CDM RuleFrame nowa jakość tworzenia aplikacji trójwarstwowych

Tomasz Kazimierski
Oracle Polska
e–mail: Tomasz.Kazimierski@oracle.com

Abstrakt. Custom Development Method (CDM) jest sprawdzoną, strukturalną metodą rozwoju aplikacji używających technologii Oracle. Metoda rozwija się wraz z firmą od wielu lat. Obecnie, tak jak większość produktów Oracle, znalazła zastosowanie w Internecie jako CDMi.
Tematem tego referatu jest tylko fragment CDMi: CDM RuleFrame - jako interesujący pomysł implementacji reguł biznesowych w architekturze trójwarstwowej.

Wymagania dla narzędzia do implementacji reguł biznesowych

W każdej aplikacji bazodanowej występują reguły biznesowe. Model relacyjny wspiera prosto implementacje niektórych z nich (np. „linia zamówienia musi wiązać się z istniejącym zamówieniem") ale w większości proste więzy bazodanowe nie wystarczają (np. „zamówienie musi mieć co najmniej jedną linię", albo „nie można dodawać linii do zamówienia w stanie 'Zamknięte'").

Wsparcie implementacji reguł bazodanowych zostało wprowadzone w Generatorze Server w Oracle Designerze. Można tam przy pomocy tak zwanego TAPI (API Tabeli) implementować automatycznie takie reguły jak prosta denormalizacja, automatyczne numerowanie z sekwencji czy zapamiętywanie informacji o czasie i użytkowniku zmieniającym rekord. Jednak wciąż istnieją reguły przy których generator nam nie pomoże.

Często podczas implementacji reguł pojawiają się problemy które powodują, że projektanci mniej lub bardziej świadomie rezygnują z umieszczenia ich w bazie danych na rzecz kodu aplikacji klienta. Problemami tymi są na przykład: (1) niemożność łatwego dostępu do wierszy tabeli z wyzwalacza bazodanowego opartego o nią (problem 'mutating table'), problemem jest także po prostu (2) złożoność kodu w bazie w porównaniu z podobnym kodem po stronie klienta, albo (3) obsługa wyjątków zgłaszanych przez taki kod. Jeśli reguły nie są w bazie muszą być implementowane w każdej aplikacji która z danych korzysta. Nie możemy być wówczas pewni, że za każdym razem reguły działają identycznie, każda implementacja może wnosić swoje własne błędy.

Kolejnym aspektem jest utrzymanie kodu obsługi reguł. Często sposób implementacji jest pozostawiony poszczególnym programistom, wówczas zmiana jednej reguły może powodować konieczność zmiany kodu w wielu miejscach i w różny sposób.

Sposobem rozwiązania powyższych problemów ma być RuleFrame. Podczas jego tworzenia zostały przyjęte następujące założenia wynikające z kwestii związanych z samą implementacją reguł i kwestii związanych z ich utrzymaniem.

Uwaga o terminologii: Przyjmując za językiem angielskim zwykle mówi się o wymuszaniu reguł biznesowych. Jednak, ponieważ reguły mogą być o wiele bardziej skomplikowane niż proste sprawdzenie (np. regułą biznesową jest aktualizacja sumy całkowitej faktury przy dodaniu do niej pozycji) słowo „wymuszanie" wydaje mi się niezręczne. Dlatego, dalej będziemy używać sformułowania wykonanie reguł mając na myśli wykonanie kodu implementującego określoną regułę.

Implementacja reguł

RuleFrame powinien efektywnie wspierać powstawanie implementacji reguł. Dlatego przyjęto następujące założenia:

1. Szybkie tworzenie kodu – najlepiej automatyczna generacja.
2. Łatwe dodawanie reguł poprzez określenie zdarzenia, danych oraz kodu.
3. Użycie informacji zebranych na etapie analizy i zapisanych w repozytorium.
4. Użycie standardowych mechanizmów upraszczających złożoność kodu. RuleFrame opiera się na funkcjonalności bazy danych Oracle 8 ale koncepcje tu opisywane mogą zostać także użyte, po pewnych zmianach, dla bazy Oracle 7.
5. Implementacja reguł zawarta jest całkowicie w bazie danych, jest niezależna od kodu aplikacji klienta. Kod klienta, bez względu na to czy jest to kod Web Forms, kartrydż PL/SQL czy JAVA, jest odpowiedzialny wyłącznie ze obsługę błędów z implementacji reguł. Założenie to oznacza wydzielenie warstwy implementacji reguł.
6. Mechanizm powinien zapewniać możliwość wykonania reguł na poziomie transakcji. Gwarantujemy spełnienie reguł na końcu transakcji ale podczas jej przeprowadzania reguły nie koniecznie muszą być spełnione. Przykładem jest reguła „każde zamówienie ma zawsze co najmniej jedną linię", ponieważ nie da się zrobić zamówienia bez jej chwilowego naruszenia. Implikuje to konieczność stworzenia mechanizmu transakcji rozszerzającego standardowe działanie bazy danych.
7. Sprawdzanie reguł można zablokować. Założenie to wynika z konieczności wykonania działań wsadowych, kiedy sprawdzanie reguł prowadziło by do problemów wydajnościowych.

Utrzymanie reguł

Reguły łatwo wspierać jeśli:

8. Kod implementacji ma podobną strukturę (tu znów założenie o generacji),
9. Struktura implementacji powinna być możliwie odporna na zmiany reguły lub struktur danych. Założenie to prowadzi do zastosowania technik obiektowych w implementacji.
10. Powinna istnieć łatwa możliwość oczyszczania z błędów kodu implementacji.

CDM RuleFrame

W części tej opisywana jest architektura rozwiązania i poszczególne jego elementy.

Struktura

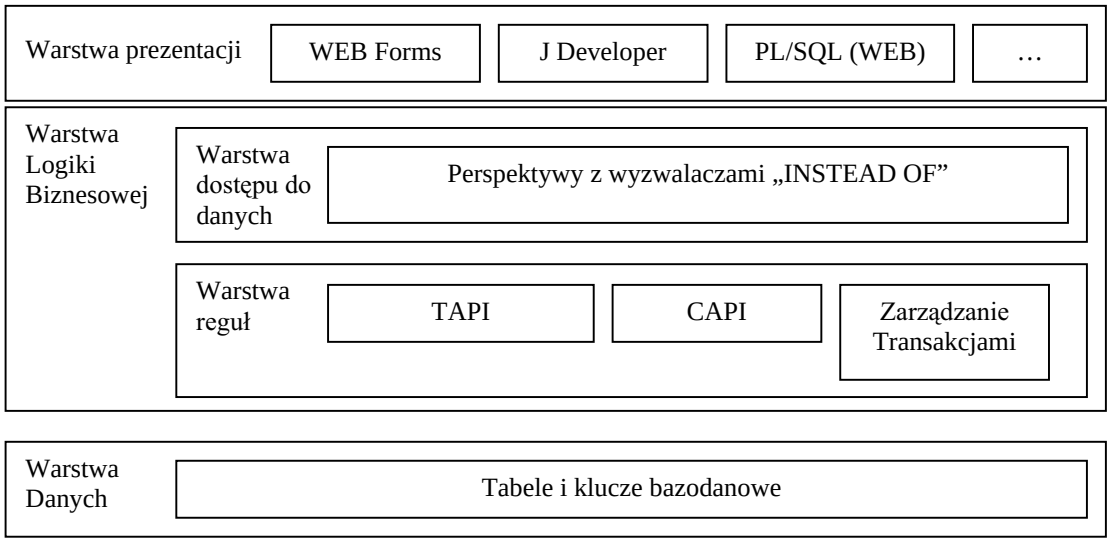
Zgodnie z założeniami opisanym w poprzedniej części RuleFrame stanowi dodatkową warstwę logiki aplikacji pomiędzy warstwą danych (baza danych utrzymywana przez Oracle 8) i warstwą prezentacji (kod aplikacji pokazujący dane i przekazujący działania wykonywane przez użytkownika).

Warstwę RuleFrame nazywamy warstwą logiki biznesowej.

Warstwa logiki biznesowej ma dalej dwie podwarstwy:

- Warstwę dostępu do danych,
- Warstwę reguł.

Pełną strukturę przedstawia poniższy rysunek:



Ważnym komponentem, nie wchodzącym w skład RuleFrame jest kompleksowy mechanizm obsługi błędów. Mechanizm ten został wprowadzony w pakiecie Headstart.

Dalej opisywane są poszczególne komponenty warstwy logiki biznesowej.

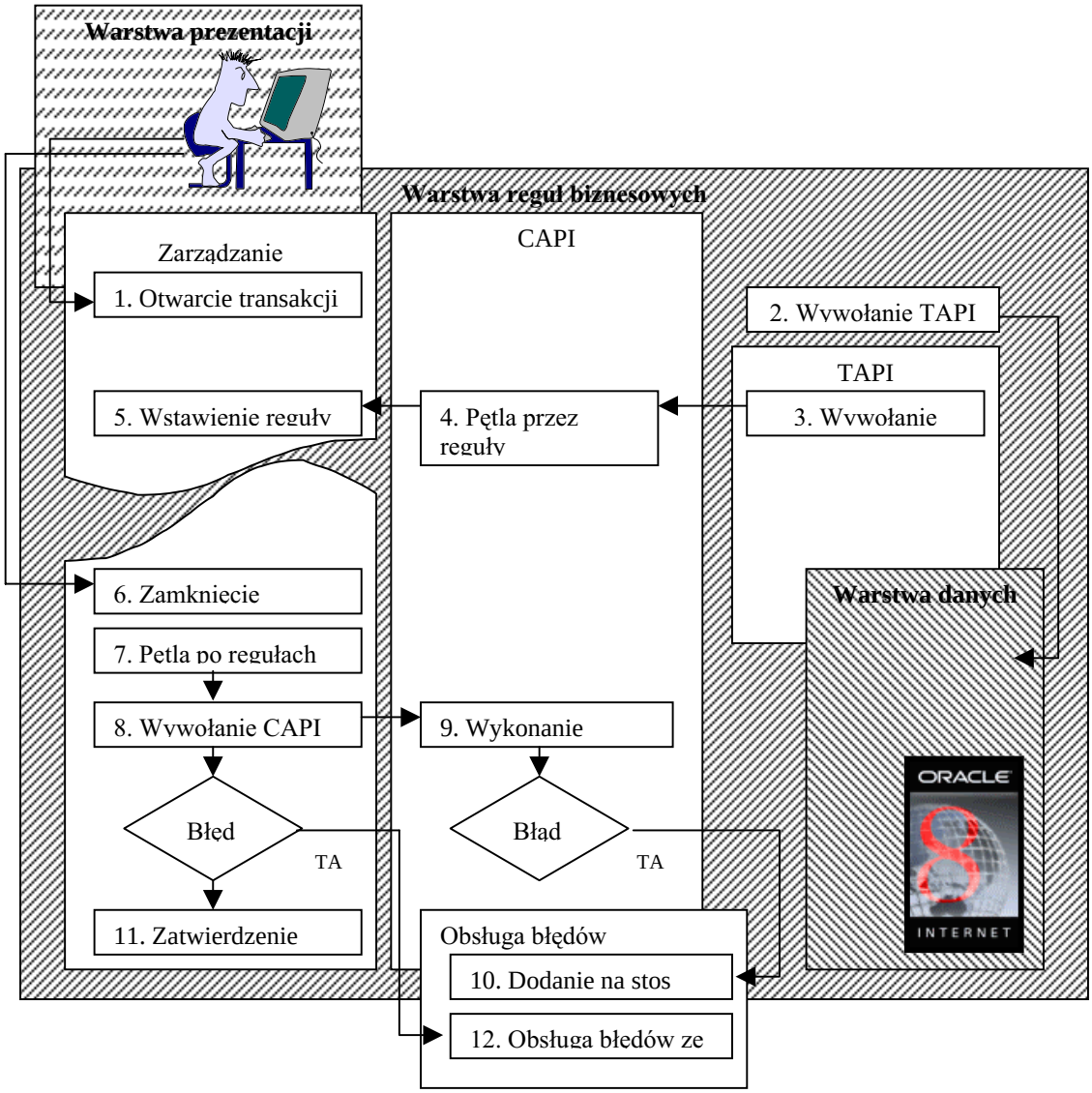
Warstwa reguł

Warstwa reguł składa się z trzech części: TAPI, CAPI i zarządzania transakcjami. Poszczególne części będą szczegółowo opisane dalej poniżej tylko krótki opis który powinien pomóc zrozumieć diagram.

TAPI jest to kod pakietów i wyzwalaczy generowanych przez Generator Serwera Oracle Designera. RuleFrame dodaje w repozytorium dodatkowy kod zapewniający współpracę TAPI z pozostałymi komponentami.

CAPI (Custom API) pakiety PL/SQL implementujące reguły oraz dodatkowe usługi. Mogą być częściowo generowane przez narzędzia RuleFrame.

Moduł zarządzania transakcjami jest gotowym pakietem PL/SQL zawartym w RuleFrame.



Powyższy diagram przedstawia w sposób uproszczony jak poszczególne części współpracują ze sobą (diagram nie zawiera warstwy dostępu).

1. Aplikacja użytkownika poprzez warstwę dostępu rozpoczyna transakcję.
2. TAPI obsługuje DML wykonany przez aplikację.
3. TAPI wywołuje procedurę CAPI
4. Procedura CAPI sprawdza czy istnieją reguły bazodanowe, które wymagają wykonania podczas tego wywołania.
5. Jeśli tak to odpowiednia informacja umieszczana jest na stosie transakcji. Dalej w ramach tej transakcji mogą nastąpić kolejne wywołania TAPI aż do…
6. …chwili kiedy aplikacja użytkownika zgłasza koniec transakcji.
7. Mechanizm transakcyjny przegląda stos sprawdzając czy zostały tam umieszczone wywołania CAPI dla obsługi reguł.
8. Dla każdej pozycji stosu wywołuje odpowiednią procedurę CAPI.
9. Procedura CAPI wykonuje akcje wynikające z reguły. Jeśli procedura natrafia na błąd informacja o nim jest przekazywana do mechanizmu obsługi błędów.

10. Mechanizm obsługi błędów umieszcza informacje na stosie błędów. W zależności od rodzaju błędu jest zwracany albo nie jest zwracany wyjątek do obsługi transakcji.
11. Jeśli obsługa transakcji otrzymała błąd albo nie ma więcej informacji na stosie transakcji sprawdzane jest czy istnieją błędy na stosie błędów.
12. Jeśli tak wywoływany jest mechanizm obsługi błędów.
13. Jeśli nie transakcja kończy się zatwierdzeniem.

TAPI

Warstwa składa się z kodu generowanego przez Generator Serwera Oracle Designera. W jego skład wchodzi kod realizujący podstawowe akcje DML (UPD, DEL, INS), odczytu danych (SEL) oraz blokowania (LCK).

Generator Serwera może umieścić w kodzie TAPI kod zapewniający wykonanie następujących dodatkowych działań:

1. Wypełnianie kolumn wartościami z sekwencji bazodanowych. Kod pobiera wartość z sekwencji związanej z pole tabeli podczas wstawiania rekordu.
2. Generowanie wartości domyślnych dla kolumn na podstawie informacji z repozytorium.
3. Utrzymywanie informacji o dacie i użytkowniku wstawiającym dane oraz dacie i użytkowniku modyfikującym dane.
4. Konwersja danych do dużych liter.
5. Utrzymanie tabel żurnalowych. Tabele zawierają informacje o zmianach danych z tabeli źródłowej. Podczas modyfikacji i wstawiania rekordu poprzedni stan wstawiany jest do tabeli żurnalowej.
6. Utrzymywanie denormalizacji: (1) prostej – przepisanie danych z rekordu nadrzędnego do podrzędnego (np. nazwa departamentu przepisywana do rekordu pracownika departamentu obok klucza obcego na numerze departamentu). (2)Wyliczeniowej kiedy wartość kolumny rekordu nadrzędnego uzyskiwana jest z działania na rekordach podrzędnych (np. utrzymanie sumy całkowitej w fakturze na podstawie wartości poszczególnych linii faktury).
7. Sprawdzanie dopuszczalnych wartości pól na podstawie dziedzin zdefiniowanych w repozytorium.
8. Wykonywanie dodatkowych akcji związanych z kluczami obcymi. Generator umożliwia aby w chwili zmiany klucza głównego została wykonana jedna z poniższych akcji (1) aktualizacja pól kluczy obcych na nową wartość, (2) ustawienie pól kluczy obcych na wartość pustą albo (3) ustawienie pól kluczy obcych na wcześniej określoną wartość domyślną. Podobnie podczas usuwania rekordu nadrzędnego można wymusić (1) usunięcie rekordów podrzędnych albo ustawić wartość kluczy obcych w tych rekordach na wartość (2) pustą albo wartość (3) domyślną.
9. Wymuszanie łuków. Łuk jest jedną z dróg implementacji podtypów. Na podstawie nadtypu (encji zewnętrznej) tworzona jest tabela podrzędna T0 dla tabel utworzonych z podtypów (tabele T1, T2…). Klucze obce pomiędzy T0 a T<N> wykluczają się wzajemnie. Na przykład nadtypem jest osoba, jego podtypami jest osoba fizyczna i osoba prawna. Wówczas powstają tabele OSOBY, OSOBY_FIZYCZNE i OSOBY_PRAWNE. Każdy rekord z tabeli OSOBY jest związany z rekordem z tabeli OSOBY_FIZYCZNE albo OSOBY_PRAWNE nigdy z dwoma na raz.

Poza funkcjonalnością opisaną powyżej do TAPI można dodawać swój własny kod jako „User logic". Własność ta jest wykorzystana przez RuleFrame do podłączenia wywołań funkcji CAPI.

Wywołania funkcji CAPI są wpisywane do repozytorium automatycznie przez narzędzia RuleFrame.

Zarządzanie transakcjami

Zarządzanie transakcjami zostało zaimplementowane jako generyczny pakiet bazodanowy wspólny dla wszystkich CAPI zawierający usługi otwarcia, zamknięcia transakcji, zapamiętywania reguł potrzebnych podczas transakcji na stosie transakcji, wykonanie reguł odroczonych przy zmykaniu transakcji. Przed omówieniem poszczególnych usług konieczne jest wprowadzenie pojęcia poziomu wykonania reguł.

Poziom wykonania reguł

Każda modyfikacja bazy danych odbywa się w kontekście transakcji bazodanowej. Wewnątrz transakcji może nastąpić jedna bądź wiele modyfikacji, jednej bądź wielu tabel.

1. Reguły mogą zostać wykonane dla każdego modyfikowanego rekordu. Wówczas mówimy, że wykonanie reguł następuje na poziomie wiersza.
2. Inne reguły mogą wymagać wykonania po każdej modyfikacji. Mówimy wtedy, że wykonanie reguł następuje na poziomie polecenia.
3. I w końcu wykonanie reguł może odbywać się dla całej transakcji czyli na poziomie transakcji.

Istnieją reguły, które mogą być wykonywane tylko na poziomie transakcji. Przykładem jest problem zamówienia z co najmniej jedną linią (obligatoryjny klucz obcy). Ale odwrotnie, każda reguła która może być wykonana na poziomie wiersza bądź polecenia może zostać również wykonana na poziomie transakcji. Stąd podczas konstrukcji RuleFrame przyjęto zasadą użycia tylko tego poziomu dla wykonania reguł. Podejście takie ma dodatkowe zalety:

1. Wykonanie reguł zawsze tylko na jednym poziomie czyni kod prostszym i bardziej czytelnym. Do tego nie trzeba podejmować za każdym razem decyzji gdzie daną regułę zaimplementować.
2. Możliwa jest spójna obsługa błędów. Ponieważ wszystkie reguły zostaną sprawdzone dla transakcji można uniknąć sytuacji kiedy użytkownik jest informowany o każdym problemie z osobna.

Usługi mechanizmu zarządzania transakcjami.

Z założenia o wykonaniu reguł na poziomie transakcji wynika, że mechanizm musi być wyposażony w usługi zapewniające zbieranie informacji o regułach, które transakcja musi wykonać. Stąd cztery usługi mechanizmu:

- Otwieranie transakcji

Otwarcie transakcji jest konieczne dla prawidłowego działania RuleFrame.

W przypadku użycia aplikacji w narzędziach takich jak Oracle Forms, które są świadome transakcyjności przetwarzania, można wykorzystać zdefiniowane tam mechanizmy dla otwarcia i zamknięcia transakcji. Na przykładzie Forms będzie to para wyzwalaczy PRE-COMMIT i POST-FORM-COMMIT (modyfikacja klasy własności biblioteki obiektowej wprowadzi taki kod do całej aplikacji).

Przy użyciu innych narzędziach na kod klienta spada konieczność otwarcie transakcji. Jeśli jednak klient nie dokona tego jawnie przy pierwszym dostępie do danych (wykonaniu TAPI) transakcja zostanie automatycznie otwarta.

- Zbieranie danych o regułach

Usługa ta jest automatycznie wywoływana przez kod CAPI. Służy do zapisania informacji o konieczności sprawdzenia reguły.

- Zamykanie transakcji

Dla wszystkich pozycji zapisanych na stosie reguł wywoływane są odpowiednie funkcje CAPI implementujące reguły. Jeśli pojawiły się błędy zostaną one zwrócone jako wyjątek do programu macierzystego. Jeśli nie wystąpiły błędy sterowanie wraca do programu macierzystego, który może teraz zatwierdzić zmiany.

- Anulowanie transakcji

Czyści stos reguł i wycofuje zmiany.

Przeciwdziałanie przed porzuceniem transakcji

Przy pomocy odroczonych więzów bazodanowych został zaimplementowany mechanizm zapewniający, że każda otwarta transakcja musi zostać zamknięta lub anulowana. Mechanizm ten działa wyłącznie w bazie danych i jest absolutnie niezależny od kodu aplikacji użytkownika. Dzięki niemu baza danych może przeciwdziałać błędom aplikacji polegających na porzuceniu transakcji mechanizmu zarządzania transakcjami albo niekontrolowanemu zatwierdzeniu modyfikacji bez wywołania usługi zamknięcia transakcji.

Ten sam mechanizm zapewnia anulowanie transakcji przy wycofaniu zmian.

CAPI

CAPI jest tworzone jako pakiety PL/SQL dla każdej tabeli bazodanowej znajdującej się pod kontrolą RuleFrame.

Proces tworzenia CAPI jest oparty o dokumentację reguł biznesowych dokonaną w repozytorium w fazie analizy. Reguły są wówczas zapisywane jako funkcje biznesowe.

Składowe CAPI

Każda reguła biznesowa jest implementowana jako trzy podprogramy w pakiecie CAPI:

- Funkcja sprawdzająca czy reguła musi być wykonywana

Zadaniem tej funkcji jest sprawdzenie czy mechanizm zarządzania transakcji będzie musiał w przypadku bieżącej modyfikacji wymuszać regułę. Jeśli tak odpowiednia informacja zostanie umieszczona na stosie reguł.

Przykład funkcji:

```
function need_br_ord001_tpl
  return boolean
is
begin
  return g_inserting
  or
  ( g_updating
    and
    (
      g_clr_orders_ind.delivery_date
      or
      g_clr_orders_ind.orderdate
    )
  );
end need_br_ord001_tpl;
```

- Funkcja wykonująca regułę,

Tutaj następuje faktyczne wykonanie reguły (w przykładzie jest to tylko sprawdzenie). Jeśli sprawdzanie kończy się sukcesem zwracana jest prawda.

```
function br_ord001_tpl
( p_order_date in date
,p_delivery_date in date
) return boolean
is
begin
  return p_order_date < p_delivery_date -4
end br_ord001_tpl;
```

- Procedura obsługi zewnętrznej.

Procedura jest wywoływana przez mechanizm zarządzania transakcji.

Przykład:

```
procedure br_ord001_tpl
is
begin
  if not
    br_ord001_tpl
   ( g_clr_orders_row.orderdate
    ,g_clr_orders_row.delivery_date )
  then
    qms_transaction_mgt.push_message(<message_code>,'E');
  end if;
end br_ord001_tpl;
```

Kod CAPI jest składowany w repozytorium, tam też dokumentowany jest związek kodu z funkcjami biznesowymi z etapu analizy.

Usługi obiektowe

Jednym z wymagań dla mechanizmy była odporność kodu implementującego reguły na zmiany modelu danych i samych reguł.

Odporność ta może zostać uzyskana poprzez wykorzystanie technik obiektowych przy dostępie do danych przy konstruowaniu podprogramów CAPI. Odwołania do danych powinny tu następować przy pomocy składowych odpowiedniego pakietu TAPI zamiast poprzez bezpośrednie odwołania do tabel. TAPI choć nie jest samo zaimplementowane jako obiekt staje się mechanizmem kapsułkowania i ukrywania, zatem umożliwia pisanie kodu w sposób obiektowy.

Automatyzacja tworzenia CAPI

Jedną z głównych zalet użycia TAPI/CAPI jest możliwość automatycznej generacji kodu.

Jeśli udało by się w sposób formalny udokumentować reguły w repozytorium, generator mógłby użyć tej definicji do utworzenia kodu CAPI. Przykładem języka opisującego reguły jest OCL (Object Constraint Language), jest on proponowanym rozszerzeniem UML.

Narzędzia RuleFrame są wyposażone w mechanizmy generacji kodu CAPI oraz fragmentów TAPI odpowiedzialnych za integracje z CAPI. Kod jest generowany bezpośrednio do repozytorium.

Warstwa dostępu do danych

Dostęp do danych może się odbywać na jeden z poniższych sposobów:

- Poprzez wykonywanie zdań DML z kodu aplikacji. Przy pomocy wyzwalaczy bazodanowych związanych z TAPI uruchamiany jest mechanizm RuleFrame.
- Poprzez bezpośrednie wywoływanie składowych TAPI.
- Poprzez wywołanie składowych MAPI (module API) które dalej wywołują TAPI

- Poprzez warstwę dostępu do danych VAPI. VAPI jest to zbiór perspektyw wraz z wyzwalaczami typu INSTEAD-OF ukrywający przed aplikacją użytkownika mechanizm RuleFrame.

RuleFrame nie wymusza użycie VAPI jednak jest to droga z wielu względów najwygodniejsza.

VAPI

Najprostszym przykładem VAPI jest perspektywa o strukturze identycznej z tabelą bazodanową. Aby zapewnić, że DML będzie przechodził przez procedury TAPI konieczne jest zdefiniowanie wyzwalaczy bazodanowych, które wywołają odpowiednie funkcje.

Przykład wyzwalacza INSTEAD-OF:

```
create or replace trigger hsd_v_dep_io
  instead of delete or insert or update
  on hsd_v_departments
begin
  declare
    cg$rec cg$hsd_departments.cg$row_type;
    cg$ind cg$hsd_departments.cg$ind_type;
    cg$pk cg$hsd_departments.cg$pk_type;
begin
  if inserting or updating
  then
    cg$rec.id:= :new.id;
    cg$rec.name:= :new.name;
    cg$rec.location:= :new.location;
  end if; -- inserting or updating
  if inserting
  then
    cg$ind.id:= true;
    cg$ind.name:= true;
    cg$ind.location:= true;
    cg$hsd_departments.ins(cg$rec, cg$ind);
  end if; -- inserting
  if updating
  then
    cg$ind.id:= updating('id');
    cg$ind.name:= updating('name');
    cg$ind.location:= updating('location');
    cg$hsd_departments.upd(cg$rec, cg$ind);
  end if; -- updating
  if deleting
  then
    cg$pk.id:= :old.id;
    cg$hsd_departments.del(cg$pk);
  end if; -- deleting
end;
end hsd_v_dep_io;
```

Perspektywa złożona

Prawdziwą zaleta zastosowania VAPI jest dopiero użycie perspektyw opartych o złączenie wielu tabel. Można w ten sposób budować dostęp do danych niezależnie od prawdziwej ich struktury.

Przykład perspektywy złożonej:

```
create or replace force view hsd_v_emp_primary_address
   (id
```

```
   ,name
   ,job
   ,hire_date
   ,dep_id
   ,street
   ,city
   ,zip_code
   ,country
   )
as select
   emp.id id,
   emp.name name,
   emp.job job,
   emp.hire_date hire_date,
   emp.dep_id dep_id,
   adr.street street,
   adr.city city,
   adr.zip_code zip_code,
   adr.country country
from hsd_employees emp
,hsd_addresses adr
where emp.id = adr.emp_id
and adr.seqno = 1
```

Wyzwalacze INSTEAD-OF używają TAPI obu tabel składowych.

```
create or replace trigger hsd_v_emp_primary_address_io
   instead of delete or insert or update
   on hsd_v_emp_primary_address
begin
   declare
      cg$emp_rec cg$hsd_employees.cg$row_type;
      cg$emp_ind cg$hsd_employees.cg$ind_type;
      cg$emp_pk cg$hsd_employees.cg$pk_type;
      cg$adr_rec cg$hsd_addresses.cg$row_type;
      cg$adr_ind cg$hsd_addresses.cg$ind_type;
      cg$adr_pk cg$hsd_addresses.cg$pk_type;
   begin
      if inserting or updating
      then
         cg$emp_rec.id:= :new.id;
         cg$emp_rec.name:= :new.name;
         cg$emp_rec.job:= :new.job;
         cg$emp_rec.hire_date:= :new.hire_date;
         cg$emp_rec.dep_id:= :new.dep_id;
         cg$adr_rec.emp_id:= :new.id;
         cg$adr_rec.seqno:= 1;
         cg$adr_rec.street:= :new.street;
         cg$adr_rec.city:= :new.city;
         cg$adr_rec.zip_code:= :new.zip_code;
         cg$adr_rec.country:= :new.country;
      end if; -- inserting or updating
      if inserting
      then
         cg$emp_ind.id:= true;
         cg$emp_ind.name:= true;
         cg$emp_ind.job:= true;
         cg$emp_ind.hire_date:= true;
```

```
        cg$emp_ind.dep_id:= true;
        cg$adr_ind.emp_id:= true;
        cg$adr_ind.seqno:= true;
        cg$adr_ind.street:= true;
        cg$adr_ind.city:= true;
        cg$adr_ind.zip_code:= true;
        cg$adr_ind.country:= true;
        cg$hsd_employees.ins(cg$emp_rec, cg$emp_ind);
        cg$hsd_addresses.ins(cg$adr_rec, cg$adr_ind);
      end if; -- inserting
      if updating
      then
        cg$emp_ind.id:= updating('id');
        cg$emp_ind.name:= updating('name');
        cg$emp_ind.job:= updating('job');
        cg$emp_ind.hire_date:= updating('hire_date');
        cg$emp_ind.dep_id:= updating('dep_id');
        cg$adr_ind.emp_id:=false;
        cg$adr_ind.seqno:=false;
        cg$adr_ind.street:= updating('street');
        cg$adr_ind.city:= updating('city');
        cg$adr_ind.zip_code:= updating('zip_code');
        cg$adr_ind.country:= updating('country');
        cg$hsd_employees.upd(cg$emp_rec, cg$emp_ind);
        cg$hsd_addresses.upd(cg$adr_rec, cg$adr_ind);
      end if; -- updating
      if deleting
      then
        cg$emp_pk.id:= :old.id;
        cg$adr_pk.emp_id:= :old.id;
        cg$adr_pk.seqno:= 1;
        cg$hsd_addresses.del(cg$adr_pk);
        cg$hsd_employees.del(cg$emp_pk);
      end if; -- deleting
    end;
end hsd_v_emp_primary_address_io;
```

Zalety i wady użycia VAPI jako warstwy dostępu

Użycie VAPI ma następujące zalety:

- VAPI może być używane przez wszystkie narzędzia do tworzenia aplikacji użytkownika, dzięki czemu implementacja reguł pozostaje w jednym miejscu.
- Istniejące aplikacje nie wymagają żadnych modyfikacji dla integracji z RuleFrame jeśli perspektywy zostaną utworzone na podstawie tabel (bez złączeń).
- Użycie VAPI czyni warstwę reguł całkowicie przezroczystą dla kodu aplikacji klienta. Programiści aplikacji klienta mogą być całkowicie nieświadomi istnienia tej warstwy.
- Aby używać warstwy reguł z kodu aplikacji klienta nie trzeba posługiwać się PL/SQL. Dostęp do danych z punktu widzenia aplikacji odbywa się przy pomocy czystego SQL, SQLJ czy pakietu Oracle Business Components for Java (który używa wewnętrznie SQLJ).
- VAPI pozwala bardzo dokładnie zarządzać bezpieczeństwem danych.
- Użycie perspektyw VAPI dla DML i zapytań czyni kod klienta prawdziwie bezpieczny przed zmianami struktur danych leżących pod spodem.

- Dowolne oprogramowanie innych dostawców niż Oracle używające standardowego SQL może łatwo integrować się z warstwą reguł.

Użycie VAPI ma następujące wady:

- Kod perspektyw musi zostać utworzony i utrzymywany w repozytorium (istnieje automatyczny mechanizm jego generacji).
- Generator Web Oracle Designera nie wspiera bezpośrednio VAPI (istnieje obejście aby go do tego zmusić).
- Użycie VAPI komplikuje środowisko bazodanowe.

Dostęp do danych poprzez wyzwalacze TAPI

Wyzwalacze bazodanowe mogą powstać przy pomocy generatora serwera podczas generacji API. Dla każdej tabeli powstaje dwanaście wyzwalaczy (dla każdej akcji na poziomie wiersza i polecenia przed i po zdarzeniu).

Wady i zalety użycia wyzwalaczy TAPI jako warstwy dostępu

Zalety:

- Wyzwalacze TAPI mogą być używane przez wszystkie narzędzia do tworzenia aplikacji użytkownika, dzięki czemu implementacja reguł pozostaje w jednym miejscu.
- Istniejące aplikacje nie wymagają żadnych modyfikacji dla integracji z RuleFrame.
- Użycie wyzwalacze TAPI czynią warstwę reguł całkowicie przezroczystą dla kodu aplikacji klienta. Programiści aplikacji klienta mogą być całkowicie nieświadomi istnienia tej warstwy.
- Aby używać warstwy reguł z kodu aplikacji klienta nie trzeba posługiwać się PL/SQL. Dostęp do danych z punktu widzenia aplikacji odbywa się przy pomocy czystego SQL, SQLJ czy pakietu Oracle Business Components for Java (który używa wewnętrznie SQLJ).
- Wyzwalacze TAPI pozwalają bardzo dokładnie zarządzać bezpieczeństwem danych.
- Dowolne oprogramowanie innych dostawców niż Oracle używające standardowego SQL może łatwo integrować się z warstwą reguł.

Jedyną ważną wadą wyzwalaczy TAPI w stosunku do VAPI jest to, że nie jest to faktycznie dodatkowa warstwa izolująca kod aplikacji od logiki składowanej w bazie. Kod aplikacji użytkownika dalej zależy od zmian w strukturze bazy.

Dostęp do danych poprzez procedury TAPI

Nie może być rozważana jako w pełni funkcjonalna warstwa dostępu ze względu na dużą złożoność użycia i uzależnienie do możliwości wywoływania kod PL/SQL. Jedynym powodem dla którego jest omawiana jest użycie tego sposobu przez generator Web Oracle Designera.

Dostęp do danych poprzez procedury MAPI

Podobnie jak procedury TAPI nie metoda dostępu poprzez procedury MAPI nie może być uważana w pełni za warstwę dostępu do danych. W porównaniu do TAPI ma tą zaletę, że umożliwia przetwarzanie całych macierzy rekordów zamiast poszczególnych rekordów. Metoda może być użyta przy generacji Forms z Designera.

Zastosowanie Business Components for JAVA

Business Components for JAVA (BC4J) jest narzędziem umożliwiającym efektywne rozwijanie przenośnych i elastycznych aplikacji bazodanowych.

Zastosowanie RuleFrame z warstwą dostępu zaimplementowaną przez VAPI może łatwo posłużyć do integracji z JAVĄ przy pomocy BC4J. Obiekty klasy JAVY odpowiadające perspektywom VAPI stanowią doskonały obiektowy interfejs do danych w bazie Oracle 8.

Narzędzia

Podstawowym narzędziami potrzebnymi do budowy RuleFrame jest Oracle Designer oraz Headstart.

Headstart jest pakietem narzędzi stworzonym przez konsultantów Oracle doświadczonych w użyciu Oracle Designera we wszystkich fazach projektu. W jego skład wchodzą międzyinnymi narzędzia do generacji kod CAPI, TAPI oraz VAPI. Twórcy Headstart: Sandra Müller, Lauri Boyd i Steven Davelaar są równocześnie architektami RuleFrame.

Bibliografia

1. CDM Standards and Guidelines Library: Vol.2 Generation of Multi-Tier Web Applications
2. Headstart Oracle Designer - CDM RuleFrame