STUDIA INFORMATICA 2003
Volume 24 Number 4 (56)

Marcin GORAWSKI, Przemysław SIÓDEMAK
Politechnika Śląska, Instytut Informatyki

GRAFICZNE PROJEKTOWANIE APLIKACJI ETL

Streszczenie. W celu zoptymalizowania procesu ekstrakcji danych często projektuje się specjalizowaną, dostosowaną do wymagań konkretnego systemu hurtowni danych, własną aplikację ETL [10,15]. W pracy przedstawiono projekt i realizację graficznego środowiska rozwojowego ETL/JB^S do tworzenia aplikacji ETL. Prezentowane środowisko minimalizuje nakłady czasowe realizacji aplikacji ETL w stopniu wyższym niż w środowisku rozwojowym C++ [11, 13] lub ETL/JB [14].

Słowa kluczowe: ekstrakcja danych, hurtownie danych, modele ETL

GRAPHIC DESIGN OF ETL APPLICATIONS

Summary. When optimizing data extraction it is common to create ETL [10, 15] application specialized and adjusted to a particular data warehouse system. In the following paper we present a project and realization of the ETL/JB^S graphic development environment for creating ETL applications. Presented environment minimizes amount of time needed to create ETL application, in a higher degree then C++ [11, 13] or ETL/JB [14] development environment.

Keywords: extraction process, data warehouses, ETL models

1. Wprowadzenie

Zasadniczą koncepcją hurtowni danych (*ang. Data Warehouse - DW*) jest zintegrowanie danych z różnych źródeł w jedną odpowiednią kolekcję danych, którą aktualizujemy w pewnych odstępach czasu w celu ich bieżącej analizy dla wspomagania podejmowania decyzji w organizacji [1, 2, 4, 5, 6, 7, 8]. Uporządkowany zbiór zadań związanych z

przenoszeniem danych ze źródeł danych do DW zwany jest potocznie ekstrakcją danych. W procesie ekstrakcji danych wyróżnia się trzy odrębne etapy:

- pobranie danych z systemu źródłowego (*ang. Extraction*),
- przekształcenie danych do pożądanej postaci (*ang. Transformation*),
- załadowanie danych do hurtowni (*ang. Load*).

Stąd procesy ekstrakcji danych określane są zwykle mianem procesów ETL [3, 9].

W pracy [14] opisano sposób tworzenia aplikacji procesów ETL (w skrócie: aplikacje ETL) poprzez udostępnienie zbioru komponentów ETL w postaci pliku .jar w środowisku rozwojowym ETL/JavaBeans (ETL/JB). Ten sposób tworzenia aplikacji ETL polega na określeniu: potrzebnych komponentów, kolejności ich wykonania, atrybutów wejściowych i wyjściowych każdego komponentu oraz właściwości z dodatkowymi metodami. Generowany jest plik .java, który następnie jest kompilowany. Otrzymany plik lub pliki .class stanowią aplikację ETL. Uruchomienie odbywa się z wykorzystaniem pliku java.exe. W praktyce korzystanie ze środowiska ETL/JB jest dość uciążliwe – wielokrotnie trzeba definiować te same zmienne, łatwo popełnić błędy, które uniemożliwią prawidłową realizację procesu ekstrakcji [15].

Niniejsza praca przedstawia graficzny sposób projektowania aplikacji ETL, który eliminuje ww. wady, stanowiąc zarazem kolejny etap rozwoju metod projektowania procesu ekstrakcji. Podstawą jest środowisko rozwojowe ETL/JavaBeans/JavaSwing (ETL/JB^S). Środowisko ETL/JB^S łączy model konceptualny procesu ETL z jego logicznymi i fizycznymi odpowiednikami, ze szczególnym uwzględnieniem (a) zależności pomiędzy zadaniami ETL i rozważanymi źródłami danych; (b) uchwycenia połączeń strumienia procesów w scenariuszu ETL oraz (c) optymalizację scenariuszy wykonania.

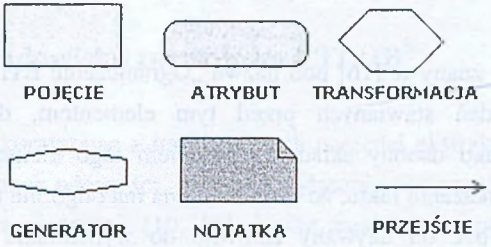
2. Model konceptualny procesu ETL

W pracy [16] zaproponowano model konceptualny cechujący się tym, że:

- opisuje mechanizm odkrywania zależności pomiędzy atrybutami i odpowiednimi zadaniami ETL we wczesnych etapach projektu DW,
- konstruowany jest w konfigurowalny i rozszerzalny sposób, tak aby projektant mógł wzbogacić go o własne sekwencje zadań ETL,
- wskazuje paletę najczęściej używanych zadań ETL, tj.: przydzielanie zastępczych identyfikatorów, sprawdzenie wartości typu null itd.

Model ten opisuje ogólne jednostki zdolne do reprezentowania każdego scenariusza ETL oraz warstwę wzorca, która jest zbiorem „wbudowanych" typowych jednostek scenariuszy ETL.

Na rysunku 1 przedstawiono symbole graficzne reprezentujące podstawowe elementy modelu konceptualnego. Atrybuty i pojęcia są pierwszoplanowymi elementami tego modelu.



Rys. 1. Symbole graficzne elementów modelu konceptualnego
Fig. 1. Graphic symbols of conceptual model

Atrybuty

Reprezentują najmniejszy, pojedynczy zbiór informacji (np. kolumnę danych) definiowany przez nazwę oraz typ. Założono, że atrybut może być jednym z trzech typów: *łańcuch znaków* (string), *liczba całkowita* (int), *liczba zmiennoprzecinkowa* (float). Typom atrybutów przyporządkowano odpowiednie operacje.

Pojęcia

Reprezentują jednostkę w bazie źródłowej lub docelowej. Wyróżnia się dwa typy pojęć: a) *źródłowe*, z których pobierane są dane lub b) *docelowe*, do których zapisywane są dane. Konkretnym wystąpieniem mogą być pliki w bazie źródłowej, tablice faktów i wymiarów w DW itd. Formalnie pojęcie definiowane jest przez nazwę oraz zbiór związanych z nim atrybutów.

Transformacje

Reprezentują pewną abstrakcję opisującą kody wykonujące pewne zdefiniowane operacje na danych. Ogólnie można je podzielić na transformacje filtrujące i czyszczące dane (filtry, funkcje) oraz na transformacje używane do przetwarzania danych (funkcje, agregacja, złączenie). Pierwszy typ charakteryzuje się tym, że jest używany do usuwania rekordów, które nie spełniają zadanych przez użytkownika warunków. Z kolei drugi typ może wpływać na schemat przechodzących przez niego danych – dodawać, zmieniać i usuwać kolumny. Funkcje są uważane za należące do obu tych grup, gdyż z jednej strony mogą operować na pojedynczym atrybucie, przygotowując go do operacji filtrowania (np. poprzez kapitalizacje liter i usunięcia dodatkowych znaków), a z drugiej strony mogą tworzyć, zmieniać istniejące kolumny danych lub dodawać nowe na podstawie istniejących. W przypadku modelu

konceptualnego zastosowanego w tej pracy z transformacjami mogą być związane pewne atrybuty reprezentujące kolumny, jakie powstają dodatkowo (lub zamiast) w wyniku działania danej transformacji w stosunku do kolumn danych wejściowych.

Generator

Element ten jest znany w [16] pod nazwą „Ograniczenie ETL". Generator obejmuje tylko część zadań stawianych przed tym elementem, dlatego też został on wprowadzony jako osobny składnik. Zadaniem tego elementu jest umożliwienie projektantowi pokazania faktu, że dana kolumna ma odgórnie narzuconą strukturę lub wartość. Może być on używany zarówno do wypełniania pojedynczą wartością podanego atrybutu (kolumny w danych), jak i do przydzielania unikalnych wartości wg określonej zasady generacji.

Notatka

Są to znaczniki używane do opisu modelu. Mogą zawierać dodatkowe komentarze umieszczane przez projektanta w czasie fazy projektowania. Notatki mogą być przyłączane do każdego innego elementu z modelu konceptualnego. Są one przedstawione jako prostokąt z zawiniętym rogiem. Najczęściej element ten to proste komentarze wyjaśniające semantykę stosowanych transformacji (np. w przypadku złączenia notatka może zawierać opis warunków złączenia) oraz opis pewnych ogólnych ograniczeń całego procesu (np. założenie, że czas całego ładowania nie może być dłuższy niż 4 godziny).

Przejście

Jest to element odzwierciedlający kolejne etapy przekształcania wybranego atrybutu. Przejście zawsze zaczyna się w atrybucie, a kończy albo na wejściu transformacji (co oznacza, że wskazany atrybut będzie podlegał wybranej operacji), albo na atrybucie w pojęciu docelowym (co oznacza fakt bezpośredniego odwzorowania danego atrybutu w atrybut w pojęciu docelowym). Jest to uproszczenie elementu przedstawionego w [16] jako tzw. zależność udostępniająca (ang. *provider relationship*).

Opisane elementy mogą dotyczyć dwóch poziomów projektowania procesu ETL. Przede wszystkim są używane na poziomie najbardziej szczegółowym, czyli są to zależności pomiędzy atrybutami. Jednak niektóre z nich (np. transformacje) mogą operować także na poziomie wyższym, wskazując powiązania pomiędzy całymi pojęciami.

Z powyższego wynika, że zaprezentowany model konceptualny jest zbiorem ogólnych jednostek zdolnych do zaprezentowania każdego scenariusza ekstrakcji danych. Każdy ww. element może zostać zaimplementowany w dowolny sposób, specyficzny dla wymagań

konkretnego procesu ETL. Mogą być również tworzone podklasy, używane tylko w pewnych określonych sytuacjach.

3. Graficzne środowisko rozwojowe ETL/JB[S]

Zwykle projektant korzystając z uniwersalnych narzędzi ekstrakcji, mając udostępnione pewne operacje i określone własności, definiuje wyłącznie połączenia pomiędzy elementami oraz konkretne zadania ekstrakcji [10, 12]. Także model konceptualny udostępnia zbiór komponentów dowolnych implementacji scenariuszy ETL [16]. Kierując się tym modelem oraz opierając się na prowadzonych pracach [11, 12, 13, 14] utworzono następujące reprezentatywne komponenty:

- TSourceFile – pojęcie źródłowe (wczytywanie danych z pliku),
- TDestFile – pojęcie docelowe (zapisywanie danych do pliku),
- TFilter – transformację eliminacji rekordów nie spełniających określonych wymagań (możliwość definiowania warunków logicznych),
- TFunction – transformację wykonującą dowolną operację na podanych atrybutach wraz z możliwością tworzenia nowych atrybutów (obligatoryjne tworzenie własnych procedur transformacji każdego rekordu wejściowego),
- TJoin – transformację złączenia dwóch źródeł danych na podstawie określonych kryteriów złączenia wg wskazanych kolumn danych wchodzących do elementu,
- TSort – transformację sortującą dane wg określonych kryteriów,
- TAggregate – transformację grupującą i agregującą dane według określonych kryteriów oraz funkcji, tj.: SUM, MIN, MAX, AVG, MOUNT,
- TGenerator – generator unikalnych indeksów oraz stałych,
- TNote – notatkę opisu procesu,
- TUnion – sumę dwóch zbiorów danych (wskazany do przetwarzania pojęć a nie atrybutów).

Komponenty, tj.: TSourceFile, TDestFile, TFilter, TFunction, TAggregate, TGenerator do poprawnego działania wymagają znajomości tylko bieżącego rekordu. Z kolei komponent sortujący TSort musi pobrać wszystkie rekordy wejściowe przed rozpoczęciem właściwej operacji sortowania. Podobnie funkcjonuje komponent TJoin, z tą różnicą, że konieczne jest wczytanie do komponentu zawartości drugiego wejścia tak, aby mogło się odbywać łączenie z elementami przychodzącymi z pierwszego wejścia. Elementy TJoin oraz TUnion będą udostępniały dwa wejścia oraz jedno wyjście.

Mimo dużej różnorodności zaimplementowanych elementów można wyodrębnić pewne cechy wspólne, tj.:

- nazwa elementu i unikalny identyfikator;
- reprezentacja graficzna, zmieniająca się w zależności od warunków i właściwości;
- kolumna lub kolumny wejściowe i/lub wyjściowe, scharakteryzowane przez nazwę oraz typ kolumny (integer, float, string);
- bufory wejściowe oraz wyjściowe będą tablicami, dzięki czemu można osiągnąć maksymalną szybkość odczytu i zapisu danych oraz minimalną zajętość pamięci (w porównaniu z kontenerami udostępnianymi przez język Java);
- mechanizm komunikacji oparty na zasadzie wymiany komunikatów w JavaBeans;
- każdy komponent implementuje minimum jeden osobny wątek. Wątek ten jest uruchamiany w momencie otrzymania powiadomienia od poprzedniego elementu, że są już dane w buforze wejściowym. Następnie odbywa się właściwe przetwarzanie. Wątek kończy działanie, gdy wykryje znak końca danych wśród danych wejściowych. Komponenty TFilter oraz TFunction udostępniają możliwości uruchomienia kilku wątków filtrowania oraz transformacji przychodzących rekordów;
- każdy komponent będzie tworzył swój log tekstowy, w którym będą umieszczane informacje dotyczące działania i otrzymanych wyników.

Komponenty te umieszczone są w graficznym środowisku rozwojowym, które zapewnia za pomocą panelu projektowego tworzenie projektów i aplikacji ETL oraz ich uruchamianie.

Środowisko to zaimplementowano w technologii JavaBeans [17] oraz Java Swing [18] wspierając budowę graficznego interfejsu użytkownika. Stąd nazwa graficzne środowisko rozwojowe ETL/JavaBeans/JavaSwing (ETL/JB^S), w skrócie środowisko ETL/JB^S.

Stworzone środowisko ETL/JB^S składa się z dwóch odrębnych, ale wzajemnie uzupełniających się warstw. Pierwsza z nich tworzy zbiór komponentów ETL zaimplementowanych w technologii JavaBeans. Pojedyncze zadanie ekstrakcji należy rozumieć jako ciąg zadań zaczynający się od pobrania danych z plików źródłowych, a kończący się na zapisie przetworzonych danych do plików docelowych. W tym celu wykorzystywane są komponenty: TSourceFile i TDestFile. Pomiędzy tymi komponentami projektant aplikacji ETL może umieścić inne komponenty, takie jak: TFilter, TFunction, TSort, TJoin, TAggregate, TGenerator, TUnion, TNote. Drugą warstwę tworzy panel projektowy „ETL Builder" zrealizowany w technologii Java Swing. Panel pozwala zarządzać projektami i arkuszami projektowymi, zbudować system ekstrakcji danych z dostępnych komponentów oraz uruchamiać zadania ekstrakcji.

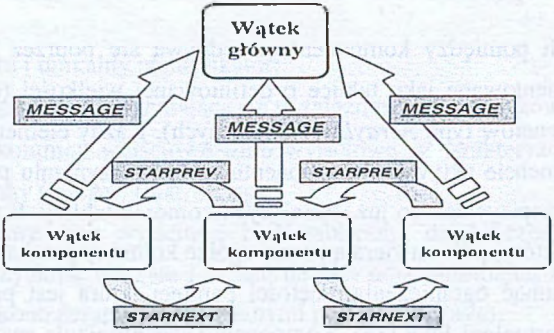
3.1. Zasady wymiany danych i komunikatów pomiędzy komponentami

Wymiana danych pomiędzy komponentami odbywa się poprzez bufory wejściowe i wyjściowe zaimplementowane jako tablice o definiowanej wielkości (dostęp do tablic jest szybszy niż do kontenerów typu *ArrayList* i podobnych). Każdy element tablicy inicjowany jest tylko raz w momencie aktywacji komponentu lub po otrzymaniu pierwszych danych, a nowe elementy są przepisywane do już istniejących komórek tablicy. Każda komórka tablicy jest również tablicą, której pola zawierają poszczególne kolumny przekazywanej danej. W ten sposób udało się usunąć ograniczenia zajętości pamięci, która jest przyczyną powolnego działania maszyny wirtualnej Javy [14]. Komponenty synchronizują swoje działanie poprzez wymianę komunikatów „PropertyChangeSupport", znając element poprzedni i kolejny. Każdy komponent komunikuje się dodatkowo z wątkiem nadzorującym całość procesu odpowiedzialnego za odpowiednią konfigurację elementów, wizualizację informacji i zamknięcie procesu. Komponent implementuje swój wątek, który realizuje działanie odpowiednie dla danego elementu. Wątki działają do czasu przetworzenia wszystkich rekordów (czekają na semaforach, gdy nie mają danych, i nie zajmują czasu procesora). Dane przekazywane są z komponentu do komponentu. Element, który skończy swoje operacje, informuje swojego następnika, że bufor wyjściowy jest pełny i można działać. Ponadto wysyłany jest komunikat do elementu poprzedzającego, który również może zacząć przetwarzać swoje dane wejściowe. Komponent odbiera komunikat i uaktywnia związany wątek (przez zwolnienie semafora lub ustawienie flagi). Dzięki temu można zwiększyć ilość wątków operujących w ramach komponentu TFilter oraz TFunction.

W grafie wymiany komunikatów (rys. 2) wyróżniamy:

- START – uruchomienie wątku związanego z komponentem,
- STOP – zatrzymanie i koniec przetwarzania,
- STARTNEXT – komponent informuje swojego następnika, że jego bufor wejściowy jest pełny i może zacząć przetwarzanie,
- STARTPREV – komponent informuje swojego poprzednika, że jego bufor wyjściowy jest pusty i może zacząć przetwarzanie bez obaw, że nadpisze jeszcze nieprzetworzone dane,
- MESSAGE – informacja dla wątku głównego związanego z procesem; zostanie ona wyświetlona w okienku prezentującym przebieg procesu.

Dodatkowo jest jeszcze używany znacznik DATAFINISHED, który zostaje umieszczony w pustym buforze. Po napotkaniu takiego znacznika wątek komponentu kończy swoje działanie.



Rys. 2. Graf wymiany komunikatów
Fig. 2. Communications exchange graph

Aby mogła wystąpić wymiana komunikatów pomiędzy komponentami, musi najpierw nastąpić faza rejestrowania słuchaczy. Jest to wykonywane w momencie budowania modelu, w panelu projektowym „ETL Builder". I tak, wątek główny staje się słuchaczem każdego komponentu. Jest to połączenie jednokierunkowe. Poza tym każdy komponent jest łączony z komponentem poprzednim oraz następnym w grafie ekstrakcji. W tym przypadku występuje połączenie w obu kierunkach.

3.2. Wybrane aspekty implementacyjne

Warto zaprezentować pewne wybrane rozwiązania implementacyjne, między innymi pseudokody komponentów TFilter i TJoin, ich konfigurowanie oraz panel projektowy ETL Builder.

3.2.1. Pseudokody komponentów TFilter i TJoin

Poniżej przedstawiono pseudokody dwóch komponentów TFilter i TJoin

- TFilter

Zawartość metody run():

startnext – zmienna oznaczająca czy był komunikat STARTNEXT

startprev – zmienna oznaczająca czy był komunikat STARTPREV

```
powiadom wątek główny procesu
while stan wątku różny od FINISHED
     if startnext==true && startprev==true do
          startnext==false;
          startprev==false;
          copyData();
     else do
          przejdź w stan zawieszenia;
     end if
```

```
end while
powiadom wątek główny procesu
```

Zawartość metody copyData():

counterIn – licznik z bufora wejściowego

counterOut – licznik z bufora wyjściowego

liveThreads – liczba żywych wątków

packSizeOut – wielkość bufora wyjściowego

packSizeIn – wielkość bufora wejściowego

```
if counterIn==0 do
      odczytaj wiersz z pozycji 0;
      counterIn++;
      if pierwszy raz do
            zsynchronizuj indeksy;
            zapamiętaj strukturę rzędu;
      end if
end if

if counterOut==0 do
      zapisz strukturę rzędu na pozycję 0;
      counterOut++;
end if
puść wątki pomocnicze;

stój na semaforze - czekaj aż wszystkie wątki skończą działanie;

if liveThreads==0 do
      wpisz DATAFINISHED do bufora wyjściowego;
      counterOut++;

      wyślij STARTNEXT;
      ustaw stan wątku na FINISHED;
else do
      if counterOut==packSizeOut do
            counterOut=0;
            wyślij STARTNEXT;
      else do
            startprev=true;
      end if

      if counterIn==packSizeIn do
            counterIn=0;
            wyślij STARTPREV;
      else do
            startnext=true;
      end if
end if
```

Algorytm wątku pomocniczego:

```
while true do
      odczytaj wiersz z bufora wejściowego;
      if wiersz == null do
            poinformuj że wątek czeka;
            zawieszenie;
      else if wiersz==DATAFINISHED do
```

```
            break;
      else do
            wynik = sprawdź wiersz;
            if wynik != null do
                  próbuj zapisać do bufora wyjściowego;
                  if błąd przy zapisie do
                        poinformuj że wątek czeka;
                        zawieszenie;
                        wpisz element do bufora wyjściowego;
                  end if
            end if
      end if
end while
poinformuj że wątek skończył;
```

Wątek elementu rozpoczyna działanie po otrzymaniu komunikatów STARTNEXT od elementu poprzedniego (są dane wejściowe) oraz STARTPREV od elementu kolejnego (bufor wyjściowy został już przetworzony – można nadpisywać). W tym momencie uruchamiane są wątki realizujące filtrowanie. W każdym wierszu wyszukiwany jest najpierw atrybut, który ma podlegać filtrowaniu, następnie jest on rzutowany na konkretny typ i sprawdzany najpierw przez funkcję realizującą warunek prosty, a później przez funkcję z warunkiem zewnętrznym zdefiniowanym przez użytkownika (tylko wtedy, gdy jest aktywny). Jeżeli na wyjściu tych funkcji pojawi się wartość null, to rekord jest traktowany jako błędny i zapisywany do pliku raportu. W przeciwnym przypadku analizowany wiersz jest umieszczany w buforze wyjściowym. Takie przetwarzanie trwa tak długo, aż zabraknie danych (wtedy wątek zakończy działanie) lub, gdy będzie pełny bufor wyjściowy (wtedy wysyła STARTNEXT i wait), lub, gdy skończą się dane w buforze wejściowym (wtedy wysyła STARTPREV i wait). Może się zdarzyć, że równocześnie będzie wysłane STARTPREV i STARTNEXT.

- TJoin

Zawartość metody run():

a, b – flagi

canStartJoining – flaga decydująca czy skończyło się już kopiowanie do pamięci i można przystąpić do łączenia

joiningFlag – flaga określająca czy jest STARTNEXT od pierwszego źródła

startnext – czy było STARTNEXT do drugiego źródła

startprev – czy było STARTPREV od kolejnego komponentu

```
powiadom wątek główny procesu
while stan wątku różny od FINISHED
      if startnext==true do   //od drugiego źródła
            startnext=false;
            copyToMemory();
      else do
            a=true;
```

```
      end if

      if startprev==true && canStartJoining==true && joiningFlag==true do
            startprev=false;
            joinData();
      else do
            b=true;
      end if

      if a==true && b==true do
            przejdź w stan zawieszenia;
            a= false;
            b= false;
      end if
end while
powiadom wątek główny procesu
```

Zawartość copyToMemory():

counter – licznik z bufora wejściowego drugiego źródła danych

memoryPackSize – wielkość bufora drugiego źródła danych

```
while true
      odczytaj wiersz z bufora drugiego z pozycji counter;
      if wiersz różny od DATAFINISHED do
            if pierwszy raz && counter==0 do
                  synchronizuj indeksy dla wejścia 2;
            end if

            if counter!=0 do
                  zapisz do pamięci;
            end if
      else
            break;
      end if
      counter++;
end while

if wiersz==DATAFINISHED do
      posortowanie danych w pamięci;
      canStartJoining=true;
else
      wyślij STARTPREV do komponentu z drugiego wejścia;
end if
```

Ogólny zarys łączenia (1 do 1) joinData():

counterOut – licznik do bufora wyjściowego

packSizeOut – wielkość bufora wyjściowego

firstCounter – licznik do bufora wejściowego 1

joiningPackSize – wielkość bufora 1

```
odczytaj wiersz z bufora 1 z pozycji 0;
if wiersz!=DATAFINISHED do
      while counterOut<packSizeOut
            if counterOut==0 do
                  zapisz strukturę danych do bufora wyjściowego;
                  counterOut++;
```

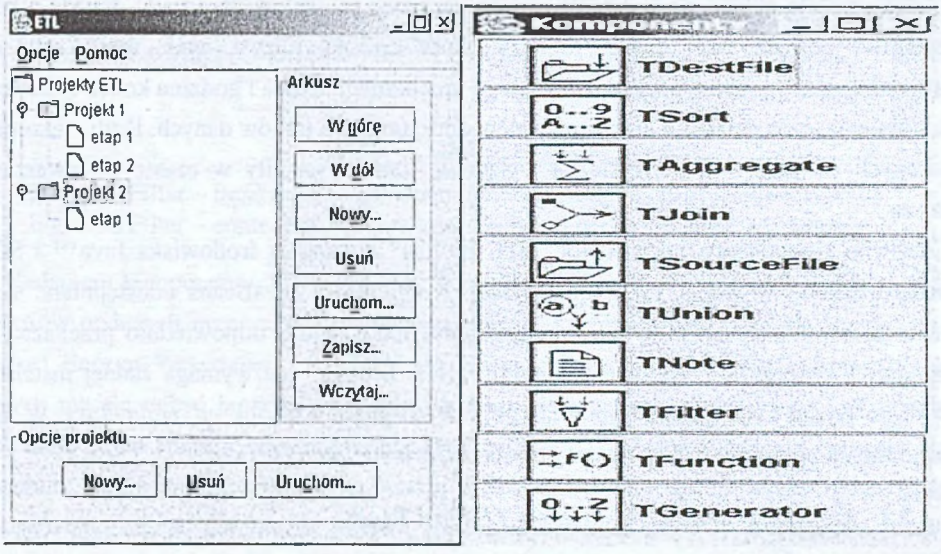
```
            end if
            odczytaj wiersz z bufora 1;
            if wiersz != DATAFINISHED do
                  if pierwszy raz && firstCounter==0 do
                        synchronizuj indeksy;
                        firstCounter++;
                  else
                        połącz element;
                        zapisz do bufora wyjściowego;
                        counterOut++;
                        firstCounter++;
                  end if

                  if firstCounter==joiningPackSize do
                        break;
                  end if
            else
                  break;
            end if
      end while
end if

if wiersz==DATAFINISHED do
      zapisz znacznik końca do bufora wyjściowego;
      counterOut++;
end if
joiningFlag=false;

if counterOut==packSizeOut || wiersz==DATAFINISHED do
      counterOut=0;
      wyślij STARTNEXT;
else
      startprev=true;
end if

if firstCounter==joiningPackSize do
      firstCounter=1;
      wyślij STARTPREV;
else
      joiningFlag=true;
end if
if wiersz==DATAFINISHED do
      stan wątku na FINISHED;
end if
```

Wątek główny operuje na dwóch źródłach danych, stąd potrzebne są dwa bufory wejściowe. W momencie przeciągania strzałek zapamiętywane są identyfikatory źródeł. Dzięki temu można łatwo określić, który element nadał komunikat STARTNEXT. W pierwszej fazie działania odbywa się wczytywanie danych z drugiego źródła do pamięci. Jako pamięć jest wykorzystany obiekt TArrayList, który implementuje kontener za pomocą tablicy (pozwala to zaoszczędzić na zajętości pamięci w porównaniu z kontenerami takimi jak ArrayList). Gdy wszystkie dane z drugiego zbioru zostaną wczytane, rozpoczyna się ich sortowanie. Operacja ta używa wyszukiwania binarnego do szybkiej lokalizacji podobnych rekordów. W tym momencie rozpoczyna się faza właściwego łączenia. Dla każdego rekordu przychodzącego z pierwszego źródła wyszukiwany jest rekord pasujący. Jeżeli udało się go

znaleźć, to tworzony jest nowy rekord zawierający: kolumny, po których odbywało się złączenie, pozostałe kolumny z elementu pierwszego, pozostałe kolumny z elementu drugiego. Taki rekord zostaje zapisany do bufora wyjściowego. W przeciwnym przypadku rekord jest pomijany. Komponent modyfikuje nieznacznie obsługę komunikatów (w stosunku do TFilter i TFunction) tak, aby można było obsłużyć dwa komponenty wejściowe bez groźby zawieszenia wątku głównego. Ustawiane są odpowiednie flagi, które decydują, w jakim momencie przetwarzania jest wątek.

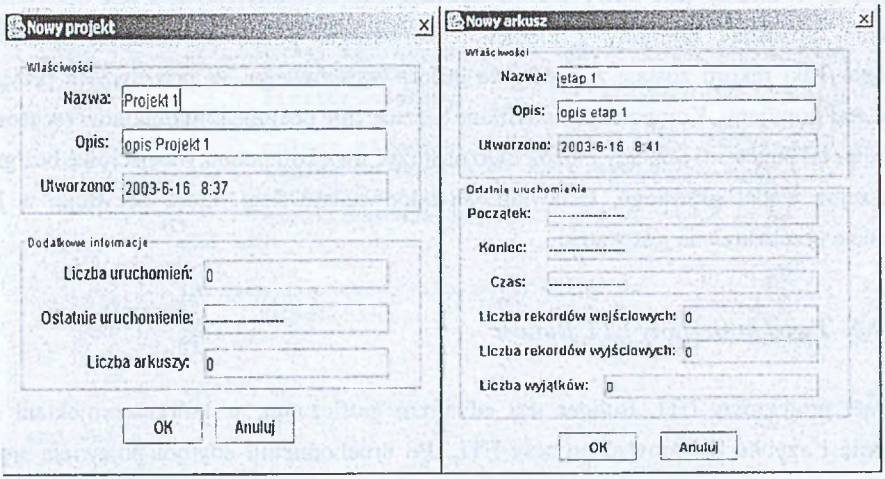
3.2.2. Panel projektowy ETL Builder

Panel projektowy ETL Builder jest edytorem graficznym, w którym projektant może wygodnie i szybko definiować procesy ETL. Po uruchomieniu edytora pojawiają się dwa okna (rys. 3). Pierwsze z nich zatytułowane „ETL" zawiera listę bieżących projektów oraz projektowych arkuszy ETL, a drugie okno stanowi paleta dostępnych komponentów.



Rys. 3. Okno z listą bieżących projektów i okno palety dostępnych komponentów
Fig. 3. Window with list of current projects & available components window

„Projekt ETL" grupuje arkusze projektowe, na których mamy kolejne etapy procesu ekstrakcji danych. Pojedynczy projekt ETL charakteryzuje nazwa, opis, data i godzina utworzenia, dodatkowe informacje oraz cztery podstawowe operacje (rys. 4).



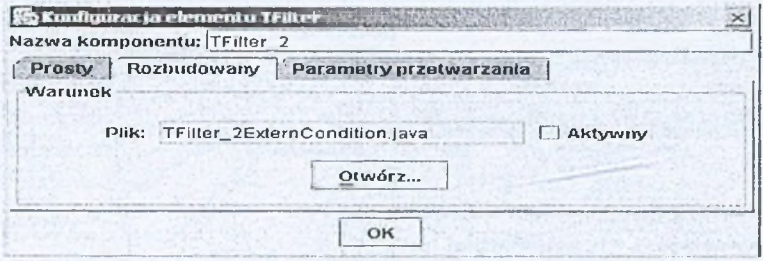
Rys. 4. Cechy pojedynczego projektu i pojedynczego arkusza projektowego
Fig. 4. Features of single project & single project sheet

Każdy projekt składa się z dowolnej liczby arkuszy projektowych (rys.4). Każdy arkusz projektowy posiada swój obszar roboczy, który opisują: nazwa, opis, data i godzina utworzenia, data i godzina początku ostatniego uruchomienia, data i godzina końca ostatniego uruchomienia, czas działania, liczba rekordów odczytanych z plików danych, liczba rekordów zapisanych do plików danych, liczba wyjątków, które wystąpiły w czasie przetwarzania arkusza.

Zarówno komponenty, jak i edytor „ETL Builder" wymagają środowiska Java™ 2 SDK Standard Edition w wersji 1.4.0 lub wyższej. Komponenty JavaBeans udostępniane są w postaci archiwum o nazwie *„etl_components.jar"*, które należy odpowiednio przekazać do programu – katalog *jars*. Natomiast program „ETL Builder" nie wymaga żadnej instalacji. Jest to po prostu zbiór plików typu class, z których główny to plik *etl_main.class*. W celu uruchomienia programu należy wydać polecenie: *java etl_main.class*.

3.2.3. Konfigurowanie komponentów TFilter i TJoin

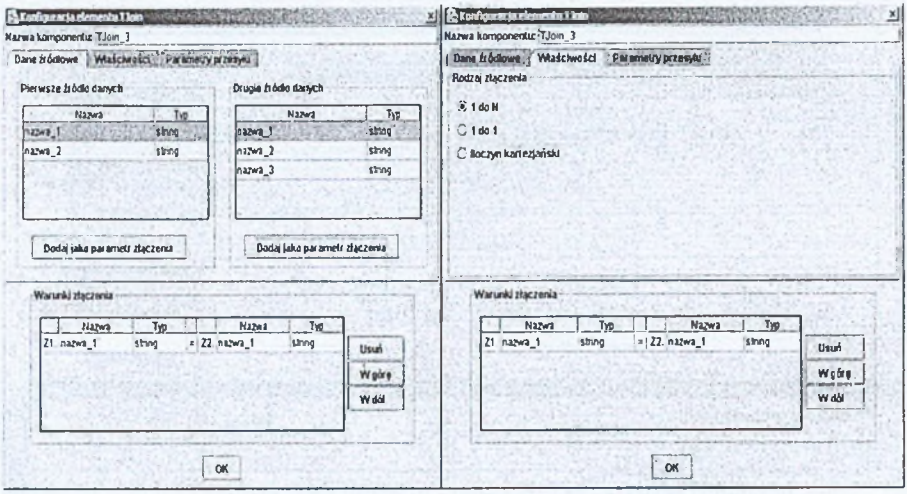
Komponent TFilter cechuje się możliwością zwielokrotnienia wątków wykonujących filtrowanie. Służą do tego zakładka „parametry przesyłu" i pole „liczba wątków". Domyślnie komponent używa jednego wątku filtrującego, ale można tę liczbę dowolnie modyfikować. Należy jednak uwzględnić fakt, że w przypadku maszyny jednoprocesorowej nie zawsze spowoduje to szybsze przetwarzanie danych.



Rys. 5. Filtr rozbudowany
Fig. 5. Extended filter

Rys. 6. TFilter – treść metody „doOperation" wykonującej filtrowanie rekordów
Fig. 6. TFilter – content of „doOperation" method performing record filtering

Zadaniem komponentu TJoin jest realizacja złączenia rekordów z dwóch źródeł, według kryteriów podanych przez użytkownika. Możliwe są 3 typy łączenia: a) 1 do N, b) 1 do 1, oraz c) iloczyn kartezjański. Konfiguracja TJoin polega na wskazaniu kolumn, według których ma się odbyć łączenie oraz na określeniu typu wykonywanej operacji. Rysunek 7 prezentuje panel służący do konfiguracji elementu oraz wybór typu operacji łączenia. Aby złączenie mogło być zrealizowane, muszą się zgadzać typy kolumn łączonych. Dzięki posortowaniu elementów można łatwo i szybko wyszukiwać rekordy przyłączane. Później następuje właściwe łączenie, a uzyskany rekord zostaje zapisany na wyjście. Ze względu na taki algorytm działania komponentu ważne jest, aby jako drugie źródło danych (przyłączane) wskazać pojęcie reprezentujące mniejszą ilość danych niż pierwsze – w ten sposób można ograniczyć zajętość pamięci w maszynie wirtualnej Javy i przyspieszyć działanie całego procesu ekstrakcji danych.

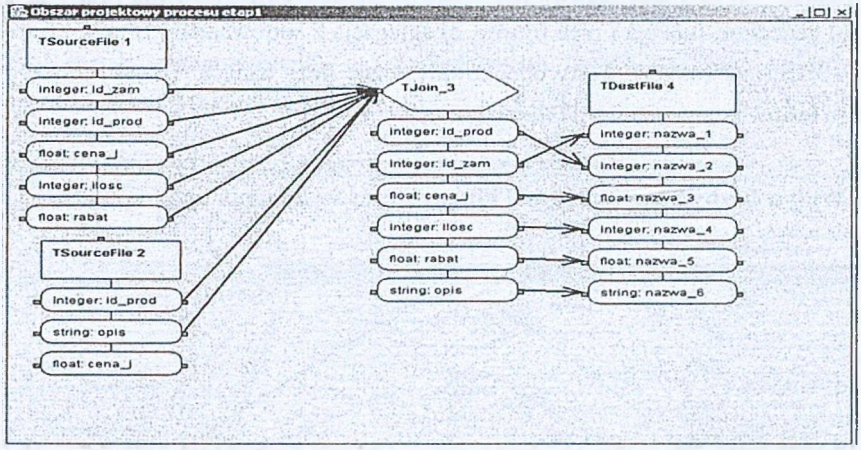


Rys. 7. TJoin - definiowanie kolumn i rodzaj złączenia
Fig. 7. TJoin – defining columns and joins types

3.3. Projektowanie procesu ETL

Projektowanie procesu ETL za pomocą edytora ETL Builder polega na projektowaniu i zarządzaniu arkuszami projektowymi. Podstawowe operacje dostępne na pojedynczym arkuszu to: utworzenie nowego, usunięcie istniejącego, przesunięcie w górę w hierarchii (pozycja w hierarchii decyduje o kolejności uruchamiania arkuszy przy uruchomieniu całego projektu), przesunięcie w dół w hierarchii, zapis struktury do pliku, odczyt struktury z pliku, uruchomienie wybranego arkusza, podgląd właściwości. Struktura projektów oraz arkuszy jest zapisywana w czasie zamykania programu do pliku konfiguracyjnego. Podczas ponownego uruchomienia jest ona odtwarzana i prezentowana na ekranie.

W momencie tworzenia nowego arkusza ETL na ekranie pojawia się okno będące obszarem projektowym dla tego procesu. W oknie tym można umieszczać dowolne elementy pobrane z palety, dowolnie je przemieszczać, konfigurować oraz tworzyć powiązania pomiędzy nimi. Przykładowy schemat ekstrakcji został zaprezentowany na rys. 8. Mamy tu dwa źródła danych: TSourceFile1 (integer: id_zam, integer: id_prod, float: cena_j, integer: ilosc, float: rabat), TSourceFile2 (integer: id_prod, string: opis, float: cena_j). Są one łączone na podstawie parametru id_prod, z pominięciem argumentu cena_j z pojęcia TSourceFile2. Otrzymany wynik złączenia będzie zapisany do pliku wynikowego (symbolizowanego przez element TDestFile4) z zachowaniem określonej kolejności atrybutów.



Rys. 8. Przykładowy schemat ekstrakcji danych
Fig. 8. Example schema of data extraction

Gdy uruchomiony zostaje schemat, pojawia się okno skojarzone z wybranym procesem. W oknie są prezentowane informacje o komponentach, o przebiegu budowy modelu i rezultatach testowania oraz komunikaty o uruchomieniu wątków i ich zatrzymaniu.

4. Ocena wpływu komponentów ETL na czas przetwarzania

Podstawowym zadaniem prowadzonych testów było określenie wpływu różnych komponentów na czas realizowania procesu ETL. Do testów użyto następujących plików tekstowych (tab. 1) oraz środowiska PC/Duron 600/256 MB RAM/Windows XP.

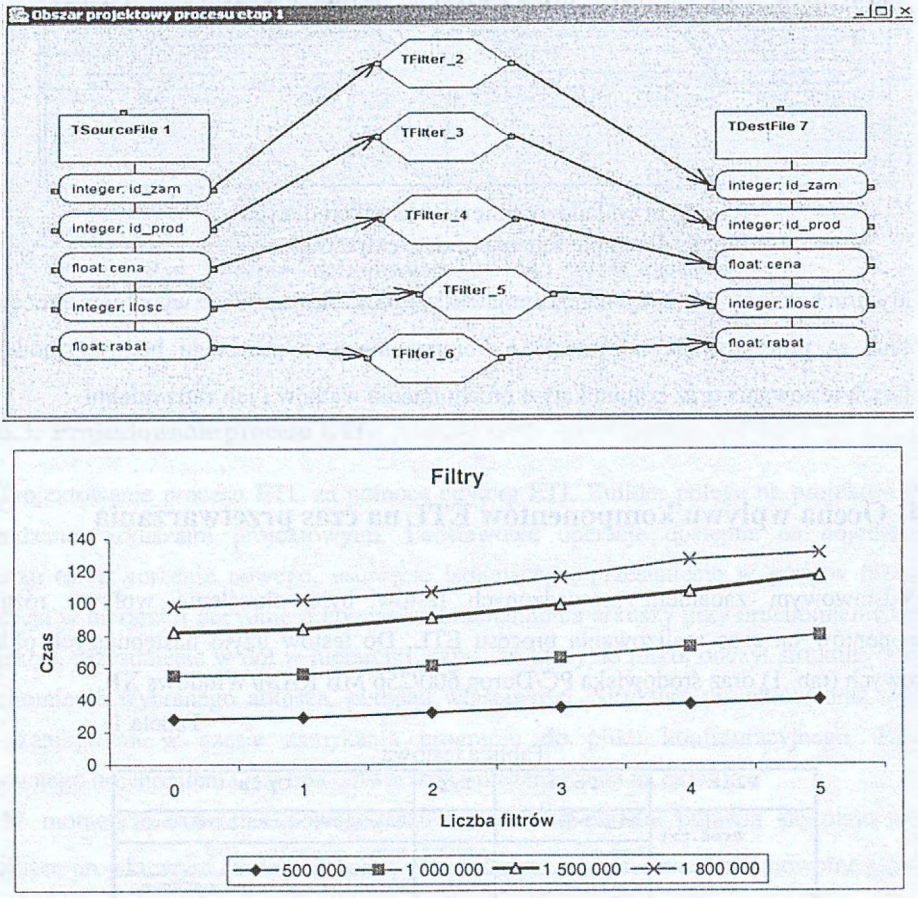
Tabela 1

Tablica testowa

Plik	Pole	Typ	Opis
Prod.txt	Id_produkt	INT	Unikatowy numer produktu
	Nazwa	STRING	Nazwa produktu
	Cena_jedn	FLOAT	Aktualna cena jednostkowa
Opis_zam.txt	Id_zamowienie	INT	Numer zamówienia
	Id_produkt	INT	Zakupiony produkt
	Cena_jedn	FLOAT	Cena jednostkowa kupna
	Ilosc	INT	Ilość zakupionych jednostek
	Rabat	FLOAT	Przysługujący rabat
Zamowienia.txt	Id_zamowienie	INT	Unikatowy numer zamówienia
	Id_klient	INT	Identyfikator kupującego
	Data	STRING	Data zamówienia

Wykonano wiele testów z różnymi konfiguracjami komponentów, tj.: a) funkcja i pięć filtrów, b) generator, funkcja i pięć filtrów, c) agregacja z sortowaniem, filtrem i funkcją, d) TUnion +2*filtr +2*funkcja. Testy obejmowały różne liczy danych. Bliżej przyjrzyjmy się rezultatom testów komponentów TFilter i TJoin.

4.1. Wpływ liczby komponentów TFilter na czas wykonania aplikacji ETL

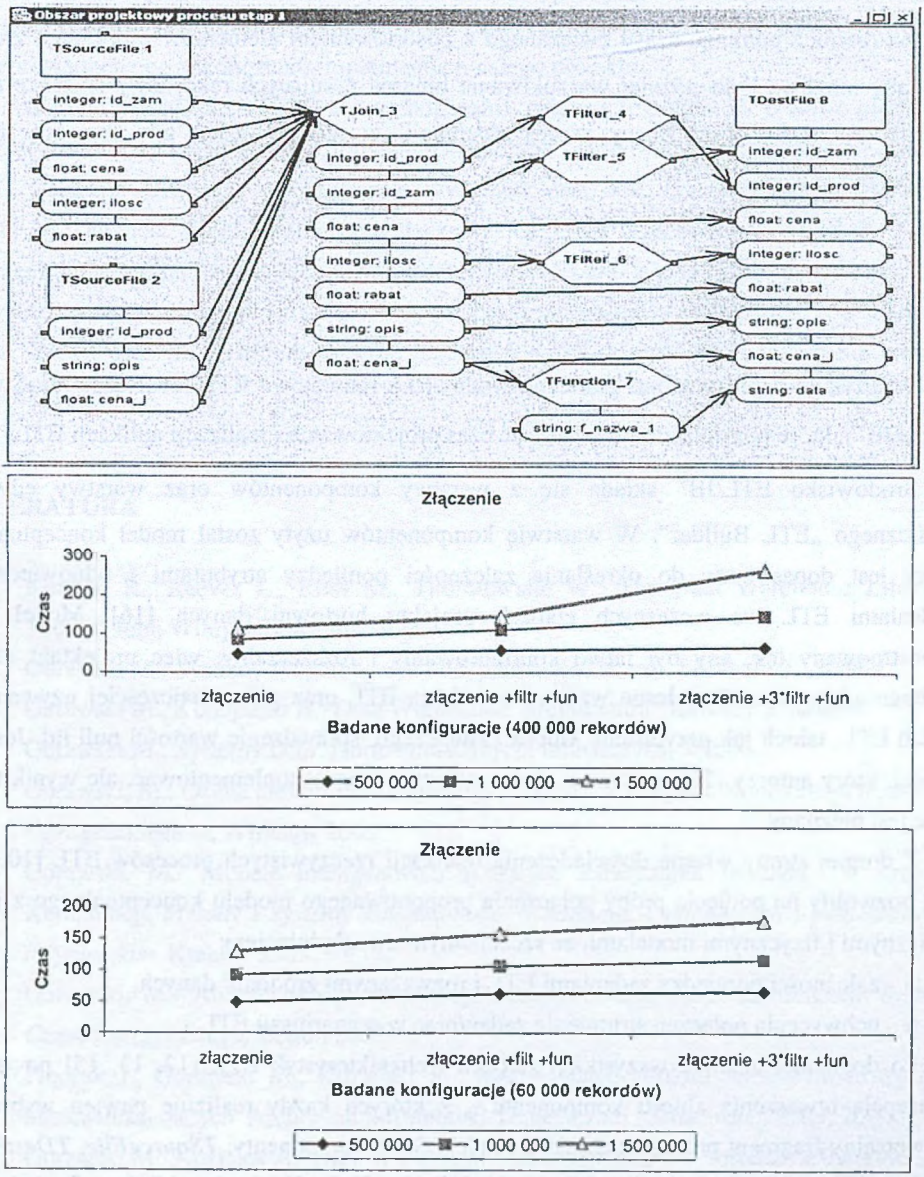


Rys. 9. Schemat i wykres dla aplikacji ETL/5 TFilter
Fig. 9. Schema and figure for ETL/5 TFilter application

Każdy filtr realizuje prosty warunek logiczny. Można zauważyć, że zwiększenie liczby filtrów powoduje wydłużenie czasu realizacji aplikacji ETL, przy czym narzut wprowadzany przez nowy komponent nie jest duży (rys. 9). Potwierdza się zależność, że w przypadku dużej liczby danych pogarszają się parametry czasowe procesu ekstrakcji.

4.2. Wpływ komponentu TJoin na czas wykonania aplikacji ETL

Testy zostały przeprowadzone z uwzględnieniem dwóch wielkości zbiorów przyłączanych: 60 000 i 400 000 rekordów (rys. 10).



Rys. 10. Złączenie Opis_zam.txt z Prod.txt wg id_prod (z Zamowienia.txt wg id_zam)
Fig. 10. Join Opis_zam.txt with Prod.txt by id_prod (from Zamowienia.txt by id_zam)

Złączenie okazało się kosztowną operacją, szczególnie w przypadku przyłączania dużej liczby danych. Powodem jest fakt wczytania wszystkich danych z drugiego źródła przed właściwym łączeniem pojedynczych rekordów. Ulega zwiększeniu zapotrzebowanie na pamięć przez maszynę wirtualną Javy, co powoduje spowolnienie całego procesu ETL. Nie można również pominąć czasu związanego z posortowaniem elementów z drugiego źródła tak, aby możliwe było później wyszukiwanie binarne pasujących rekordów. To także daje dodatkowy narzut wydłużający proces ekstrakcji, w szczególności, gdy dane są silnie nieuporządkowane.

5. Podsumowanie

W artykule przedstawiono graficzne środowisko rozwojowe ETL/JavaBeans/JavaSwing (ETL/JB^S), które pozwala zminimalizować czas projektowania i realizacji aplikacji ETL.

Środowisko ETL/JB^S składa się z warstwy komponentów oraz warstwy edytora graficznego „ETL Builder". W warstwie komponentów użyty został model konceptualny, który jest dopasowany do określania zależności pomiędzy atrybutami i odpowiednimi zadaniami ETL we wczesnych etapach projektu hurtowni danych [16]. Model jest skonstruowany tak, aby był łatwo konfigurowalny i rozszerzalny, więc projektant może wzbogacać go o swoje własne wzorcowe zadania ETL oraz paletę najczęściej używanych zadań ETL, takich jak przypisanie klucza zastępczego, sprawdzenie wartości null itd. Jest to model, który autorzy [16] zgodnie z zapowiedziami chcą zaimplementować, ale wynik tych prac jest nieznany.

Z drugiej strony własne doświadczenia realizacji rzeczywistych procesów ETL [10, 11, 14] pozwoliły na podjęcie próby połączenia proponowanego modelu konceptualnego z jego logicznymi i fizycznymi modelami, ze szczególnym uwzględnieniem:

- zależności pomiędzy zadaniami ETL i rozważanymi źródłami danych,
- uchwycenia połączeń strumienia zadań/prac w scenariuszu ETL .

Po dogłębnej analizie wszystkich założeń i charakterystyk ETL [12, 13, 15] powstała koncepcja utworzenia zbioru komponentów, z których każdy realizuje pewien wybrany, uniwersalny fragment procesu ekstrakcji danych. Są to komponenty: *TSourceFile, TDestFile, TFilter, TFunction, TAggregate, TGenerator, TSort, TJoin, Tunion* oraz *TNote*. Dodatkowo pojawiła się również konieczność zaimplementowania autonomicznej aplikacji umożliwiającej swobodne i wygodne definiowanie oraz uruchamianie wybranego procesu. Warstwa "ETL Builder" pozwala wizualizować zarządzanie projektami i arkuszami

projektowymi poprzez przyjazny dla użytkownika interfejs. Zgodnie z najnowszymi trendami związanymi z tworzeniem oprogramowania zarówno komponenty, jak i aplikacja wspierają przetwarzanie równoległe za pomocą wątków. Przeprowadzono wiele testów budowania aplikacji ETL za pomocą środowiska ETL/JB^S.

Ostatecznym sprawdzianem poprawności oraz wydajności stały się całościowe testy przeprowadzone po zakończeniu implementacji całego projektu.

Naturalna wydaje się możliwość wzbogacania palety komponentów o nowe elementy. Każdy z nowo wprowadzonych komponentów mógłby realizować inne, specyficzne zadanie ETL. Zalecana jest także optymalizacja wydajnościowa ww. komponentów. Przykładem może być element Tjoin, realizujący złączenia, na które rezerwowany jest duży obszar pamięci dla wczytania wszystkich rekordów z jednego źródła. Warto byłoby zastosować pliki zewnętrzne do przechowywania rekordów przyłączanych wtedy, gdy ich liczba jest bardzo duża. Innym ważnym aspektem rozwoju tego środowiska są problemy odtwarzania procesu ETL [19].

LITERATURA

1. Kimball R., Reeves L., Ross M., Thornthwaite W.: The Data Warehouse Lifecycle Toolkit. John Wiley & Sons, Inc 1998.
2. Gorawski M.: Systemy Wspomagania Podejmowania Decyzji. Software 2. ,5/1999
3. Gorawski M., Konopacki A.: Data Warehouse: Architektura. Software 2. ,6/1999.
4. Gorawski M., Systemy DSS: Hurtownia Danych, Informatyka, 3/2000.
5. Gorawski, M.: Ocena efektywności architektur OLAP. V Krajowa Konferencja *Inżynieria Oprogramowania,* Wrocław 2003.
6. Gorawski, M.: Modele inteligentnych systemów zarządzania strategią. IV Krajowa Konferencja *Metody i systemy komputerowe w badaniach naukowych i projektowaniu inżynierskim,* Kraków 2003.
7. Gorawski, M.: Architektura przemysłowych hurtowni danych. X Konferencja *Systemy Czasu Rzeczywistego,* Ustroń 2003.
8. Frączek J., Gorawski M., Kozielski S..: Modelowanie struktur wielowymiarowych w hurtowniach danych. Archiwum Informatyki Teoretycznej i Stosowanej PAN, 2000.
9. Gorawski M., Koziatek A.: Data Warehouse: Ekstrakcja danych. Software 2.09 /1999.
10. Gorawski M., Koziatek A.: Systemy DSS: Projekt ekstrakcji danych. Informatyka 7-8/2000.

11. Gorawski M., Grzywacz M.: Systemy DSS: Wybrane zagadnienia implementacji procesu ekstrakcji danych przy użyciu C++. Informatyka 9/2000.
12. Gorawski M., Świtoński A.: Systemy DSS: Uniwersalne narzędzia ETL. Informatyka 10/2000.
13. Gorawski, M.: Charakterystyka procesu ekstrakcji danych. Studia Informatica, ZN Pol. Śl. Vol.24, No 4(56), Gliwice 2003.
14. Gorawski M., Piekarek M.: Rozwojowe środowisko ETL/JavaBeans. Studia Informatica, ZN Pol. Śl. Vol.24, No 4(56), Gliwice 2003.
15. Gorawski, M.: Modelowanie procesu ekstrakcji danych. IV KK *Metody i systemy komputerowe w badaniach naukowych i projektowaniu inżynierskim*, Kraków 2003.
16. Vassiliadis, P., Simitsis, A., Skiadopoulos S., „Conceptual Modeling for ETL Processes". ACM International Workshop on Data Warehousing and OLAP, DOLAP 2002.
17. Java 2 Platform, Standard Edition, v 1.4.0, API Specification, Sun Microsystems, 2002
18. Creating a GUI with JFC/Swing.
19. Gorawski M., Wocław A.: Implementacja algorytmu odtwarzania Design-Resume w technologii JavaBeans. Studia Informatica, ZN Pol. Śl. Vol. 24, No 1(52), Gliwice 2003.

Recenzent Dr inż. Arkadiusz Sochan

Wpłynęło do Redakcji 19 sierpnia 2003 r.

Abstract

ETL process (data extraction) in practice can take even up to 70% of the overall time destined for data warehouse realization. This paper presents ETL/JavaBeans/JavaSwing (ETL/JB^S) graphic development environment that makes process of building new ETL application much easier by use of friendly user interface. ETL/JB^S environment consists of components layer and „ETL Builder" graphic editor layer. Based on our practical experience, we have decided to create components that are responsible for extraction, transformation and data loading operations. Those are *TSourceFile, TDestFile, TFilter, TFunction, TAggregate, TGenerator, TSort, TJoin, TUnion* and *Tnote* components. "ETL Builder" layer enables visualization of projects and projects sheets management. Many ETL applications building tests and analysis of their efficiency were performed using ETL/JB^S environment.

Adresy

Marcin GORAWSKI: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-100 Gliwice, Polska, M.Gorawski@zti.iinf.polsl.gliwice.pl.

Przemysław SIÓDEMAK: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-100 Gliwice, Polska, duzysiod@zeus.polsl.gliwice.pl