Distributing CORBA Views From an OODBMS

Eric Viara
SYSRA
523, place des Terrasses
91000 EVRY, France.
Eric.Viara@sysra.com

Guy Vaysseix
CRI INFOBIOGEN
523, place des Terrasses
91000 EVRY, France
Guy.Vaysseix@infobiogen.fr

Emmanuel Barillot
GENOPLANTE
523, place des Terrasses
91000 EVRY, France
Emmanuel.Barillot@infobiogen.fr

Abstract

The need to distribute objects on the Internet and to offer views from databases has found a solution with the advent of CORBA. Most database management systems now offer CORBA interfaces which are generally simple mapping of the database schema to the CORBA world. This approach does not address all the problem of database interoperation because (i) such a view is static (ii) its semantic is completely bound to the semantic of the schema and it is not possible to re-model it (iii) only one view per database can be offered (iv) access may be limited to reading and no mechanism is given to write in the database through the view. To solve these problems, we have designed a language, the Interface Mapping Definition Language (IMDL) and some tools, grouped in the Interface Mapping Service (IMS). IMDL is used to define CORBA views from OODBMS, while IMS generates an IDL construct and a full CORBA implementation from an IMDL construct and a database schema.

1. Introduction

The problem of data interoperation remains one of the greatest challenge in the domain of database science. To tackle this issue, the Object Management Group [17] promotes the Common Object Request Broker Architecture (CORBA) [9, 15, 17, 18, 20, 23] for several years. CORBA specifies a common language and protocol for the description and distribution of objects: the Interface Definition Language (IDL) and the Internet Inter-ORB Protocol (IIOP). They allow for a language and platform independence of servers and clients. Besides, CORBA offers a great variety of services such as Naming or Trading. For all these reasons, CORBA has been successfully adopted in several domains as different as banking, electronic commerce, scientific research and transportation.

Most database management systems (DBMS) now provide a CORBA interface. This interface allows the manipulation of objects (or rows) from a database through an ORB layer. Generally, the interface is schema oriented in the sense that the IDL interface is a direct mapping of the database schema. Each class (or table) is mapped on a CORBA interface in a one-to-one mapping: each class attribute is mapped to an interface attribute and each class method is mapped to an interface method.

This approach enables a standardized distribution of objects but does not address all the requirements of database interoperation. For example, it may be preferable not to export some parts of the schema (some classes, attributes, or methods) which have only an internal purpose. When dealing with biological information, the domain whose database interoperation needs drove our developments, it is often necessary to offer several views from one database, simply because Biology is not yet a unified science but presents many facets : for example a physician, a molecular biologist and a geneticist would not be interested in the same aspect of the information known on a given gene. Also there are a myriad of databases storing data on the same knowledge domain but with a different perspective, which means that the database schemas present a semantic gap that has to be bridged. To address this issue, the different data providers may wish to use a common interface definition, or a standard interface may preexist that will enforce the database manager to map a database schema to a given target IDL. All these points are not addressed by a one-to-one mapping of the database schema to an IDL.

To solve these problems, we have designed a language, Interface Mapping Definition Language (IMDL) and tools, grouped in the Interface Mapping Service (IMS), to define and produce CORBA views from an OODBMS [2, 26, 14] in an automated and flexible way.

The current work has been designed to be as generic as possible: ideally, the IMDL language and IMS should not depend neither on the OODBMS nor on the ORB used. This is true for IMDL: the IMDL language is quite independent from the OODBMS and the ORB used. However, the In-

terface Mapping Services (IMS), whose main purpose is to generate the CORBA bridge for the given OODBMS and ORB, both depend on the OODBMS and the ORB used.

Because ORBs follow the CORBA standard, there is a large common denominator to all the ORB implementations: the differences appear essentially in the form of the provided API and the quantity of services provided. IMS has been designed so that only a couple of days is necessary to port IMS from one ORB to another.

The diversification in the set of the OODBMS is larger: although ODMG specifies and promotes a standard for OODBMS (see 2.2), only a few OODBMS are ODMG compliant and very few of them implement strictly the C++ language binding. IMS can be adapted to any OODBMS that supports at least single inheritance and provides a query language, but the adaptation time is far more important than for an ORB adaptation.

The EYEDB OODBMS has been choosen to realize the first implementation of the Interface Mapping Services for two main reasons: the first one is that EYEDB is closely based on the ODMG concepts and so, it makes future ODMG-based OODBMS adaptation easier; the second more pragmagtic reason is that the designers of IMDL and IMS are the designers of the EYEDB OODBMS.

The actors of the problem of distributing CORBA views are presented in section 2, while those intervening in the solution (IMDL and IMS) are described section 3. Section 4 exposes the details of use of IMDL and IMS through a basic example and section 5 discusses about related work and approaches.

2. CORBA view distribution: Actors of the problem

2.1. CORBA

CORBA specifications have been designed by the Object Management Group (OMG) at the beginning of the '90. These specifications mainly include:

1. **architecture specification**: the OMG Object Management Architecture (OMA). The major components of this architecture are the clients, the servers and the Object Request Broker (ORB) which is the inescapable mediator between clients and servers.

2. **interface specification**: services provided by servers to clients are described in a platform independent interface language named Interface Definition Language (IDL),

3. **protocol specification**: object are distributed accross the network using a standard protocol named Internet Inter-ORB Protocol (IIOP).

4. **standard services**: a great variety of standard services gathered in a set called *CORBAservices* are specified: Lifecycle, Relationship, Persistent Object, Externalization, Naming, Trading, Event, Transaction, Concurrency, Property, Query, Security and Licensing Services.

CORBA specifications are currently implemented by many ORB products: `Orbix` , `Orbacus` , `VisiBroker` , `HP ORB Plus` , `SUN Neo` and so on. All this products implements the OMA, the IDL and the IIOP protocol but, generally, do not implement currently all the standard services gathered in the *CORBAservices*.

2.2. Object Oriented Database Management Systems

The domain of the Object Oriented Database Management Systems (OODBMS) is large and diversified; the OODBMS concept includes several meanings. However, all the OODBMS have a number of common points:

1. they provide an object model which allows for the definition of complex data:

   (a) single or multiple inheritance is provided,

   (b) classes include both attributes and methods,

   (c) class attributes can take the form of literal types, arrays, collections or object references.

   (d) one-to-one, one-to-many and many-to-many relationships are often supported.

2. each database object is identified in a unique way by an Object IDentifier (OID),

3. integration with at least one programming language: C++, Smalltalk, Java,

4. a query language adapted to the object model.

Because of the great diversity of the OODBMS, the Object Data Management Group (ODMG) [11] has been created in the early 1990 whose aim is to specify and promote a standard for the OODBMS. ODMG gathers the main actors of the OODBMS. ODMG standard (currently version 2.0) specifies mainly the following points:

1. an object model,

2. an object definition language (ODL),

3. an object query language (OQL),

4. C++, Java and Smalltalk language bindings.

ODMG specifies several level of compliance: object model compliance, ODL compliance, OQL compliance, C++ binding compliance and so on.
Several product databases are currently partially compliant with the ODMG standard: O2[2], ObjectStore [10], POET [24], VERSANT [12], ONTOS [4], Objectivity [16], EYEDB [30] and so on.

2.3. The EYEDB OODBMS

The EYEDB OODBMS [30] is an OODBMS based on the ODMG concepts. It is currently used in several projects related to genetics and molecular biology and undergoes testing in several locations, including industrial companies. Online information and a trial version of EYEDB can be obtained from http://www.eyedb.com.

The key features of the EYEDB OODBMS are

- **standard OODBMS features** [2, 26, 14]: persistent typed data management; client/server model; transactional services; recovery system; expressive object model; inheritance; integrity constraints; methods; triggers; query language; application programming interfaces,
- **language orientation**: a definition language based on the ODMG [11] Object Definition Language (ODL); a query language based on the ODMG Object Query Language (OQL); C++ and Java bindings; PHP and PERL bindings,
- **genericity and orthogonality of the object model**: inspired by the SmallTalk, LOOPS, Java and ObjVlisp object models (i.e. every class derives from the class `object` and can be manipulated as an object); type polymorphism; binary relationships; literal and object types; transient and persistent objects; method and trigger overloading; template-based collections (set, bag and array); multi-dimensional and variable size dimensional arrays,
- **support for data distribution**: CORBA binding; multi-database objects,
- **support for large databases**: databases up to several Tb (terabytes),
- **efficiency**: database objects are directly mapped within the virtual memory space; object memory copies are reduced to the minimum; clever caching policies are implemented,
- **scalability**: programs are able to deal with hundred of millions of objects without loss of performance.

2.4. Distributing CORBA Views from an OODBMS

2.4.1 What is a CORBA View?

A database CORBA view (named CORBA view for shortcut) allows the user to manipulate database objects through an ORB. A CORBA view is mainly composed of an interface definition (using the IDL language) and an implementation of this IDL dealing with an OODBMS.
We call this a view by analogy to the well known relational database management systems (RDBMS) view concept, but there are some conceptual differences between an RDBMS and a CORBA view.
An RDBMS view is a table-oriented exportation of the RDBMS content through an SQL statement: a relational view denotes a set of rows on which queries can be performed.
A CORBA view is essentially an instance-oriented interface to the DBMS: objets are built on the fly by the CORBA server and one cannot restrict the extension of a class, every instance in the DBMS from a mapped class will be accessible through the interface. Differently said, the production rules are separated from the view and reside in factories, where the filtering is done if needed. As a consequence, no query is possible in CORBA views.

The main other differences between are:

1. the CORBA view allows for the distribution of database objects,
2. the CORBA view allows for the manipulation of object in different databases with different schemas,
3. the CORBA view allows read and write access to the database objects,
4. the CORBA view does not depend on the DBMS used,
5. the creation of a CORBA view is done outside the DBMS.

To define and distribute CORBA views from an OODBMS, we provide two approaches whose spirit differ fundamentally. The first approach is driven by the OODBMS source schema, while the second is driven by the target CORBA view defined in IDL. The first approach is named *source schema driven* and the second one is named *target view driven*.

2.4.2 Source Schema driven CORBA views

From a database schema, one defines, using the IMDL language, hints about the view to be built, for example:

- class, attribute or method visibility restrictions,

- class, attribute or method renaming,
- interface attribute or method mapping to an OQL construct,
- interface attribute or method mapping to a C++ construct,
- addition of attributes or methods in a target interface.

In this case, as shown in Figure 1, the starting point is:

1. a database schema,
2. hints about the view to be built expressed as an IMDL construct.

the result is:

1. a generated IDL,
2. a full or partial CORBA implementation.

2.4.3 Target View driven CORBA views

In this case, the target view is given a priori in IDL. From a database schema, one defines, using IMDL, the way to map the target IDL from the source schema.

In this second case, as shown in Figure 2, the starting point is:

- a database schema,
- an IDL,
- hints about the way to map the target view from the input schema expressed as an IMDL construct.

the result is:

- a full or partial CORBA implementation of the IDL.

The mapping hints are expressed using IMDL and the CORBA view is generated using IMS services.

3. CORBA view distribution: Actors of the solution

3.1. The Interface Mapping Definition Language (IMDL)

IMDL is a strict superset of the OMG IDL [17, 18]. A few constructs have been added to specify the mapping from the database schema to the IDL view. These constructs allow the user to give the following directives to IMS compiler:

1. to hide database classes, database class attributes or methods in the target CORBA view,
2. to rename database classes, database class attributes or method in the target CORBA view,
3. to extend database classes in the CORBA view by adding new attributes or methods in the target CORBA view,
4. to map an IDL interface attribute from a specific database class attribute,
5. to map an IDL interface attribute from an Object Query Language (OQL) construct,
6. to map an IDL interface attribute from a C++ expression or C++ code.
7. to map an IDL interface method from a database method,
8. to map an IDL interface method from an OQL construct,
9. to map an IDL interface method from a C++ expression or C++ code.

Note that IMDL does not depend on the OODBMS nor on ORB used: as soon as the OODBMS provides a data description language and a query language, IMDL can be used.

3.2. The Interface Mapping Services (IMS)

The main actor of IMS is the IMS compiler which generates an IDL and a full or partial CORBA implementation for a given ORB from a database schema and an IMDL construct (Figure 1) or which generates a full or partial CORBA implementation from a target IDL, a database schema and an IMDL construct (Figure 2). The IMS compiler may generate also a standard interface factory to build instances of any generated interface.
The IDL generated with IMS does not depend on the OODBMS nor on the ORB used; but the generated C++ code depends on both the OODBMS and the ORB. IMS has been implemented for the `Orbix` and `Orbacus` ORBs and for the EYEDB [27, 30] OODBMS.

3.3. CORBA View Runtime Architecture

The main actors of the runtime architecture of a CORBA view are (Figure 3):

1. the clients of the CORBA view using the services defined in the IDL,

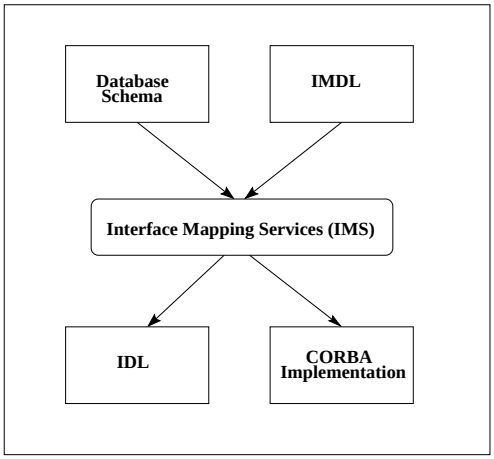

Figure 1. Principles of a source schema driven CORBA view

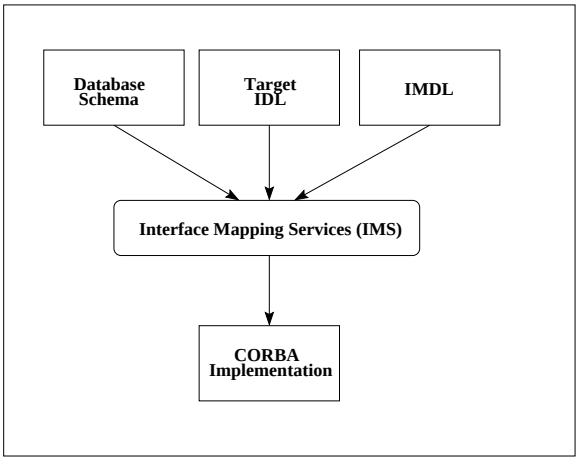

Figure 2. Principles of a target IDL driven CORBA view

2. the Object Request Broker,
3. the server implementing the CORBA IDL; the clients dialog with this server through the ORB layer; this server is a client of the OODBMS server,
4. the OODBMS server.

Clients manipulate CORBA objects while the server manipulates both CORBA and OODBMS objects. More precisely, there are three kind of objects: (Figure 3):

1. The CORBA objects, instances of the IDL interfaces: for each client CORBA object, there is one `corba object` in the server side which is the implementation of the client `corba object`. Those objects are denoted as client and server `corba object` s.
2. The OODBMS runtime objects, instances of the ODL classes, bound to the database objects. There is one `oodbms object` in the server side for each `oodbms object` in the client side. Those objects are denoted as client and server `oodbms object` s.
3. The OODBMS database objects residing in a database. Those objects are denoted as `database object` s. Note that the `oodbms object` s and the `database object` s are tightly linked together through the OODBMS runtime layer.

A `corba object` *cobj* is said to be mapped from an `oodbms object` *iobj* when:

1. at least one attribute selection or modification on *cobj* refers to *iobj* or
2. at least one method invocation on *cobj* refers to *iobj*.

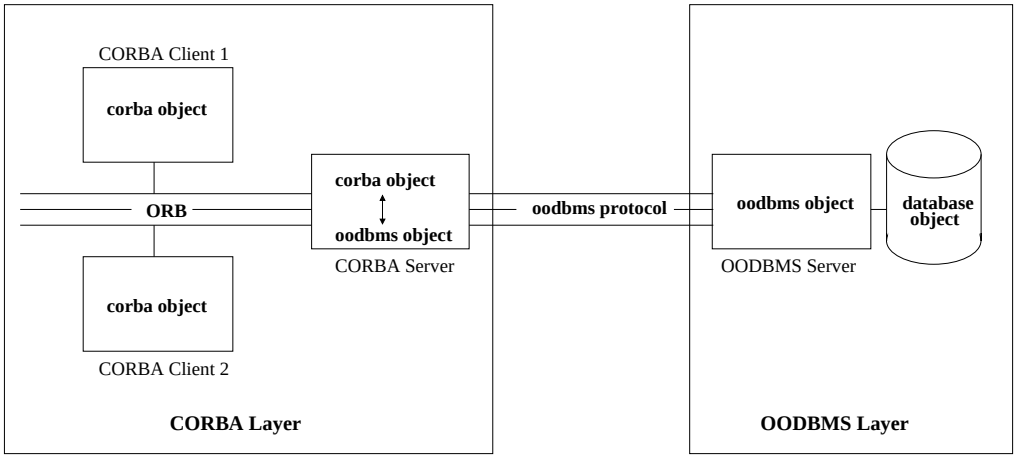

Figure 3. CORBA View Runtime Architecture

4. Designing a CORBA View from an OODBMS

As presented previously, to design a CORBA view from an OODBMS, one needs to define the mappings from the database classes to the target CORBA interfaces using the IDML language. The IMS compiler then generates the CORBA implementation for a given ORB and OODBMS from the IMDL construct.

To illustrate this paper and give more details on the principles of IMS, let's consider a simple schema that is described here using the EYEDB Object Definition Language (ODL) close to the ODMG ODL [11].

4.1. A Simple Schema

```
enum CivilState {
  Lady = 0x10,
  Sir = 0x20,
  Miss = 0x40
};

class Address {
  attribute string street;
  attribute string<32> town;
};

class Person {
  attribute string name;
  attribute int age;
  attribute Address addr;
  attribute CivilState cstate;
  relationship Person * spouse
    inverse Person::spouse;
  attribute set<Car *> * cars;
  attribute Person *children[];
};

class Car {
  attribute string brand;
  attribute int num;
};

class Employee extends Person {
  attribute long salary;
};
```

This ODL construct defines one enumerated type, `CivilState`, and four classes, `Address`, `Person`, `Car` and `Employee`. The class `Adress` is composed of a variable size literal attribute, `street`, and one fixed size literal attribute `town`. As specified by the ODMG, a literal attribute value has no identifier, while an object attribute value has an identifier.

The class `Person` is composed of four literal attributes, `name`, `age`, `addr`, and `cstate`, a relationship object attribute, `spouse`, a collection object attribute, `cars`, and a variable size object attribute array, `children`. Note that as the `spouse` attribute is a relationship, the OODBMS maintains its referential integrity. This means that if an object that participates in a relationship is removed, then any traversal path to that object is also removed.

As the object attributes `cars` and `children` are not relationships, the OODBMS does not maintain their referential integrity. Such a unidirectionnal reference is called an object-valued attribute.

4.2. Source Schema driven study

4.2.1 Creating a CORBA view with IMDL

From the previous database schema, we want to build a CORBA view where several attributes are hidden, some oth-

ers are renamed and some methods are created; for example, we want:

- to hide the following attributes:
  - `cstate` and `children` in the class `Person`,
  - `street` in the class `Address`.
- to rename the following attributes:
  - `name` in the class `Person` renamed as `lastname`,
  - `age` in the class `Person` renamed as `how_old_is_he`
- to add an updatable attribute in the class `Person` named `spname` mapped from the `name` of the `spouse` of the calling instance of `Person`,
- to add a readonly attribute in the class `Person` named `same_age_person_list` which gives the list of the instances of `Person` which have the same age as the calling instance of `Person`.
- to add a method in the class `Employee` named `is_he_rich()` which returns `true` if the salary of the instance of `Employee` is greater than a constant amount.
- to add an updatable attribute in the class `Employee` named `euro_salary` mapped from the `salary` attribute converted to the *euro* currency,

IMDL and IMS allow the user to implement such a CORBA view in a quite automatic and simple way. The user only has to define hints in IMDL and to invoke IMS compiler to generate both the target IDL and the CORBA implementation of that IDL for `Orbix` or `Orbacus`.
Using IMDL, the description of the hints for the CORBA view proposed above is as follows:

```
module People {

  implicit *;

  hide Person::cstate,
       Person::children,
       Address::street;

  rename Person::name to Person::lastname;
  rename Person::age to
                  Person::how_old_is_he;

  typedef sequence<Person> PersonList;

  extend Person {
   map attribute spname
    from spouse.name;
   map readonly attribute
     PersonList same_age_persons
     from %oql{select x from Person x where
                x.age = this.age; %};
  };

  extend Employee {
   map boolean is_he_rich() from
       expr("return salary() > 10000");
   map attribute euro_salary from
       // get C++ expression
       expr("salary() / 6.55957")
       : // token delimiter
       // set C++ expression
       expr("salary(euro_salary * 6.55957)");
  };
};
```

Let's explain each detail of the previous IMDL construct:

- this IMDL construct is composed of an IDL module, named `People`, in which view hints are defined.
- the first statement within the module declaration is `implicit *`. This statement means that all the database schema types defined in the input ODL will be mapped in an implicit way. An implicit mapping for an ODL class is a one-to-one mapping:
  - the ODL class has a corresponding generated IDL interface with the same name,
  - each of its attributes (resp. methods) has a corresponding attribute (resp. method) within the generated IDL interface with the same name and the corresponding type (resp. signature).

  For example, the implicit mapping of the ODL class `Address` is:

  ```
  interface Address {
    attribute string street;
       // mapped from Address::street
    attribute string town;
       // mapped from Address::town
  };
  ```

  The directive `implicit *` ensures a one-to-one mapping between each ODL class and each generated IDL interface. Nevertheless, the IMDL directives `hide`, `rename` and `extend` may be used to alter this implicit mapping.
- the `hide` directive allows the user to hide an ODL class, an ODL class attribute or method: this means that the pointed class, attribute or method will not be present in the generated IDL. In our example, three attributes are hidden in the generated IDL view:

Person::cstate . Person::children and Address::street . This means that the previously shown IDL interfaces becomes:

```
interface Address {
  attribute string town;
};

interface Person {
  attribute string name;
  attribute long age;
  attribute Address addr;
  attribute Person spouse;
  attribute CarList cars;
  // ...
```

- the rename directive allows the user to rename an ODL class, an ODL class attribute or method: this means that the pointed class, attribute or method will be renamed in the generated IDL. In our example, the interface Person becomes:

```
interface Person {
  attribute string lastname;
     // mapped from Person::name
  attribute long how_old_is_he;
    // mapped from Person::age
    ...
```

- the extend directive allows the user to extend an existing mapping by adding some attributes or methods to it. In our example, an attribute named same_age_persons of type sequence of Person is added to the interface Person and a method named is_he_rich() is added to the interface Employee .

- the map from IMDL directive allows the user to map an IDL attribute or method from one of the following constructs:

  1. an ODL attribute,
  2. an OQL construct,
  3. a C++ expression,
  4. some C++ code.

  This example introduces the first, the second and the third mapping:

1. mapping from an attribute:

   As we have seen previously, the attributes of the Person IDL interface are mapped from attributes of the ODL Person class thanks to the implicit * and rename directives. These view attributes are updatable as they are directly mapped to well identified mono valued database attributes.

   The attribute spname is also mapped from an attribute, more exactly from an attribute path expression (path expression for shortcut): spouse.name . A path expression is a sequence of attribute names separated by dots. It can also contain array modifiers under the form [const_int_expr] . For instance, the following path expression mapping construct is valid:

```
map attribute long xage from
  spouse.children[(2+3)>>1].spouse.age;
```

   Note that an IDL attribute mapped from an ODL path expression is always updatable unless it has the readonly qualifier.

2. mapping from an OQL construct:

```
map readonly attribute
   PersonList same_age_persons
   from %oql{select x from Person x
             where x.age =
               this.age; %};
```

   This construct means that the same_age_persons attribute is assigned to the evaluation of the OQL statement select x from Person x where x.age = this.age where this denotes the corresponding database object. For instance, if the age of the database person object is 32 , the executed OQL statement select x from Person x where x.age = 32 will return the list of persons in the database whose age is equal to 32 , including of course, the calling person; this returned list will be bound onto the attribute same_age_persons .

   Contrary to the mapping from an attribute, such an attribute is not updatable for two reasons:

   - this attribute is readonly . Although this reason is sufficient to forbid the update, this is not the structural reason.
   - the structural reason is that this attribute is mapped from an arbitrary OQL construct (in this case a select statement) and the system is not able to convert automatically any OQL statement to a “reverse” update statement. Furthermore, even if it will be able, the update of all the person instances in the list is perharps not wanted.

   To allow the user to update the same_age_persons attribute, one has to give explicitely the OQL construct for update in the IMDL. For example:

```
map attribute
  PersonList same_age_persons
```

```
from // the get OQL construct:
 %oql{ select x from Person x where
         x.age = this.age; %}
: // token delimiter
// the set OQL construct:
%oql{ for (x in
            (select x from Person x
             where x.age = this.age))
        x.age := same_age_persons; %}
```

Note that the IMS does not perform any semantic control of the update function: you can do whatever you want in the update function.

3. mapping from a C++ expression:

```
map boolean is_he_rich() from
  expr("return salary() > 10000");
```

This construct means that the `is_he_rich()` method is assigned to the evaluation of the C++ expression `salary() > 10000` Note that the function call `salary()` returns the `salary` attribute value of the calling instance. So, if the `salary` attribute value is greater than `10000` , the calling instance is said to be a rich person instance, and poor otherwise.

The `euro_salary` is an attribute mapped from a C++ expression: it is an updatable attribute as a C++ expression for update has been explicitely given in the IMDL as follows:

```
map attribute euro_salary from
    // get C++ expression
    expr("salary() / 6.55957")
    :
    // set C++ expression
    expr("salary(euro_salary * 6.55957)");
```

The `euro_salary` attribute is mapped from the evaluation of the C++ expression `salary() / 6.55957` for reading and from `salary(euro_salary * 6.55957)` for writing.

4.2.2 The generated IDL

The complete generated IDL is as follows (forward declaration are skipped):

```
#include <eyedb/corba/eyedb.idl>

module People {
  typedef sequence<People::Car>
     CarList;
  typedef sequence<People::Person>
     PersonList;
  typedef sequence<People::Address>
     AddressList;
  typedef sequence<People::Employee>
     EmployeeList;

  enum CivilState {
    Lady,
    Sir,
    Miss
  };

  interface Address : EyeDB_ORB::idbStruct {
    attribute string town;
  };

  interface Person : EyeDB_ORB::idbStruct {
    attribute string lastname;
    attribute long how_old_is_he;
    attribute string spname;
    attribute People::Address addr;
    attribute People::Person spouse;
    attribute People::CarList cars;
    readonly attribute
       People::PersonList same_age_persons;
  };

  interface Car : EyeDB_ORB::idbStruct {
    attribute string brand;
    attribute long num;
  };

  interface Employee : People::Person {
    attribute People::CivilState cstate;
    attribute People::PersonList children;
    attribute long salary;
    attribute euro_salary;
    boolean is_he_rich();
  };

  interface Factory {

    People::Address AddressQueryFirst
      (in EyeDB_ORB::idbDatabase db,
       in string query);

    People::AddressList AddressQuery
      (in EyeDB_ORB::idbDatabase db,
       in string query);

    People::Address asAddress
      (in EyeDB_ORB::idbObject o);

    People::Address AddressCreate
      (in EyeDB_ORB::idbDatabase db);

    // similar methods for Person,
    // Car and Employee
  };
};
```

This generated IDL calls for a few remarks:

1. as previously introduced, the generated IDL includes by default the file `eyedb.idl` which contains the generic interface of EYEDB. Each generated interface inherits directly or indirectly from the `EyeDB_ORB::idbObject` interface.

2. by default, an interface factory is generated which contains a few methods for each generated interface:

   (a) the method `Address AddressQueryFirst(db, query)` returns the first `Address` instance which matches the input OQL `query` argument.

   (b) the method `AddressList AddressQuery(db, query)` returns all the instances of `Address` which matches the input OQL `query` argument.

   (c) the method `Address asAddress(EyeDB_ORB::idbObject o)` builds an `Address` instance from a generic `idbObject` instance if and only if the generic instance is of dynanic type `Address`. Otherwise a null object is returned.

   (d) the method `Address AddressCreate(db)` creates an empty runtime `Address` instance.

4.2.3 Using the generated CORBA view

To use the CORBA view, the user must:

1. generate the CORBA view by giving the database schema and the IMDL construct to the IMS compiler.

2. generate the CORBA stubs and skeleton by giving the generated IDL to the ORB IDL compiler,

3. compile all the generated code,

4. write and compile the client programs.

Here is a simple example of a client program to get, display and update all the instances of the class `Employee`:

```
// making a OQL query
EyeDB_ORB::idbObjectSeq_var obj_seq =
  db->queryObjects("select Employee");

for (int i = 0; i < obj_seq->length(); i++) {
  // building an IDL SpecialEmployee from
  // an ODL Employee
  People::Employee_var empl =
    person_factory->asEmployee(obj_seq[i]);
   // displaying Employee attributes
  cout << "Employee #" << i << endl;
  cout << "\tname = '" << empl->lastname()
       << "';\n";
  // making this Employee rich
  while (!empl->is_he_rich())
    empl->salary(empl->salary() + 100)
  cout << "\tsalary = " << empl->salary()
       << ";\n";
  cout << "\teuro_salary = " <<
          empl->euro_salary()
       << ";\n";
  // incrementing his age
  empl->age(empl->age() + 1);
  cout << "\tage = " << empl->age()
       << ";\n";
}
```

And to create an instance of `Person` and store it in the database:

```
// creating a Person instance
People::Person_var p =
   person_factory->PersonCreate(db);

// setting the person lastname
p->lastname("martin");

// setting martin's age
p->how_old_is_he(32);

// storing martin into the database
p->store();
```

5. Related work and approaches

There have been other proposals for view support in the context of OODBMS [1, 21, 22, 3].

The O2 approach [1] is based on the concept of *virtual class*: a *virtual class* is populated with objets selected through a query in the OQL language on the root class extension. Part of the information visible in the original base class may be hidden in the view and some renaming may also take place. Furthermore, virtual attributes, defined by OQL constructs, may be added to the view. This view mechanism (O2 Views) is based on the query language - at least to define the extension of the virtual class - and on a specific view language to add virtual attributes, hide or rename attributes in the original base class. A mechanism for updating views in OODBMS, implemented on top of O2 Views, is exposed in [3].

Another view approach, implemented on top of the COCOON model, is presented in [22] and [21]: contrary to the O2 Views approach, this view system uses the standard way of defining views by nothing else than query language expressions.

In those both approaches, as in the traditionnal RDBMS view approach and contrary to our approach, the view gathers "production" and "filtering" rules:

- the "production" rules, expressed in the query language, are used to select objets from one or several classes.
- the "filtering" rules are used to filter objets by, for instance, adding virtual attributes, hiding or renaming attributes in the original base class. The filtering rules are expressed in the query language or in a specific language depending on the system.

In our approach, the production rules are separated from the filtering rules: the production rules "reside" in factories while the filtering rules "reside" in the view interface. The production mechanism is orthogonal to the filtering (or view) mechanism: one produces an object using a factory and then one applies the filter (view) to this object, keeping its original database identifier.
About this point, our approach is conceptually more similar to *aspects* [19] which provides mechanisms to extend existing objects with new state and new behavior while maintaining the same object identity.

On the other hand, our approach for the update of views is close to the O2 one described in [3]. In the both approaches:

- one keeps the original database identifier (OID) in the "viewed" object: this is the core of the update process.
- some attributes can be straightforwardly updated: O2 allows update of any attribute for which the reverse mapping method can be automatically derived from the mapping expression. Our approach allows update of any attribute mapped from an ODL path expression, which is, in a certain sense, more restrictive than the O2 approach.
- the user must supply update methods for other attributes: in our approach, the user must supply an update method for the attributes mapped from an OQL or a C++ expression.

Lastly, our approach has a structural difference with the previous ones: our view management is done "outside" the DBMS, thus, our system is structurally more independent from the DBMS used than the other systems and it can be - more or less easily - ported to other DBMS included relational DBMS. Futhermore, the choice of CORBA as our view architecture made our "viewed" objects easily and standardly distributable.

6. Conclusion

We have defined a powerful language, IMDL, and tools named IMS that enable the distribution of CORBA views from an OODBMS. As opposed to the traditional RDBMS views, our approach allows one to customize with flexibility the CORBA views, and to offer several different views or to map a database schema onto a pre-existing IDL definition. Another important improvement on traditional views lies in a mechanism that enables an update access to the objects mapped in the view.

The IMDL language and IMS service were used at Infobiogen to develop a standard interface to a database of human genome maps: HuGeMap [8, 6]. This database was built with the EyeDB OODBMS. A common IDL for genome maps was defined as a consensus by the genome community [7] and HuGeMap had to offer a CORBA view implementing this IDL. This target IDL consists of 19 interfaces that were mapped to the database schema in a 170 line IMDL file. The resulting generated implementation of the CORBA server contains about 1,500 lines of code. Only two man-days were spent to implement this CORBA view using the IMDL language and IMS compiler. Without this tool, the realization of such an implementation is estimated to take about two man-weeks.

IMS was implemented for the EYEDB OODBMS and the `Orbix` and `Orbacus` ORBs. But IMDL is a language that does not depend on the ORB or on the DBMS. The concepts and the language presented here can be used with any ORB and any DBMS, relational or object-oriented.

We now plan to extend IMDL and IMS to allow for the definition and implementation of a single and integrated CORBA view of several databases with different schemas. This will achieve a real interoperation of multiple databases with heterogeneous and complementary semantics.

More information on the IMDL language, the Interface Mapping Services and the EYEDB OODBMS can be found at the EYEDB home page [27, 30]. This page contains the full online programming manual, links to related publications and a trial version for Solaris can be downloaded. The page `http://www.eyedb.com/corba_views` [29] encloses material related to this paper: the IMDL grammar, the source schema driven example given in this paper, a target view driven example, the common IDL for genome maps and the IMDL file used to map the HuGeMap database to this IDL.

7. Acknowledgements

The Interface Mapping Definition Language (IMDL) and the Interface Mapping Services (IMS) have been developped at CRI Infobiogen [25] with funding from the European Commission (BIO4-CT96-0346).

The EYEDB OODBMS has been developped at Sysra [28] with partial funding from the Agence National de la Valorisation de la Recherche (ANVAR) [5] and the Conseil Regional d'Ile de France [13].

References

[1] S. Abiteboul and A. Bonner. Objects and views. pages 238–247, 1991.

[2] M. Adiba and C. Collet. *Objets et bases de données, le SGBD O2*. Hermès, 1993.

[3] S. Amer-Yahia, P. Breche, and C. S. dos Santos. Object views and updates.

[4] T. Andrews and all. *The ONTOS Object Database*. Ontologic, Inc, Burlington, Massachusetts, 1989.

[5] ANVAR. L'Anvar, votre partenaire pour l'innovation. http://www.anvar.fr/, 1998.

[6] E. Barillot. The Hugemap Home Page. http://www.infobiogen.fr/services/Hugemap/, 1998.

[7] E. Barillot, U. Leser, P. Lijnzaad, C. Cussat-Blanc, K. Jungfer, F. Guyon, G. Vaysseix, C. Helgesen, and P. Rodriguez-Tomé. A proposal for a CORBA interface for genome maps. *BIOINFORMATICS*, 15, 1999.

[8] E. Barillot, S. Pook, F. Guyon, C. Cussat-Blanc, E. Viara, and G. Vaysseix. The HuGeMap database: Interconnection and Visualisation of Human Genome Maps. *Nucleic Acids Research*, 27:119–122, 1999.

[9] R. Ben-Natan. *CORBA a Guide to Common Object Request Broker Architecture*. Computing McGraw-Hill, 1995.

[10] C. Lamb et al. The objectstore database system. *Communications of the ACM, 34(10)*, pages 50–63, 1991.

[11] C. G. Cattell and al. *Object Database Standard, ODMG 2.0*. Morgan Kaufmann, 1997.

[12] V. Corporation. Versant Corporation. http://www.versant.com/.

[13] CRIF. Conseil Régional d'Ile de France. http://www.cr-ile-de-france.fr/, 1998.

[14] H. F. Korth and A. Silberschatz. *Database system concepts*. MacGraw-Hill, 1991.

[15] T. J. Mowbray and R. Zahavi. *The Essential CORBA*. John Wiley & Sons, Inc., 1995.

[16] Objectivity. Welcome to objectivity. http://www.objectivity.com/.

[17] OMG. Object Management Group Home Page. http://www.omg.org/, 1997.

[18] A. L. Pope. *The CORBA Reference Guide*. Addison Wesley, 1998.

[19] J. Richardson and P. Schwarz. Aspects: Extending objects to support multiple, independent roles. pages 298–307, 1991.

[20] D. H. Robert Orfali and J. Edwards. *The Essential Distributed Objects, Survival Guide*. John Wiley & Sons, Inc., 1996.

[21] M. H. Scholl, C. Laasch, and M. Tresch. Updatable Views in Object-Oriented Databases. (566), 1991.

[22] M. H. Scholl and H.-J. Schek. Supporting views in object-oriented databases. *Data Engineering Bulletin*, 14(2):43–47, 1991.

[23] J. Siegel. *CORBA Fundamentals and Programming*. John Wiley & Sons, Inc., 1996.

[24] P. Software. Data management for the Internet Age. http://www.poet.com/.

[25] I. Staff. The INFOBIOGEN Home Page. http://www.infobiogen.fr/, 1998.

[26] M. Stonebraker and J. M. Hellerstein, editors. *readings in database systems*. Morgan Kaufmann, 1998.

[27] E. Viara. The EYEDB Home Page. http://www.eyedb.com/, 1998.

[28] E. Viara. The SYSRA Home Page. http://www.sysra.com/, 1998.

[29] E. Viara. Material for the article *Distributing CORBA Views from an OODBMS*. http://www.eyedb.com/corba_views/, 2001.

[30] E. Viara, E. Barillot, and G. Vaysseix. The EYEDB OODBMS. *IEEE publications*, 1999. International Database Engineering and Applications Symposium (IDEAS), Montreal, 2-4 August 1999.