STUDIA INFORMATICA 2003
Volume 24 Number 1 (52)

Marcin GORAWSKI, Andrzej WOCŁAW
Politechnika Śląska, Instytut Informatyki

IMPLEMENTACJA ALGORYTMU ODTWARZANIA DESIGN-RESUME W TECHNOLOGII JavaBeans

Streszczenie. Hurtownia danych konsoliduje, agreguje i gromadzi gigabajty danych przetwarzanych analitycznie celem wsparcia podejmowania decyzji. Hurtownie ładowane są danymi pochodzącymi zwykle z różnych źródeł danych. Taki proces ładowania może być przerwany przez wystąpienie błędów programowych lub uszkodzeń sprzętowych. Obecnie najlepsze rezultaty odtwarzania procesu ładowania hurtowni zapewnia algorytm Design-Resume zaimplementowany w języku C na Uniwersytecie Stanforda w 1999 roku. W naszej pracy przedstawiamy implementację tego algorytmu z wykorzystaniem technologii JavaBeans.

Słowa kluczowe: algorytmy odtwarzania, proces ekstrakcji danych, hurtownie danych.

IMPLEMENTATION OF DESIGN-RESUME RECOVERY ALGORITHM USING JavaBeans TECHNOLOGY

Summary. Data warehouse consolidates, aggregates and gathers gigabytes of data, which are processed analytically in order to support decision-making. Data warehouses are loaded with data that usually come from various data sources. Programming errors or hardware failures can interrupt this loading process. The best results in recovery of the loading process are provided at present by Design-Resume algorithm implemented in C language at Stanford University in 1999. In our research we present another implementation of this algorithm using JavaBeans technology.

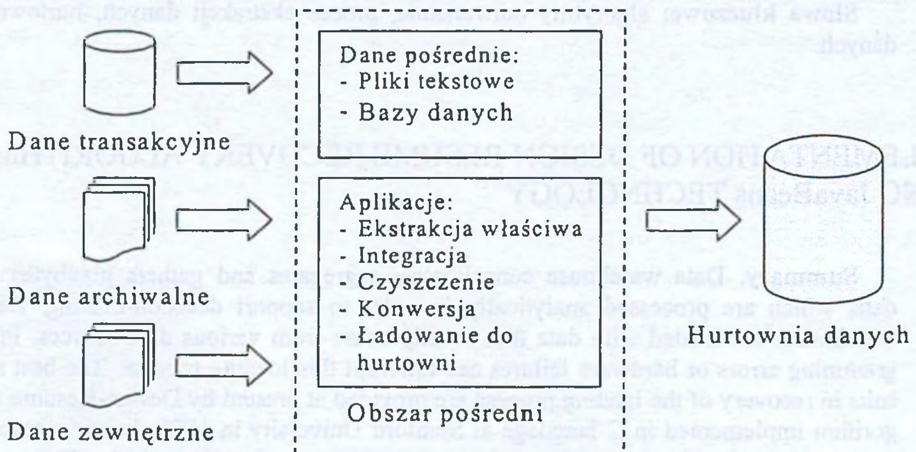
Keywords: recovery algorithms, extraction process, data warehouses.

1. Odtwarzanie procesu ekstrakcji danych

1.1. Proces ekstrakcji danych na tle tworzenia hurtowni danych

Hurtownie danych są tworzone w celu zbierania, integrowania i przechowywania danych na potrzeby analizy i eksploracji danych. Proces pozyskiwania danych dla hurtowni danych potocznie nazywany jest ekstrakcją danych lub procesem ETL. Tworzą go procesy składowe, tj. ekstrakcja, transformacja i ładowanie danych (ang. *Extraction, Transformation and Loading, ETL*). Praktyczna realizacja procesu ekstrakcji jest często bardzo złożona [1].

Hurtownie danych gromadzą ogromne ilości danych, które pochodzą z rozproszonych i najczęściej heterogenicznych źródeł danych (np. relacyjne bazy danych, pliki tekstowe, Internet). W czasie typowego ładowania danych do hurtowni ilość przetwarzanych informacji mierzy się w gigabajtach. Proces ładowania może zajmować dziesiątki godzin i korzystać z bardzo wielu skomplikowanych transformacji, takich jak np.: wyszukiwanie duplikatów, usuwanie niespójności w danych, konwersja danych z różnych źródeł do wspólnego formatu, dodawanie unikalnych kluczy, wykonywanie złączeń i filtrowanie danych. Procesy ekstrakcji tworzą tzw. obszar pośredni (ang. *staging area*) [1,2] pomiędzy systemami źródłowymi a bazą hurtowni danych. W obszarze pośrednim dane są przygotowywane i przetwarzane, a następnie przenoszone do właściwej hurtowni danych (rys. 1).



Rys. 1. Obszar pośredni w procesie ekstrakcji danych
Fig. 1. Staging area in data extraction process

Proces ładowania danych do hurtowni jest realizowany w ściśle określonym czasie i zwykle wtedy, gdy systemy z danymi źródłowymi są mało aktywne. W przypadku organizacji o zasięgu lokalnym najczęściej ładowanie hurtowni odbywa się nocą lub w weekend. W organizacji o zasięgu globalnym dobowy przyrost nowych danych jest zwykle bardzo znaczący, a charakter wykonywanych analiz wymaga maksymalnej aktualności danych w hur-

towni. W takich sytuacjach stosuje się specjalne mechanizmy przyrostowego ładowania danych.

1.2. Zakłócenia procesu ekstrakcji danych

Zakłócenia procesu ekstrakcji przerywają operację ładowania hurtowni danych. Zakłócenia mogą być spowodowane wystąpieniem błędów (np. nieprawidłowe dane, upadek RDBMS) lub uszkodzeń platform sprzętowych (np. awaria zasilania, uszkodzenie procesora) średnio co trzydziesty proces ładowania danych zostaje przerwany [5]. W takim przypadku, zwykle dane już przetworzone są usuwane, a cały proces powinien być natychmiast wznowiony - w przeciwnym wypadku baza hurtowni będzie nieaktualna i niekompletna. Wtedy wykonanie przerwanego procesu przekłada się na kolejny cykl ładowania.

Akcja wznowienia procesu ekstrakcji polegająca na dokończeniu przerwanego procesu ładowania danych nazywa jest odtwarzaniem. W wyniku odtwarzania zbiór danych wynikowych ekstrakcji powinien się pokrywać ze zbiorem danych, który powstałby, gdyby nie doszło do jej przerwania. Algorytmy poszczególnych podprocesów ekstrakcji wspomagających odtwarzanie są modyfikowane, co nie tylko komplikuje projekty, ale przede wszystkim obniża wydajność przetwarzania.

2. Algorytmy odtwarzania procesu ekstrakcji danych

2.1. Tradycyjne metody odtwarzania

2.1.1. Metoda podziału danych wejściowych (ang. batching)

W tej metodzie dane wejściowe dzielone są na bloki danych (ang. *batch*) przetwarzane sekwencyjnie, blok po bloku. W razie przerwania procesu ekstrakcji odtwarzanie jest wznawiane od uszkodzonego bloku, pozostałe poprawnie załadowane bloki nie wymagają powtórnego przetwarzania. Wadą tej metody jest potrzeba separacji danych na niezależnie przetwarzane bloki i wzrost czasu przetwarzania wraz z rosnącą liczbą bloków. Dodatnią cechą metody jest brak potrzeby modyfikacji kodu transformacji.

2.1.2. Metoda kopii migawkowych i punktów powrotu

Metoda opisuje sposób tworzenia i wykorzystania okresowych kopii migawkowych przetwarzanych danych oraz punktów powrotu (ang. *snapshots and savepoints*). W razie przerwania ekstrakcji każda transformacja danych wznawia swoje przetwarzanie od ostatniego punktu powrotu. Implementacja punktów powrotu wymaga zapisywania sekwencji krotek przesyłanych między transformacjami w specjalnych kolejkach (ang. *persistent*

queues). Praktyczne stosowanie tej metody jest uciążliwe, wymaga złożonych modyfikacji transformacji, powoduje dodatkowe duże obciążenie przetwarzania i trudności w implementacji kolejek przechowujących kopie migawkowe. Wymaga także dobrej znajomości semantyki transformacji potrzebnej do tworzenia punktów powrotu i ich aktywowania.

2.1.3. Metoda podziału procesu ekstrakcji (ang. staging)

Metoda polega na dekompozycji procesu ekstrakcji na powiązane logicznie grupy podprocesów. Dane wyjściowe danej grupy są zapisywane i jednocześnie podawane na wejście kolejnej grupy podprocesów. Odtwarzanie polega na restarcie grupy podprocesów obejmujących wcześniej wykonane kopie danych wejściowych tej grupy. Nie jest wymagane powtórne uruchamianie procesów z zakończonych poprawnie grup podprocesów. Metoda ta wnosi dodatkowe duże obciążenie procesu ekstrakcji związane z tworzeniem kopii danych oraz wywołuje szczególnie niekorzystne zjawisko utraty współbieżności.

2.1.4. Metoda powtarzania procesu ekstrakcji od początku (ang. redo)

W metodzie powtarzania procesu ekstrakcji od początku (zamiennie: odtwarzanie typu Redo) wznawiany jest od samego początku każdy przerwany proces ekstrakcji, natomiast proces ładujący dane do hurtowni filtruje przetworzone już krotki tak. Dzięki temu do bazy hurtowni nie trafiają dane już tam zapisane przed wystąpieniem awarii. Metoda nie wykorzystuje jednak krotek już zapisanych w hurtowni i wszystkie dane wejściowe są przetwarzane jeszcze raz. Odtwarzanie Redo nie nakłada żadnego obciążenia na poprawnie przebiegający proces ekstrakcji, stąd jest często stosowana w komercyjnych systemach ETL, np. Informatica [6].

2.2. Algorytm odtwarzania Design-Resume

2.2.1. Cechy charakterystyczne algorytmu

Algorytm Design-Resume (DR) opisujący zmodyfikowaną metodę odtwarzania typu Redo został opracowany na Uniwersytecie Stanforda w 1999 roku [5]. Algorytm DR wykorzystuje dane już przetworzone i załadowane do hurtowni danych przez przerwany proces ekstrakcji, eliminując potrzebę powtórnego przetwarzania wszystkich danych wejściowych. Dane te są wykorzystywane w czasie filtrowania danych wejściowych i danych przetwarzanych przez wznowiony proces ekstrakcji. Takie podejście zasadniczo skraca czas odtwarzania. Podstawowym problemem jest określenie, które krotki wejściowe można bezpiecznie odfiltrować. Aby je określić, należy każdej krotce załadowanej do miejsca docelowego przez przerwaną ekstrakcję wyznaczyć zbiór krotek wejściowych, które ją tworzą (w całości lub częściowo), oraz bezpiecznie je odfiltrować z sekwencji krotek wejściowych.

Krotki są bezpiecznie filtrowane, gdy sekwencja krotek wynikowych odtwarzanego procesu ekstrakcji jest taka sama jak sekwencja krotek wynikowych procesu ekstrakcji, który miał przebieg nie zakłócony. Algorytm DR wykorzystuje zbiór krotek załadowanych do miejsca docelowego przed momentem przerwania oraz informacje o własnościach i atrybutach przetwarzanych sekwencji krotek. Algorytm DR posiada następujące wyróżniające go cechy odtwarzania:

- selektywne filtrowanie pozwala uniknąć powtórnego przetwarzania wielu krotek wejściowych (minimalizuje czas odtwarzania) oraz powtórnego przetwarzania bardzo dużej ilości danych bez korzystania z dzielenia danych wejściowych na bloki,
- nie wnosi żadnego dodatkowego obciążenia w czasie przetwarzania bezawaryjnego (w przeciwieństwie do algorytmów odtwarzania z punktami powrotu),
- nie wymaga również modyfikacji kodu transformacji (algorytm DR może być stosowany zarówno dla prostych, jak i złożonych procesów ekstrakcji).

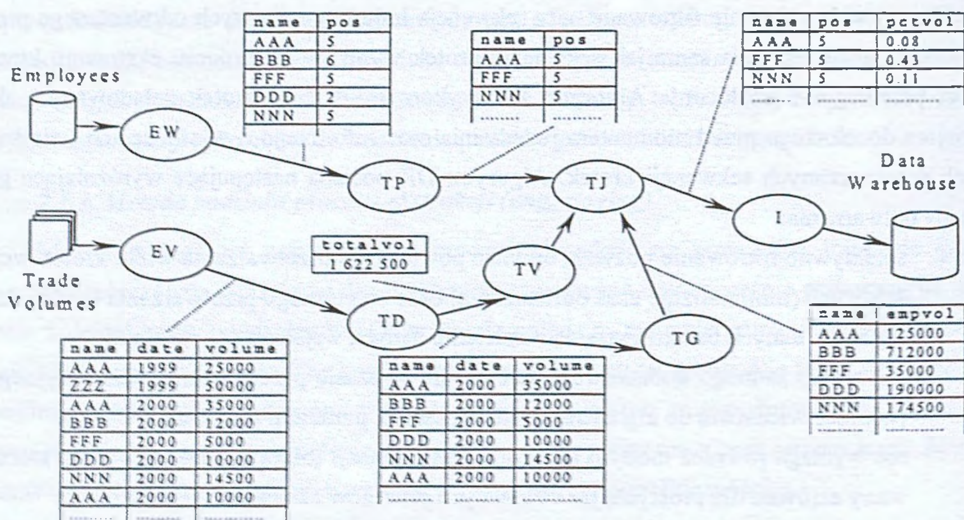
2.2.2. Graf ekstrakcji danych

Do prezentacji i analizy dowolnego procesu ekstrakcji można posłużyć się grafem acyklicznym skierowanym (ang. *Directed Acyclic Graph*, DAG). Węzły tego grafu reprezentują poszczególne komponenty procesu ekstrakcji, a jego krawędzie kierunki przepływu przetwarzanych danych. Algorytm DR wyróżnia następujące typy węzłów:

- ekstraktory (E) odpowiedzialne za wydobycie danych źródłowych,
- transformacje (T) przetwarzające wydobyte dane,
- insertory (I) ładujące przetworzone dane do miejsca docelowego.

Na rys. 2 przedstawiony został przykładowy graf ekstrakcji wraz z danymi.

Konstruowanie grafu ekstrakcji polega na określeniu zbioru węzłów i ich wzajemnych połączeń. Każdy graf ekstrakcji musi posiadać jeden węzeł ładowania danych oraz przynajmniej po jednym węźle ekstraktora i jednym węźle transformacji. Każdy węzeł algorytmu DR jest opisany wartościami charakteryzującymi jego własności oraz parametrami jego wejść niezbędnymi do odtwarzania. Konstruktor grafu ekstrakcji określa zadania wykonywane przez poszczególne węzły transformacji, wskazuje dane źródłowe dla każdego ekstraktora oraz format i miejsce docelowe danych wynikowych.



Rys. 2. Przykład grafu ekstrakcji
Fig. 2. Example of extraction graph

2.2.3. Bezpieczne filtrowanie

Poniżej przedstawione zostaną własności, które pozwalają stwierdzić, czy krotki z sekwencji wejściowej danego węzła grafu można bezpiecznie odfiltrować [5].

2.2.3.1. Własności wejść węzłów transformujących

- Własność *map-to-one*

Własność ta opisuje stan, w którym każda krotka z sekwencji wejściowej przyczynia się do powstania co najwyżej jednej krotki w sekwencji wyjściowej. Własność tę spełniają następujące operacje: selekcja, projekcja, unia, agregacja oraz niektóre złączenia.

- Własność *suffix-safe*

Własność ta wskazuje, czy prefiks sekwencji wejściowej może być bezpiecznie odfiltrowany. Własność *suffix-safe* jest spełniona, gdy dowolny prefiks sekwencji wyjściowej danej transformacji może być utworzony z odpowiedniego prefiksu sekwencji wejściowej. Inaczej, kolejność produkowania krotek sekwencji wyjściowej jest taka sama jak kolejność krotek w sekwencji wejściowej. Transformacje, które spełniają tę właściwość, to selekcja, projekcja, unia oraz agregacja wykonana na posortowanej sekwencji wejściowej.

- Własność *set-to-seq*

Jest spełniona, jeżeli kolejność krotek w sekwencji wejściowej nie ma wpływu na kolejność krotek w sekwencji wyjściowej. Własność ta jest spełniona dla transformacji sortujących krotki sekwencji wejściowej.

❑ Własność *no-hidden-contributors*

Własność jest spełniona, gdy każda krotka z sekwencji wyjściowej jest utworzona z całości lub części krotki (lub krotek dla przypadku wielu wejść) sekwencji wejściowej. Własność tę spełniają operacje: projekcja, agregacja oraz unia. Wiele transformacji typu złączenie również spełnia tę własność.

2.2.3.2. Własności węzłów transformujących

❑ Własność *in-det-out*

Własność jest prawdziwa, gdy transformacja generuje zawsze tę samą sekwencję wyjściową przy podaniu na wejściu tej samej sekwencji wejściowej (sekwencje są te same, gdy kolejność krotek również jest taka sama w obu sekwencjach). Spełniają ją wszystkie transformacje deterministyczne.

❑ Własność *no-spurious-output*

Własność jest prawdziwa, gdy na każdą krotkę z sekwencji wyjściowej składa się przynamniej jedna krotka (cała lub część) z każdej sekwencji wejściowej. Jedynie unia nie spełnia tej własności.

❑ Własność *same-set*

Własność jest spełniona, gdy transformacja daje zawsze ten sam zbiór krotek wyjściowych przy podaniu na wejściu tego samego zbioru krotek wejściowych. Większość transformacji ją spełnia. Przykładem transformacji, która jej nie spełnia, może być transformacja, która oblicza sekwencję wyjściową, bazując na kolejności krotek sekwencji wejściowej. Aby uczynić sekwencję wyjściową tej transformacji deterministyczną, można umieścić przed nią transformację sortującą.

2.2.4. Budowa algorytmu DR

Algorytm DR składa się z dwóch części: Design i Resume. Po zbudowaniu grafu ekstrakcji G i określeniu dla każdego parametru wejściowego jego własności i atrybutów oraz wskazaniu atrybutów kluczowych, uruchamiana jest procedura Design, która konstruuje graf odtwarzania G' używany przez procedurę Resume w czasie odtwarzania. Graf odtwarzania jest uzupełniony w stosunku do grafu ekstrakcji G o pewne dodatkowe cechy (obliczone przez procedurę Design). Są nimi:

- nazwy procedur reekstrakcji w ekstraktorach grafu odtwarzania oraz
- nazwy filtrów przypisane do parametrów wejściowych węzłów grafu odtwarzania.

Procedura Design określając wartości powyższych cech bazuje na własnościach, atrybutach i atrybutach kluczowych zdeklarowanych podczas projektowania grafu G. Gdy proces ekstrakcji zostanie przerwany, uruchamiana jest procedura Resume, która inicjuje procedury reekstrakcji oraz filtry w grafie G' bazując na sekwencji krotek już załadowanych w miejsce

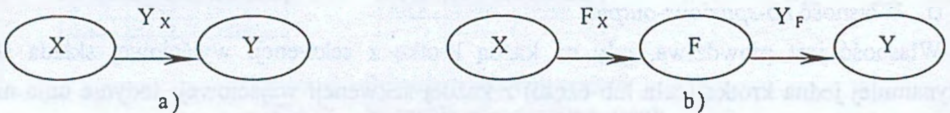
docelowe, a następnie wznawia proces ekstrakcji. Procedury Design i Resume nie działają podczas normalnego przetwarzania danych.

2.2.4.1. Węzły filtrujące algorytmu DR

Procedura Design tworząc graf odtwarzania G' przypisuje, jeżeli jest to możliwe, filtry do parametrów wejściowych węzłów grafu ekstrakcji. Wyróżnia się dwa typy filtrów:

- filtry typu *Subset* (*ClearSubset* i *DirtySubset*) usuwające z sekwencji wejściowej krotki, które zostały już przetworzone przed awarią procesu ekstrakcji,
- filtry typu *Prefix* (*ClearPrefix* i *DirtyPrefix*) usuwające prefiks (pierwszych *n* krotek) z sekwencji wejściowej, które można bezpiecznie odfiltrować.

W każdym z tych przypadków filtr otrzymuje sekwencję wyjściową danego węzła X i tworzy sekwencję wynikową Y_X dla węzła Y.



Rys. 3. Węzły X i Y przed (a) i po (b) przypisaniu filtru F
Fig. 3. Nodes X and Y before (a) and after (b) assignment of filter F

2.2.4.2. Procedury reekstrakcji

Procedury reekstrakcji mają budowę filtrów. W szczególności, procedury *GetSuffix* i *GetSubset* filtrują krotki w ten sam sposób, jak czynią to odpowiednio filtry *ClearSuffix* i *ClearSubset*. Pozostałe dwa filtry odpowiadają procedurze *GetDirtySuffix* i *GetDirtySubset*. Rodzaj przypisanej przez Design procedury reekstrakcji zależy od typu i możliwości danego ekstraktora. Jeżeli żaden filtr nie może być przypisany, procedura Design wybiera *GetAll* lub *GetAllReordered*, w zależności od typu ekstraktora.

3. Implementacja algorytmu odtwarzania DR

3.1. Język programowania i środowisko programistyczne

Do implementacji algorytmu DR wykorzystano język programowania Java. Programy napisane w Javie posiadają cechy, takie jak: obiektowość, przenośność kodu, wielowątkowość, przejrzystość kodu, bogate biblioteki klas w ramach Java Foundation Class (JFC), i można je uruchomić na wszystkich platformach sprzętowych, dla których powstała Wirtualna Maszyna Javy (ang. *Java Virtual Machine*, JVM). Obecnie wszystkie liczące się systemy operacyjne posiadają swoją własną implementację JVM.

Ze względu na bardzo dużą ilość przetwarzanych danych minimalizowanie czasu wykonania zadań ekstrakcji jest ważnym zagadnieniem. Ze względu na uniwersalność kodu

programy napisane w Javie mogą być wolniejsze w działaniu niż programy skompilowane pod dany system operacyjny. Wynika to z narzutu czasowego, jaki wnosi warstwa oprogramowania JVM konieczna do uruchomienia kodu bajtowego Javy (ang. *B-code, bytecode*). Jednak najnowsze kompilatory i wysoko zoptymalizowane interpretery Javy mogą dostarczyć wydajności porównywalnej z kodem napisanym w C++ bez utraty uniwersalności.

Java doskonale wspomaga tworzenie aplikacji wielowątkowych. Tworzenie i sprawne zarządzanie wątkami w połączeniu ze skutecznie zrealizowaną obsługą dostępu do pamięci współdzielonej spełnia wymagania nałożone na środowisko o architekturze wielowątkowej. Java dostarcza również bardzo elastycznych struktur do reprezentacji danych wraz z różnorodnymi metodami dostępu do nich. Platforma Java w ramach technologii JavaBeans bardzo mocno wspomaga tworzenie zaawansowanych komponentów pozwalających na późniejsze ich wykorzystanie w tworzonych aplikacjach. Powyższe cechy w połączeniu z możliwością przenoszenia kodu między różnymi systemami przyczyniły się do wyboru Javy jako języka do realizacji projektu.

Do implementacji została wykorzystana technologia JavaBeans [4].

3.2. Technologia JavaBeans wykorzystana w implementacji algorytmu DR

3.2.1. Cechy komponentów JavaBeans

Do budowy komponentów algorytmu DR w technologii JavaBeans został wybrany pakiet Forte for Java (Sun Microsystems) wykorzystujący pakiet SDK Java 2 SE 1.4. Przy pomocy JavaBeans API można tworzyć zaawansowane technologiczne i niezależne od platformy systemowej komponenty, które można następnie wykorzystać w środowiskach programistycznych IDE wspomagających JavaBeans. Komponenty udostępniają swoje właściwości, publiczne metody i zdarzenia narzędziom typu IDE, które na podstawie ich analizy pozwalają na konfigurację komponentu w czasie programowania. Użycie komponentu JavaBeans najczęściej polega na wybraniu go z odpowiedniej palety komponentów i umieszczeniu na formatce projektu. W kolejnym etapie przy pomocy edytora własności ustawia się właściwości komponentu i jego interakcję z pozostałymi komponentami.

Komponent JavaBeans może zapisywać i odtwarzać swój stan dzięki zastosowaniu serializacji obiektów w ramach mechanizmu Java Object Serialization. Własności danego komponentu, których stan powinien być pamiętany, powinny być serializowalne. Robi się to poprzez implementację interfejsu *java.io.Serializable* w każdej klasie, której obiekty tworzą własności komponentów.

3.2.2. Własności komponentów

Własności komponentów JavaBeans dają programiście możliwość konfigurowania komponentu w czasie programowania. Przy pomocy odpowiednich edytorów można ustawiać i odczytywać wartości danej własności.

W realizacji projektu wykorzystano trzy rodzaje własności:

- własności zwykłe (ang. *simple properties*): można je ustawiać i odczytywać, a skutki zmiany własności są widoczne tylko w obrębie instancji danego komponentu;
- własności powiązane (ang. *bound properties*): zmiana własności jest widziana przez inne obiekty, jeżeli te implementują interfejs *java.beans.PropertyChangeListener* i są zarejestrowane na liście nasłuchu (ang. *listeners*) obiektu, do którego należy własność typu *bound*;
- własności indeksowane (ang. *indexed properties*): podobne do własności zwykłych, służą do reprezentacji kolekcji wartości, do których dostęp jest realizowany przez indeksy (jak w tablicach). Posiadają dwa rodzaje metod *get* i *set*: metody indeksowe i metody zwykłe.

Narzędzia typu IDE „odkrywają" konfigurowalne własności danego komponentu w ramach procesu zwanego introspekcją (ang. *introspection*). W projekcie zrealizowano introspekcję poprzez wykorzystanie skojarzonej z danym komponentem klasy implementującej interfejs *java.beans.BeanInfo*. Klasa ta zawiera listę własności, metod i zdarzeń, które będą udostępnione użytkownikowi na etapie konfiguracji komponentu.

W projekcie do konfigurowania komponentów (ang. *customising*) wykorzystano specjalnie w tym celu stworzone edytory własności (ang. *properties editors*). Zrealizowano to poprzez stworzenie nowej klasy implementującej interfejs *java.beans.PropertyEditor*.

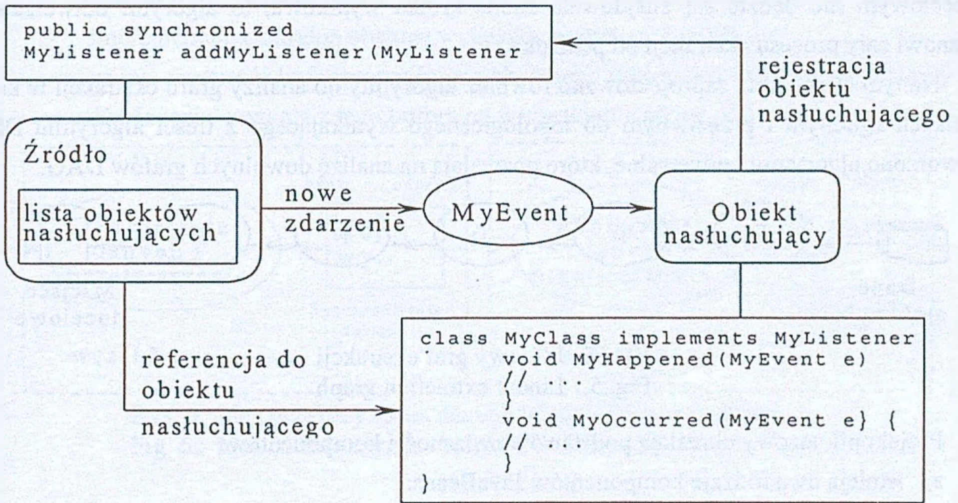
3.2.3. Zdarzenia generowanie przez komponenty

Komponenty JavaBeans używają zdarzeń (ang. *events*) do komunikacji z innymi komponentami. Komponent, który chce otrzymywać zdarzenia (ang. *event listener*) rejestruje się w komponencie, który jest źródłem danego zdarzenia (ang. *event source*). Do rejestracji i jej usuwania służą następujące metody, które muszą być dostarczone przez klasę będącą źródłem zdarzenia:

```
public void add<EventListenerType>(<EventListenerType> a)
public void remove<EventListenerType>(<EventListenerType> a)
```

Aby dany komponent mógł się zarejestrować, musi implementować jeden z interfejsów dziedziczących po *java.util.EventListener*. Musi on być odpowiedni dla danego typu zdarzenia. W projekcie wykorzystano interfejs *java.beans.PropertyChangeListener*, który implementuje wszystkie komponenty tworzące węzły grafu ekstrakcji. Interfejs ten pozwala informować o zmianach własności danego komponentu.

Każde zdarzenie jest reprezentowane przez obiekt dziedziczący po klasie *java.util.EventObject* (np. obiekt typu *PropertyChangeEvent*), który zawiera informacje o zdarzeniu oraz o źródle zdarzenia. Jest on przekazywany jako argument metodzie obsługującej zdarzenie w obiekcie nasłuchującym (np. metodzie propertyChange()). Takie źródło zdarzeń może mieć wiele obiektów nasłuchujących, a pojedynczy obiekt może być zarejestrowany do wielu źródeł zdarzeń. Oczywiście, obiekt nasłuchujący może być również źródłem zdarzeń. Na rys. 4 został przedstawiony model zdarzeń realizowany w JavaBeans.



Rys. 4. Model zdarzeń w technologii JavaBeans [3]
Fig. 4. Events model in JavaBeans technology [3]

3.3. Realizacja projektu

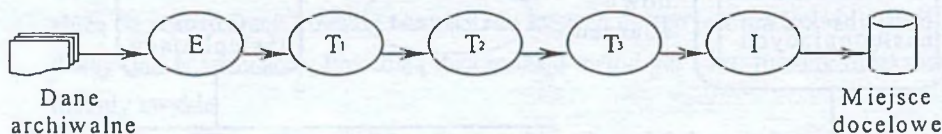
Realizacja projektu, którego celem była implementacja i ocena wydajności odtwarzania opartego o algorytm DR wymagała opracowania rozwojowego środowiska komponentów odpowiedzialnych za poszczególne etapy ekstrakcji danych, takie jak: selekcja danych źródłowych, transformowanie ich i ładowanie do miejsca docelowego. Komponenty ekstrakcji należało tak zaprojektować, by realizowały proces odtwarzania oparty na algorytmie DR w stopniu wystarczającym do przeprowadzenia jego rzetelnej oceny. Szczególna uwaga została poświęcona komponentom pozyskania i zapisu danych do plików tekstowych.

Środowisko rozwojowe powstawało dwuetapowo: w trakcie prowadzenia Projektu pilotażowego, a następnie Projektu docelowego.

3.3.1. Projekt pilotażowy

W pierwszym etapie nazwanym *Projektem pilotażowym* ograniczono projekt, ze względu na bardzo dużą złożoność algorytmu DR, do odtwarzania w zakresie liniowych grafów ekstrakcji (rys. 5). Na tym etapie przyjęto, że tworzony zbiór komponentów do przetwarzania wielowątkowego będzie „niewidoczny" dla projektanta grafu ekstrakcji. Odtwarzanie algorytmu DR jest tym bardziej skuteczne, im więcej przetworzonych krotek znajdzie się w miejscu docelowym przed przerwaniem. Jeżeli w chwili przerwania procesu ekstrakcji w miejscu docelowym nie będzie się znajdowała żadna krotka wynikowa, to algorytm odtwarzania wznowi cały procesu ekstrakcji od początku.

Na tym etapie prac zaprojektowano również algorytmy do analizy grafu ekstrakcji w kierunkach zgodnym i przeciwnym do topologicznego wynikającego z treści algorytmu DR. Stworzono algorytmy uniwersalne, które pozwalają na analizę dowolnych grafów DAG.



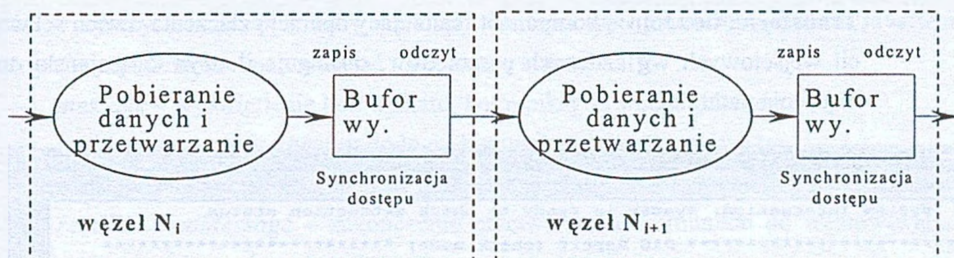
Rys. 5. Liniowy graf ekstrakcji
Fig. 5. Linear extraction graph

Projekt pilotażowy określają podstawowe własności komponentów:

a) istnieją dwa rodzaje komponentów JavaBeans:
   - komponenty-węzły reprezentujące pojedyncze węzły grafu ekstrakcji DAG,
   - komponent-kontener przechowujący komponenty-węzły,

b) każdy komponent-węzeł:
   - realizuje pojedyncze podzadanie ekstrakcji danych,
   - obsługuje jedno wejście i jedno wyjście,
   - implementuje algorytm DR,
   - posiada synchronizowany bufor wyjściowy typu lista (rys. 6),
   - implementuje osobny wątek, który:
     i. jest uruchamiany przez węzeł-wątek poprzedzający go w grafie wtedy, gdy pojawią się dla niego dane wejściowe,
     ii. kończy swoje działanie, gdy przetworzy wszystkie dane wejściowe i zapisze dane wynikowe oraz gdy poprzedzający go wątek w grafie ekstrakcji również skończy działanie,
     iii. komunikuje się z pozostałymi wątkami za pomocą komunikatów JavaBeans,

c) każdy komponent-kontener
- sprawdza poprawność skonstruowanego grafu ekstrakcji,
- tworzy graficzny interfejs użytkownika,
- implementuje główny wątek aplikacji i uruchamia proces ekstrakcji poprzez uruchomienie wątków ekstraktorów,
- implementuje procedury *Design* i *Resume* algorytmu DR,
- implementuje obsługę komunikatów tekstowych generowanych przez procesy ekstrakcji i odtwarzania,
- implementuje centralną obsługę wyjątków ekstrakcji danych,
- usuwa zawartość buforów wyjściowych danego wątku, gdy otrzyma komunikaty o zakończeniu czytania z tego bufora od wszystkich wątków, które z niego czytały.



Rys. 6. Idea przekazywania danych między dwoma wątkami
Fig. 6. Idea of data transfer between two threads

3.3.2. Projekt docelowy

Projekt docelowy jest rozwinięciem projektu pilotażowego, w którym usuwa się jego ograniczenia. W stosunku do projektu pilotażowego rozbudowany został zbiór komponentów. Wszystkie komponenty-węzły grafu ekstrakcji obsługują wiele wejść i wiele wyjść (rys. 2).

Każdy komponent został wyposażony w odpowiednie edytory własności, pozwalające między innymi ustawiać własności węzłów wynikające z algorytmu DR. Komponenty zostały tak zaprojektowane, aby w sposób „niewidoczny" dla użytkownika implementowały algorytm odtwarzania DR i stosują przetwarzanie wielowątkowe.

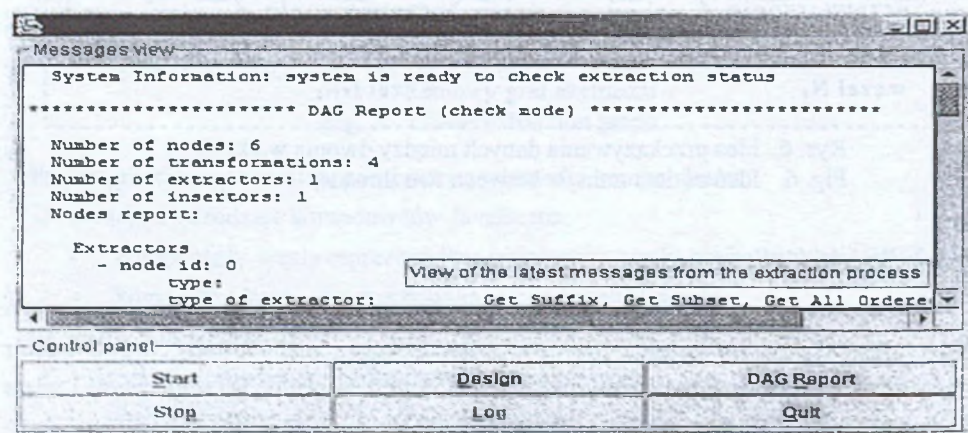
Stworzone środowisko obejmuje następujące komponenty:

a) **JDAGPanel** – komponent-kontener, obowiązkowy element każdej aplikacji ekstrakcji danych, jedyny komponent wizualny w zestawie stworzonych komponentów (rys. 7);

b) **FileExtraction** – komponent odpowiedzialny za realizację ekstrakcji właściwej z dowolnego pliku dyskowego;

c) **FileInsertor** – komponent odpowiedzialny za zapis wynikowej sekwencji krotek do pliku dyskowego o zdefiniowanym przez użytkownika formacie;

d) komponenty transformacji:
  - **TransformationFilter** – komponent realizujący filtrowanie oraz dowolne transformowanie pojedynczych krotek z sekwencji wejściowej (dodanie, modyfikacja, usunięcie atrybutu);
  - **TransformationGroup** – komponent realizujący operację grupowania całej sekwencji wejściowej wg zadanych parametrów, posiada funkcjonalność frazy GROUP BY wraz z funkcjami grupującymi (MAX, MIN, SUM, COUNT, AVG);
  - **TransformationAggr** – komponent realizujący agregację sekwencji wejściowej wg zadanych parametrów;
  - **TransformationSort** – komponent sortujący wejściową sekwencję krotek wg zadanych parametrów, obsługuje sortowanie numeryczne i leksykograficzne;
  - **TransformationJoin** – komponent realizujący operację złączenia dwóch sekwencji wejściowych wg zadanych parametrów; obsługuje iloczyn kartezjański oraz złączenia naturalne.



Rys. 7. Interfejs użytkownika (komponent **JDAGPanel**)
Fig. 7. User interface (**JDAGPanel** component)

Komponenty reprezentujące węzły grafu ekstrakcji tworzą hierarchię, która może być rozbudowywana o kolejne komponenty. Klasą bazową jest klasa *AbstractNode*, która implementuje mechanizm wymiany komunikatów między węzłami tworzącymi graf ekstrakcji. Dołączenie poprzez mechanizm dziedziczenia nowej klasy (komponentu) nie wymaga modyfikacji kodu procedur *Design* i *Resume* zaimplementowanych w komponencie **JDAGPanel**. Dodane komponenty mogą obsługiwać wiele wejść.

W skład środowiska wchodzą również następujące klasy pomocnicze:

a) klasy reprezentujące węzły filtrujące w algorytmie DR (obiekty tych klas są generowane dynamicznie w czasie działania procedury *Resume* algorytmu DR), mają one strukturę i funkcjonalność zbliżoną do komponentu **TransformationFilter**. Stwo-

rzono następujące klasy filtrów: **ClearPrefixFilter, DirtyPrefixFilter, ClearSubsetFilter, DirtySubsetFilter,**

b) klasy wizualne implementujące dedykowane edytory własności komponentów, używane przez narzędzia IDE,

c) klasy **BeanInfo** służące do odpowiedniej prezentacji stworzonych komponentów,

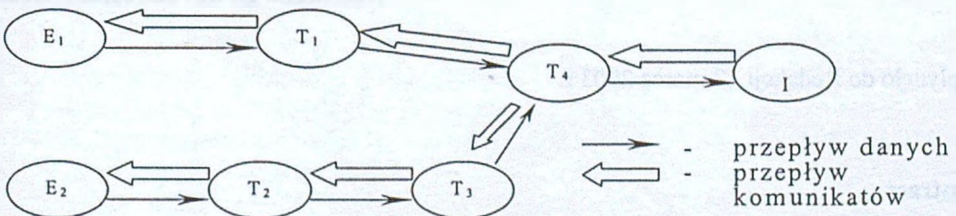
d) klasy reprezentujące wejścia węzłów grafu ekstrakcji.

3.3.3. Komunikacja międzywęzłowa

Przetwarzanie danych realizowane przez wątki-węzły grafu ekstrakcji jest oparte na wielowątkowości. Zastosowanie takiego podejścia wymagało stworzenia skutecznego mechanizmu wymiany komunikatów między wątkami grafu ekstrakcji. Skorzystano z komunikatów wysyłanych podczas zmiany wartości własności typu *bound* komponentu JavaBeans. Każdy z komponentów może wysyłać następujące komunikaty:

- *message* – wyświetlenie komunikatu; komunikat do wątku głównego,
- *logMessage* – zapis wyjątku do logu; komunikat do wątku głównego,
- *resultCode* – zakończenie ekstrakcji; komunikat do wątku głównego,
- *readFinishedMessage* – zakończenie przetwarzania; komunikat do węzłów-wątków poprzedzających dany węzeł-wątek.

Aby zdarzenie wysyłania i odbierania komunikatów mogło nastąpić, musi być ono poprzedzone etapem rejestracji nasłuchu. Rejestracja obiektów nasłuchujących jest realizowana w dwóch etapach przed uruchomieniem procesu ekstrakcji. Pierwszy etap ma miejsce zaraz po utworzeniu i pokazaniu formatki głównej aplikacji. Tworzone są wówczas połączenia między obiektem głównym aplikacji (klasy **JDAGPanel**) a każdym węzłem grafu ekstrakcji. Połączenia pozwalają na dwukierunkową wymianę komunikatów między wątkami węzłów a wątkiem głównym. Drugi etap ustawia połączenia między połączonymi bezpośrednio w grafie ekstrakcji węzłami. Jest on realizowany pod koniec wykonywania procedur *Design* i *Resume* algorytmu DR. Połączenia są jednokierunkowe – komunikaty będą przesyłane z węzła wysyłającego komunikat do wszystkich bezpośrednio go poprzedzających węzłów.



Rys. 8. Komunikacja międzywęzłowa
Fig. 8. Node-Node communication

4. Podsumowanie

W obszarze odtwarzania procesów ekstrakcji danych do hurtowni danych poszukuje się algorytmów, które minimalizują czas realizacji obsługi odtwarzania w aspekcie:

- dodatkowego obciążenia procesów ekstrakcji oraz
- modyfikacji kodu transformacji.

I pod tym względem najlepsze obecnie wyniki zapewnia algorytm Design-Resume.

Zaproponowany zbiór komponentów implementujących algorytm DR tworzy rozwojowe środowisko komponentów pozwalające na tworzenie złożonych aplikacji realizujących zadania ekstrakcji danych. Uzyskanie poprawnych wyników pozwoliło stwierdzić, że działanie aplikacji ekstrakcji i mechanizmu odtwarzania zostało poprawnie zaimplementowane. Stworzone środowisko jest punktem wyjścia do dalszych prac nad szukaniem efektywnego algorytmu odtwarzania danych.

Ocena efektywności i skuteczności odtwarzania przerwanych procesów ekstrakcji danych algorytmem DR oraz wskazanie możliwości jego dalszego rozwoju zostanie przedstawiona w kolejnym artykule.

LITERATURA

1. Gorawski M., Koziatek A.: Data Warehouse: Ekstrakcja danych, Software 2000.
2. Gorawski M., Koziatek A.: Systemy DSS: Projekt ekstrakcji danych, Informatyka 8/2000.
3. Java™ 2 SDK, Standard Edition, Documentation, Sun Microsystems, 2002.
4. JavaBeans™ API Specification, Sun Microsystems, 1997.
5. Labio W., Wiener J., Garcia-Molina H., Gorelik V.: Efficient Resumption of Interrupted Warehouse Loads, Stanford University, Sagent Technologies.
6. Powermart 4.0. Technical Overview, Informatica, 1999.

Recenzent: Dr inż. Arkadiusz Sochan

Wpłynęło do Redakcji 12 marca 2003 r.

Abstract

Data warehouse is a subject-oriented database that consolidates, aggregates and gathers gigabytes of data for online analytical processing. A data warehouse is a foundation for deci-

sion-making support systems. Data warehouses are loaded with data that usually come from various data sources. Programming errors or hardware failure can interrupt this long-lasting loading process. Classic recovery algorithms often bring in not acceptable additional processing burden and require long time to complete. The best results in recovery of the loading process are provided at present by Design-Resume algorithm implemented in C language at Stanford University in 1999. In our research we present another implementation of this algorithm using JavaBeans technology.

Adresy

Marcin GORAWSKI: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-101 Gliwice, Polska, M.Gorawski@zti.iinf.polsl.gliwice.pl.

Andrzej WOCŁAW: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-101 Gliwice, Polska, A.Woclaw@zti.iinf.polsl.gliwice.pl.