A UML/OCL framework for design of mediated data federations

Balsters, H.

A UML/OCL Framework for Design of Mediated Data Federations

H. Balsters
University of Groningen
Faculty of Management and Organization
P.O. Box 800
9700 AV Groningen, The Netherlands
h.balsters@bdk.rug.nl

SOM theme A Primary processes within firms

Abstract

This paper describes a general semantic framework for precise specification of so-called mediating systems; such systems provide for tight coupling on a global level of a collection of heterogeneous component databases to a federated database. A mediating system maps in a uniform and systematic manner the underlying database schemas of the component systems to a separate, newly defined integrated database schema. This integrated database is completely virtual, and will constitute the actual federated database. That is, queries posed against the federated system will be posed against this virtual integrated database; these global queries will then be mapped by the mediator to actual local queries against the existing (legacy) component databases. Our approach is based upon the UML/OCL data model. UML is the *de facto* standard language for analysis and design in object-oriented frameworks, and is being employed more and more for analysis and design of information systems, in particular information systems based on databases and their applications. Database specifications often involve specifications of constraints, and the Object Constraint Language (OCL) - as part of UML - can aid in the unambiguous modelling of database constraints. One of the central notions in database modelling and in constraint specifications is the notion of a database view; a database view closely corresponds to the notion of derived class in UML. We will employ OCL and the notion of derived class as a means to treat database constraints and database views in a federated context. The paper will demonstrate that our particular mediating system integrates component schemas without loss of constraint information. Furthermore, we will discuss a UML/OCL representation of relational databases.

Keywords Data modeling, database design, federated databases, mediators, interoperable databases, database views, query languages, constraint integration, UML, derived classes, OCL, UML/OCL representation of relational databases

(also downloadable) in electronic version: http://som.rug.nl/

1. Introduction

Modern information systems are often distributed in nature. Data and services are often spread over different component systems wishing to cooperate in an integrated setting. Cooperation of component systems in one integrated information system is becoming more and more important since information is often spread over different databases in one organization (or even spread over different organizations). Such information systems involving integration of cooperating component systems are called *federated information systems*; if the component systems are all databases then we speak of a federated database system (FDB). In current applications, there is more and more a tendency *not* to develop stand-alone, monolithic database systems; rather, the tendency is to employ existing (legacy) components by letting them work together in a single integrated environment. This tendency to build integrated, cooperating systems is often encountered in applications found in EAI (Enterprise Application Integration), which typically involve several, usually autonomous, component (data and service repositories) systems, with the desire to query and update information on a global, integrated level. In this paper we will address the situation where the component systems are so-called legacy systems; i.e. systems that are given beforehand and which are to interoperate in a integrated single framework in which the legacy systems are to maintain as much as possible their respective autonomy.

A major obstacle in designing interoperability of legacy systems is the heterogeneous nature of the legacy components involved. This heterogeneity is caused by the design autonomy of their owners in developing such systems. Legacy systems were typically designed to support local requirements, under constraints imposed by local rules, and often without taking into account any future cooperation with other systems. To address the problem of interoperability the term *mediation* has been defined [Wie95]. A database federation can be seen as a special kind of mediation, where all of the data sources are (legacy) databases, and the mediator offers a mapping to a (virtual) DBMS-like interface. This interface offers the application the possibility to approach the federation via this integrated virtual database, which offers the user the illusion that he is interacting with an actual homogeneous, monolithic database. The mediator then maps queries against this virtual integrated database on to actual component databases. In our paper we will consider a tightly-coupled approach to database mediation, in which a global integrated schema of the federation is maintained, which can be accessed by a global query language. We base our notion of querying on the "Closed World Assumption" (CWA, [Rei84]), where the integrated database is to hold -in some manner- the "union" of the data in the underlying component databases. Central theme in our approach is that the integrated database on the federated level is completely *virtual*. The user of the federated system is offered the illusion that he is working with a monolithic homogeneous database system, while in fact this system basically resembles an interface, mapping interactions on the federated level to actions on the existing local database components. More precisely, the federated database will consist of an integrated *database view* on top of the existing legacy database components. For an overview of work on the virtual approach to database federation, we refer to [Hull97].

We concentrate on problems concerning integration of component legacy schemas on the level of the mediator. Schema integration requires the definition of relationships between schema elements of component systems. Detection and definition of such relationships can be heavily complicated by so-called *semantic heterogeneity* [DKM93,GSC96]. Semantic heterogeneity refers to disagreement about the meaning, interpretation, or intended use of related data. It has been widely agreed upon that schema integration cannot be fully automated [ShL90], as this would require full knowledge of the semantics of the component schema elements. In order to tackle the problem of integrating semantic heterogeneity, we employ the UML/OCL data model. UML is the *de facto* standard language for analysis and design in object-oriented frameworks, and is being employed more and more for analysis and design of Information systems, in particular information systems based on databases and their applications. Database specifications often involve specifications of constraints, and the Object Constraint Language (OCL) - as part of UML - can aid in the

unambiguous modelling of database constraints. One of the central notions in database modelling and in constraint specifications is the notion of a *database view*, where a database view closely corresponds to the notion of derived class in UML. In this paper we will employ OCL and the notion of derived class as a means to treat database constraints and database views in a federated context. In [Bal02] it is demonstrated that the notion of derived class can be given a formal basis in OCL, and that derived classes in OCL have the expressive power of the relational algebra. Hence, OCL has the explicit power to emulate basic features of the relational query language SQL. The paper will demonstrate that our particular mediating system integrates component schemas without loss of constraint information; i.e., no loss of constraint information available at the component level may take place as result of integrating on the level of the virtual federated database.
This paper heavily exploits the concept of the so-called *homogenizing function* (first introduced in [BB01]). This function provides the necessary mapping from the (legacy) components to the virtual integrated database on the federated level, while adhering to the principle that no *integration loss* may take place. Furthermore, following the approach given in [Bal02], we have in principle a mapping of queries posed against a federated database (specified in terms of derived classes in UML/OCL) to SQL-code, thus providing the link to actual database implementations.

2. UML/OCL as a specification language for databases

Information systems, and in particular information systems based on databases and their applications, rely heavily on sound principles of analysis and design. This paper focuses on particular principles of analysis and design related to database applications. Following [BP98], we can state that object-oriented (OO) modelling can prove to be very beneficiary in (relational) database applications. A database is a permanent, self-descriptive repository of data stored in files. A database is self-descriptive in the sense that it not only contains the data, but also a description of the data structure, or *schema.* In databases, the data usually change rapidly, while the schema stays relatively static. A database management system (DBMS) consists of software managing access to the data. DBMSs provide generic functionality for a broad range of applications; one of the foremost features of a DBMS is the availability of a *query language* offering an interactive means for reading and writing data from the database. A relational database has data represented as tables, and a relational DBMS manages access to tables of data and associated structures in a highly effective and efficient manner. (Relational databases use SQL as a data manipulation language, and tables are called *relations* in SQL.) Relational database applications can benefit substantially from OO modelling. The OO paradigm provides a uniform framework for both the design of database code and programming code. Database and their applications can thus be developed in one and the same conceptual framework. In fact, one can say that integrating relational databases into object-oriented applications is state of the art in software development practice. OO data models offer high-level modelling primitives leading to clear and concise specifications of database schemas. A high-level description of a database schema in terms of an OO data model can easily be mapped to a relational database schema employed by a conventional relational DBMS [BP98]. Hence, the analysis and design stage of a (relational) database can be separated in a clear and meaningful fashion.

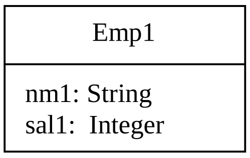
The most important OO modeling language is UML, being the de facto standard for OO analysis and design of information systems [OMG99]. Recently, researchers have investigated possibilities of UML as a modeling language for (relational) databases. [BP98] describes in length how this process can take place, concentrating on schema specification techniques. [DH99, DHL01] investigate further possibilities by employing OCL (the Object Constraint Language [WK99]) for specifying constraints and business rules within the context of relational databases. The idea is that OCL provides expressiveness in terms of relatively abstract set definitions that should prove to be sufficient to capture the general notion of (relational) database view. This idea of employing abstract object-oriented set definitions to captures views and constraints has also been pursued on the full level of object-oriented databases, be it not in the context of UML/OCL language, but rather in the context of an experimental OODB user language in combination with an underlying theoretical semantics [BBZ93, BV92]. In the more specific context of relational databases and OCL, [DH99] offer a framework for representing constraints within the relational data model. Some researchers take a very general approach investigating possibilities of UML/OCL; e.g., [AB01] treat OCL as a general query language for UML data

models, and [EP00] use OCL as a general language for business modeling. Current research, however, has not yet shown an effective way to deal with an important aspect of (relational) database modeling, namely modeling of so-called database views. A (database) view is a derived table (or derived relation, in SQL), meaning that a view does not exist as a physical relation; rather a view is defined by an expression much like a query [GUW02]. Views, in turn, can be queried as if they existed physically, and in some cases, we can even modify view content. That is, a user is offered the impression that a view is some base relation inside the database, but in fact it is a derived (or virtual) relation defined in terms of the actual base relations constituting the database. View definitions are an important asset in database applications, because users are usually only interested in a part of the database, and not in the complete underlying corporate database. Hence, it is important that users have access to that part of the database considered relevant for their category of database applications. Our application area for views is focused on Federated Databases, where legacy databases are to interoperate by employing a so-called mediating system. This mediating system can be considered as an integration of a set of certain database views defined on the component legacy database systems.
Database views and query languages are strongly related, since views basically are no more than named queries. [GR97] is one of the first papers to investigate the possibilities of a general query language for UML; further investigations can be found in [AB01] and [MC99]. [AB01] have attempted to demonstrate that OCL can offer the basis for a general query language for UML data models by showing how to represent Cartesian products and projections in OCL, thus paving the way to the claim that OCL has the same expressive power as the so-called relational algebra [D00, GUW02]. By demonstrating such a result, one could also claim to have a basis for representing views within OCL. In [Bal02] it is demonstrated that the expressiveness of OCL actually includes that of the relational algebra. This is done by showing how to offer the notion of *derived class* a formal basis within the framework of UML/OCL, and subsequently using this notion of derived class to represent the notions of Cartesian product and (relational) join. This result establishes that OCL includes the expressiveness of the relational algebra, without resorting to language extensions of OCL. Once it is established that OCL includes the

expressiveness of the relational algebra, then we also have provided a basis for representing the general notion of (relational) database view.
A derived class is a device for denoting a virtual class, defined in terms of already existing (base) classes (and possibly other derived classes). Views can be queried independently, with a semantics explained entirely in terms of queries on base classes. [Bal02] also offers a mapping to SQL-code [D00, GUW02], providing implementation support for our approach.
The paper ends with a short summary of our results.

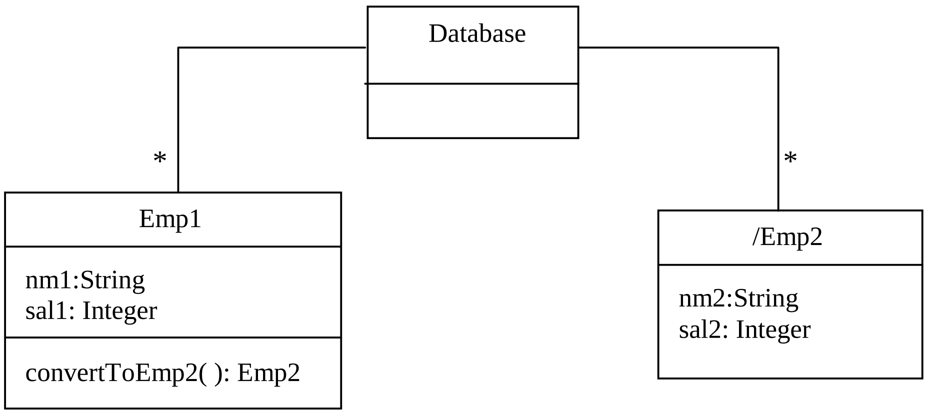
3. Basic principles: Databases and views in UML/OCL

Databases are basically a set of related tables. Tables in UML are represented by classes. Classes have attributes and corresponding domain values, while we can also have complex-valued attributes (i.e. non-first normal form) in UML by allowing for enumerated sets as domains for attributes, and to employ UML-style relations to represent directly references to other objects in tables without residing to foreign-key constructs (to indirectly enforce this kind of modelling facility). Views, as derived tables, can also be represented in UML, which we will describe below.
Let’s consider the case that we have a class called Emp1 with attributes nm1 and sal1, indicating the name and salary of an employee object belonging to class Emp1



Now consider the case where we want to add a class, say Emp2, which is defined as a class whose objects are completely derivable from objects coming from class Emp1. The calculation is performed in the following manner. Assume that the attributes of Emp2 are nm2 and sal2 respectively (indicating name and salary attributes for Emp2 objects), and assume that for each object e1:Emp1 we can obtain an object e2:Emp2 by stipulating that e2.nm2=e1.nm1 and e2.sal2=(2 * e1.sal1). By definition the total set of instances of Emp2 is the set obtained from the total set of

instances from Emp1 by applying the calculation rules as described above. Hence, class Emp2 is a *view* of class Emp1, in accordance with the concept of a view as known from the relational database literature. In UML terminology [BP98], we can say that Emp2 is a *derived class,* since it is completely derivable from other already existing class elements in the model description containing model type Emp1.
We will now show how to faithfully describe Emp2 as a derived class in UML/OCL in such a way that it satisfies the requirements of a (relational) view. First of all, we must satisfy the requirement that the set of instance of class Emp2 is the result of a calculation applied to the set of instances of class Emp1. The basic idea is that we introduce a class called Database that has associations to classes Emp1 and Emp2. A database object will reflect the actual state of the database, and the system class Database will only consist out of one object in any of its states. Hence the variable *self* in the context of the class Database will always denote the actual state of the database that we are considering. In the context of this database class we can then define the calculation obtaining the set of instances of Emp2 by taking the set of instances of Emp1 as input.



Note that we have used a prefix-qualification by adding a slash to Emp2 indicating that Emp2 is a derived class definition [BP98]. Moreover, we have added an operation, called convertToEmp2, meant to coerce an arbitrary Emp1-object to an Emp2-object. This operation can be defined by the following OCL-specification

```
context   Emp1::convertToEmp2( ): Emp2
post:     self.convertToEmp2.nm2 = self.nm1  and
          self.convertToEmp2.sal2 = (2*self.sal1)
```

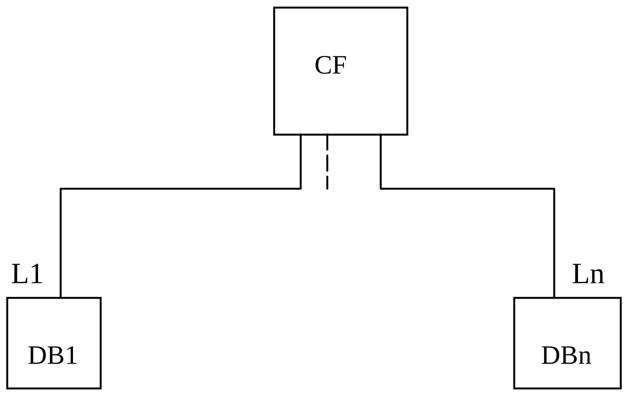
We now have all the ingredients necessary to specify the relation coupling the derived class Emp2 to the original class Emp1. This is done by including an invariant specification in the class Database telling us how to calculate the set of instances of Emp2 from the set of instances of Emp1

```
context  Database  inv:
self.Emp2 = self.Emp1→ collect(e:Emp1 | e.convertToEmp2) and
Emp1.allInstances = self.Emp1  and
Emp2.allInstances = self.Emp2
```

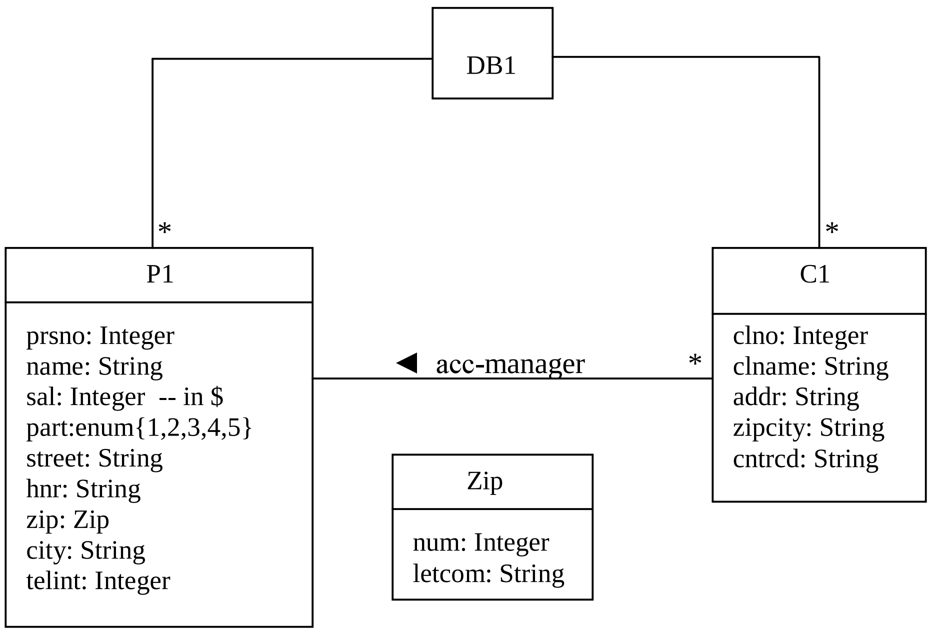
In this way we explicitly specify Emp2 as the result of a calculation performed on Emp1, and we also stipulate that the only Emp1- and Emp2-objects in the database are those obtained from the links starting from the database-object *self*.

4. Component frames

We can also consider a complete collection of databases by looking at so-called component frames, where each (labelled) component is an autonomous database system (typically encountered in legacy environments)



As an example consider a component frame consisting of two separate component database systems: the CRM-database (DB1) and the Sales-database (DB2):

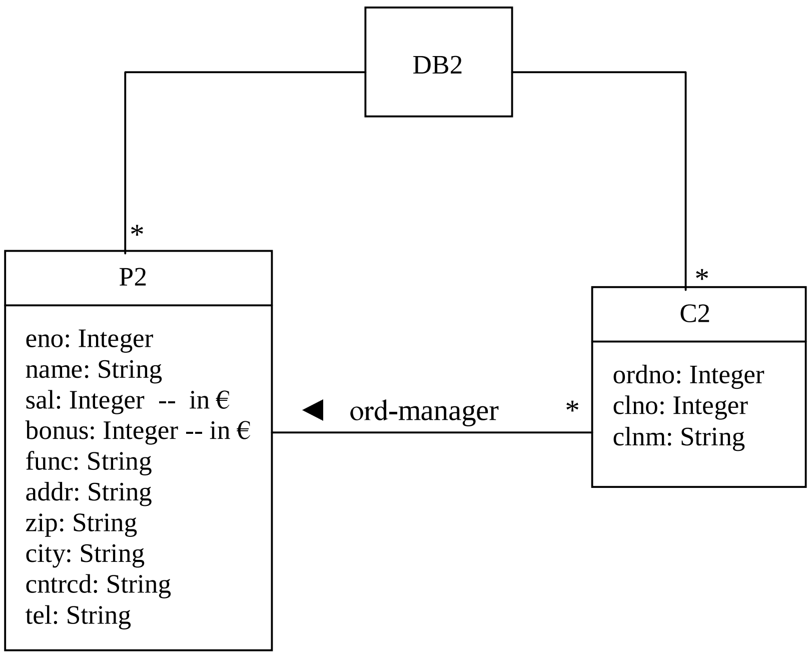


Constraints

```
context P1 inv:
P1.allInstances --> isUnique (p: P1 | p.prsno)
sal <= 1500
telint >= 1000  and  telint <= 9999

context C1 inv:
C1.allInstances --> isUnique (c: C1 | c.clno)
cntrcd.size <= 5
```

```
context Zip inv:
num >= 1000  and  num <= 9999
letcom.size = 2
```

The Sales-database: DB2

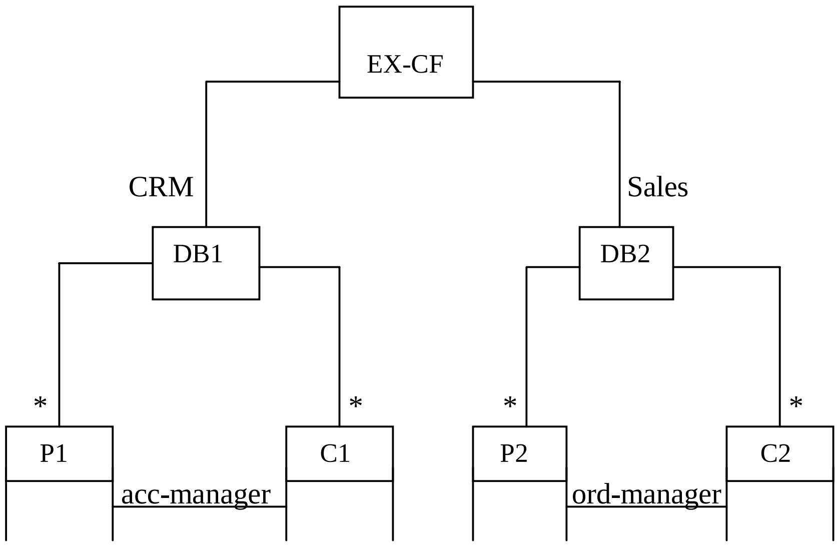


Constraints

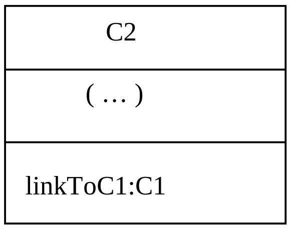
```
context P2 inv:
P2.allInstances --> isUnique (p: P2 | p.eno)
sal >= 1000
bonus >= 0
tel.size <= 16

context C2 inv:
C2.allInstances --> isUnique (c: C2 | c.ordno)
```

```
C2.allInstances   -->   forall(c:   C2   |   c.ord-manager.func   =
"Sales")
cntrcd.size <= 5
```

The example component frame EX-CF now combines the two database DB1 and DB2 into one component frame



The two databases DB1 and DB2 are –in the case of this example- related, in the sense that an order-object residing in class C2 is associated to a certain client-object in the class C1. On the component frame level, we can define an auxiliary function mapping a client-order object in class C2 to a client object in class C1. We do this by assuming an operation in the class C2, called linkToC1



with the following post conditions

```
context   C1::linkToC1( ): C1
post      self.linkTo.clno = self.clno
```

Since the attribute clno has unique values, the link from C2 to C1 is properly defined (assuming that there always exists a corresponding clno-value in the class C1 for each clno-value in the class C2).

5. Semantic heterogeneity; the integrated database DBINT

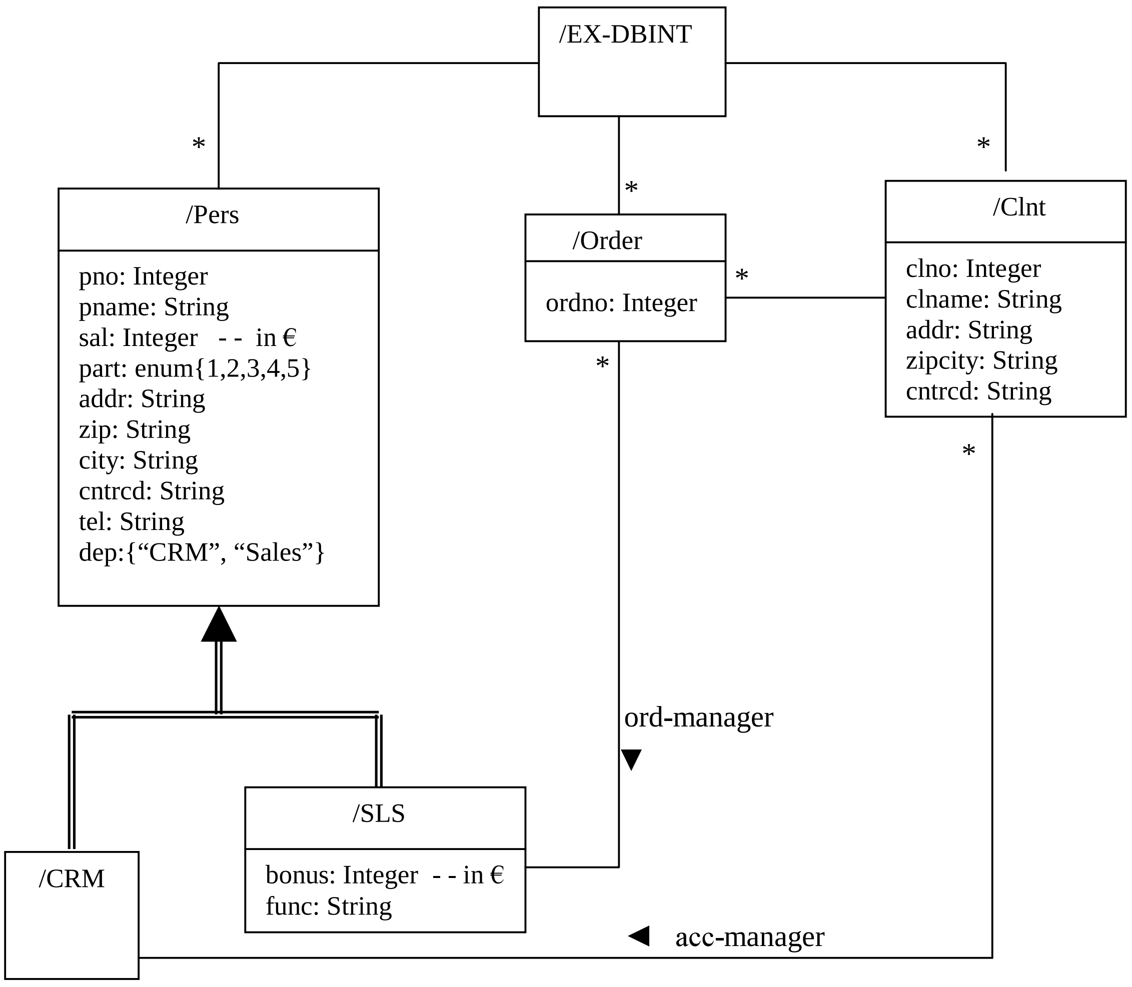
The problems we are facing when trying to integrate the data found in legacy component frames are well-known and are extensively documented (cf. [ShL90]). We will focus on one of the large categories of integration problems coined as *semantic heterogeneity* (cf. [Ver97]). Semantic heterogeneity deals with differences in intended meaning of the various database components. Integration of the source database schemas into one encompassing schema can be a tricky business due to

1. *renaming* (homonyms and synonyms)
2. *data conversion* (different data types for related attributes)
3. *default values* (adding default values for new attributes)
4. *missing attributes* (adding new attributes in order to discrimate between certain classes)

We will illustrate each of these cases in the context of our example databases. Important thing to know at this moment is that with **homonyms** we mean that certain names may at first sight- look the same (same syntax), but actually have a different meaning (different semantics). **Synonyms**, on the contrary, refer to certain names that are different in the sense that they have a different syntax, but that the actually mean the same (same semantics). Homonyms and synonyms occur extremely often in integration processes. In general, we will adopt the following solution to resolve these naming conflicts: different semantics call for different names, and equal semantics (intended meaning) call for equal names.

First consider our construction of a virtual database, represented in terms of a derived class in UML/OCL. (For an at length treatment of derived classes in UML/OCL we refer to [Bal02].

The database we describe below, intends to capture the integrated meaning of the features found in the component frame described earlier.



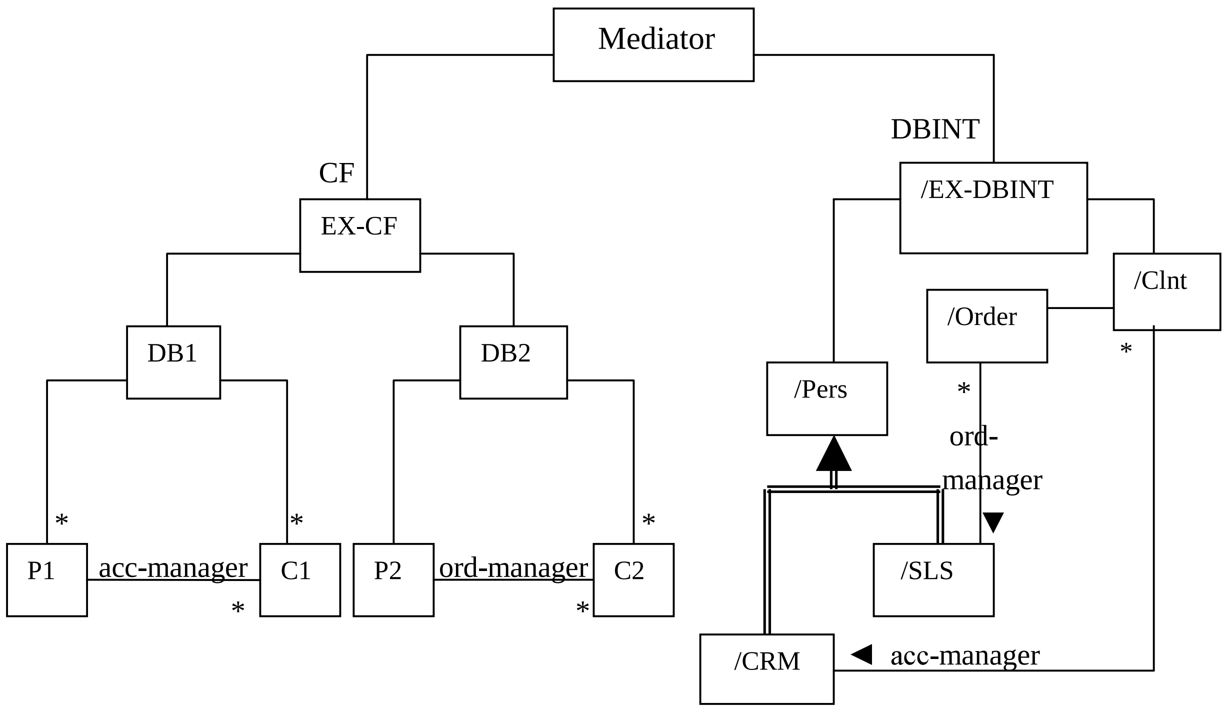
Constraints

```
context Pers inv:
Pers.allInstances -->
forall(p1,  p2:  Pers  |  (p1.dep=p2.dep  and  p1.pno=p2.pno)
implies  p1=p2)
Pers.allInstances -->
forall(p:Pers | p.sal > 1500  implies  p.oclIsTypeOf(SLS))
sal >= 1000
tel.size <= 16
cntrcd.size <= 5

context SLS inv:
bonus >= 0

context Clnt inv:
Clnt.allInstances --> forall(c1, c2: Clnt | (c1.clno=c2.clno
                      implies  c1=c2)
cntrcd.size <= 5

context Order inv:
Order.allInstances --> isUnique (o: Order | o.ordno)
```

We are now faced with the problem to explicitly link the component frame to this integrated (and virtual) database described above

6. Getting the mediator to do its work

Consider the following UML model containing a class, called the mediator, explicitly relating the component frame EX-CF and the virtual integrated database EX-DBINT



The mediator now has to correctly link the component frame EX-CF to the (virtual) database EX-DBINT. This is not a trivial task and involves a precise mapping of component elements to the virtual database. The mapping also has to take into account various constraint conditions which rule inside EX-CF. We do this by introducing suitable conversion operations inside the classes.

Our guiding principle for a successful conversion from the component frame CF to the integrated database DBINT is:***CWA-INT:***

> the integrated database DBINT is intended to hold exactly the "union" of the data in the source databases in CF

Typically, requirement CWA-INT displays our conformance to the traditional Closed World Assumption (CWA) found in the database literature ([Rei84]). This requirement has to be further investigated for consequences when applied to querying and to updating. In more mathematical terms, we will demand that

UoD(Mediator.CF) $\cong$ UoD(Mediator.DBINT)

In words, the universe of discourse of component frame CF and the universe of discourse of the integrated database DBINT are, in a mathematical sense, isomorphic. (Actually, an *endomorphic embedding* from the universe of discourse of component frame CF and the universe of discourse of the integrated database DBINT will do.)
Another matter that needs some attention, is the way that modifications on the source databases are taken care of, once they have become members of the federation. We will stipulate that all modifications on the source databases will now run through the virtual integrated database DBINT. By this we mean that an insert on a database inside the component frame CF can from now on only take place as the effect of an initial insert inside the integrated database DB-INT. Users will only view the (virtual) integrated database DBINT, and an insert on DBINT will be translated to a (collection of) insert(s) inside database components of the component frame CF. The same holds for a delete (and –hence- an update).

In order to support this stipulation, we will have to prove that any allowed insert (delete) on DBINT will result in a allowed insert (delete) within CF.

As mentioned earlier, integration of the source database schemas into one encompassing schema can be a tricky business due to

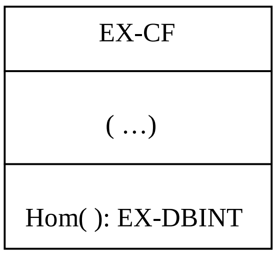
1. renaming
2. data conversion
3. default values

4. missing attributes

We will illustrate each of these cases in the context of our example databases. Key to the solution that we offer, is the introduction of a so-called homogenizing function which will actually provide for the linking of all relevant features in the component frame to features in the integrated database.

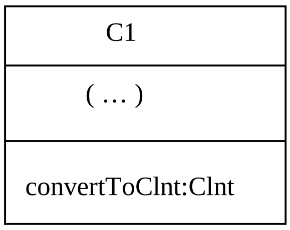
7. Introducing the homogenizing function: mapping the component frame to the virtual integrated database

What we do is that we add a method, called Hom, to the top-level EX-CF class resulting in an element (database state) of the integrated database EX-DBINT



```
context   EX-CF::Hom( ):EX-DBINT
post      self.Hom.Clnt.allInstances =
          self.CRM.C1.allInstances --> collect(c: C1 |
                                           c.convertToClnt)
```

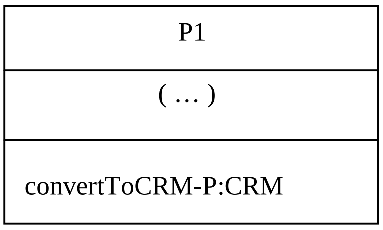
Here we have assumed the existence of a conversion function convertToClnt within the class C1



with the following post conditions

```
context   C1::convertToClnt( ): Clnt
post      Cl.attributes -->
          forall (d: String | self.convertToClnt.d = self.d)
          and
          (self. ConvertToClnt.acc-manager =
          self.acc-manager.convertToCRM)
```

we have now furthermore assumed the existence of a conversion function convertToCRM residing within the P1-class resulting in an object from the class CRM in the DBINT-database

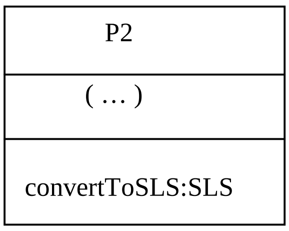


This conversion function has the following post conditions

```
context   P1::convertToCRM( ): CRM
post      self.convertToCRM.pno   = self.prsno
          and
          self.convertToCRM.pname = self.name
          and
          self.convertToCRM.sal   = self.sal.convert$To€
          and
          self.convertToCRM.part  = self.part
          and
          self.convertToCRM.addr = (self.street)^(“ ”)^
                                    (self.hnr)
          and
          self.convertToCRM.zip   = (self.zip.num)^(“ ”)^
                                    (self.zip.let)
          and
          self.convertToCRM.city  =  self.city
          and
          self.convertToCRM.tel   = (“31-50-363-”)^(“ ”)^
                                    (self.telint)
          and
          self.convertToCRM.cntrc = “NL”
          and
          self.convertToCRM.dep   = “CRM”
```

Notice that the function convertToCRM is injective!

Analogously, we can define a function converting the objects in the P2-class to corresponding objects in the SLS-class of DBINT, by assuming the existence of a conversion function convertToSLS within the class P2:

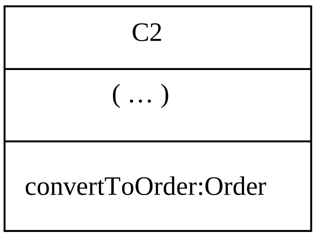


with the following (rather trivial) post conditions

```
context   P2::convertToSLS( ): SLS
post      self.convertToSLS.pno   = self.eno
          and
          self.convertToSLS.pname = self.name
```

```
and
self.convertToSLS.sal   = self.sal.
and
self.convertToSLS.part  = self.part
and
self.convertToSLS.addr  = self.addr
and
self.convertToSLS.zip   = self.zip
and
self.convertToSLS.city  = self.city
and
self.convertToSLS.tel   = self.tel
and
self.convertToSLS.cntrc = self.cntrc
and
self.convertToSLS.dep   = “SLS”
and
self.convertToSLS.bonus = self.bonus
and
self.convertToSLS.func   = self.func
```

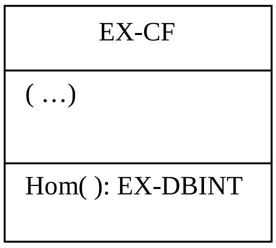
A bit more difficult is the definition of a function converting the objects in the C2-class to corresponding objects in the Order-class of DBINT. We do this by assuming the existence of a conversion function convertToOrder within the class C2:



with the following post conditions

```
context   C2::convertToOrder( ): Order
post      self.ConvertToOrder.ordno =  self.ordno
          and
          self.convertToOrder.ord-manager =
         (self.ord-manager).convertToSLS
          and
          self.convertToOrder.Clnt =
         (self.linkToC1).convertToClnt
```

where the previously defined operation linkToC1 provides the link to the unique C1-object associated to a given C2-object.
We now have a complete set of conversion functions mapping objects in the component frame CF to objects in DBINT. The homogenizing function Hom defined in the class EX-CF can now be given its full definition as offered below:



```
context    EX-CF::Hom( ):EX-DBINT
post      (self.Hom).Clnt.allInstances  =
           self.CRM.C1.allInstances   --> collect(c: C1 |
                                           c.convertToClnt)
           and
          (self.Hom).SLS.allInstances   =
           self.Sales.P2.allInstances --> collect(p: P2 |
                                           p.convertToSLS)
           and
          (self.Hom).CRM.allInstances   =
           self.CRM.P1.allInstances   --> collect(p: P1 |
                                          p.convertToCRM)
           and
          (self.Hom).Order.allInstances =
           self.Sales.C2.allInstances --> collect(o: C2 |
                                          o.convertToOrder)
```

With this set of mappings we can define the missing link providing the mapping of objects inside the component frame CF to objects inside the virtual database DBINT. We do this by adding appropriate constraints to the mediator class.

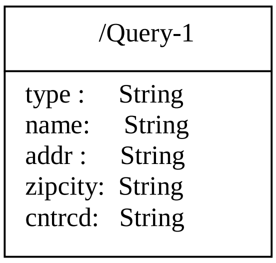
```
context Mediator inv:

self.DBINT.CRM.allInstances   = (self.CF.Hom).CRM.allInstances
and
self.DBINT.SLS.allInstances   = (self.CF.Hom).SLS.allInstances
and
self.DBINT.Clnt.allInstances  = (self.CF.Hom).Clnt.allInstances
and
self.DBINT.Order.allInstances =
(self.CF.Hom).Order.allInstances
```

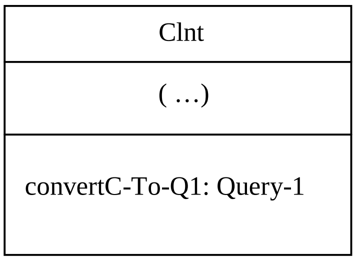
8. Querying the virtual integrated database through the mediator

Consider the following example query posed against the integrated database EX-DBINT

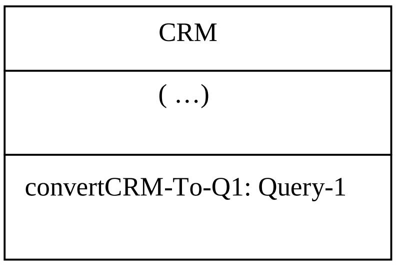
“Give the combined list of all clients and CRM-employees”

Following [Bal02], a query in UML is specified in terms of a view definition, where a view is conceived as a derived class. We define the following derived class, called /Query-1:





```
context    Clnt::convertC-To-Q1( ): Query-1
post       self.convertC-To-Q1.type    = `CL'  and
           self.convertC-To-Q1.name    = self.clname  and
           self.convertC-To-Q1.addr    = self.addr  and
           self.convertC-To-Q1.zipcity = self.zipcity  and
           self.convertC-To-Q1.cntrcd  = self.cntrcd
```



```
context     CRM::convertCRM-To-Q1( ): Query-1
post        self.convertCRM-To-Q1.type = `CRM'  and
            self.convertCRM-To-Q1.name    = self.pname  and
            self.convertCRM-To-Q1.addr    = self.addr   and
            self.convertCRM-To-Q1.zipcity =(self.zip) ^(" ")^
                                           (self.city)  and
            self.convertCRM-To-Q1.cntrcd  = self.cntrcd
```

We then add appropriate constraints to EX-DBINT

```
context  EX-DBINT inv:
Query-1.allInstances =
(Clnt.allInstances --> collect(c : Clnt | c.convertC-To-Q1))
.Union(CRM.allInstances -->
collect(p : CRM | p.convertCRM-To-Q1))
```

By now expanding the definition of Clnt and CRM, we obtain the definition of this query in terms of the original database components found in the component frame EX-CF, but then in terms of the homogenizing function Hom within the context of the *mediator* (hence , the `self` referred to in the OCL specification below, is the self in the context of the Mediator)

```
self.DBINT.Query-1.allInstances =
((self.CF.Hom).Clnt.allInstances  -->
  collect(c : self.DBINT.Clnt | c.convertC-To-Q1))
.Union((self.CF.Hom).CRM.allInstances  -->
  collect(p : self.DBINT.CRM | p.convertCRM-To-Q1))
```

By expanding the definitions of (self.CF.Hom).Clnt.allInstances and (self.CF.Hom).CRM.allInstances one level deeper, we obtain the definition of this query in terms of the original components

```
(self.CF.Hom).Clnt.allInstances  =
 self.CF.CRM.C1.allInstances   --> collect(c: self.CF.C1 |
                                              c.convertToClnt)
```

and

```
(self.CF.Hom).CRM.allInstances =
 self.CRM.P1.allInstances   --> collect(p: self.CF.P1 |
                                              p.convertToCRM)
```

Hence, the query is now expressed completely in terms of the original database components found in the component frame EX-CF!

Summary

We describe a general semantic framework for precise specification of so-called mediating systems; such a system provides for tight coupling on a global level of a collection of heterogeneous component databases to a federated database. This mediating system integrates, by means of a so-called homogenizing function, in a uniform and systematic manner the underlying data models of the component systems to a global data model, including constraint specifications. Our focus has been on solving the problems caused by semantic heterogeneity of component systems. The integration process is based on the notion of database views. The mediating system allows for global queries that can be decomposed in a uniform and systematic manner into local queries on component databases. Our approach is based upon the UML/OCL data model. UML is the de facto standard language for analysis and design in object-oriented frameworks, and is being employed more and more for analysis and design of Information Systems based on databases and their applications. The Object Constraint Language (OCL) - as part of UML - can aid in the unambiguous modelling of database constraints. One of the central notions in database modelling and in constraint specifications is the notion of a database view; a database view closely corresponds to the notion of derived class in UML. We employ OCL and the notion of derived class as a means to treat database constraints and database views in a federated context. The paper demonstrates that our

particular mediating system integrates component schemas without loss of constraint information. Furthermore, we offer a setting in which to describe a UML/OCL-representation of relational databases.

Acknowledgements:
I wish to thank Bert de Brock of the Faculty of Management and Organization, for numerous discussions, corrections and valuable insights.

References

[AB01] Akehurst, D.H., Bordbar, B.; On Querying UML data models with OCL; «UML» 2001 - The Unified Modeling Language, Modeling Languages, Concepts, and Tools, 4th International Conference, Toronto, Canada, 2001, Proceedings. Lecture Notes in Computer Science 2185, Springer, 2001

[Bal02] Balsters, H. ; Derived classes as a basis for views in UML/OCL data models; SOM Research Series 02A47, University of Groningen, 2002

[BB01] Balsters, H., de Brock, E.O.; Towards a general semantic framework for design of federated database systems ; SOM Research Series 01A26, University of Groningen, 2002

[BBZ93] Balsters, H., de By, R.A., Zicari, R.; Sets and constraints in an object-oriented data model; Proceedings Seventh European Conference on Object-Oriented Programming (ECOOP), Kaiserslautern, Germany, July, 1993.

[BP98] Blaha, M., Premerlani, W.; Object-oriented modeling and design for database applications; Prentice Hall, 1998

[BV92] Balsters, H., de Vreeze, C.C.; A semantics of object-oriented sets; Third International Workshop on Database Programming Languages (DBPL;eds. Abiteboul, Kannelakis), Morgan Kaufmann Publishers, California USA, 1992.

[CGW96] S.S. Chawathe, H. Garcia-Molina, J. Widom; A toolkit for constraint maintenance in heterogeneous information systems. 12th International Conference on Data Engineering (ICDE96); IEEE Press, 1996

[Co70] E.F. Codd; A relational model of data for large shared data bank; Communications of the ACM, vol. 13(6), 1970

[DaH84] U. Dayal, H.Y. Hwang; View definition and generalization for database integration in a multidatabase system; IEEE Transactions on Software Engineering 10, 1984

[D00] Date, C.J.; An introduction to database systems; Addison Wesley, 2000

[DH99] Demuth, B., Hussmann, H.; Using UML/OCL constraints for relational database design; «UML»'99: The Unified Modeling Language - Beyond the Standard, Second International Conference, Fort Collins, CO, USA, 1999, Proceedings. Lecture Notes in Computer Science 1723, Springer, 1999

[DHL01] Demuth, B., Hussmann, H., Loecher, S.; OCL as a spevcification Unified Modeling Language, Modeling Languages, Concepts, and Tools, 4th International Conference, Toronto, Canada, 2001, Proceedings. Lecture Notes in Computer Science 2185, Springer, 2001

[DKM93] P. Drew, R. King, D. McLeod, M. Rusinkievicz, A. Silberschatz; Report of the workshop on semantic heterogeneity and interoperation in multidatabase systems; SIGMOD RECORD 22, 1993

[EN94] R. Elmasri and S.B. Navathe; Fundamentals of database systems; Benjamin/Cummings, Redwood City (CA), 1994

[EP00] Eriksson, H., Penker, M.; Business modeling with UML; OMG 2000

[GR97] Gogolla, M., Richters, M.; On constraints and queries in UML; Proceedings UML'97 Workshop "The Unified Modeling Language – Techniques and Applications", 1997

[GSC96] M. Garcia-Solica, F. Saltor, M.Castellanos*;* Semantic heterogeneity in multidatabase systems; Object-oriented multidatabase systems; Bukhres, Elmagarid (eds.), Prentice Hall, 1996

[GUW02] Garcia-Molina, H., Ullman, J.D., Widom, J.; Database systems; Prentice Hall, 2002

[Hull97] Hull, R.; Managing Semantic Heterogeneity in Databases; ACM PODS'97, ACM Press 1997.

[Ken91] W. Kent; Solving domain mismatch and schema mismatch problems with an object-oriented database programming language; 7th International

Conference on Very Large Databases (VLDB97), 1997

[KoC95] J.L. Ko, A.L.P. Chen; A mapping strategy for querying multiple object databases with a global object schema; IEEE RIDE -DOM,1995

[MC99] Mandel, L., Cengarle, M.V.; On the expressive power of OCL; FM'99 Formal Methods, World Congress on Formal Methods in the Development of Computing Science; Lecture Notes in Computer Science 1708, Springer, 1999

[MeY95] W. Meng, C. Yu; Query processing in multidatabase systems; Modern database systems; Kim (ed.), ACM Press, 1995

[OMG99] Object Management Group; Unified Modelling Language Specification, version 1.3; June 1999; **http://omg.org**

[Rei 84] Reiter, R.; Towards a logical reconstruction of relational database theory. In: Brodie, M.L., Mylopoulos, J., Schmidt, J.W.; On conceptual modeling; Springer Verlag, 1984

[ShL90] A.P. Sheth, J.A. Larson*;* Federated database systems for managing distributed, heterogeneous and autonomous databases; ACM Computing surveys 22, 1990

[SSR94] E. Sciori, M. Siegel, A. Rosenthal; Using semantic values to facilitate interoperability among heterogeneous information systems; ACM Transactions on database systems 19, 1994

[SQL 92] ISO 9075-1992(E); Database language SQL; ISO/IEC JTC1/SC21, 1992

[Ver97] M. Vermeer; Semantic interoperability for legacy databases. Ph.D.-thesis, University of Twente, 1997.

[Wie95] G. Wiederhold; Value-added mediation in large-scale information ystems; IFIP Data Semantics (DS-6), 1995

[WK99] Warmer, J.B., Kleppe, A.G.; The object constraint language; Addison Wesley, 1999