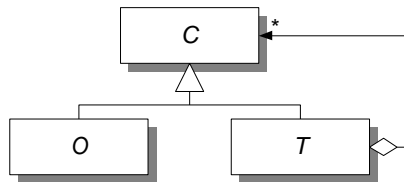


*Experiences from Object-relational
Programming in Oracle8*
COT/4-06-V1.4



Centre for Object Technology

*Centre for
Object Technology*

Revision history:	16.08.98	v.0.1	First review
	09.09.98	v.1.0	Second review
	23.10.98	v.1.2	Third review
	03.11.98	v.1.3	Ready for public review
	15.02.99	v.1.4	Final version

Authors: Johnny Olsson, WM-data
Allan R. Lassen, Rambøll

Status: final

Publication: public

Summary:

This report describes experiences from research experiments mapping an object-oriented model into the object-relational programming environment Oracle8.

As a cookbook, the different tasks creating object types, tables and encapsulation packages for the different object-oriented abstraction mechanisms are explained. It is shown where Oracle8 supports object orientation and where it lacks for support.

It is argued that Oracle8 does not give significant improvements for object storage purposes compared with traditional relational databases.

Readers should have knowledge of Oracle SQL and PL/SQL concepts and object-oriented language mechanisms.

The Centre of Object Technology (COT) is a 3 year project concerned with research, application and implementation of object technology in Danish companies. The project is financially supported by The Center of IT-Research (CIT) and the Danish Ministry of Industry.

Participants are:
Maersk Line, Maersk Training Center, Bang & Olufsen, WM-data, Rambøll, Danfoss, Systematic Software Engineering, Odense Stålskibsværft, A.P. Møller, Aarhus Universitet, Odense Universitet, Københavns Universitet, Dansk Teknologisk Institut og Dansk Maritimt Institut

The Centre of Object Technology (COT) is a 3 year project concerned with research, application and implementation of object technology in Danish companies. The project is financially supported by The Center of IT-Research (CIT) and the Danish Ministry of Industry.

Participants are:

Maersk Line, Maersk Training Center, Bang & Olufsen, WM-data, Rambøll, Danfoss, Systematic Software Engineering, Odense Stålskibsværft, A.P. Møller, Aarhus Universitet, Odense Universitet, Københavns Universitet, Dansk Teknologisk Institut og Dansk Maritimt Institut

1. Introduction

This paper outlines the results of experiments with mapping of object oriented models into object-relational models - as it is available in the database system Oracle8.0.4.0.0.

There are several proposals how to map object oriented models into the entity-relational schema. [5, 9]. This paper investigates the object opportunities in the object-relational model as it is current in Oracle8. The work extends earlier work described in [1].

Based on a simple model of how to use object facilities in Oracle8, it is shown how we map the object oriented abstractions: instantiation, association and aggregation into an object-relational environment. The mapping process is interesting as part of an automatic code generation based on a formal specification - such as UML. Automatic code generation based on an object model is interesting not only for the database, but also for the host languages such as Java. However, this report concerns the database only.

To understand in what context the automated code exists, we describe a system architecture known from [7] in Figure 1.

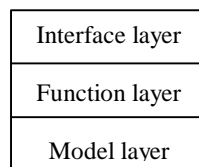


Figure Error! Unknown switch argument.. System architecture.

The model layer contains the model of the system, i.e. the classes, structure and behaviour of the system. The function layer holds the functionality that is applied on the system to change the model layer. The interface layer ties the system together with the users in terms of reports and forms that make it possible for the user to initiate changes of the model. The architecture is regarded logical rather than physical, i.e. a program component may contain code from all three layers.

The model layer consists of a persistent and a transient sub-layer. The persistent sub-layer is the model as a database schema and the storage mechanisms. The transient sub-layer contains the model in a host language. This sub-layer expresses more of the systems behaviour than the persistent sub-layer.

The schema could be an Oracle8 schema applying object-relational facilities with PL/SQL and SQL, and the application code could be a host language such as Java. As already mentioned, this paper discusses only the mapping of UML to the database schema and the storage mechanisms. Issues about code generation for the host language are not concerned.

The architecture shown in Figure 2 is our proposal for the context for the experiments we have carried through. Other proposals for an architecture would be as relevant as this

one. The purpose with the architecture is only to examine the object facilities in Oracle8.

Figure 2 shows the process of automatic code generation from specification to the model layer of an application.

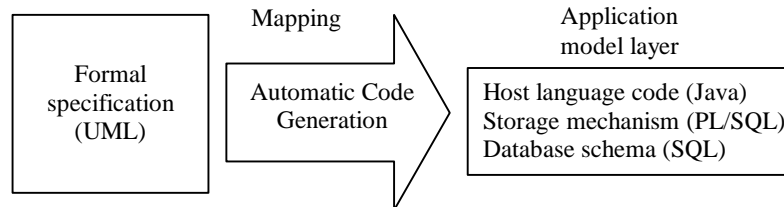


Figure 2. From a formal specification to an application.

The results of this paper can be seen both as an evaluation of object facilities in Oracle8 and as a cookbook on how to map an object oriented model into an object-relational schema. The cookbook is of course limited to the suggested architecture and the limits of Oracle8.

The model presented in this paper concerns only one-way mapping of an object oriented model into an object-relational model. Integration with existing entity-relationship/object-relational models has not been investigated.

1.1 Overview

Section 1.2 describes an ongoing example used throughout this report. Section 2 gives an introduction to the object model of Oracle8. The principles of transferring an object oriented model into the object-relational model supported by Oracle8 are explained in Section 3.

It is shown how to map the object oriented abstractions: instantiation, association, and aggregation into the object-relational model. Section 4 shows how to map instantiation. Section 5 is divided into 2 subsections concerning associations with cardinality many-many (N-N) and one-many (1-N), respectively. Only binary associations without attributes have been examined. This is due to time constraints.

Presently, Oracle8 does not support specialization. Forthcoming versions are supposed to do so [4], hence there is no reason to investigate them now.

The mapping of aggregation into the framework of Oracle8 is shown in Section 6. Section 7 discusses some issues about object modeling in Oracle8 using constraints and triggers.

Suggestions for further work is presented in Section 8, and in Section 9, the conclusions of the work are outlined.

The experiments has been carried out on the following installation: Oracle Enterprise Edition version 8.0.4.0.0 for Windows NT with the Oracle8 Object Option. According to [4], future releases of Oracle8 will support more object oriented mechanisms such as specialization and polymorphism. Despite the immature level of the product, we find it interesting to examine Oracle8 now, because it is assumed that the basic principles of object technology will remain in future releases. We find that the basic object principles are crucial for success.

1.2 Working example

The work described in this paper was based on the model illustrated in the following UML [8] diagram in Figure 3. The model constitutes the formal specification of the model to be mapped into an object-relational schema.

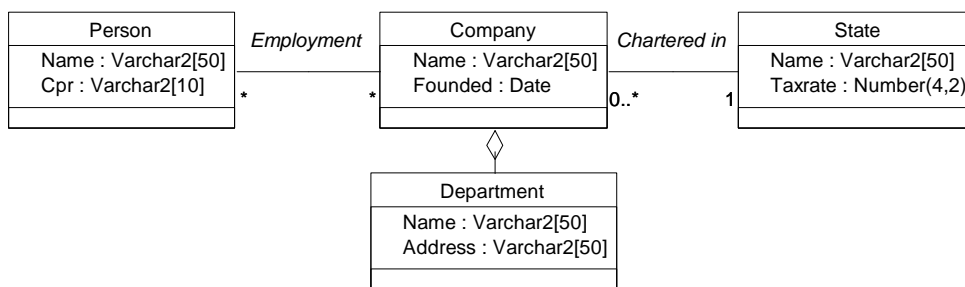


Figure 3. Working example.

Person is N-N associated with Company. A person can be employed with a number of companies, and a company has an arbitrary number of employees.

A company is chartered in one state/country, i.e. where the company is registered. A state may charter any number of companies. This is expressed with an 1-N association.

A company consists of a number of departments. Logically, a department is within a company, and therefore company is an aggregate of departments.

The number of attributes has been minimised for simplicity reasons. No methods are shown in the diagram.

2. The Object Model of Oracle8

According to Oracle8 documentation [2], “An object type is a user-defined composite datatype that encapsulates a data structure along with the functions and procedures needed to manipulate the data. The variables that form the data structure are called attributes. The functions and procedures that characterize the behavior of the object type are called methods.”

An object type is specified as follows [2]: “Like a package, an object type has two parts: a specification and a body. The specification is the interface to your applications;

it declares a data structure (set of attributes) along with the operations (methods) needed to manipulate the data. The body fully defines the methods, and so implements the specification.”

In Oracle8, the concept of records has been extended with new features. All records in tables have a unique object identifier - throughout all tables. The object identifier is not a visible column. The identifier of an object can be read through the REF function in a select statement, which is shown later. In Oracle8, columns of the user defined object types and object references can be specified as traditional columns. This is illustrated in later sections.

New objects can be created by a constructor method that is implicit given for all object types. Aggregation is supported by introducing collection types. Currently, Oracle8 does not support specialization, although Oracle says that it will be in forthcoming versions of Oracle8 [4].

The object oriented abstraction mechanisms are examined in the following sections.

It is important to realize in what context objects are persistent. Object types are defined in the Data Definition Language (DDL) as classes. Based on the object types, the database schema is defined. Objects stored in the database are as any other record in a table restricted to be manipulated by the SQL commands SELECT, INSERT, UPDATE and DELETE (the Data Manipulation Language, DML, commands).

As manipulable objects (method access etc.) they exist only as object variables in the programming environment of PL/SQL. When an object is selected from the database it is copied into an object variable in a PL/SQL-block. Method invocation of the selected object has impact only on the copy of the object - though it shares object id with the persistent object. Method invocation on one object cannot have side-effects on other objects.

To make changes persistent, an update of the object in the database is needed. In other words, the copy held in the PL/SQL-block is a transient value copy of the database object similar to traditional PL/SQL programming where database records are loaded into PL/SQL variables. Attributes may be changed and the old database objects updated with the new attribute values. The principle is illustrated in the following figure.

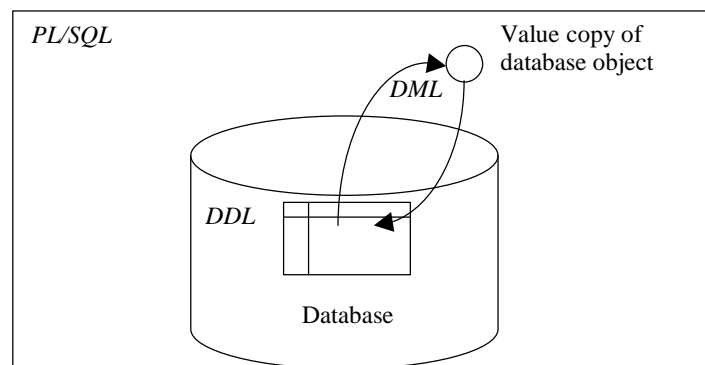


Figure 4. Objects are persistent in the database. An object fetched into an object variable in a PL/SQL-block is transient.

The syntax for creating object types and applying objects are shown in the following sections.

3. The ORM Step Model

The general framework for mapping (object-relational modeling = ORM) an object model into Oracle8 is as described in Figure 5. The figure illustrates the process for a single class Person, which is the simple case. More complex situations are described in the following sections.

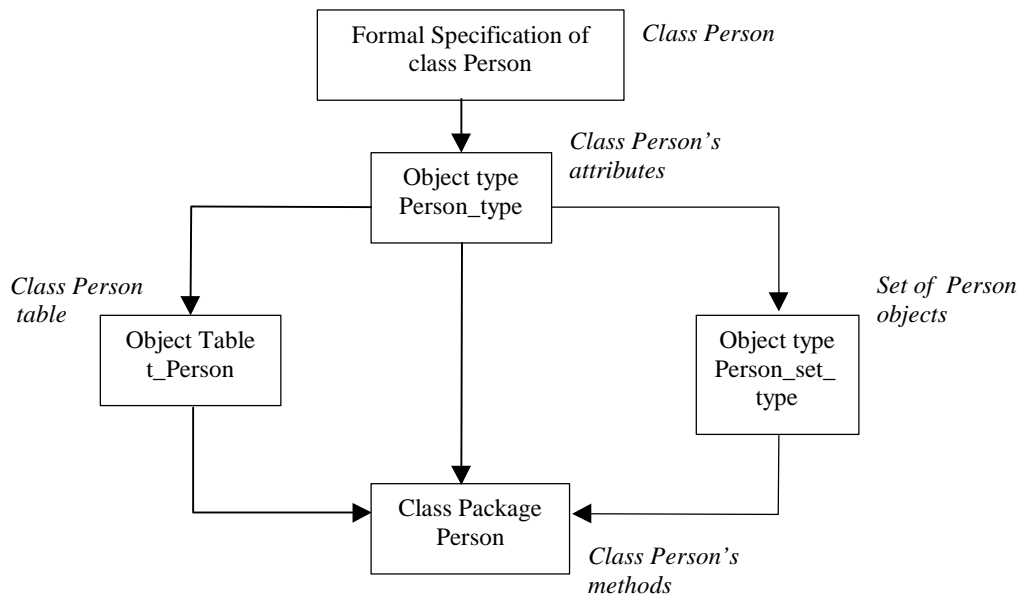


Figure 5. General framework for mapping a class into an object-relational domain. The lines indicate dependencies. The object type and the object table require the formal specification of a class etc.

The mapping is based on an object oriented specification (such as UML). A class' attributes are mapped into an object type. The methods of a class are gathered in a package - in the following named a *class package*. As described in the previous section, object type methods cannot have side effects. We need methods that may change state and references for other objects. This is the reason why our methods are organised separate from the object type. The class package is the encapsulation mechanism by convention. All access to objects of the object type is done through the class package. Technically, it is possible to access a table directly, but that is not the intention, and as such unconventional usage.

Prior to creation of the class package, the storage needs to be declared. A class is mapped into a table - named an *object table*.

Often, it is convenient to operate on a set of objects of the same type. This is illustrated in later sections. Although, for now we claim the demand for set types based on a object type, as shown in the figure.

3.1 Sample Code

This section describes part of the PL/SQL code for the object-relational model of class Person. As a first step, the object type of person is defined in the following DDL-statement.

```
create or replace type Person_Type as object (  
    Name varchar2(50));
```

Additional attributes of Person could appear after the Name attribute.

The set type of Person looks as follows:

```
create or replace type Person_Set  
    as table of ref Person_Type;
```

Creation of the storage for objects of type Person_type looks as follows:

```
create table t_Person of Person_Type
```

The class package of Person is defined as in the following:

```
create or replace package Person as  
function New(aName t_Person.Name%TYPE)  
    return ref Person_type;  
function Get(aName t_Person.Name%TYPE)  
    return Person_set;  
procedure Put (aName in t_Person.Name%TYPE);  
End Person;
```

The class package body looks as follows.

```
create or replace package body Person as  
  
function New(aName t_Person.Name%TYPE) return ref  
    Person_type is  
    person_ref ref Person_type;  
begin  
    insert into t_person Values ( Person_type(aName));  
    select ref(p)  
    into person_ref  
    from t_person p  
    where p.Name = aName;  
    return person_ref;  
end New;  
  
function Get(aName t_Person.Name%TYPE) return Person_set is  
    p_set Person_Set;  
    n binary_integer := 0;  
begin  
    p_set := Person_Set();  
    for Person_cursor in  
        (select ref(p) pp from t_person p where p.name = aName)  
    loop  
        /* loads selected objects into p_set */  
        n := n + 1;
```

```
        p_set.Extend;
        p_set(n) := Person_cursor.pp;
    end loop;
    return p_set;
end Get;

procedure Put ( aName in t_Person.Name%TYPE) is
begin
    insert into t_Person
    values ( Person_Type( aName));
end;

End Person;
```

Notice that when declaring variables of type of the attribute Name, they are not referred via the Person_type. That is because Oracle8 does not allow variable declarations based on an attribute of an object type. In other words, the %TYPE function does not work for attributes on object types. A desirable type declaration of a variable/parameter would be such as:

```
aName Person_type.Name%TYPE
```

Where %TYPE refers to the type of the Name attribute. However, that is not possible.

After the above DDL-statements, the database schema and storage mechanisms are ready to use. The next sections show how to instantiate person objects and how to store them.

4. Instantiation

Instantiation is a method, New, on the class package. The function creates an object in a PL/SQL variable, stores it, and returns an object reference to the stored object.

```
function New(aName t_Person.Name%TYPE) return ref Person_type is
    person_ref ref Person_type;
begin
    insert into t_person Values ( Person_type( aName));
    select ref(p)
    into person_ref
    from t_person p
    where p.Name = aName;
    return person_ref;
end New;
```

To create an object in the context of PL/SQL, Oracle8 offers a method named after the object type itself. In the example above the method is Person_type with parameter aName.

Instantiation of a new person looks as follows. First, a reference variable for a person is declared.

```
person_ref REF PERSON_TYPE;
```

Then a person is instantiated, stored and referred by person_ref.

```
person_ref := Person.New('Johnny Olsson');
```

5. Associations

Associations are mapped into reference tables and object reference columns according to the cardinality of the associations. The concept of associations are initially discussed by Rumbaugh [6]. In previous work [1], we proposed implementations of associations in the object oriented language Smalltalk and in Oracle8. The following is based on this work.

Associations of cardinality N-N are mapped into a table with object references to the objects in the participating object tables. 1-N associations are mapped by extending the N-participant object type table with an object reference to the 1-participant object. The impact of associations on the ORM step model is described for those two situations in the following.

The concept of collections of associations has not been investigated.

Because of the previously mentioned problems with methods having side effects, the methods on the association are not compiled into the participating classes.

Deletion of objects leading to data inconsistency is discussed in Section 7.

5.1 N-N Associations

The Employment association between Company and Person is a N-N association. As a N-N relation in entity-relationship models, the association is modeled as a table with references to the participants in the association. In Oracle8, the relation table consists of object references as shown:

```
create or replace type Employment_Type as object (  
    Employee ref Person_Type,  
    Employer ref Company_Type);
```

Employee and Employer are the role names of person and company, respectively. The storage of Employment relations are defined as follow:

```
create table Employment_Rel of Employment_Type;
```

The relation package of Employment looks as follows:

```
create or replace package Employment as  
    procedure Add(AnEmployee ref Person_Type,  
                AnEmployer ref Company_Type);  
    function Test_member(AnEmployee ref Person_Type,  
                       AnEmployer ref Company_Type) return boolean;  
    function getEmployees(AnEmployer ref Company_Type) return  
Person_Set;  
    function getEmployers(AnEmployee ref Person_Type) return  
Company_Set;  
    Employment;
```

The procedure Add is the only procedure/function shown in the package body of the Employment.

```
procedure Add(AnEmployee ref Person_Type,  
             AnEmployer ref Company_Type) is  
begin  
  insert into Employment_Rel  
  values( Employment_Type( AnEmployee, AnEmployer));  
end Add;
```

Declarations of references for a person and a company look as follows:

```
ref_t_p ref Person_type;  
ref_t_c ref Company_type;
```

After instantiation of person and company objects, references can be inserted into the association:

```
ref_t_p := Person.New('Johnny2');  
ref_t_c := Company.New('WM-data');  
Employment.Add( ref_t_p, ref_t_c);
```

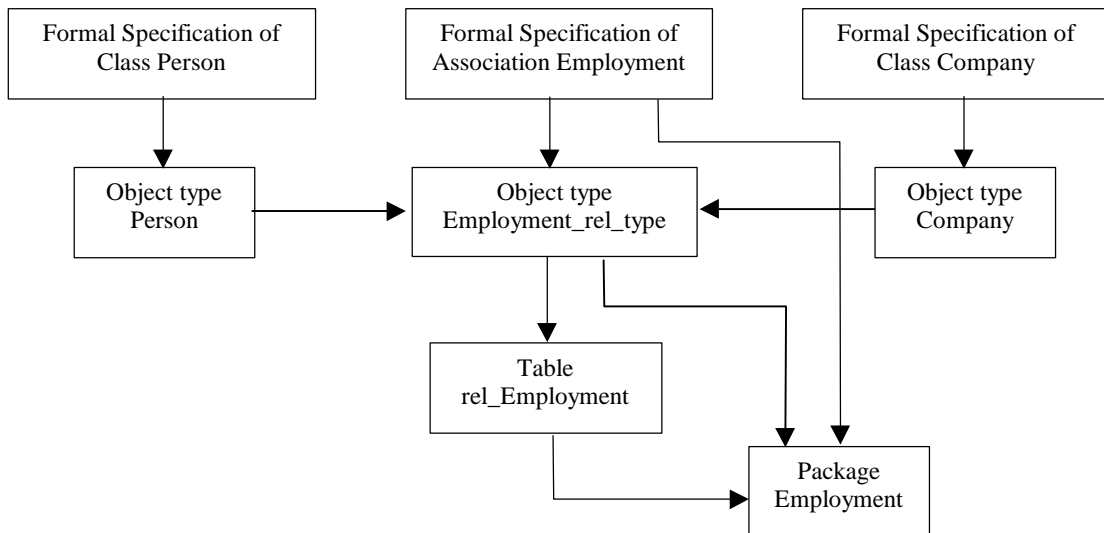


Figure 6. Mapping of N-N association.

5.2 1-N Associations

The ongoing example has an 1-N association between Company and State. A company is chartered in one state. A state may have any number of companies.

The 1-N association can be mapped into at least two models. The one is the same as for the N-N association where the association is mapped into an object type which is instantiated in a relation table. Access methods and update methods concerning the association are placed in the chartered_in package.

The second solution is less conceptual founded. The solution extends the object type of the N-participant with an object reference to the 1-participant. This solution is very similar to foreign keys for 1-N relations in entity-relationship modeling. The advantage of this solution is performance and simpler SQL-statements rather than conceptual support.

Opposed to the N-N association, the 1-N association has an impact on the N-participant object type of the association. This is due to the chosen implementation of the mapping where the association is an object reference in the N-participant object type. Having all associations implemented as relation tables would give the same ORM step model as for N-N associations. However, we consider the practical solution of the 1-N association to be a basic requirement for performance purposes.

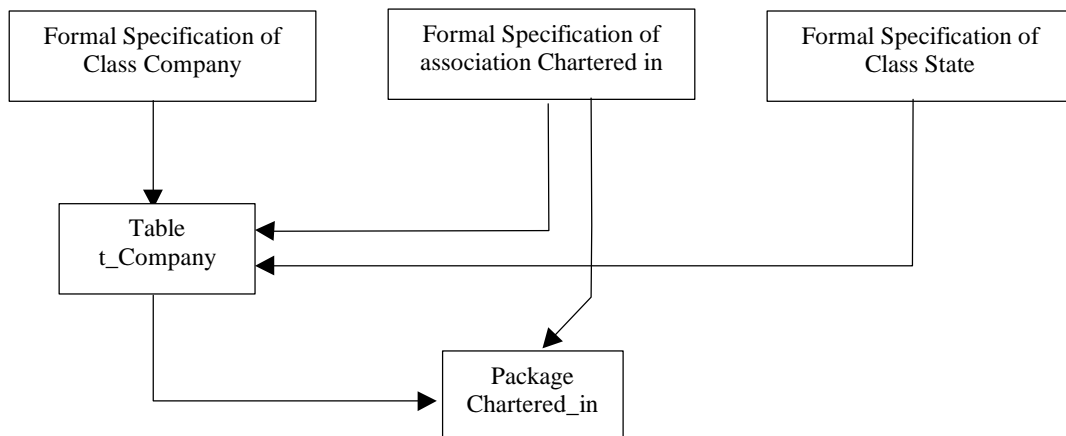


Figure 7. Mapping of an 1-N association.

Unfortunately, it is not possible to alter object types and alter tables based on an object type after their initial creation. In other words, schema evolution for object tables is not an option.

This is an important because it leads to heavy consideration whether Oracle8's object facilities can be used in commercial system if maintenance is restricted by no schema evolution.

Due to the chosen implementation solution of the association, no object type and table of chartered in is created. Creation of the company type with the reference to a state object looks as follows:

```
create or replace type Company_Type as object (  
    Name varchar2(50),  
    State ref State_Type);
```

The instantiation procedure of a company object is extended to include an initial null reference to a state object.

```
function New(aName t_Company.Name%TYPE) return ref Company_type is  
    Company_ref ref Company_type;  
    aDepartment_Set Department_set;
```

```

begin
  insert into t_Company
  Values ( Company_type( aName,null));
  select ref(p)
  into Company_ref
  from t_Company p
  where p.Name = aName;
  return Company_ref;
end New;

```

State objects are created and stored in a way similar to Person objects as shown above.

If `ref_t_s` is a reference to a state object and `ref_t_c` is a reference to a company object, an association between the company and the state is established as follows:

```
Chartered_in.Add( ref_t_s,ref_t_c);
```

6. Aggregation

Aggregation is easily supported by Oracle8 in terms of collections. A collection is an ordered group of elements, all of the same type. PL/SQL offers two kinds of collections: nested tables and varrays (short for variable-size arrays). In our experiments, we have looked only at nested tables, as they show the strongest conceptual support for aggregation. Varrays have a fixed upper bound, but nested tables are unbounded. So, the size of a nested table can increase dynamically.

Aggregation is sometimes implemented as an 1-N association. In our experiments, we have only been interested in mapping aggregations into Oracle8 collections. The 1-N association mapping of aggregation is similar to the one described in Section 5.2.

As we only investigate aggregation as collections, we do not discuss the situation where there is an explicit defined number of elements in the aggregate. This corresponds to the situation where the elements in the specification have role names.

The figure below illustrates the dependencies from the specification to the object type and the tables. The dashed line indicates the aggregate relationship with objects of class B.

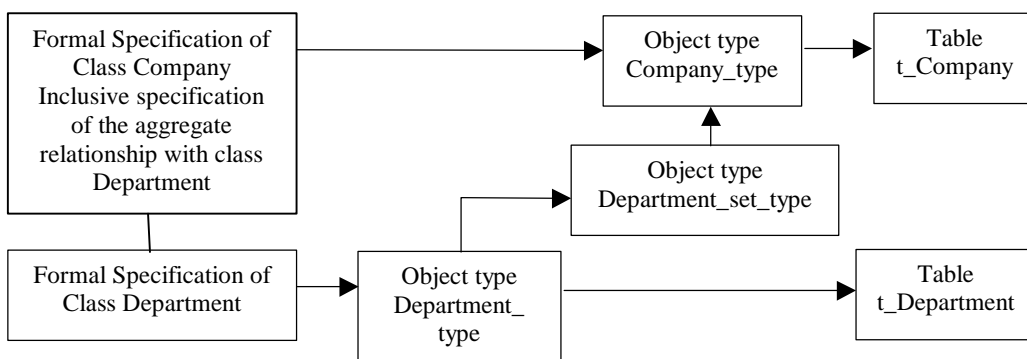


Figure 8. Mapping of an aggregation.

*Centre for
Object Technology*

In the ongoing example, a company has a set of departments. This is defined in the following statement (the former defined reference to state is not included here):

```
create or replace type Company_Type as object (  
    Name varchar2(50),  
    Department Department_Set);
```

Department_Set is defined as follows:

```
create or replace type Department_Set as table of ref Department_Type;
```

A table for departments is created as usual:

```
create table t_Department of Department_Type
```

To create a table of the Company type with departments, the following statement is required:

```
create table t_Company of Company_Type  
    NESTED TABLE Department store as t_Department_Set;
```

Where store as t_Department_Set names the internal table containing the sets of departments for each company. The table is not accessible except implicit select in t_company. Consequences of this have not been studied further. However, it may be a problem when deletion of an object triggers changes of other objects through the aggregation.

To add a department into a company the following statement is executed:

```
Company.AddDepartment( ref_t_c, ref_t_d);
```

Where ref_t_c refers to a company, and ref_t_d refers to a department company.

From the package body the procedure for adding a new department is shown:

```
procedure AddDepartment(aCompany_r in ref Company_type,  
    aDepartment in ref Department_Type) is  
    aDepartment_Set Department_Set;  
begin  
    select C.Department  
    into aDepartment_Set  
    from t_Company C  
    where aCompany_r = ref(c);  
    aDepartment_Set.Extend;  
    aDepartment_Set(aDepartment_Set.last) := aDepartment;  
    update t_Company C  
    set C.Department = aDepartment_Set;  
end;
```

Here is an example of how to treat that objects in PL/SQL which are copies of objects in the database. The set of departments is loaded into PL/SQL, a new department added, and the original record is updated with the new value of the Department column.

7. Constraints and triggers

In many situations, it is desirable to have implicit check on data consistency and automatic creation and deletion of objects. For instance, when deleting a person, the associations whereof the person is part should also be deleted. Otherwise, dangling references will occur.

The usual way to handle this situation in Oracle databases is by integrity constraints and database triggers. Integrity constraints take care of strict referential constraints, whereas triggers control more complex dependencies between objects described in PL/SQL blocks. Typically, triggers are used for dependencies that require procedural tasks to be established. This section describes experiences with triggers and constraints in Oracle8.

Oracle8 does support constraints on foreign keys. However, constraints on object references are not supported. This is an important lack of functionality in Oracle8 as an object-relational database system. The missing support for integrity checks and referencing objects means that all check of consistency needs to be done explicit at a higher program level.

Another strategy to maintain data consistent is usage of database triggers. However, Oracle8 does not support object references on correlation names. For instance, there is no support for statements like this:

```
DELETE FROM Employment_Rel w
WHERE REF(w) = REF(:old);
```

Where REF(:old) refers to the object identifier of the triggering object.

There are other issues when talking about triggers and constraints in a partly object oriented world. For instance, should the UML specification contain information about possible cascade deletion? Deletion of an aggregate object may cause deletion of all objects within the aggregate etc. This is an issue for further investigation.

Another more architectural consideration is whether the trigger PL/SQL-block should have access to the class package layer. Or, is it a violation of the layered model described in the introduction?

8. Further work

This report has been concerned only with mapping of the class diagram of an UML model. The event list and state diagrams which also are a basic part of an object-oriented model have not been taken into account. Above, we have not had much use of methods on the object type. The mapping of the event lists and state diagrams could probably require greater use of methods on the object types. This is an issue for further investigation.

The experiments shows guidelines for mapping various object oriented abstraction mechanisms. Next step will be to move the ORM step model into a case tool, for instance Rational Rose (www.rational.com), and experience automatic code generation for a larger model. The model described in [3] is a candidate for such an experiment.

Another issue is to get experience from applying the class packages and methods on object types from a host language such as Java. This experiment would give an idea about whether the suggested mapping is useful from an object oriented perspective.

And as discussed in the previous sections, it would be a matter of interest to study the layered model. Where do triggers and PL/SQL-blocks belong in the model?

9. Conclusion

The section summarizes the results described in this paper.

In Section 2, the object facilities of Oracle8 are discussed. As an important issue, it is explained at what level objects are persistent and at what level they are transient and subject for changes. Object types are defined at DDL level. Persistency is handled by DML. Objects are instantiated and may change state in the transient PL/SQL environment.

In Section 3, the ORM Step Model is described. The model describes the steps from a formal specification, to Oracle8's object types, class tables and class packages to encapsulate and ensure persistence of objects.

Section 4, 5, 6 explains how to model instantiation, associations and aggregations in Oracle8 in the ORM Step Model. Specialization has not been investigated, because it is not supported, but forthcoming versions of Oracle8 are supposed to support it.

In Section 7, it is described that Oracle8 has the usual Oracle-facilities for constraints and triggers, but Oracle8 does not support constraints on object references, and correlation names in triggers cannot contain object references.

The following list shows what we have found of unsupported features in Oracle8:

- Object types cannot be altered
- Tables based on object types cannot be altered
- Methods on object types cannot contain DML-statements
- Constraints cannot be defined for object references
- Triggers: Correlation names for object references does not exist

As described above, there are important problems with the object facilities in Oracle8. The question is whether the Object8 facilities should be used for object-storage at all. One alternative is to use the traditional mapping into a pure relational schema. Another alternative would be to let the integrity control be handled at a higher level, for instance in the host language.

We find that the current maturity of object-oriented facilities in Oracle8 does not give significant benefits enough to warrant use commercial products. Triggers, integrity constraints and schema evolution are required for commercial use. Oracle8 does not support these concepts for object facilities.

As we have not investigated other new features of Oracle8, we are not able to evaluate more general reasons to use Oracle8.

10. References

- [1] A. R. Lassen, J. Olsson, K. Østerbye. *Object Relational Modeling*, 1998. COT/4-04-v1.0.
- [2] On-line system documentation for Oracle 8.0.4.0.0. for Windows NT.
- [3] J. Olsson, K. H. Nielsen, K. Østerbye, A. R. Lassen. *Objektorienteret Analyse og Design af Udvalgte dele af STADS*. COT/4-03.V2.0
- [4] Oracle8 Object-Relational Data Server: The Next Generation of Database Technology. An Oracle Business White Paper, p. 6., June 1997
- [5] Michael Thomsen, *Persistent storage of OO-models in relational databases*. COT/4-02-V1.5, 1998
- [6] James Rumbaugh. *Relations as Semantic Constructs in an Object-Oriented Language*, 1987, Proceedings of OOPSLA'87. Pages 466-481
- [7] L. Mathiassen, A. Munk-Madsen, P. A. Nielsen, J. Stage, 1997, *Objektorienteret Analyse og Design*, Marko, Aalborg
- [8] Rational, 1997, *UML version 1.1*, www.rational.com
- [9] Kyle Brown and Bruce G. Whitenack. *Crossing Chasms: A Pattern Language for Object-RDBMS Integration "The Static Patterns"*. Technical Journal www.ksccary.com/Articles/ObjectRDBMSPattern/ObjectRDBMSPattern.htm