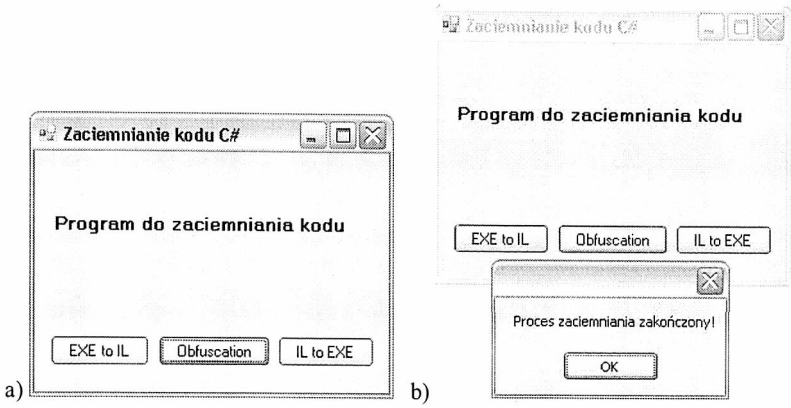
Po uruchomieniu tak przygotowanego programu, widoczne są trzy opcje do wyboru:

- EXE to IL, umożliwia dekompilację programu wykonywalnego do postaci kodu pośredniego,
- Obfuscation, uruchamia proces generowania podpisanych ciągów znakowych, oraz dokonuje zaciemnienia nazw zmiennych, klas i funkcji w kodzie pośrednim.
- IL to EXE, dokonuje kompilacji zaciemnionego kodu pośredniego do programu w postaci wykonywalnej. Poniżej przedstawiono fragment kodu źródłowego realizującego zaciemnianie kodu pośredniego:

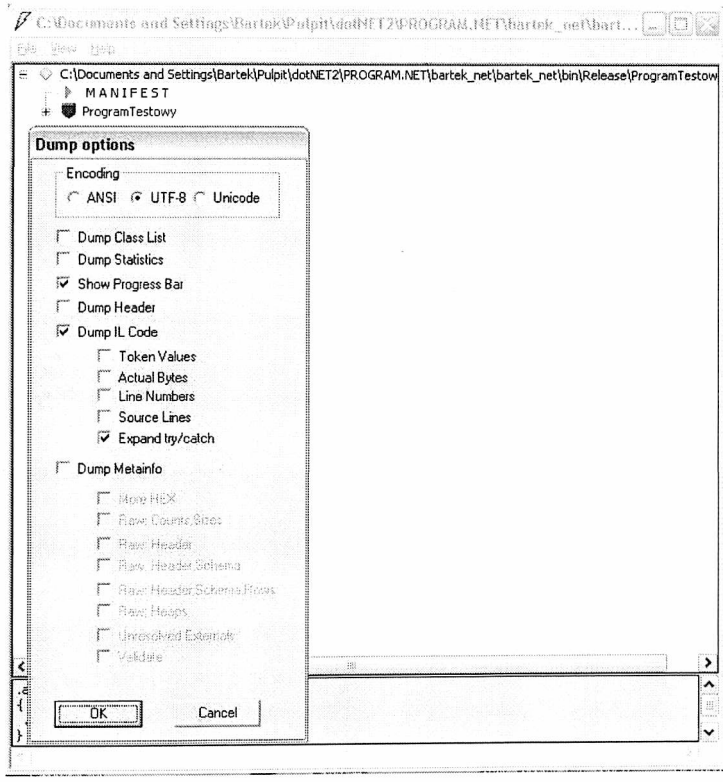
```
...
StreamReader streamReader;
            streamReader = File.OpenText(@"C:\Documents
and Set-
tings\Bartek\Pulpit\dotNET2\PROGRAM.NET\bartek_net\bartek
_net\bin\Release\zm.il");
            {
                for (int i = 0; i < ile; i++)
                {
                    line = streamReader.ReadLine();

                    if (line.Contains("field") &&
line.Contains("string"))
                    {
                        if (sf1 == sf11)
                        {
```

```
                        ost = line.Remove(0, line.Length -
1);
                    bez_ost = line.Remove(line.Length -
1, 1);
                            ost = obfus[sf1].ToString();
                            tab[i] = line.Replace(line,
bez_ost + obfus[sf1].ToString());
                            sf1++;
                            sf11++;
                        }
                    }

                    else if (line.Contains("field") &&
line.Contains("int32"))
                    {
                        if (if1 == if11)
                        {
                            ost = line.Remove(0,
line.Length - 1);
                            bez_ost = li-
ne.Remove(line.Length - 1, 1);
                            ost = obfus[if1].ToString();
                            tab[i] = line.Replace(line,
bez_ost + obfus[if1].ToString());
                            if1++;
                            if11++;
                        }
...
StreamWriter streamWriter = Fi-
le.CreateText(@"C:\Documents and Set-
tings\Bartek\Pulpit\dotNET2\PROGRAM.NET\bartek_net\bartek
_net\bin\Release\zm.il");
            for (int i = 0; i < ile; i++)
            {
               streamWriter.WriteLine(tab[i].ToString());
            }
            MessageBox.Show("Proces zaciemniania zakoń-
czony!");
            streamWriter.Close();
```

Na rysunku 25 przedstawiono główne okno programu do zaciemniania i niewidzialnego sygnowania kodu programu na poziomie kodu pośredniego.



Rys. 25. Główne okno: a) programu do zaciemniania i podpisywania kodu C#, b) z informacją po zakończeniu procesu obfuskacji i podpisaniu programu

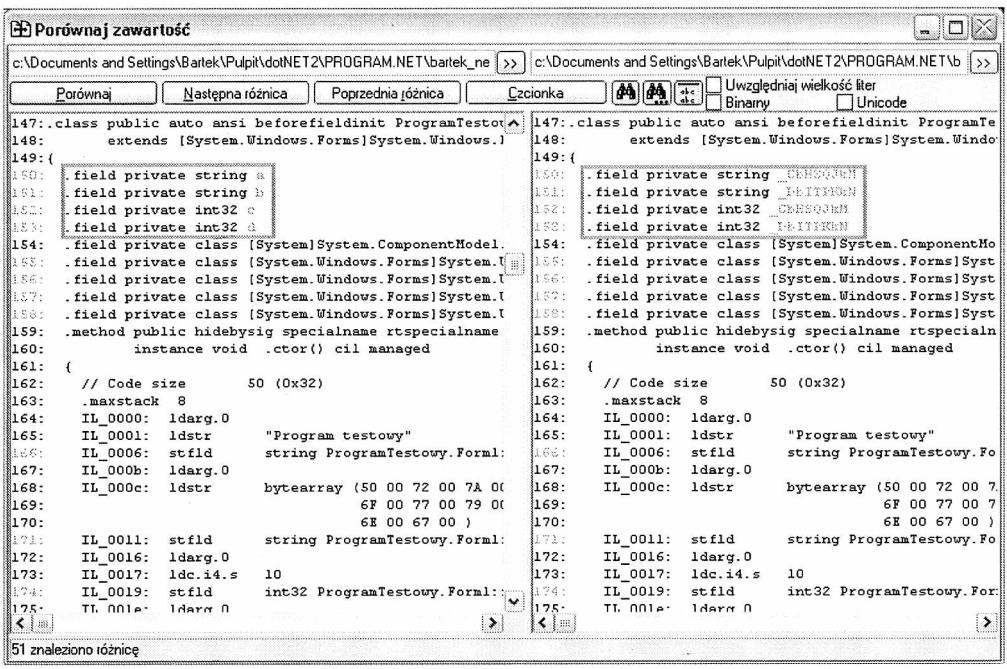


Rys. 26. Okno programu ILdasm.exe, umożliwiającego dezasemblację programów wykonywalnych do kodu pośredniego

3.3. ANALIZA ZABEZPIECZENIA

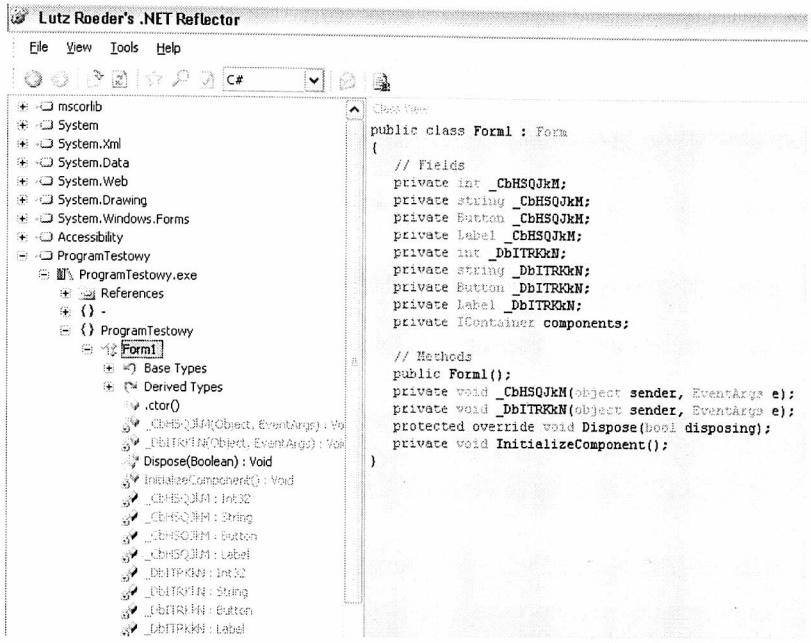
Analizę zabezpieczenia można zacząć od porównania kodu pośredniego oryginalnego programu z kodem programu zaciemnionego. Widoczne jest tu, że nazwy zmiennych zostały zastąpione ciągami znaków wygenerowanych poprzez specjalnie przygotowaną funkcję. Można również zauważyć, że druga i przedostania litera w każdym ciągu jest taka sama. Jest to podpis wykorzystany jako dodatkowe zabezpieczenie. Do dalszej analizy można wykorzystać program *Reflector*, prezentujący efektywność zaciemniania kodu.

Zaciemnienie kodu widać dokładnie w klasie inicjalizującej komponenty dla programu.

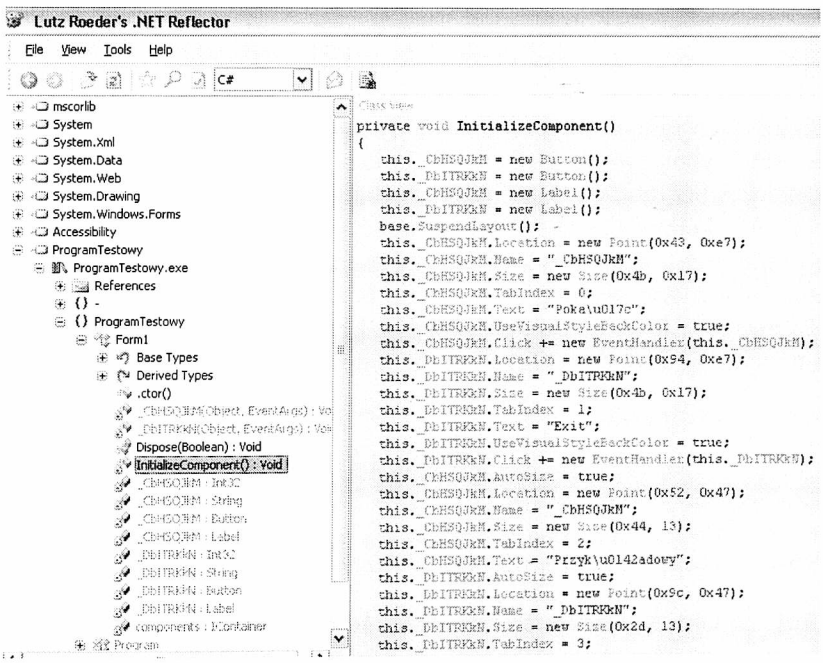


Rys. 27. Porównanie oryginalnego i zaciemnionego kodu pośredniego

W rozdziale zaproponowano dodatkową metodę zabezpieczania poprzez dodanie podpisu w ciągu znaków używanych do zaciemniania. Jest to zupełnie inne podejście do zabezpieczania oprogramowania polegające na personalizacji oprogramowania. Jeżeli zostanie podjęta próba usunięcia obfuskacji a zostanie przynajmniej jeden zaciemniony ciąg znakowy, to logicznym jest, że oprogramowanie zostało rozpowszechnianie nielegalnie.



Rys. 28. Zaciemnienie zmiennych, klas i funkcji



Rys. 29. Zaciemnienie elementów inicjalizujących

Połączenie wyżej opisanego rozwiązania z zabezpieczeniami opisanymi w poprzednich rozdziałach skutkuje utworzeniem oprogramowania z wysokim poziomem bezpieczeństwa. Stwierdzono, że pseudolosowe sygnatury z powodzeniem nadają do niedostrzegalnego umieszczania informacji o prawach autorskich do kodu źródłowego.

4. PODSUMOWANIE

W ramach pracy przeprowadzono analizę modeli zabezpieczeń oprogramowania przeniesionych na platformę .NET. Badania przeprowadzone zostały na programach zaimplementowanych w języku C#. Wykonano badania typowych metod zabezpieczania programów, takich jak: weryfikacji numeru seryjnego, weryfikacji numeru seryjnego i nazwy, weryfikacji pliku klucza, weryfikacja plikowego licznika uruchomień, weryfikacja rejestrowego licznika uruchomień, weryfikacja daty instalacji w rejestrze systemowym, identyfikacji klucza sprzętowego.

W wyniku przeprowadzonej analizy zabezpieczeń stwierdzono, że wykonanie dobrego zabezpieczenia oprogramowania realizowanego w języku C# z wykorzystaniem klasycznych metod nie jest łatwym zadaniem. Udowodniono, iż programy pisane pod platformę .NET nie dostarczają metod zabezpieczania oprogramowania na wystarczającym poziomie. W ostatnim rozdziale pracy zaproponowano nietypowe podejście do zabezpieczania oprogramowania, polegające na umieszczeniu niewidocznej sygnatury kodu programu podczas procesu obfuskacji. Wykonano program realizujący zaciemnianie kodu z dodatkowym zabezpieczeniem polegającym na podpisaniu aplikacji.

Pseudolosowe sygnatury zaproponowane w pracy mogą służyć do zabezpieczania praw autorskich kodu programu. Podejście takie jest możliwe w przypadku gdy oprogramowanie jest przygotowywane indywidualnie dla każdego klienta. Podczas procesu zaciemniania kodu pośredniego generowane są pseudolosowe ciągi znaków zawierające niewidoczny podpis. Stwierdzono, że pseudolosowe sygnatury z powodzeniem nadają się do niedostrzegalnego umieszczania informacji o prawach autorskich do kodu źródłowego. Efekty otrzymane w wyniku realizacji pracy mogą służyć do opracowania nowych typów zabezpieczeń oprogramowania.

LITERATURA

[Cerven 2001] Cerven P., *Cracking jak się przed nim bronić*. Warszawa, MIKOM, 2001.

[Kaczor 2004] Kaczor P., *Hacking, cracking, phreacking czyli ochrona przed cyberoszustami*. Warszawa, MIKOM, 2004.

[Liber 2006a] Liber A., Krawecki P., Otremba K., *Modelowanie zabezpieczeń programów komputerowych w systemie operacyjnym Windows*. Współczesne Problemy Informatyki, Legnica, Wydawnictwo WSM. 2006.

[Liber 2006b] Liber A., Kolackov I., Weiske K., *Modelowanie zabezpieczeń programów komputerowych za pomocą numeru seryjnego i zabezpieczania sprzętowego w systemie operacyjnym Linux.* Współczesne Problemy Informatyki, Legnica, Wydawnictwo WSM. 2006.

[Liber 2006c] Liber A., Kolackov I., Weiske K., *Modelowanie zabezpieczeń programów w systemie operacyjnym Linux w językach C/C++ i JAVA*, Współczesne Problemy Informatyki, Wydawnictwo WSM, Legnica 2006.

[Liber 2006d] Liber A., Krawecki P., Otremba K., *Pseudolosowe sygnatury programów w postaci źródłowej w języku asembler x86 i ich zastosowanie do autoryzacji zasobów w sieciach*, Współczesne Problemy Informatyki, Wydawnictwo WSM, Legnica 2006.

[Zemanem 2004] Zemanem J., *Cracking bez tajemnic*, HELION, Gliwice 2004.

PROPERTIES OF THE SOFTWARE PROTECTIONS AGAINST ILLEGAL SPREAD IMPLEMENTED ON THE .NET PLATFORM WITH USE C#

In frames of the work one introduced results of research of authors in the range of the construction of protections of computer programmes against the illegal spread. Described and examined was become by implementations of protections leaning softwares on algorithms: of the verification of the serial number, the verification of the serial number and names, the verification of the file of the key, the verification of bundle date of actuations, the verification of register meter of actuations, the verification of the date of the installation in the system – register, the identification of the hardware key. All implementations became realized on the platform .NET in the C# language. From the regard on the character of the work prepared algorithms became in the form of to a maximum simplified, particularly while to algorithms one did not introduce cryptographical liable additional securities very considerably to enlarge them power. In the final part of the work one introduced obfuscation algorithm proposed by authors.

Prace Naukowe Instytutu Informatyki
Politechniki Wrocławskiej

Zeszyt 10 2009

SOA,
aplikacje biznesowe,
usługi webowe,
koordynacja,
transakcje

Zbigniew FRYŹLEWICZ*

MECHANIZMY TRANSAKCJI W ŚRODOWISKU USŁUG WEBOWYCH

W pracy przedstawiono koncepcje architektoniczne wspierające koordynację i realizację transakcji w środowisku usług webowych, a opartych o paradygmat SOA. Szczegółowo przeanalizowano model architektoniczny opracowany w ramach prac OASIS, W3C, WS-I i zdefiniowany w specyfikacjach WS-Coordination, WS-Atomic Transaction, WS-Business Activity oraz języka BPEL.

1. WSTĘP

Architektura zorientowana na usługi (ang. *Software Oriented Architecture*, SOA) znajduje się wśród sześciu najbardziej strategicznych obszarów nowoczesnych technologii i usług informatycznych, które mają szanse na dynamiczny rozwój w ciągu najbliższych lat [Berger 2007]. Obecnie obserwuje się zmiany trendów w projektowaniu aplikacji, odchodząc od sztywnych struktur w kierunku elastycznych rozwiązań, opartych na luźno powiązanych usługach. SOA daje możliwość realizacji systemów umożliwiających tworzenie elastycznych modeli biznesowych, które pozwalają lepiej dostosowywać się do zmian zachodzących w środowisku biznesowym. Przejście od ściśle powiązanych aplikacji do luźno powiązanych usług zwiększa uniwersalność i zakres współdziałania aplikacji, czyniąc je bardziej atrakcyjnymi z punktu widzenia przedsiębiorstw.

Każda nietrywialna aplikacja biznesowa zrealizowana w środowisku usług webowych, wymaga skoordynowanej współpracy wszystkich uczestniczących w niej usług. W rozproszonym i zawodnym środowisku usługowym podstawowym mechanizmem koordynacji są protokoły transakcyjne. Współpraca czołowych producentów i dostaw-

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

ców oprogramowania zaowocowała opracowaniem standardów definiujących architekturę oraz mechanizmy niezawodnej koordynacji operacji w środowisku usług webowych. Należą do nich mechanizmy wsparcia transakcji języka BPEL [ChaSchMe 2007], koordynator opisany w specyfikacji WS-Coordination [WS-Coord] oraz dwa protokoły transakcyjne zdefiniowane w specyfikacjach WS-Atomic Transaction [WS-AT] i WS-Business Activity [WS-BT] z nim współdziałające.

Praca ma charakter analityczny. Jej zasadniczym celem jest analiza koncepcji pozwalających na realizację w systemach rozproszonych transakcji atomowych, jak i rozciągniętych w czasie transakcji biznesowych. Na tym tle przeanalizowano protokoły standaryzowane w ramach W3C [W3C, http://w3.org] i OASIS [OASIS, http://oasis-open.org], które zapewniają różne modele transakcji w środowisku usług webowych i zgodnych z paradygmatem SOA (ang. *Service-Oriented Architecture*).

2. USŁUGI SIECI WEB

Przed systemami IT wspierającymi działania współczesnych przedsiębiorstw stawiane są coraz to nowe wymagania, pozwalające na realizację nowych potrzeb. Jednym z nich jest konieczność łączenia technologii dostarczanych przez różnych dostawców. Współpraca niekompatybilnych ze sobą systemów i technologii powoduje powstawanie nadmiarowych komponentów, redundancję funkcjonalności i danych. Wzrasta także stopień złożoności, co przekłada się na awaryjność systemów. Rozwój przedsiębiorstw powoduje także konieczność ciągłego dostosowywania zbudowanych wcześniej aplikacji, by mogły sprostać zmieniającym się potrzebom i dostosować się do otoczenia biznesowego. Architekturą pozwalającą na realizację tych wymagań, jednocześnie eliminującą problemy integracji oprogramowania różnych dostawców, a także zapewniającą wymianę danych pomiędzy odmiennymi systemami oraz odpowiednią elastyczność przy równocześnie wysokim poziomie bezawaryjności jest architektura oparta o paradygmat SOA. Architektura SOA jest realizowana na bazie technologicznej usług XML sieci Web (ang. *Web Services*), w skrócie usług webowych [FrySal 2008].

Usługi webowe pozwalają na utworzenie pomostu łączącego różniące się od siebie aplikacje biznesowe, umożliwiając bezpieczną ekspozycję logiki biznesowej poza obszar zamkniętych systemów. Główną cechą usługi webowej – podkreślaną w różnych jej definicjach – jest to, że tworzy zwarty, samodokumentujący się komponent programowy, który może być wyszukiwany, publikowany i wywoływany z wykorzystaniem protokołów internetowych.

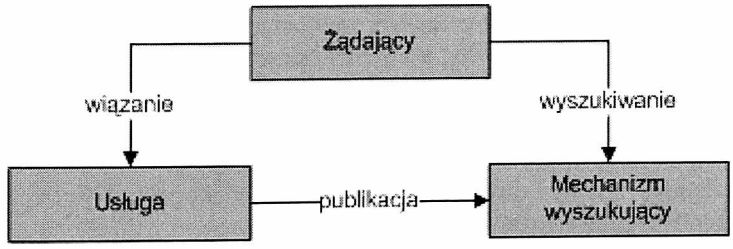
Komunikacja z usługą webową wymaga luźnego wiązania pomiędzy odbiorcą a dostawcą usługi, to znaczy, że usługa może być używana i integrowana z różnymi odbiorcami, a szczegóły jej implementacji, takie jak język programowania czy platforma sprzętowo-programowa są ukryte przed zewnętrznymi aplikacjami. Tym samym usługa staje się bardziej uniwersalna i może być wykorzystywana przez wielu

odbiorców, a dodanie kolejnych usług nie powoduje konieczności wprowadzenia zmian. Usługi webowe dają możliwość budowania zestawu pojedynczych elementów – usług, z których, jak z „klocków", projektanci mogą składać potrzebne w danej chwili złożone aplikacje. Komunikacja pomiędzy nimi realizowana jest poprzez wymianę komunikatów, którą wspiera rozbudowana warstwa pośrednia zaopatrzona w odpowiednie adaptery. To one obudowują współdziałające elementy, przysłaniając szczegóły implementacyjne. Dodatkowo z samym kanałem, łączącym dostawcę usługi z jej odbiorcą, można związać wiele cech dotyczących jakości usługi transportowej i transformacji formatu przesyłanych wiadomości. Możliwe jest to za sprawą protokołu SOAP [WeeCurLey 2005], który stanowi podstawową infrastrukturę komunikacyjną dla usług webowych.

2.1. ARCHITEKTURA USŁUGOWA

Z biznesowego punktu widzenia architektura SOA, to kompozycja funkcji, zdefiniowanych jako usługi posiadające dobrze określony interfejs, których wywoływanie pozwala na realizację złożonego procesu biznesowego [MarSha 2007]. Można powiedzieć, że koncepcja ta polega na wydzieleniu najczęściej wykorzystywanych fragmentów logiki aplikacji i nadania im formy niezależnych usług dostępnych w sieci. Z technicznego punku widzenia, SOA to elastyczna, łatwo adaptowalna i modyfikowalna architektura systemu rozproszonych obiektów, pozwalająca na ich luźne powiązania, pod którym rozumie się wykluczenie jakiejkolwiek wiedzy i założeń dotyczących platform, narzędzi i szczegółów implementacyjnych (dostawców i odbiorców usług). Dzięki tym cechom możliwa jest szybka reakcja na zmiany zachodzące w wymaganiach biznesowych. Wystarczy bowiem – w przypadku pojawienia się nowego wymagania biznesowego – stworzyć nowy, być może niewielki komponent i udostępnić go jako nową usługę webową.

SOA określa trzy główne komponenty środowiska, w jakim powinny funkcjonować usługi: dostawca usługi (ang. *Service Provider*), odbiorca usługi (ang. *Service Requester*) i rejestr usługi (ang. *Service Registry*). Ujęcie koncepcyjne architektury SOA przedstawiono na rysunku 1.



Rys. 1. Ujęcie koncepcyjne architektury SOA

Dostawca jest odpowiedzialny za udostępnienie usługi, kontrolę dostępu, a także wytworzenie jej abstrakcyjnej definicji i opublikowanie jej tak, aby umożliwić przyszłym odbiorcom uzyskanie szczegółowej informacji o samej usłudze i sposobach komunikacji z nią. Potencjalni odbiorcy, stosując odpowiednie narzędzia, odszukują usługi, które potem mogą wykorzystać. Opis usługi jest przeznaczony do przetwarzania maszynowego i jest zawarty w dokumencie WSDL (ang. *Web Services Description Language*) [WSDL].

W przypadku, gdy opis zdefiniowany w dokumencie WSDL jest niewystarczający, to może być rozszerzany zgodnie ze specyfikacją WS-Policy Framework, która standaryzuje sposób definiowania i dołączania tzw. zasad polityki. Opis usługi może być dodatkowo uzupełniony o jej charakterystykę biznesową. Jest on dołączany do opisu WSDL i umieszczany w rejestrach UDDI (ang. *Universal Description Discovery and Intergration*) [UDDI]. UDDI jest specyfikacją, która umożliwia publikację, rejestrację i wyszukiwanie usług webowych w uniwersalnym rejestrze usług. Znając lokalizację „centralnego" rejestru UDDI potencjalni odbiorcy mogą go przeszukiwać stosując różne kryteria.

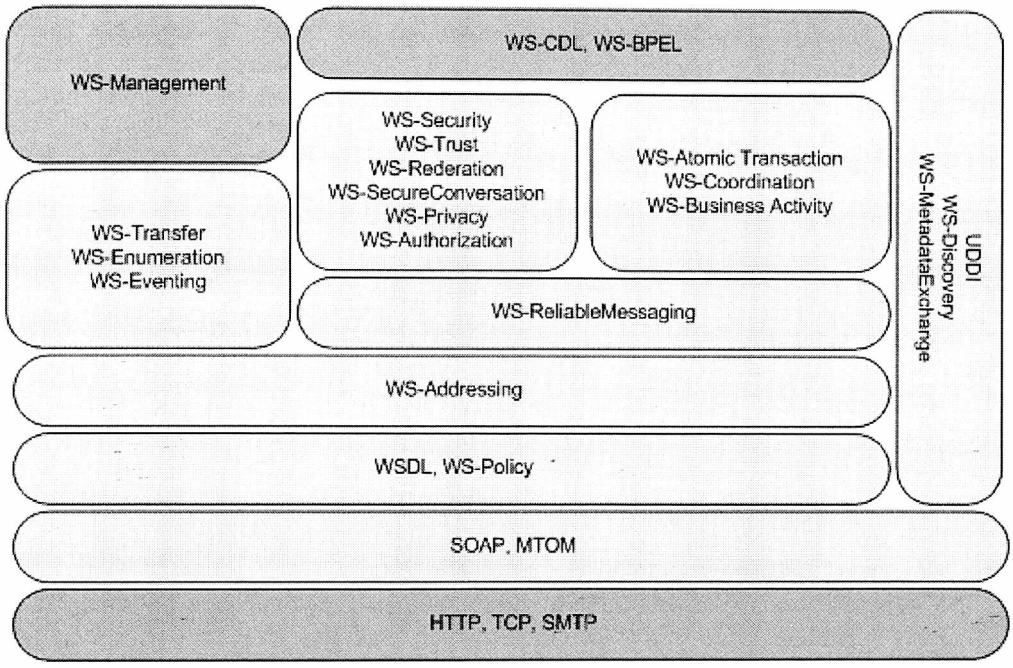
2.2. STANDARDY WS-*

Technologie typu SOAP rozwiązują problem „opakowania" informacji, protokół HTTP odpowiedzialny jest za ich transport, a opis w formie dokumentu WSDL pozwala zautomatyzować tworzenie kodu dla programów klienckich. To pozwala jednak na realizacje najprostszych usług webowych, które nie rozwiązują takich problemów jak bezpieczeństwo, niezawodne dostarczanie wiadomości, czy transakcyjność. Aby móc je rozwiązać i wykorzystać usługi webowe dla zastosowań biznesowych wprowadzono dodatkowe protokoły, zwane umownie stosem WS-* [FrySal 2008]. Ich standaryzacja jest prowadzona w ramach kilku konsorcjów, przede wszystkim W3C i OASIS. Sam proces standaryzacji jest dość złożony, wymaga powiem akceptacji wszystkich większych uczestników rynku informatycznego. Dodatkowo, sama przydatność opracowanych standardów weryfikowana jest przez rynek, co nie zawsze kończy się szerokim zastosowaniem opracowanych standardów. Przykład, to umiarkowane powodzenie standardu UDDI, a właściwie zaniechanie jego stosowania poza zasięg pojedynczej korporacji. Nadmiar i niekompatybilność poszczególnych protokołów rozwijanych w ramach W3C i OASIS spowodowały konieczność ich selekcji i wybrania zestawu protokołów interoperacyjnych oraz zdefiniowania ich użytecznych profili. Oba te działania są realizowane w ramach organizacji WS-I [WS-I, http://ws-i.org].

Efektem działań wymienionych ciał standaryzacyjnych jest obszerny zestaw protokołów o architekturze przedstawionej na rysunku 2.

Najniższa warstwa dostarcza usługi transportowe, pozwalające na komunikację pomiędzy usługami a ich klientami. Może być ona realizowana przy wykorzystaniu

różnych protokołów takich jak na przykład HTTP (najczęściej używany), TCP, SMTP etc. Co ważne, protokół warstwy transportowej jest niewidoczny dla warstwy komunikacyjnej opartej na SOAP. Protokół SOAP w łatwy sposób może zostać rozszerzony o mechanizmy przesyłania załączników binarnych (zgodnie ze specyfikacją MTOM), czy adresowanie punktów komunikacyjnych (zdefiniowane w [WS-Addr]).



Rys. 2. Warstwowa architektura usług webowych [FrySal 2008]

Kolejna warstwa jest odpowiedzialna za opis usług webowych i jej zasadniczą składową jest WSDL. Opis zawarty w dokumencie WSDL może zostać rozszerzony i powiązany z WS-Policy. Ten ostatni standard wskazuje, w jaki sposób funkcjonalny opis może zostać poszerzony o dowolne informacje i zasady (zwane politykami) wymagane podczas korzystania z usług. Przykładowo, dołączona do opisu WSDL polityka, może określać konkretny mechanizm zastosowany do szyfrowania przesyłanych wiadomości.

Powyżej znajdują się specyfikacje zgrupowane pod względem funkcjonalności w trzy bloki. Są to transfer i zarządzanie zasobami, bezpieczeństwo i transakcyjność. Dodatkowo muszą one współdziałać ze standardami służącymi do opisu metadanych, które zawierają informacje o możliwościach i ograniczeniach odkrywanych przez klientów usług. Oprócz pozyskiwania informacji z „centralnego" rejestru usług (UDDI) możliwe jest ich pozyskiwanie bezpośrednio z punktu komunikacyjnego danej usługi, pomijając w ten sposób pośredników. Ten mechanizm opisuje specyfi-

kacja WS-MetadataExchange. Nieco inne rozwiązanie oferuje standard WS-Discovery, który umożliwia dynamiczne odnajdywanie usług w środowisku rozproszonym.

WS-Transfer i specyfikacje powiązane zapewniają mechanizmy tworzenia, aktualizacji, pobierania i usuwania XML-owej reprezentacji dowolnych zasobów w środowisku usług webowych.

Kolejny blok dotyczy bezpieczeństwa. Można go podzielić na dwie grupy: zabezpieczeń podstawowych, specyfikacja WS-Security, oraz zaawansowanych, zdefiniowanych przez pięć pozostałych specyfikacji. Pierwsza z nich przedstawia ogólny model zabezpieczeń i opisuje architekturę ramową bezpieczeństwa, dostarczając potrzebnych mechanizmów autoryzacji i uwierzytelniania. WS-Trust opisuje model i mechanizmy budowy relacji zaufania pomiędzy dwoma partnerami interakcji. Kolejne ze specyfikacji przedstawiają mechanizmy ustawiania kontekstu bezpieczeństwa i zapewnienia długotrwałej, bezpiecznej interakcji (WS-SecureConversation), a także złożone modele federacji relacji zaufania (WS-Federation).

Ważnym aspektem współpracy pomiędzy partnerami jest zapewnienie niezawodności. Specyfikacja WS-Reliable Messaging określa architekturę, która umożliwia budowę modułowego protokołu w celu niezawodnego przesyłania wiadomości pomiędzy partnerami. Inne standardy pozwalające zapewnić niezawodność na poziomie architektury, to WS-Coordination, WS-Atomic Transaction i WS-Business Activity, które specyfikują protokoły wpierające transakcje. Te trzy protokoły przeanalizowano szczegółowo w dalszej części artykułu na tle ogólniejszych wymagań zwanych właściwościami ACID [CouDolKin 2005, GarUllWid 2006].

Na najwyższym poziomie umiejscowiono standardy pozwalające na zarządzanie zasobami i modelowanie procesów biznesowych. Jak już wcześniej nadmieniono, usługi webowe mogą być komponentem atomowym lub elementem składowym większego procesu biznesowego. Aby umożliwić zarządzanie szeregiem powiązanych usług wymagane są pewne mechanizmy wpierające kompozycję. Aktualnie dostępne są dwa rozwiązania opisane w specyfikacjach WS-BPEL i WS-CDL.

Modułowa budowa architektury, rysunek 2, pozwala aplikacjom biznesowym wykorzystywać tylko część specyfikacji znajdujących się w warstwach niższych. Tym samym mogą korzystać z wybranych standardów w zależności od wymaganej funkcjonalności, które nawzajem się uzupełniają. W przypadku konieczności rozszerzenia funkcjonalności, ich zbiór może być rozszerzony o nowe specyfikacje. Przykładowo, jeden z systemów może wymagać wykorzystania mechanizmów zdefiniowanych przez WS-Security i pokrewnych, inny mechanizmów transakcyjności zdefiniowanych w WS-BusinessActivity, a jeszcze inny tylko z funkcjonalności zdefiniowanej przez WS-Transfer.

Udostępnienie funkcjonalności istniejących aplikacji w postaci usług webowych umożliwia użytkownikom tworzenie nowych, silniejszych aplikacji, które wykorzystują usługi jako swoje składniki. Wykorzystanie stosu protokołów WS-* pozwala z sukcesem wykorzystać usługi webowe w zastosowaniach biznesowych.

3. KOORDYNACJA I TRANSAKCJE W ŚRODOWISKU USŁUG WEBOWYCH

Podobnie jak w przypadku scentralizowanego systemu, tak i rozproszonego, transakcje powinny odnosić się do odtwarzalnych danych i z reguły być niepodzielne (atomowe). Kiedy operacja zostanie zakończona pomyślnie, jej skutki muszą być trwale zapisane. W przeciwnym przypadku, wymagane jest odtworzenie stanu sprzed rozpoczęcia transakcji. Aby transakcja spełniała powyższe wymagania powinna realizować postulaty zaproponowanej przez Haidera i Reutera [GraReu 1993] zasady ACID, czyli: atomowość (ang. *atomicity*), spójność (ang. *consistency*), izolację (ang. *isolation*) i trwałość (ang. *durability*).

Atomowość oznacza, że operacje składowe wchodzące w skład transakcji są wykonane w całości lub nie są wykonywane wcale. Spójność wymaga od transakcji, aby przeprowadzała bazę z jednego stanu spójnego przed transakcją, do innego stanu spójnego po zatwierdzeniu transakcji. Izolacja gwarantuje niezależne wykonanie się współbieżnych transakcji w taki sposób, którego efekt jest taki sam, jakby były wykonywane w sposób szeregowy. Stopień separacji współbieżnych transakcji może być różny i jest określany jako tzw. poziom izolacji. Trwałość transakcji oznacza, że po pozytywnym jej zakończeniu wyniki muszą być trwale zapisane.

W systemach rozproszonych zagwarantowanie powyższych własności nie jest zadaniem tak łatwym jak w przypadku systemu scentralizowanego, dlatego też wprowadzono specjalne mechanizmy ułatwiające przetwarzanie rozproszonych transakcji spełniających własności ACID. Modele i mechanizmy zapewnienia transakcji proponowane i rozwijane pierwotnie w różnych systemach rozproszonych mają swoje odbicie w specyfikacjach i standardach OASIS, W3C oraz WS-I a dotyczących transakcji atomowych i biznesowych w środowisku usług webowych.

3.1. TRANSAKCJE W SYSTEMACH ROZPROSZONYCH

Mechanizmem zaproponowanym do realizacji transakcji rozproszonych jest technika dwufazowego zatwierdzenia (ang. *2-phase commit*) [BerNew 1997]. Mechanizm opiera się na realizacji operacji zatwierdzania w dwóch fazach. W pierwszej z nich, określanej jako faza przygotowania (ang. *prepare*), systemy biorące udział w transakcji przygotowują się do zatwierdzenia lokalnych części transakcji rozproszonej (globalnej). W momencie gdy wszyscy uczestnicy transakcji potwierdzą swą gotowość do zatwierdzenia transakcji, realizowana jest druga faza, określana mianem fazy rzeczywistego zatwierdzania (ang. *commit*). Cały ten proces musi być koordynowany. Koordynator, działając w imieniu całej rozproszonej aplikacji, jest odpowiedzialny za komunikację ze wszystkimi uczestnikami i ustalenie jednoznacznego wyniku transakcji.

W ustaleniu wyniku biorą udział wszyscy uczestnicy. W momencie gdy koordynator, w odpowiedzi na wysłane żądania przygotowania się, otrzyma od wszystkich uczestników gotowość do zatwierdzenia transakcji, ogłasza pozytywny wynik głosowania i wysyła żądanie trwałego zatwierdzenia zmian.

Jeżeli jednak w wyniku fazy pierwszej koordynator otrzyma chociażby jedną odpowiedź negatywną, cała transakcja nie może zostać zatwierdzona i w fazie drugiej wysyłana jest wiadomość o wycofaniu zmian.

W wielu przypadkach, technika dwufazowego zatwierdzenia jest rozwiązaniem blokującym, gdyż w wielu miejscach wykonywania dwufazowego zatwierdzania procesy oczekują na komunikaty, które z różnych przyczyn (np. awaria, opóźnienia w sieci) mogą dotrzeć z dużym opóźnieniem lub w ogóle nie dotrzeć. Na wypadek takich ewentualności stosuje się mechanizm anulowania zadań po upływie określonego czasu (ang. *timeout*), tym samym unika się nieskończenie trwającej blokady procesów, gdyż oczekiwanie procesów po upływie tego czasu jest przerywane i proces musi podjąć pewną akcję (ang. *timeout action*).

3.2. DŁUGOTRWAŁE TRANSAKCJE BIZNESOWE

Procesy biznesowe często są uruchamiane przez długi okres czasu, czasami liczony nawet w dniach lub tygodniach. Wykorzystanie transakcji o cechach ACID w przypadku tak długich procesów niesie ze sobą kilka problemów. Jednym z nich jest próba zachowania izolacji, która musiałaby spowodować zablokowanie zasobów przez dłuższy czas, co niemal uniemożliwia równoczesne wykonywanie akcji na pojedynczym zasobie. Izolacja określa, że środki wykorzystywane w transakcji są zablokowane, dopóki nie zostanie ona zatwierdzona w całości lub wycofana. Oprócz blokowania, długotrwałe procesy biznesowe mając dostęp do wielu zasobów mogą powodować wzrost zakleszczeń. Biorąc pod uwagę niepodzielność wymaganą przez ACID, błędy wewnątrz instancji procesu biznesowego powodują, iż cała transakcja (całego procesu biznesowego), musi zostać wycofana. W przypadku długotrwałych procesów może to oznaczać wycofanie zadań wykonanych podczas tygodni trwania procesu, niwelując w ten sposób wyniki długotrwałych czynności. W celu przezwyciężenia niedoskonałości wymagań ACID dla procesów długotrwałych proponuje się kilka rozwiązań, takich jak transakcje zagnieżdżone czy też sagi.

3.2.1. TRANSAKCE ZAGNIEŻDŻONE

W poprzednim punkcie opisano tzw. płaskie transakcje (ang. *flat transaction*), czyli takie, które nie posiadają wewnętrznej struktury. Atomowość i synchronizacja opiera się na całej transakcji, dlatego też albo cała transakcja zostanie zatwierdzona, albo cała zostanie wycofana.

Istnieją jednak rozwiązania, które dopuszczają wykonywanie transakcji wewnątrz innych. Posiadają one, niezależny od zewnętrznej transakcji, swój początek i koniec, mogąc również niezależnie od zewnętrznej transakcji być zatwierdzane lub wycofywane [GraReu 1993]. Rozwiązanie takie nosi nazwę transakcji zagnieżdżonej (ang. *nested transaction*). Wewnątrz nich mogą być umieszczone inne transakcje, zwane podtransakcjami (ang. *subtransactions*), w których mogą znajdować się także inne transakcje. W ten sposób można konstruować zagnieżdżone transakcje o wielu poziomach zagnieżdżeń.

Koncepcję zagnieżdżonych transakcji warto stosować, w przypadku gdy mamy do czynienia z transakcją zawierającą wiele operacji. Grupy podtransakcji mogą być na przykład podzielone względem operacji, które mogą być równocześnie wykonywane lub które odzwierciedlają strukturę modułową transakcji.

3.2.2. ZAMKNIĘTE TRANSAKCJE ZAGNIEŻDŻONE

W tak zwanych zamkniętych transakcjach zagnieżdżonych (ang. *closed nested transaction*) [CouDolKin 2005] podtransakcje rezygnują z trwałości rozumianej jako jednej z własności transakcji ACID. Po poprawnym wykonaniu operacji, podtransakcja nie dokonuje zatwierdzenia widocznego na zewnątrz, lecz odbywa się tzw. lokalne zatwierdzanie (ang. local commit). Oznacza to, że wyniki stają się widoczne dla rodziców transakcji, ale nie dla innych transakcji spoza zagnieżdżonej transakcji. W przypadku, gdy główna transakcja (będąca korzeniem w reprezentacji transakcji zagnieżdżonej w postaci drzewa) jest gotowa do zatwierdzenia (czyli wszystkie jego podtransakcje zatwierdzono lokalnie), podtransakcje moją zostać zatwierdzone ostatecznie i trwale, a tym samym cała transakcja zagnieżdżona zostaje zatwierdzona. Jeżeli któraś z transakcji z drzewa transakcji zostanie przerwana, wszystkie jej transakcje zagnieżdżone również zostają przerwane.

Zagnieżdżone transakcje pozwalają na równoległe przetwarzanie podtransakcji na różnych maszynach. Ponadto, jeżeli wystąpi błąd w jednej z podtransakcji, to tylko ta podtransakcja (wraz z transakcjami zagnieżdżonymi w niej) jest przerywana. Następnie rodzic w drzewie transakcji, w którym wystąpił błąd w podtransakcji decyduje o obsłudze błędu. Obsługa może polegać na zignorowaniu przerwanej podtransakcji lub niezatwierdzeniu całej transakcji zagnieżdżonej. Takie własności transakcji zagnieżdżonych mogą być szczególnie przydatne w przypadku transakcji procesów biznesowych, ponieważ pojedyncze niepowodzenie jakiejkolwiek operacji w ramach procesu biznesowego, nie oznacza konieczności, powrotu do punktu wyjściowego i wycofania wszystkich wyników pomyślnie wykonanych czynności.

3.2.3. OTWARTE TRANSAKCJE ZAGNIEŻDŻONE

Otwarte transakcje zagnieżdżone (ang. *open nested transactions*) [EliMos 2006] są podobne do zamkniętych transakcji zagnieżdżonych, w odróżnieniu jednak do nich nie

gwarantują izolacji. W zamkniętych transakcjach zagnieżdżonych lokalne zatwierdzenie podtransakcji powoduje, iż wyniki widoczne są tylko dla rodzica transakcji. To oznacza, że podtransakcje tej samej transakcji - rodzic mogą korzystać z różnej wersji tego samego źródła. Pozostałe transakcje spoza zagnieżdżonej transakcji muszą czekać na dostęp do zasobów do momentu, aż transakcje zagnieżdżone zostaną zatwierdzone lub wycofane. Koncepcja otwartych transakcji zagnieżdżonych, w odróżnieniu do zamkniętych transakcji zagnieżdżonych, zapewnia, iż wyniki podtransakcji są widoczne od razu po lokalnym zatwierdzeniu zmian. Zaletą takiego rozwiązania jest to, że powyższe transakcje mogą korzystać z zasobów, w których podtransakcje innych transakcji zagnieżdżonych dokonały lokalnego zatwierdzenia (a nie ostatecznego, całej transakcji zagnieżdżonej). Wadą takiego rozwiązania jest utrata zakresu izolacji. W rezultacie możliwy jest tzw. brudny odczyt. Dodatkowo w przypadku, gdy transakcja zagnieżdżona kończy się niepowodzeniem, na wynikach podtransakcji muszą zostać wykonane odpowiednie akcje kompensacyjne.

3.2.4. TRANSAKCJE KOMPENSACYJNE

W przypadku transakcji zagnieżdżonych, czasami wymagane jest wykonywanie pewnych czynność mających za zadanie wycofanie zmian jakie zaszły podczas zatwierdzonej już transakcji. Jednak zgodnie z własnością ACID, modyfikacje wprowadzone w ramach transakcji muszą być trwale zapisane. A tym samym pojawia się problem z ich wycofaniem. Rozwiązanie tego problemu stanowią tzw. transakcje kompensacyjne. Transakcja kompensacyjna jest zbiorem operacji odwrotnych w stosunku do operacji wykonanych podczas zatwierdzonej transakcji. Tym samym transakcje kompensacyjne odwracają skutki wykonanych transakcji. Jeżeli np. zatwierdzona transakcja miała za zadanie kupno książki w sklepie internetowym, to transakcja kompensacyjna polega na rezygnacji z zamówienia i wycofaniu wszystkich czynności z tym związanych, np. anulowanie płatności.

3.2.5. SAGI

Podana koncepcja wspiera długotrwałe transakcje (LLT) i odszkodowania. Sagi pozwalają zdefiniować rekompensatę dla każdej podtransakcji. Saga jest transakcją zagnieżdżoną zawierającą jeden poziom podtransakcji. Podobnie jak w otwartych transakcjach zagnieżdżonych, zasoby są dostępne tuż po zatwierdzeniu zmian przez podtransakcję. Ta koncepcja zakłada również, że jako podtransakcje mogą być wykorzystywane tylko przez takie działania, których wyniki mogą być widoczne przed końcem całej transakcji. W przypadku przerwania sagi, transakcje kompensacyjne są wywoływane dla wszystkich podtransakcji wykonanych przed przerwaniem. Transakcje kompensacyjne dla zakończonych transakcji wykonywa-

ne są w kolejności odwrotnej do zakończonych transakcji [CouDolKin 2005, StrKar 2002].

Na wypadek awarii, sagi zapisują stan po każdej zatwierdzonej transakcji. Dzięki takiemu rozwiązaniu, w przypadku jakiejkolwiek awarii, sagi mają przywrócić najnowszy, trwale zapisany punkt, w którym była transakcja przed awarią i kontynuować pracę.

3.2.6. PODSUMOWANIE

Tego typu transakcje nie zachowują wszystkich cech ACID. Dokładnie mówiąc ograniczenia związane z izolacją, trwałością i atomowością są luźniejsze, nie tak restrykcyjnie wymagane, jak w przypadku normalnych transakcji. Dzięki takiemu rozluźnieniu można zachować dużą współbieżność równolegle wykonywanych operacji.

3.3. KOMPOZYCJA USŁUG

Po fazie tworzenia aplikacji na podstawie komponentów nadchodzi era systemów będących zbiorem usług. Sama koncepcja aplikacji jako zestawu usług nie jest nowa. Najwięksi producenci oprogramowania przywołują ją od kilku lat w kontekście usług webowych (ang. Web Services). Idea jest prosta, użytkownik określa jakie usługi są mu potrzebne i na podstawie tych informacji wybiera z katalogu usługodawców tego, który dane najlepszą gwarancję wykonania zadania spełniającego jego potrzeby. W rzeczywistości jednak sprawa jest bardziej skomplikowana. Usługa webowa nie musi być bowiem atomowa. Może składać się z wielu innych usług stając się usługą złożoną. Aktualnie wyróżnia się głównie dwa modele kompozycji usług: orkiestrację i choreografię. Pojęcie choreografii serwisów często bywa mylone z terminem orkiestracji. Oba bowiem dotyczą tego samego: modelowania współpracy usług webowych zorientowanej na realizację określonego procesu biznesowego. Definiują jednak odmienne podejścia do tego zagadnienia.

3.1.1. 3.3.1. CHOREOGRAFIA USŁUG

Logika wykonywania zadań w modelu choreografii (ang. *choreography*) [JurMatSar 2006] jest rozproszona pomiędzy wszystkie usługi. Brak jest wyróżnionej nadrzędnej jednostki, która by centralnie koordynowała współpracę usług. Każda usługa zaangażowana w realizację określonego zadania, wie dokładnie jakie operacje i kiedy musi wykonać. Wszyscy uczestnicy muszą zdawać sobie sprawę w jakim procesie biorą udział, jakie czynności muszą wykonać, jakie informację muszą przesłać partnerom biorącym udział w realizacji zadania i kiedy mają to zrobić. Cały opis procesu bizne-

sowego jest zbudowany na sekwencji komunikatów, które są wymieniane pomiędzy poszczególnymi usługami. Aktualnie trwają intensywne prace nad językiem choreografii, który nosi nazwę WS-CDL.

3.3.2. ORKIESTRACJA USŁUG

Podejście zaprezentowane powyżej, niesie jednak ze sobą pewien problem. Dodanie nowej usługi wymaga dostarczenia jej informacji o pozostałych usługach biorących udział w realizacji zadania. Również pozostałe usługi muszą otrzymać informację o nowej usłudze. Dodatkowo, usługa posiadając zaszytą w sobie wiedzę o procesie biznesowym, nie może być wykorzystane przez inne procesy w ten sam sposób. Aby tworzenie zadań realizowanych przez zbiór usług było tanie i elastyczne, a także bezkonfliktowe, samo zadanie nie powinno wpływać na wewnętrzną strukturę biorących w nim udział usług. Dzięki temu nowo powstający proces biznesowy, realizujący konkretne zadanie, może wykorzystywać komponenty powstałe wcześniej w celu spełnienia innych wymagań.

Takie zalety ma drugi z modeli, mianowicie orkiestracja (ang. *orchestration*) [Erl 2005, JurMatSar 2006]. Orkiestracja opiera się na centralnym koordynatorze, którzy przechowuje całą logikę procesu biznesowego i zarządza wywołaniami poszczególnych usług. Centralnym koordynatorem jest sam proces, który przechowuje logikę przetwarzania i pilnuje, aby wszystkie wywołania usług były wykonywane w odpowiedniej dla realizowanego zadania kolejności, realizując także ewentualną synchronizację. Same usługi biorące udział w wykonywaniu zadania nie muszą posiadać szczegółowej wiedzy o pozostałych usługach partnerskich i całym procesie biznesowym, w którym biorą udział.

Podstawowym językiem kompozycji usług według modelu orkiestracji jest język BPEL. Od wersji 2.0 znany jest pod nazwą WS-BPEL.

3.4. BPEL

BPEL jest deklaratywnym językiem, służących do opisu wykonania procesów biznesowych, korzystających z usług webowych. Umożliwia realizację architektury SOA za pomocą technik kompozycji i koordynacji usług webowych. Dzięki użyciu języka BPEL, proste komponenty usług webowych mogą być wykorzystane do zbudowania usług złożonych, zwanych procesami biznesowymi. Definicja procesu w języku BPEL precyzuje kolejność, w jakiej wykonywane są szeregowe lub równoległe usługi webowe. Możliwe jest także definiowanie zachowań warunkowych, a wyniki uzyskane przez jedną usługę mogą być wykorzystane jako argumenty wywołania innej.

W celu opisania zadań, które mają zostać wykonane w ramach procesu BPEL wykorzystuje się dwa różne rodzaje aktywności, tj. aktywności proste i złożone.

Aktywności proste (atomowe) są wykorzystywane do manipulowania zmiennymi oraz do wymiany komunikatów pomiędzy procesem i jego partnerami. Przykładem takiej aktywności jest np. `assign`, która umożliwia kopiowanie danych pomiędzy zmiennymi, przypisanie zmiennym wartości wyrażeń lub skopiowanie referencji punktu komunikacyjnego do partner link. Aktywnością atomową jest również aktywność `invoke` wykorzystywana do wywołania operacji oferowanych przez usługę – partnera (komponentu Web Services), czy też `receive` i `reply`, które określają oczekiwanie na wywołanie procesu biznesowego przez partnera i wygenerowanie odpowiedzi na wywołanie.

Aktywności złożone umożliwiają kompozycję aktywności. Mogą one grupować jedną lub wiele aktywności prostych i złożonych w nową aktywność, która z kolei może być wykorzystywana podczas tworzenia kolejnych, nowych aktywności. Przy użyciu takiej aktywności można zdefiniować sekwencyjny przepływ (aktywność `sequence`) bądź równoległy (aktywność `flow`). Przykładami złożonych aktywności są m.in. `while` definiująca iterację, `if-elseif-esle` pozwalająca określić warunkowe zachowanie.

3.4.1. ZASIĘG

Specjalnym przykładem aktywności jest `scope` (zasięg). Definiuje kontekst dla umieszczonych w niej aktywności. Element ten przypomina blok kodu spotykany w tradycyjnych językach programowania ujęty w konstrukcjach klamrowych lub pary `try` i `catch`. Aktywności `scope` mogą definiować zmienne, które są widoczne tylko dla zagnieżdżonych wewnątrz nich aktywności. Analogicznie jak `catch` pozwalają na definicję sekcji obsługi błędów, a także mogą posiadać sekcję kompensacji i korelacji.

Proces BPEL bardzo często definiuje długotrwałe interakcje biznesowe. Wprowadzenie aktywności kompensacji i sekcji obsługi błędów pozwala udostępnić mechanizm zwany długotrwałą transakcją biznesową, w skrócie LRT (ang. *Long-Running Business Transaction*). Opiera się on na koncepcji otwartych transakcji zagnieżdżonych i sag. Z punktu widzenia transakcyjności instancja procesu biznesowego może być postrzegana jako zagnieżdżona transakcja, zawierająca kilka podtransakcji. Podtransakcje w procesie BPEL są reprezentowane przez aktywność `scope`. Ta z kolei może być transakcją rodzica dla innych podtransakcji, gdyż aktywności `scope` mogą być dowolnie zagnieżdżane. Końcowy wynik takiej transakcji (która może być reprezentowana przez instancję całego procesu biznesowego, czyli element `process`) opiera się na wynikach zagnieżdżonych transakcji (tj. na aktywnościach `scope` znajdujących się wewnątrz np. elementu `process`). W przypadku gdy jedna z zagnieżdżonych transakcji kończy się niepowodzeniem, to BPEL oferuje rozwiązanie oparte na akcjach kompensacyjnych, znanych z koncepcji sag. Umożliwia to zdefiniowanie

elastycznych akcji kompensacyjnych dla każdej aktywności `scope`, które pełnią rolę transakcji kompensacyjnych.

Ponadto element `scope` pozwala na definiowanie obsługi zdarzeń i zbiorów korelacji. Obsługa zdarzeń polega na wywołaniu zdefiniowanych zdarzeń i akcji z chwilą wystąpienia zdarzenia. W porównaniu do obsługi błędów i kompensacji, obsługa zdarzeń jest wykonywana w ramach normalnego wykonywania procesu. Zbiory korelacji, a dokładniej własności będących elementami tych zbiorów, służą do identyfikacji instancji procesów.

3.4.2. CYKL ŻYCIA PROCESU BIZNESOWEGO

Instancja procesu BPEL jest tworzona w momencie, gdy zostanie odebrana wiadomość przez aktywność `receive` (lub `pick`), która posiada atrybut `createInstance` i jest ustawiony na `true`. Cały proces biznesowy może zakończyć się na jeden z trzech sposobów. Pierwszy przypadek występuje, gdy proces kończy się w sposób normalny, określony w definicji procesu. W innym przypadku, kiedy błąd znajduje się na najbardziej zewnętrznym zakresie, określającym cały proces, mamy do czynienia z niepoprawnym zakończeniem, nawet gdy błąd byłby prawidłowo obsłużony na poziomie procesu. Proces może być również zakończony poprzez jawne wywołanie aktywności `terminate`, która powoduje zakończenie wszystkich aktualnie wykonywanych aktywności, tym samym przerywając działanie instancji procesu.

3.4.3. OBSŁUGA BŁĘDÓW

W procesie BPEL błędy mogą powstawać w różnych przypadkach:

- zgłoszenie błędu z wnętrza bieżącego procesu lub wykonywanej usługi,
- błąd zgłoszony automatycznie w wyniku nieprawidłowości wykonania, np. poprzez wygenerowanie błędu `joinFailure`.
- błędy powstałe podczas działania serwera BPEL związane np. z komunikacją (specyfikacja podaje kilka standardowych błędów).

Obsłużenie zgłoszonych błędów jest wykonywane przez zadania zdefiniowane w sekcji `faultHandler`. Może składać się z elementu `Catach` lub `catchAll` w zależności czy obsługiwane będą wszystkie wyjątki, czy tylko o określonej nazwie lub typie.

Sekcja obsługi błędów `faultHandler` może mieć zasięg dla elementu `process`, `scope` lub `invoke`. Powiązanie z elementem `scope` daje możliwość izolowania obsługi błędów tylko do zasięgu aktywności `scope`. Pozwala także na tworzenie hierarchii obsługi błędów w zagnieżdżonych konstrukcjach. Tym samym zagnieżdżony element może sam obsłużyć błąd jak również przekazać go do znajdującego się wyżej elementu (za pomocą elementu `rethrow`).

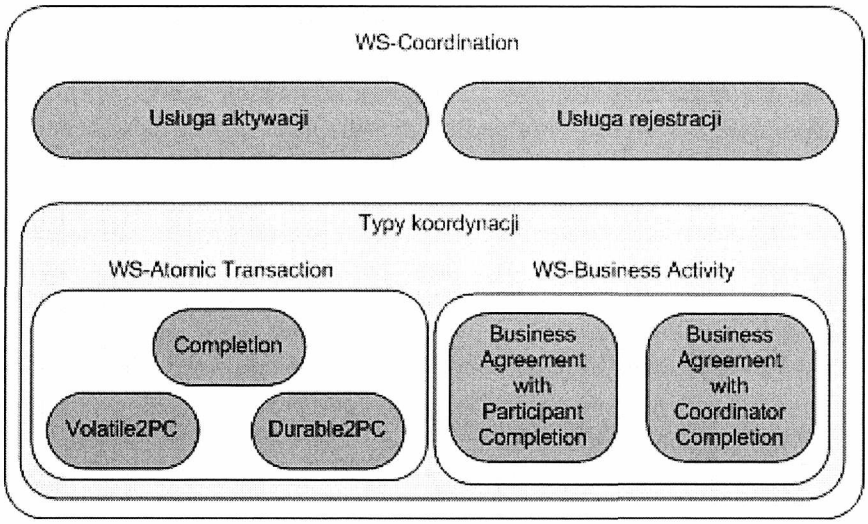
3.4.4. AKCJA KOMPENSACJI

Mechanizm kompensacji BPEL opiera się na elemencie `compensationHandler`. Zawiera on aktywność, która zostanie wykonana w celu odwrócenia efektów działania innej, powiązanej aktywności. BPEL sam w sobie nie zapewnia żadnego mechanizmu automatycznego wycofania efektów wykonanych działań, który byłby wykonywany w momencie wystąpienia błędu. Jest to związane z tym, że czynność ta jest związana z logiką wykonywanego procesu biznesowego. Dlatego też zadaniem programisty jest zapewnienie, aby w przypadku błędu, wykonano odpowiedni proces kompensacji. `CompensationHandler` może zostać wywołany za pomocą elementów `compensate` lub `compensateScope`, nazywanych aktywnościami kompensacyjnymi. Elementy te mogą zostać wykonane w obrębie sekcji obsługi błędów lub obsługi kompensacji. Dodatkowo, element `compensate` zawiera opcjonalny atrybut `scope`, który określa nazwę zakresu dla którego zostanie wykonana akcja kompensacyjna. W przypadku, gdy zostanie on pominięty wywoływane są akcje kompensacji dla wszystkich bezpośrednio zagnieżdżonych aktywności `scope`. Jeżeli w elemencie `scope` nie ma jawnie określonego elementu `compensationHandler`, wykonywane są czynności domyślne, których zachowanie opisane jest szczegółowo w specyfikacji BPEL.

Obsługa błędów w połączeniu z akcjami kompensacyjnymi pozwala, w przypadku jakiegoś błędu, na wykonanie czynności mające na celu wycofanie efektów wcześniej wykonywanych i zakończonych pomyślnie aktywności. Jedną z wad tego mechanizmu zwanego LRT jest to, że jest wykonywany tylko lokalnie, w obrębie jednego procesu biznesowego. A co za tym idzie nie możliwe jest wykorzystanie tego mechanizmu dla procesów, w których uczestniczy wiele aplikacji, często różnych organizacji i utrzymywanych na różnych platformach. W takich przypadkach zaleca się korzystanie z mechanizmów zdefiniowanych w [WS-Coord], [WS-AT] i [WS-BA].

3.5. WS-COORDINATION

Specyfikacja WS-Coordination [WS-Coord] określa ogólne założenia, które muszą być spełnione, aby mogła być realizowana koordynacja wielu usług. Ogólnie rzecz biorąc specyfikacja definiuje architekturę ramową, która umożliwia aplikacji zarządzanie, utrzymywanie i propagację informacji kontekstu pomiędzy uczestników oraz zarządzanie przetwarzaniem na zakończenie aktywności. W szczególności definiuje protokoły i usługi, które zezwalają innym usługą na rejestrację siebie jako stron, które biorą udział w realizacji jakiegoś zadania i ich identyfikację, zezwalają na wybór protokołu koordynacji, który będzie wymagany w chwili zakończenia aktywności (rys. 3).



Rys. 3. WS-Coordination – usługi i protokoły

Obecnie do najczęściej stosowanych protokołów koordynacji, które są wykonywane z chwilą końca aktywności, należą protokoły opisane w specyfikacji WS-AT i WS-BA (wsparcie dla transakcji atomowych i biznesowych). Jednak specyfikacja definiuje mechanizm koordynacji w szerszym zakresie, niż tylko wsparcie dla transakcji przy użyciu protokołów zgodnych z dwoma wcześniej wymienionymi specyfikacjami.

3.2. 3.6. WS-ATOMICTRANSACTION

Specyfikacja WS-Coordination określa pewne ramy architektoniczne dla różnych rodzajów koordynacji. WS-AtomicTransaction [WS-AT] definiuje protokoły atomowych transakcji, używanych w celu koordynacji działań typu „wszystko albo nic". Tym samym zapewnia, iż w przypadku gdy chociaż jedna z czynności wykonywana w ramach transakcji zakończy się niepowodzeniem, wszystkie zmiany dokonane w ramach transakcji nie są widoczne i trwałe. W przypadku jednak, gdy wszystkie czynności w ramach transakcji atomowej zakończą się powodzeniem, wszystkie dokonane w ramach transakcji zmiany są widoczne i trwałe. Protokoły opisane w specyfikacji WS-AT wspierają krótkotrwałe aktywności, które powinny spełniać własności ACID.

W ramach WS-AT są dostępne dwa protokoły:

- Completion – obowiązuje na styku aplikacja-koordynator i używany do zainicjalizowania końca aktywności oraz ustalenia ostatecznego wyniku (wiadomość Commit), lub wycofania całej transakcji (wiadomość Rollback) w przypadku, gdy aplikacja-inicjator nie będzie chciał zatwierdzić transakcji.

- Two-phase commit (2PC) – protokół zapewnia wszystkim jego uczestnikom udział w ustalaniu jednolitego wyniku transakcji. Składa się z dwóch etapów: przygotowania (faza głosowania) i zakończenia (ogłoszenia wyniku). WS-AT definiuje dwa typy dwufazowego protokołu zatwierdzania: Volatile2PC (używane przez uczestników mających zdolność przechowywania porcji danych w pamięci podręcznej takiej jak cache; tym samym dane są szybciej dostępne) i Durable2PC (używane przez uczestników wykorzystujących pamięć trwałą np. bazę danych).

3.7. WS-BUSINESS ACTIVITY

Specyfikacja określa protokoły transakcji biznesowych rozciągniętych w czasie. Przy takich transakcjach wymaga się aby wyniki pośrednich czynności były natychmiast widoczne, tym samym unika się kosztownego a często niemożliwego blokowania zasobów przez dłuższy czas. W przypadku, gdy mimo wcześniej zakończonych czynności i trwałych ich efektów, cała transakcja kończy się niepowodzeniem, WS-BA wykorzystuje akcje kompensacyjne, w celu wykonania pewnych czynności kompensujących i wycofania uzyskanych i widocznych już efektów pośrednich. Sama specyfikacja procesów biznesowych może nie wymagać poprawności wykonania wszystkich czynności w celu osiągnięcia poprawnego zakończenia całej aktywności. Definicja poprawnego wykonania aktywności, a także opis działań kompensacyjnych wyrażane są najczęściej za pomocą protokołów i języków kompozycji takich jak BPEL.

Standard wspiera 2 typy koordynacji:

- AtomicOutcome – ten typ koordynacji musi wskazać wszystkim uczestnikom poprawne zakończenie aktywności (wiadomość `Close`) lub konieczność wykonania czynności kompensacyjnych (wiadomość `Compensate`).
- MixedOutcome – ten typ koordynacji pozwala wskazać uczestnikom poprawne zakończenie aktywności (wiadomość `Close`) lub konieczność wykonania czynności kompensacyjnych (wiadomość `Compensate`), dodatkowo jednak pozwala na rezygnację z udziału któregoś z członków w aktywności, jeśli tylko jest to dozwolone w ramach danej aktywności.

Te dwa typy koordynacji mogą używać jednego z dwóch dostępnych protokołów:

- BusinessAgreementWithParticipantCompletion: po dokonaniu rejestracji w tym protokole uczestnik, który wykonał swoje zadanie przyjmuje na siebie odpowiedzialność powiadomienia o tym koordynatora i czekania na ewentualne dalsze polecenia.
- BusinessAgreementWithCoordinatorCompletion: protokół jest identyczny z poprzednim, z tym jednak wyjątkiem, że każde zadanie uczestnika musi czekać na wiadomość `Complete` i dopiero wtedy może zgłaszać rezultaty wszystkich swoich zadań.

4. ZAKOŃCZENIE

Problemy transakcyjności na poziomie usług webowych wprowadzają dodatkowy poziom złożoności. Tradycyjne transakcje spotykane w bazach danych zazwyczaj mają krótki okres trwania, a zatem dostęp do tabel, których dotyczy transakcja, może zostać zablokowany, dając pewność spełnienia kryteriów ACID.

W przypadku współpracujących ze sobą usług webowych transakcje trwają o wiele dłużej. Wynika to z faktu, że bardzo często współpracujące ze sobą usługi, wykonywane są w różnych środowiskach i poza obrębem jednej organizacji, a co za tym idzie wymagana jest komunikacja sieciowa pomiędzy nimi, co wydłuża samą transakcję. Same usługi mogą również być wykonywane przez wiele godzin, dni czy tygodni. Dlatego też zastosowanie dwufazowego zatwierdzania, które jest rozwiązaniem blokującym, może być złym rozwiązaniem. Dochodzi tu również problem zaufania pomiędzy usługami. Wiele, szczególnie niepublicznych usług ogranicza zakres blokowania nadzorowanych przez siebie zasobów, tak aby nie doszło do sparaliżowania działań usługi. Dlatego też często „rozluźnia" się wymagania ACID, stosując alternatywne środki zaradcze. Rozluźnienie wymagań może również wynikać z natury samej aplikacji biznesowej. Wiele aplikacji biznesowych ma nieco inne wymagania w stosunku do transakcji, niż te stawiane przez systemy baz danych. W złożonych aplikacjach biznesowych niepowodzenie jednej z aktywności, nie musi bowiem oznaczać niepoprawności całej transakcji i konieczności wycofania uzyskanych rezultatów (nierzadko osiągniętych w długotrwałych procesach). Sama aplikacja powinna móc selekcjonować usługi, które mają wpływ na końcowy wynik transakcji. Wymagać może także obsługi błędów. Chwilowa niedostępność jednej z usług nie musi powodować konieczności unieważnienia całej transakcji i wycofania wszystkich rezultatów otrzymanych wcześniej. Niepraktyczność blokowania zasobów, a czasem nawet niemożliwość blokowania, skłania do uwidocznienia rezultatów usług webowych realizujących cząstkowe zadania całej aktywności ujętej w transakcji. Pozostaje pytanie co zrobić z osiągniętymi cząstkowymi wynikami, gdy cała transakcja powinna jednak zakończyć się nie powodzeniem. Dobrym rozwiązaniem jest zastosowanie transakcji kompensacyjnych, w celu wycofania zmian jakie już zaszły. Warto jednak zauważyć, że działania kompensacyjne nie mogą być generowane automatycznie i ściśle zależą od logiki biznesowej całej aplikacji. Niektóre z zadań bowiem nie wymagają zastosowania działań odszkodowawczych (np. gdy mamy do czynienia z odczytem). W przypadku innych mogą być dość trudne wyegzekwowania, np. gdy powodują naliczanie i egzekucję kar finansowych w realnym świecie. Zwykle do opisu takich zachowań stosowane są protokoły i języki kompozycji, np. BPEL.

W [Bell 2008] zdefiniowano kryteria, które ułatwiają współpracę i interakcję usług biorących udział w transkacji w środowisku usług webowych:

- względna atomowość – pojedyncza transakcja powinna udostępniać atomowość dla transakcji krótkotrwałych, umożliwiając jednak elastyczność dla dłuższych transakcji,

- polityka odszkodowań – na wypadek niepowodzenia długotrwałych transakcji powinno się planować transakcje kompensacyjne,
- krótkie i długie aktywności – transakcje powinny składać się zarówno z krótkich, jak i długich aktywności, razem współdziałających w celu osiągnięcia pożądanego efektu,
- synchroniczne i asynchroniczne aktywności – transakcje mogą być oparte o aktywności, które mogą wykorzystywać zarówno synchroniczne i asynchroniczne protokoły,
- izolacja transakcji wraz ze względnym blokowaniem – transakcje powinny umożliwiać uzyskanie wymagania izolacji przez blokowanie zasobów przez krótkotrwałe aktywności; dla aktywności długotrwałych należy stosować łagodniejszą politykę blokowania,
- ścisłe i luźne powiązania – transakcja powinna umożliwiać wykonywanie aktywności w środowiskach usług webowych, które stosują politykę zarówno ścisłych, jak i luźnych powiązań,
- orkiestracja i choreografia – transakcje powinny obejmować zarówno orkiestrację, jak i choreografię aktywności.

LITERATURA

[Bell 2008] Bell M., *Service-oriented modeling*, John Wiley & Son Inc., Piscataway 2008.

[BerNew 1997] Bernstein P., Newcomer E., *Principles of Transaction Processing*, Morgan Kaufmann Publishers Inc., San Francisco 1997.

[ChaSchMe 2007] Charfi A., Schmeling B., Mezini M., *Transactional BPEL Processes with AO4BPEL Aspects*, Fifth European Conference on Web Services, Washington 2007.

[CouDolKin 2005] Coulouris G.F., Dollimore J., Kindberg T., Distributed *Systems – Concepts and Design.* Pearson Education Limited, Essex UK 2005.

[EliMoss 2006] Eliot J., Moss B., *Open Nested Transactions: Semantics and Support*, Department of Computer Science University of Massachusetts, Amherst 2006.

[Erl 2005] Erl T., *Service-Oriented Architecture: Concepts, Technology, and Design*, Prentice Hall PTR USA 2005.

[FrySal 2008] Fryźlewicz Z., Salamon A., *Podstawy architektury i technologii usług XML sieci WEB*, Wydawnictwo Naukowe PWN S.A., Warszawa 2008.

[GarUllWid 2006] Garcia-Molina H., Ullman J.D., Widom J., *Systemy baz danych.* Pełny wykład Wydawnictwa Naukowo-Techniczne, Warszawa 2006.

[GraReu 1993] Gray J., Reuter A., *Transaction processing: concepts and techniques*, Morgan Kaufmann Publishers, San Francisco 1993.

[JurMatSar 2006] Juric M. B., Mathew B., Sarang P., *Business Process Execution Language for Web Services*, 2nd Edition, Packt Publishing Birmingham 2006.

[KorLevSil 1990] Korth H. F., Levy E., Silberschatz A., *Formal Approach to Recovery by Compensating Transactions*, Konferecja, Brisbane, Australia 1990.

[MarSha 2007] Margolis B., Sharpe J., *SOA for the Business Developer-Concepts, BPEL, and SCA*, First Edition, MC Press, USA, 2007.

[StrKar 2002] Strandenæs T., Karlsen R., *Transaction Compensation in Web Services*, Norsk Informatikkonferanse 2002.

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

Zasoby internetowe

[Berger 2007] *Technologie o wysokim potencjale wzrostu*, Roland Berger Strategy Consultants, http://www.rolandberger.pl/

[OASIS] OASIS, Organization for Advancement of Structured Information Standards, http://oasis-open.org

[SOAP12] SOAP Version 1.2, http://www.w3.org/TR/2001/WD-soap12-20010709/

[UDDI] UDDI Version 3.0.2, http://uddi.org/pubs/uddi-v3.0.2-20041019.htm

[W3C] W3C, World Wide Web Consortium, http://w3.org

[WS-Addr] WS-Addressing 1.0, http://www.w3.org/2005/08/addressing

[WS-AT] Web Services Atomic Transaction (WS-Atomic Transaction) 1.1, 2007, http://docs.oasis-open.org/ws-tx/wstx-wsat-1.1-spec-errata-os/wstx-wsat-1.1-spec-errata-os.html

[WS-BA] Web Services Business Activity 1.1, 2007, http://docs.oasis-open.org/ws-tx/wstx-wsba-1.1-spec-errata-os/wstx-wsba-1.1-spec-errata-os.html

[WS-Coord] Web Services Coordination (WS-Coordination) 1.1, 2007, http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.1-spec-errata-os/wstx-wscoor-1.1-spec-errata-os.html

[WSDL] Web Services Description Language (WSDL) Version 2.0 http://www.w3.org/TR/wsdl20/

[WS-I] WS-I , Web Services-Interoperability Organization, http://ws-i.org/

TRANSACTIONS IN THE ENIRONMENT OF WEB SERVICES

The work presents architectural concepts which make it possible to coordinate and implement transactions in the environment of Web Services. The architectural model developed within the framework of OASIS, W3C, WS-I and defined in WS-Coordination, WS-Atomic Transaction, WS-Business Activity and BPEL has been described in details

Prace Naukowe Instytutu Informatyki
Politechniki Wrocławskiej

Zeszyt 10 2009

integracja aplikacji,
architektura zorientowana na usługi,
usługi sieciowe,

Marek MAŁECKI,
Artur WILCZEK*

USŁUGI SECIOWE REST

Duża złożoność protokołów sieciowych i standardów rodziny WS-*, stosowanych w budowie usług sieciowych, spowodowała wzrost popularności architektury usług sieciowych wykorzystującej bezstanowe mechanizmy protokołu HTTP – REpresentational State Transfer (w skrócie REST).

1. WPROWADZENIE

REST (ang. *Representational State Transfer*) jest stylem architektonicznym budowy oprogramowania, wykorzystywanym przez systemy rozproszone (również usług sieciowych) oparte na sieci Web (World Wide Web). Koncepcja REST została zaprezentowana w 2000 roku przez Roya Fieldinga, jednego z autorów standardu http [Fielding 2002].

REST odwołuje się do wykorzystania architektury sieci komputerowej oraz jej zasobów, definiowanych poprzez adresy URI (ang. *Uniform Resource Identifier*) [Berners 2005]. Termin REST jest czasami używany do opisywania prostego interfejsu umożliwiającego uzyskanie dostępu i modyfikację danych z użyciem HTTP w obrębie danej domeny, bez dodatkowej warstwy takiej jak SOAP [SOAP 1.2]. Istnieje możliwość zaprojektowania systemu opierającego się na REST bez wykorzystania sieci WWW (np. w sieciach lokalnych).

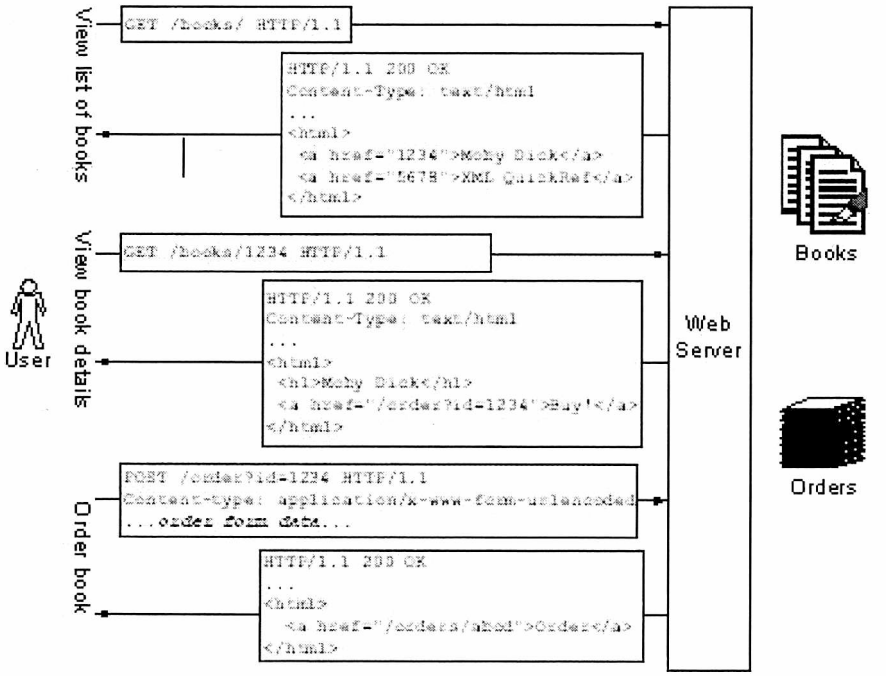
Należy zaznaczyć, że REST jest stylem architektonicznym, zestawem zaleceń, które wskazują sposób budowania oprogramowania w celu uzyskania pożądanej funkcjonalności, dzięki skalowalności protokołu HTTP i poprzez wykorzystanie jego metod (GET, POST, PUT itd.).

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

2. ZASADY REST

Konstruowanie aplikacji wykorzystujących ideę REST polega na stosowaniu następujących zaleceń [Pautasso 2008]:

1. **Identyfikacja zasobów poprzez URI** – Usługa sieciowa oparta na REST, daje dostęp jej użytkownikowi do zestawu zasobów, jednocześnie określając zasady interakcji z nimi. Zasoby są identyfikowane na podstawie URI, co daje możliwość wykorzystania dla zasobów globalnej przestrzeni adresowania w ramach danej domeny oraz ogranicza możliwość wyszukiwania tych zasobów wyłącznie do domeny globalnej.
2. **Jednorodny interfejs** – Manipulowanie zasobami odbywa się poprzez zestaw czterech, znanych z HTTP, metod, które są odpowiedzialne za wstawianie, usuwanie, pobieranie i wysyłanie danych, (PUT, GET, POST, DELETE). PUT tworzy nowy zasób, który następnie może być usunięty poprzez DELETE. GET pobiera aktualny stan zasobu, w reprezentacji zależnej od implementacji serwera. POST przesyła nowy stan zasobu do serwera powodując zmianę zasobu. Często programiści usług sieciowych modyfikują wykorzystanie tych metod w sposób zapewniający odpowiednie bezpieczeństwo aplikacji udostępniającej usługi sieciowe.



Rys. 1. Wykorzystanie interfejsu REST

3. **Samo opisujące się komunikaty** – Zasoby są oddzielone od ich reprezentacji, dzięki czemu, uzyskując do nich dostęp, możemy zadecydować jaką reprezentację mogą przyjąć. Wszystkie informacje niezbędne do odczytania komunikatu są wewnątrz komunikatu. Jednocześnie metadane powiązane z zasobem mogą zostać wykorzystane do wykrywania błędów transmisji, buforowania, autoryzacji czy kontroli dostępu.
4. **Protokół komunikacji** – protokół, który działa na zasadzie „klient–serwer", jest bezstanowy, umożliwia buforowanie. Jednocześnie powinien umożliwiać umieszczanie informacji w swojej strukturze (tak jak to ma miejsce np. w przypadku hyperlinków, do których dodawane są informacje, potem wykorzystywane przez usługi sieciowe).

2.1. ZASOBY REST

Podstawowym pojęciem wykorzystywanym w zaleceniach architektury REST jest pojęcie zasobu. Zgodnie z zaleceniami REST komunikacja pomiędzy klientem i serwerem polega na przesyłaniu reprezentacji zasobów. Zasobem może być każdy obiekt o określonej reprezentacji opisującej jego stan. Wszystkie zasoby są dostępne w protokole HTTP poprzez odpowiednie identyfikatory URI. Aby móc manipulować tymi zasobami, komponenty sieci (takie jak np. przeglądarki internetowe) komunikują się poprzez standardowy interfejs (np. HTTP) oraz wymieniają reprezentacje tych zasobów. Jednocześnie dostępu do zasobu (poprzez adres URI) może żądać wielu klientów jednocześnie. Jednak każde takie żądanie nie ma „świadomości" istnienia innych, np. poprzednich żądań – jest bezstanowe. Dzięki temu aplikacja może wchodzić w interakcję z danym zasobem, „znając" dwie informacje: identyfikator zasobu oraz akcję wymaganą do interakcji z tym zasobem.

Należy zauważyć, że elementy sieci komputerowej takiej jak: firewall, proxy, tunel itp. nie są przeszkodą w nawiązaniu komunikacji i przesyłaniu informacji zgodnie z zaleceniami REST.

2.2. WYDAJNOŚĆ

W celu poprawienia wydajności komunikacji, założenia architektoniczne REST określają możliwość wykorzystania buforowania (ang. *caching*) reprezentacji danych przesyłanych pomiędzy klientem a serwerem. Wykorzystuje się tu głownie fakt bezstanowości usługi sieciowej opartej o REST. Co za tym idzie, dane, które są wielokrotnie żądane, mogą być buforowane przez dodatkowe mechanizmy serwera usługi sieciowej, aby móc znacznie szybciej (bez dodatkowego opóźnienia) być przesyłane do żądającego je klienta. Odnosi się to tylko do danych, które zostały jawnie zadeklarowane, jako takie, które mogą być buforowane. Buforowanie umożliwia częściowe

lub całkowite wyeliminowanie interakcji pomiędzy mechanizmem przesyłającym reprezentację danych a samymi danymi, może także poprawić wydajność i skalowalność aplikacji.

Załóżmy sytuację, w której bardzo duża ilość danych musi zostać przekazana przez serwer usługi sieciowej do klienta. Jedyną możliwością poprawienia efektywności tej operacji jest ładowanie danych „strumieniowo" poprzez dosyłanie kolejnych części danych. Korzystając z buforowania danych, część danych już przetworzonych może zostać wysłana natychmiast, dzięki czemu użytkownik może zacząć z nich korzystać praktycznie natychmiast, bez potrzeby oczekiwania na pozostałe dane.

Wymogi Internetu i udostępniania w nim usług sieciowych opartych o REST sprawiły, że zaistniała potrzeba zapewnienia możliwości budowy warstwowej architektury usług. Możliwe jest komponowanie warstw w sposób hierarchiczny, zgodnie z ich zachowaniem tak, że każdy komponent jest odrębny i może (choć nie musi) być dostępny z innych warstw i komponentów w nich znajdujących się. Warstwy mogą być wykorzystane do enkapsulacji usług już istniejących względem nowych usług. Można tego dokonać poprzez proste przesunięcie dotychczasowych usług, do niższych warstw, dzięki czemu nowe usługi mają do nich pośredni dostęp. Umożliwia to zwiększenie skalowalności systemu oraz rozłożenia obciążenia usług na kilka niezależnych sieci (co z perspektywy klienta nie jest widoczne). Wadą takiego rozwiązania jest zwiększony czas oczekiwania na odpowiedź i dodatkowy czas na przetwarzanie danych, co może w dłuższej perspektywie obniżyć efektywność usług.

W architekturze REST, możliwe jest dostarczenie klientom przez system zarządzający funkcjonalności kodu na żądanie (ang. code on demand). Zamiast danego zasobu, serwer usługi sieciowej może udostępnić kod wykonywalny rozszerzający funkcjonalność oprogramowania (np. przeglądarki klienta). Przykładem takiego zachowania jest np. przesłanie kodu apletu Javy, który może rozszerzyć funkcjonalność przeglądarki internetowej, a jednocześnie posiadać dalszą zdolność współpracowania z dostępnymi usługami sieciowymi na danym serwerze.

2.3. BEZPIECZEŃSTWO

Architektura REST nie uwzględnia bezpośrednio jakichkolwiek mechanizmów zabezpieczających przesyłane dane, dostępu do zasobów czy manipulowania nimi. Wszystkie te czynniki są właściwie przerzucone na barki programistów korzystających z tej architektury. Można zapewnić odpowiedni poziom bezpieczeństwa korzystania z usług opierających się na REST. Jest to zależne od wykorzystywanego oprogramowania hostującego aplikację usługi webowej. Przesyłane dane można zabezpieczyć korzystając z szyfrowanej, z pomocą SSL, wersji protokołu HTTP (HTTPS). Jednocześnie dostęp do usług może być również autoryzowany na podstawie danych, które przesyła użytkownik podczas wywoływania żądania do danej usługi, co daje możliwość implementacji polityk bezpieczeństwa ACL i wielu innych. Dodatkowo w ar-

chitekturze REST, zakładającej wykorzystanie protokołu bezstanowego, można implementować z powodzeniem techniki komunikacyjne oparte na wzorcu pytanie-odpowiedź. Utrzymanie stanu może bazować na danych umieszczonych na maszynie klienta (np. plikach Cookiem) oraz dodatkowej identyfikacji, która jest dostarczana przez warstwę TCP/IP (adres IP), która jest nośnikiem protokołu HTTP.

3. ZALETY I WADY REST

Poniżej omówiono główne zalety i wady wynikające z zastosowania REST do budowy usług sieciowych.

3.1. ZALETY

Podstawową i główną zaletą usług sieciowych opartych na REST jest ich niezmiernie duża prostota w implementacji, wdrożeniu, utrzymaniu i dokumentowaniu. Wynika to z faktu, że REST opiera się na znanych, popularnych, dojrzałych i dopracowanych standardach W3C takich jak HTTP, XML, URI czy MIME. Jednocześnie architektura programowo sprzętowa wymagana do utrzymania usługi sieciowej opartej na REST stała się ogólnodostępna i łatwa we wdrożeniu i zarządzaniu. Serwery HTTP dostępne są praktycznie na każdy system operacyjny czy platformę sprzętową, tak samo jak klienci wykorzystujący HTTP (w ich roli mogą występować nie tylko przeglądarki, ale również zwykli klienci telnet, którzy mogą połączyć się z portem 80 serwera).

Bardzo lekka infrastruktura, w której usługi sieciowe oparte na REST mogą być tworzone, powodują, że ich całościowy koszt stworzenia i wykorzystania jest bardzo niski w porównaniu z rozwiązaniami konkurencyjnymi (chociażby SOAP), co powoduje znaczną popularyzację rozwiązań opartych o taką architekturę.

Czas tworzenia usługi sieciowej zgodnej z zaleceniami REST jest także stosunkowo niewielki, co pozwala na zwiększenie, jakości rozwiązania poprzez przeznaczenie zaoszczędzonych zasobów na dodatkowe testowanie i optymalizację kodu aplikacji.

Kolejną zaletą usługi sieciowej opartej na REST jest prostota w jej wdrażaniu, która sprowadza się do prostego dostosowania powszechnie wykorzystywanego oprogramowania (takiego jak np. serwer Apache), co nie jest trudniejsze niż wdrożenie prostej strony internetowej.

Dzięki wykorzystaniu URI oraz hiperłączy, REST udowodniło, że możliwe jest dynamicznie odkrycie dostępnych usług sieciowych bez konieczności dostarczania dodatkowych systemów, standardów i specyfikacji potrzebnych do zidentyfikowania i korzystania z usług sieciowych. Niewymagana jest również konieczność rejestrowania swoich usług w centralnym katalogu (tak jak miało to miejsce w przypadku SOAP i UDDI).

Zaletami usług sieciowych opartych na REST jest także łatwa skalowalność do dużej liczby klientów z nich korzystających, w szczególności dzięki wykorzystaniu możliwości buforowania danych, zdolności do prostego budowania i wykorzystania klastrów oraz wydajnych (a za razem prostych) mechanizmów równomiernego rozkładu obciążenia systemów.

Kolejną zaletą REST jest brak konieczności, chociaż taka możliwość też istnieje, przesyłania żądań w postaci wiadomości kierowanych na dany adres. Pobieranie reprezentacji zasobu na podstawie URI znacznie ułatwia proces korzystania z usługi sieciowej, jednocześnie nie obciąża (w porównaniu z SOAP) serwera usługi sieciowej. Istnieje możliwość korzystania z lekkich formatów wiadomości w przypadku żądań z wiadomościami, takich jak JSON (ang. *JavaScript Object Notation*). Dlatego też interfejs usług REST jest bardzo przejrzysty, prosty i umożliwia korzystanie z niego bez dodatkowych obciążeń oraz nie wymaga dodatkowych nakładów na jego identyfikację i wykorzystanie (tak jak ma to miejsce w przypadku interfejsów SOAP).

Dostęp do reprezentacji zasobu jest prosty, gdyż URI jest prostym sposobem na dostęp do danych (reprezentacji danych) a jednocześnie modyfikowanie go jest bardzo proste w porównaniu z innymi rozwiązaniami. Również sam dostęp do reprezentacji zasobu może być łatwo zmieniony tak, aby uzyskać inną reprezentację tych samych danych (zaletą jest tutaj odpowiednie wykorzystanie MIME, identyfikujące typ reprezentacji zasobu, w procesie przesyłania danych na żądanie klienta)

Poprzez zredukowanie konieczności utrzymywania stanu danych w ramach zasobu usługi sieciowej dla każdego użytkownika, REST umożliwia większą skalowalność architektury sprzętowej udostępniającej usługę. Jednocześnie istnieje możliwość prostego uzyskania stanowości, przy relatywnie niewielkim dodatkowym obciążeniu poprzez zastosowanie zapisywania ustawień sesji po stronie klienta. Można w nich przetrzymywać przez określony czas identyfikator sesji, który następnie posłuży do dodatkowych operacji na danych. Jednak żadne dane konkretnie przypisane do sesji, nie są przetrzymywane w pamięci usługodawcy.

3.2. WADY REST

REST jako zbiór reguł wskazujących jak projektować daną usługę sieciową, nie posiada własnych, popartych przez duże organizacje standardów, które byłyby zatwierdzone i uznawane jako sprawdzone wytyczne. Nie istnieje żaden sformalizowany dokument zawierający konkretną specyfikację jak i przy użyciu jakich technologii należy budować aplikacje REST. Dlatego też twórcy aplikacji poszukujący wskazówek dotyczących tworzenia oprogramowania zgodnie z zasadami REST są zdani wyłącznie na wybranie jednej z dwóch opcji: HiREST i LOREST. Rekomendacja HiREST zakłada wykorzystanie 4 metod protokołu HTTP, przesyłanie danych przy pomocy formatu POX – Plain Old XML oraz stosowanie adresów URI w sposób

opisany wcześniej. Implementacja LoREST zaleca stosowanie tylko dwóch metod z HTTP, wykorzystanie typów MIME, stosowanie dodatkowych ukrytych pól (z racji używania tylko metod GET i POST) w ramach formularzy do przesyłania do serwera dodatkowych informacji.

Kolejnym problemem są ograniczenia jakie nakładane są przez używanie metod GET lub POST. W ramach protokołu HTTP od klienta do serwera można przesyłać dane. Takie dane mogą być zawarte w ramach formularza HTML jak również mogą znajdować się jako dodatkowe atrybuty w adresie URI. W drugim przypadku zachodzi problem z ilością przesyłanych danych. Większość obecny implementacji tzw. cienkich klientów opartych o przeglądarki nie może przesyłać danych poprzez URI jeżeli przekraczają one 4KB limitu dostępnego dla tego sposobu transmisji dodatkowych atrybutów. Serwer nie jest w stanie zdekodować większej ilości danych i w zależności od implementacji, zazwyczaj występuje błąd (HTTP Error 414 – Request URI too long) lub dane są ucinane, co może powodować błędy. W drugim przypadku może to także prowadzić do załamania się działającego serwera powodując, że serwer będzie narażony na ataki związane z przepełnieniem bufora. Ostatecznym problemem dotyczącym przesyłania danych przy użyciu URI, jest fakt, że w URI nie ma możliwości przekazywania złożonych struktur danych, które mogą być wysyłane. Można to jednak rozwiązać poprzez zastosowanie alternatywnej metody przekazywania danych, poprzez ukryte pola formularza lub przesyłanie wiadomości w popularnych formatach (POX, JSON itp.). Wynika to z faktu, że metoda POST obsługująca takie wywołania nie ma ograniczeń wielkościowych (zdefiniowanych domyślnie).

Wadą REST jest również podejście do bezpieczeństwa. W standardowym założeniu wykorzystania REST do przesyłania wiadomości (czy żądań) od klienta do serwera nie ma żadnego standardu czy rekomendacji dot. bezpieczeństwa. Całość zapewnienia bezpieczeństwa przenosi się na doświadczenie i możliwości programisty. REST nie gwarantuje, że wywołania żądań czy przesyłania danych od klienta do serwera będą prawidłowe i będą pochodzić dokładnie do podanego klienta. Dane które są przesyłane mogą powodować nieprawidłowe wykonywanie się skryptów, przez co może to narazić na nieprawidłowe działanie cały serwer usług sieciowych. Dane przesyłane do serwera od klienta można łatwo przechwycić i odczytać lub zmodyfikować do własnych potrzeb. Tak samo reprezentacja zasobów przesyłana od serwera do klienta, bez dodatkowych zabezpieczeń, które nie są zdefiniowane w modelu wykorzystania REST do budowy usług sieciowych, może być narażona na przechwycenie i zmodyfikowanie. REST nie gwarantuje również żadnej formy autoryzacji czy kontroli dostępu do zasobów. Usługi sieciowe oparte na REST są zazwyczaj rozszerzane (poza ideą REST) o dodatkowe rozwiązania mające na celu zapewnienie bezpieczeństwa. SSL w przypadku przesyłania danych pomiędzy klientem a serwerem, ACL do autoryzacji i odpowiednie skrypty umożliwiające walidację danych zapewniają usługom sieciowym opartym na REST odpowiedni poziom bezpieczeństwa.

4. PODSUMOWANIE

Architektura REST sprawdza się przede wszystkim, gdy wymagane jest zastosowanie usługi opierającej się na protokole całkowicie bezstanowym. Usługi REST stosowane są z powodzeniem tam gdzie zachodzi potrzeba integracji bezpośredniej typu punkt–punkt. Oznacza to, że obie strony, usługodawca i klient mają informacje niezbędne do interpretacji zawartości komunikatów i kontekście danych dostarczanych przez taką usługę. Podstawowym argumentem za stosowaniem REST jest wymóg dostosowania możliwości komunikacyjnych usługi do limitowanej przepustowości łącza, z racji tego, że REST nie posiada dodatkowego narzutu (w porównaniu z SOAP) wynikającego z interpretacji i opakowywania danych. Czyni to REST świetnym rozwiązaniem do zastosowań w urządzeniach mobilnych.

LITERATURA

[Fielding 2002] Fielding Roy T., Taylor Richard N. (2002-05), *Principled Design of the Modern Web Architecture* (PDF), ACM Transactions on Internet Technology (TOIT), New York: Association for Computing Machinery, 2(2), 115–150.

[Berners 2005] Berners-Lee, Fielding and Masinter 2005 RFC 3986, Uniform Resource Identifier (URI), Generic Syntax.

[Pautasso 2008] *RESTful Web services vs. Big Web services: making the right architectural decision.* In: Ma W.-Y., Tomkins A., Zhang X. (Eds.), Proceedings of WWW 2008, ACM Press. pp. 805–814.

[SOAP 1.2] W3C, SOAP Version 1.2, http://www.w3.org/TR/soap/

REST WEB SERVICES

This article presents a brief overview of web services construction methods based on pure HTTP protocol features – REST.

Wydawnictwa Politechniki Wrocławskiej
są do nabycia w księgarni „Tech"
plac Grunwaldzki 13, 50-377 Wrocław
budynek D-1 PWr., tel. 071 320 29 35
Prowadzimy sprzedaż wysyłkową
zamawianie.ksiazek@pwr.wroc.pl

ISBN 978-83-7493-505-0