Java en Oracle niet weg te denken

Finance Data Portal kan niet zonder

Veel mensen zullen bij IT in de bankwereld niet gelijk denken aan innovatieve databasetechniek. Toch is die wel degelijk aanwezig. In twee artikelen proberen Kenneth Hammink en Gert-Jan Paulissen dat beeld bij te stellen. Zij beschrijven daarin de werking van de Oracle Finance Data Portal (FDP) een applicatie die de gegevensstroom goed kanaliseert en beheert. In dit nummer het tweede deel van hun tweeluik, waarin ze dieper zullen ingaan op twee onderwerpen: objectoriëntatie en Java in de database.

De Finance Data Portal ontvangt en verzendt alle in- en uitkomende data voor zowel interne als externe partijen. Het grote voordeel van de Finance Data Portal is dat alles gecentraliseerd is. De andere interne applicaties hoeven slechts één type interface te hebben: de koppeling naar het Finance Data Portal. Daarom is het een belangrijke speler in de wereldwijde verwerking van de financiële gegevens van de Nederlandse Grootbank.

Objectoriëntatie

De systemen die zijn aangesloten op de Finance Data Portal onderscheiden zich op drie manieren voor wat betreft het aanleveren en afnemen van data. Ze kunnen dat doen via bestanden, databaselinks (bijvoorbeeld views) of berichten.

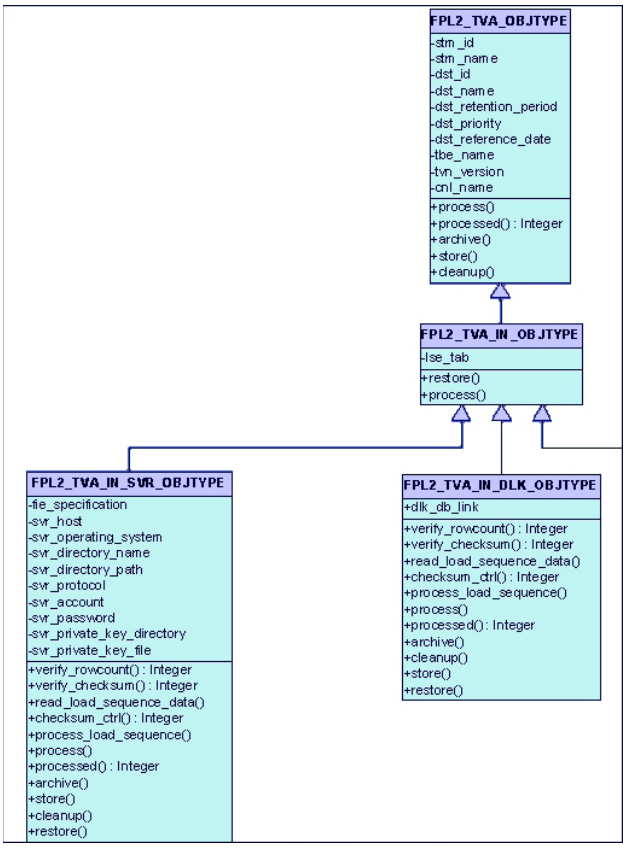
Afgezien van deze verschillen zijn dit de algemene functies binnen de applicatie:

- verwerken van inkomende of uitgaande data;
- archiveren van inkomende of uitgaande data;
- opslaan van meta-informatie van inkomende of uitgaande data;
- opschonen.

Verder verschillen systemen die bestanden aanleveren of afnemen vaak op details:

- zijn er controlegegevens zoals controlesom of aantal rijen;
- hoe wordt een controlesom berekend;
- zitten de controlegegevens in een bestand zelf of in een apart controlebestand.

Als je alle mogelijkheden procedureel gaat implementeren, dan zou er een grote hoeveelheid van if-then-else constructies ontstaan met een bijbehorende complexiteit bij onderhoud en testen. Als echter objectoriëntatie gebruikt wordt, dan kan nieuwe functionaliteit geïmplementeerd worden via overerving. Het systeem is hierdoor veel makkelijker aan te passen. De generieke software hoeft zelden aangepast te worden: een nieuw type aanlevering wordt geïmplementeerd door een nieuwe (sub)klasse.



Figuur 1: Klassendiagram

Oracle heeft sinds Oracle 8 object-relationele eigenschappen. De OO van Oracle lijkt veel op die van Java.

Overeenkomsten tussen OO van Oracle en Java:
- ondersteuning van attributen en methoden (statisch of niet);
- enkele overerving (single inheritance);
- standaard en zelf te definiëren constructors;
- geen destructor zoals in C++;
- van een klasse kan wel of niet een instantie gemaakt worden: sommige klassen beschrijven maar een deel van de functionaliteit waarop voortgeborduurd kan worden;
- een methode kan wel of geen implementatie hebben.

Verschillen tussen OO van Oracle en Java:
- er zijn geen 'private' of 'protected' attributen/methoden in Oracle. Alles is publiek;
- aanroepen van een methode uit een basisklasse wordt niet (direct) ondersteund (Java gebruikt hiervoor super.<method> of this.<constructor>);
- een Oracle object kan worden gebruikt als onderdeel van een tabel zodat opslag in de database mogelijk is. Java biedt hiervoor serialisatie.

Een basisklasse

Het object fpl2_tva_objtype wordt gebruikt voor verwerking van inkomende en uitgaande aanleveringen:

```
create or replace type fpl2_tva_objtype as object (
  stm_id number(10)
, stm_name varchar2(30)

/* some attributes deleted */

/* some methods deleted */

  /**
   * Print the attributes.
   *
   * The super keyword is not supported as in Java: so let print() call
   * print_tva_objtype().
   * Overriding print() methods can call self.print_<parent type> to
print
   * their parent.
   */
, final
  member procedure print_tva_objtype
  ( self in fpl2_tva_objtype
  )

, not instantiable
  member procedure print
  ( self in fpl2_tva_objtype
  )

) not instantiable not final
/


create or replace type body fpl2_tva_objtype as

/* some methods deleted */

final
member procedure print_tva_objtype(self in fpl2_tva_objtype)
is
begin
  dbug.print('debug', 'job: %s', job);
  dbug.print('debug', 'SYSTEM id: %s; name: %s', stm_id, stm_name);
  dbug.print
  ( 'debug'
  , 'DELIVERY SET id: %s; name: %s; retention period: %s; priority: %s;
reference date: %s'
  , dst_id
  , dst_name
  , dst_retention_period
  , dst_priority
  , to_char(dst_reference_date, 'yyyy-mm-dd hh24:mi:ss')
  );
  dbug.print('debug', 'DELIVERY TABLE SET id: %s', dtt_id);
  dbug.print('debug', 'TABLE id: %s; name: %s; control name: %s', tbe_
id, tbe_name, dtt_control_name);
  dbug.print
  ( 'debug'
  , 'TABLE VERSION id: %s; version: %s; format: %s; valid from: %s;
valid till: %s'
  , tvn_id
  , tvn_version
  , tvn_format
  , to_char(tvn_valid_from, 'yyyy-mm-dd hh24:mi:ss')
  , to_char(tvn_valid_till, 'yyyy-mm-dd hh24:mi:ss')
  );
  dbug.print
  ( 'debug'
  , 'TABLE VERSION xml schema length: %s; controlling id: %s; control-
ling indicator: %s'
  , dbms_lob.getlength(tvn_xml_schema)
  , tvn_tvn_id
  , tvn_controlling_ind
  );
  dbug.print('debug', 'channel id: %s; name: %s', cnl_id, cnl_name);
  dbug.print
  ( 'debug'
  , 'TABLE VERSION DATA timestamp: %s; valid from: %s; valid till: %s;
rowcount: %s'
  , to_char(tva_timestamp, 'yyyy-mm-dd hh24:mi:ss')
  , to_char(tva_valid_from, 'yyyy-mm-dd hh24:mi:ss')
  , to_char(tva_valid_till, 'yyyy-mm-dd hh24:mi:ss')
  , tva_rowcount
  );
  dbug.print
  ( 'debug'
  , 'data processed: %s; control data processed: %s; data archived: %s'
  , tva_data_processed
  , tva_control_data_processed
  , tva_data_archived
  );
end print_tva_objtype;

end;
/
```

Overerving

De klasse fpl2_tva_in_objtype wordt gebruikt voor alle inkomende aanleveringen:

```
create or replace type fpl2_tva_in_objtype under fpl2_tva_objtype (
  lse_tab fpl2_lse_tabtype

/* some methods deleted */

, overriding
  member function test
  ( self in fpl2_tva_in_objtype
  , p_method_call in varchar2
  )
  return varchar2
```

```
, final
  member procedure print_tva_in_objtype
  ( self in fpl2_tva_in_objtype
  )

) not instantiable not final
/


create or replace type body fpl2_tva_in_objtype as

/* some methods deleted */

final
member procedure print_tva_in_objtype
( self in fpl2_tva_in_objtype
)
is
begin
  self.print_tva_objtype(); -- call super method
  if lse_tab is not null and lse_tab.count > 0
  then
    for i_idx in lse_tab.first .. lse_tab.last
    loop
      dbug.print
      ( 'debug'
      , 'LOAD SEQUENCE seq: %s; object: %s; file: %s; locked: %s'
      , lse_tab(i_idx).lse_seq
      , lse_tab(i_idx).lse_object_owner
        || '.'
        || lse_tab(i_idx).lse_object_name
      , case
          when lse_tab(i_idx).directory_name is not null

then lse_tab(i_idx).directory_name || ':' || lse_tab(i_idx).file_name
        end
      , lse_tab(i_idx).is_locked
      );
    end loop;
  end if;
end print_tva_in_objtype;

end;
/
```

Door het gebruik van een 'final' methode kan de super constructie als in Java gesimuleerd worden zoals je kunt zien in de print methods.

Java in de database

Sinds Oracle 8 is het mogelijk om functies of procedures aan te roepen die zijn geschreven in een andere taal zoals C of Java. Dit kan nodig zijn omdat PL/SQL niet snel genoeg is of omdat het PL/SQL aan bepaalde functionaliteit ontbreekt.

Dit is een voorbeeld van een PL/SQL procedure die een Java methode aanroept:

```
procedure dump_db(p_properties in clob)
as
language java name 'com.bank.db.Dumper.dumpDB(oracle.sql.CLOB)';
```

Als test is gebruik gemaakt van een Java programma om SQL queries naar bestand te dumpen. Als voorbeeld is genomen de volgende query: select * from sys.all_objects. Het O/S:

Oracle 10.2 draait in een virtuele Linux machine op het O/S.

Wanneer er via Java in de database wordt geschreven dan kost het ongeveer tachtig seconden om de 41.000 objecten te dumpen. Als het via de command line wordt geschreven kost het vijf seconden! Duidelijk is dat dit verlies aan performance onacceptabel is.
Als oplossing is er uiteindelijk voor gekozen om via Java in de database een O/S script te starten dat via de command line het commando uitvoert. Dit geeft wel een acceptabele performance.
In Oracle 10g is het trouwens ook mogelijk om Java Native Compiled libraries (NCOMP of JAccelerator) te installeren via de Database Companion CD. Dit geeft een flinke verbetering.
In Oracle 11g is aangekondigd dat de performance van Java in de database verbetert door het gebruik van een JIT (Just In Time) compiler.

Conclusie

Het gebruik van Oracle-types en Java zijn niet voor elke ontwikkelaar dagelijkse kost. Toch kan de Finance Data Portal applicatie niet zonder: in de praktijk is gebleken dat de OO benadering leidt tot veel minder veranderingen aan bestaande programmatuur. Het heeft dus zowel een positieve invloed op de hoeveelheid ontwikkel- en testwerkzaamheden en ook op de kwaliteit van de programmatuur.
Het gebruik van OO is zeker het overwegen waard bij het ontwikkelen van Oracle applicaties.
Het gebruik van Java heeft zeker voordelen als de functionaliteit van PL/SQL niet voldoende is. De invloed op de performance moet dan wel goed worden getest.

Referenties

Application Developer's Guide - Object-Relational Features
Thread: Oracle recommends installing JAccelerator, http://forums.oracle.com/forums/thread.jspa?threadID=304269


Kenneth Hammink (l.) en Gert-Jan Paulissen werken bij een Nederlandse Grootbank aan een Data Portal.