Integration of Integrity Constraints in Database Federations

Herman Balsters, Bert de Brock
University of Groningen, Faculty of Management and Organization, P.O. Box 800, 9700 AV Groningen, The Netherlands, {h.balsters, e.o.de.brock}@bdk.rug.nl

Abstract: A database federation provides for tight coupling of a collection of heterogeneous legacy databases into a global integrated system. A major problem in constructing database federations concerns maintaining quality and consistency of data on the global level of the federation. In particular, the integration of integrity constraints within component legacy databases into a single global schema is a process that is prone to incompleteness and inconsistency. Moreover, additional inter-database constraints between the various component legacy databases often also have to be taken into account to complete the integration process. Our approach to coupling of component databases into a global, integrated system is based on the concept of mediation. Our major result is that mediation in combination with a so-called integration isomorphism integrates component schemas without loss of constraint information; i.e., integrity constraints available at the component level remain intact after integration on the global level of the federated database. Our approach to integration also allows for specification of additional inter-database constraints between the various component legacy databases. We therefore can handle consistency not only on the local level of component databases, but also consistency on the global level between the various component databases. We shall describe a general semantic framework for specification of database federations based on the UML data model. This data model will prove to offer an elegant and powerful framework for analysis and design of database federations, including integration of integrity constraints.

Key words: Databases, distribution, integrity control, legacy systems, systems integration, semantics, consistency, data modeling, object-orientation, UML

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* ([ShL90]). 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 systems (data and service repositories), 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 an 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. We are faced with major problems when trying to maintain data quality and data consistency in the integration process. Many of these problems deal with integration of the schemas of the component databases involved, and in particular with resolving consistency conflicts rising from integrity constraints ruling within the component databases.

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. In our paper we will consider a *tightly-coupled approach* to database mediation, in which a global integrated schema of the federation is maintained. We base our approach 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. The user of the federated system will be offered the impression 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. 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,Ver97]. Semantic heterogeneity refers to disagreement about the meaning, interpretation, or intended use of related data. In this paper we will focus on the UML/OCL data model to tackle the problem of integrating semantic heterogeneity. UML/OCL offers a high-level specification language and is equipped with a unique combination of high expressiveness with a large degree of precision. Also, 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. OCL ([WK03, OCL2.0]) complements UML by providing a language for formally expressing integrity constraints of a model. In this paper, we will assume that component databases -e.g. relational databases - can somehow be modeled in the UML/OCL-framework ([BP98]).

The paper describes a particular mediating system integrating 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. Furthermore, we will show how to deal with integrity constraints not only within, but also between various component databases (the so-called *inter-database constraints*). Integration conflicts are treated in a tightly-coupled environment, and are resolved by introducing a so-called *integration isomorphism* ([Bal03]).

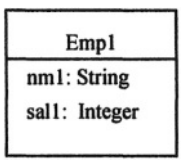
The paper is concluded by a description of general design principles for database federations.

2. UML/OCL AS A SPECIFICATION LANGUAGE FOR DATABASES

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 for specifying integrity 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. In the more specific context of relational databases and OCL, [DH99] offers a

framework for representing integrity constraints within the relational data model ([GUW02]). Our application area 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. In particular, we will concentrate on data representations in the UML data model, and employ OCL as a language for expressing accompanying constraint specifications, both on the component level and on the global level of a federation.

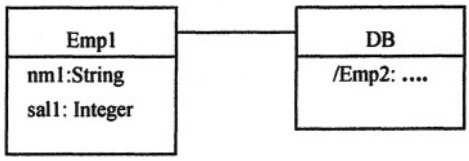
An important notion in modeling databases is that of a database view. To illustrate this notion, consider the case that we have a class called Emp1 with attributes nm1 (of type String) and sal1 (of type Integer), indicating the name and salary of an employee object belonging to class Emp1. In UML, this can be represented as follows



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=(100 * 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 (showing all salaries in cents), 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.

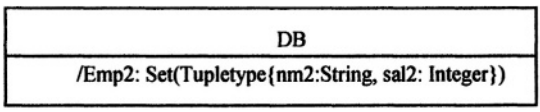
We will now show how to faithfully describe Emp2 as a derived class in UML/OCL (version 2.0) in such a way that it satisfies the requirements of a (relational) view. First of all, we must satisfy the requirement that the set of instance of class Emp2 is the result of a calculation applied to the set of instances of class Emp1. The basic idea is that we introduce a (database-) class called DB that has an association to class Emp1, and that we define within the context of the database DB an attribute called Emp2. A database object will reflect the actual state of the database, and the system class DB

will only consist out of one object in any of its states. Hence the variable *self* in the context of the class DB 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.



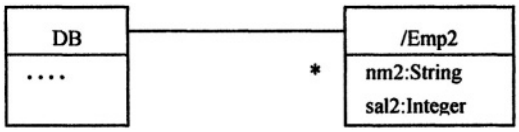

```
context  DB
def: Emp2: Set(Tupletype{nm2:String, sal2: Integer}) =
             (self.Emp1 -> collect(e:Emp1 |
            Tuple{nm2=e.nm1, sal2=(100*e.sal1)}))-> asSet
```

In this way, we explicitly specify Emp2 as the result of a calculation performed on the base class Emp1. Graphically, we could represent Emp2 as follows



(1)

where the slash-prefix of Emp2 indicates that Emp2 is a *derived* table. Since in practice such a graphical representation could give rise to rather large box diagrams (due to lengthy type definitions), we will use the following (slightly abused) graphical notation (2) to indicate this derived class

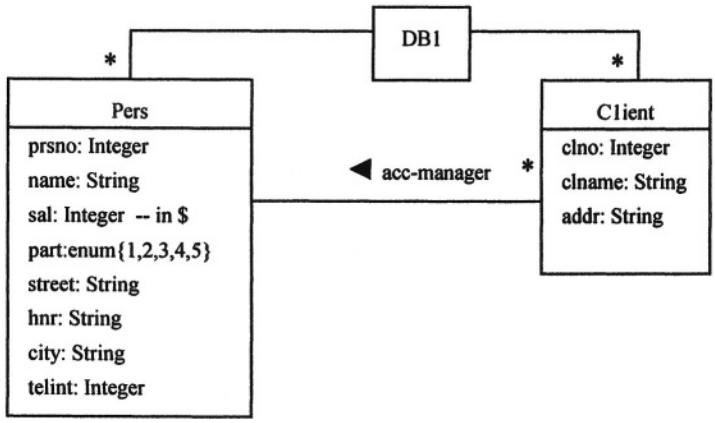


(2)

The intention is that these two graphical representations are to be considered equivalent; i.e., graphical representation (2) is offered as a diagrammatical convention with the sole purpose that it be formally equivalent *(translatable)* to graphical representation (1). Full details regarding representations of database views in UML/OCL can be found in [Bal03b].

3. COMPONENT FRAMES

We can also consider a complete collection of databases by looking at so-called component frames, where each (labeled) component is an autonomous database system (typicaly encountered in legacy environments). As an example consider a component frame consisting of two separate component database systems: the Client-Relationship-Management (CRM)-database DB1, and the Sales-database DB2 below



DB1 consists of two classes. For example, `Pers` is the class of employees responsible for management of client resource; `part` indicates that employees are allowed to work part time; hnr indicates house number; telint indicates internal telephone number; acc-manager indicates the employee (account manager) that is responsible for some client's account. The rest of the features of DB1 speak for themselves. We furthermore assume that database DB1 has the following integrity constraints

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

context Client inv:
Client.allInstances --> isUnique (c: Client | c.clno)
```

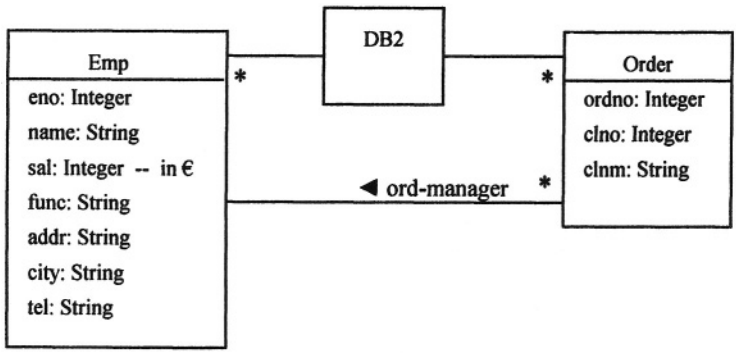
Informally, these integrity constraints state that

- Persons have unique `prsno`-values;
  their salaries may not exceed 1500 dollars;

their (internal) telephone numbers have values between 1000 and 9999

- Clients have unique `clno`-values

The second database is the so-called Sales-database DB2



Most of the features of DB2 also speak for themselves. We offer a short explanation of some of the less self-explanatory aspects: `Emp` is the class of employees responsible for management of client order; `func` indicates that an employee has a certain function within the organization; `Ord-manager` indicates the employee (account manager) that is responsible for some client's order. We assume that this second database has the following integrity constraints:

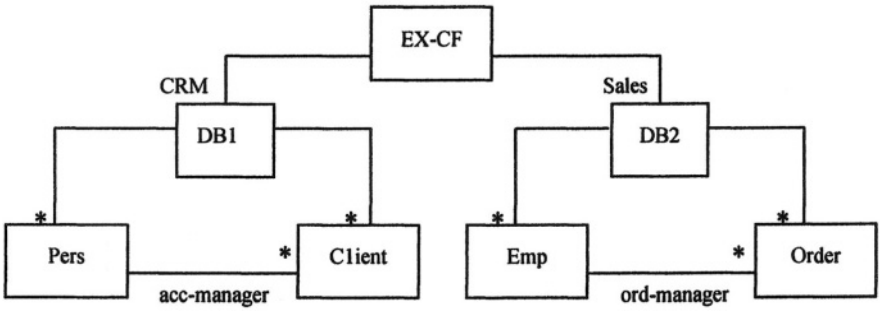
```
context Emp inv:
Emp.allInstances --> isUnique (p: Emp | p.eno)
sal >= 1000
tel.size <= 16

context Order inv:
Order.allInstances --> isUnique (c: Order | c.ordno)
Order.allInstances --> forall(c: Order | c.ord-manager.func =
'Sales')
```

Informally, these integrity constraints state that

- Employees have unique `eno`-values;
  their salaries are at least 1000 euros;
  their telephone numbers consist of strings of no more than 16 characters
- Orders have unique `ordno`-values;
  employees managing some order must have `'sales'` as `func`-value

We can now place the two databases DB1 and DB2 without confusion into one component frame EX-CF as seen below



4. SEMANTIC HETEROGENEITY

We are faced with major problems when trying to maintain data quality, data completeness and data consistency in the integration process. Many of these problems deal with integration the schemas of the component databases involved, and in particular with resolving consistency conflicts rising from integrity constraints ruling within (and possibly between) the component databases. 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: *homonyms* (solved by mapping same name occurrences to different names); *synonyms* (mapping different names to one common name); *different representations* (data conversion to a common value); *default values* (adding default values for new attributes); *missing attributes* (adding new attributes in order to discriminate between certain class objects); *absent subclasses* (creation of a common superclass and subsequent accompanying subclasses).

By homonyms we mean that certain names may be the same, but actually have a different meaning (different semantics). Conflicts due to homonyms are resolved by mapping two same name occurrences to different names in the integrated model. In the sequel, we will refer to this solution method as **hom**. Synonyms, on the contrary, refer to certain names that are different, but have in fact the same semantics. Synonyms are treated analogously, by mapping two different names to one common name; this solution method is refered to by **syn**.

In the integration process, one often encounters the situation where two attributes have the same meaning, but that their domain values are differently represented. For example, the two attributes sal in the Pers and the Emp class of databases DB1 and DB2, respectively, both indicate the salary of an employee, but in the first case the salary is represented in the currency dollars ($), while in the latter case the currency is given in euros (€). What we can do, is to convert the two currencies to a common value (e.g. $, invoking a function convertTo$). Applying a conversion function to map to some common value in the integration process, is indicated by **conv**.

Sometimes an attribute in one class is not mentioned in another class, but it could be added there by offering some suitable default value for all objects inside the second class. As an example, consider the attribute `part` in the class Pers (in DB1): it could also be added to the class Emp (in DB2) by stipulating that the default value for all objects in Emp will be 5 (indicating full-time employment). Applying this principle of adding a default value in the integration process, is indicated by **def**.

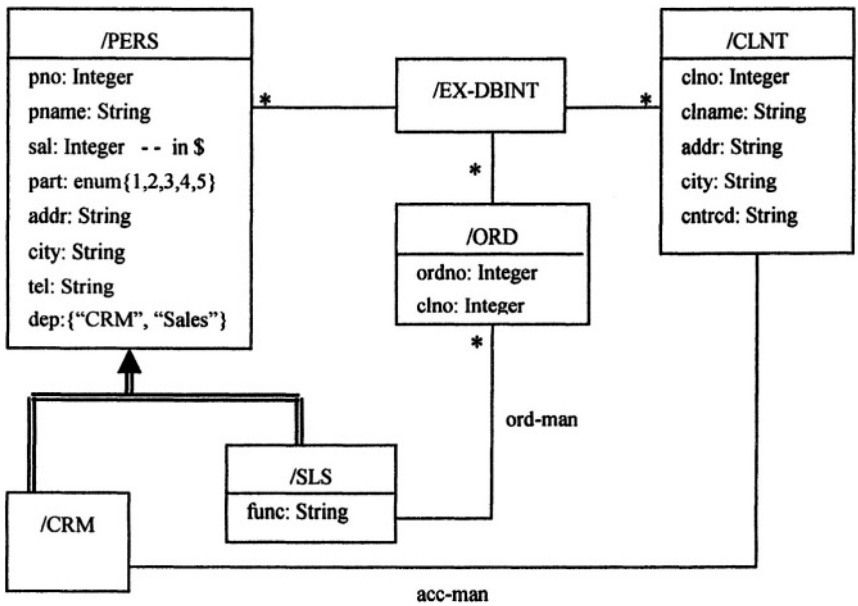
The integration of two classes often calls for the introduction of some additional attribute, necessary for discriminating between objects originally coming from these two classes. This will sometimes be necessary to be able to resolve seemingly conflicting constraints. As an example, consider the classes Pers (in DB1) and Emp (in DB2). Class Pers has as a constraint that salaries are less than 1500 (in $), while class Emp has as a constraint that salaries are at least 1000 (in €). These two constraints seemingly conflict with each other, obstructing integration of the Pers and the Emp class to a common class, say PERS. However, by adding a discriminating attribute `dep` indicating whether the object comes from the CRM or from the SLS department, one can differentiate between two kinds of employees and state the constraint on the integrated level in a suitable way. Applying the principle of adding a discriminating attribute to differentiate between two kinds of objects inside a common class in the integration process, will be indicated by **diff**. The situation of a missing attribute mostly goes hand in hand with the introduction of appropriate subclasses. For example, introduction of the discriminating attribute `dep` (as described above), entails introduction of two subclasses, say CRM and SLS of the common superclass PERS, by listing the attributes, operations, and constraints that are specific to CRM- or SLS-objects inside these two newly introduced subclasses. Applying the principle of adding new subclasses in the integration process, is indicated by **sub**.

We note that in [Ver97] the problems regarding semantic interoperability were analyzed in a high-level object-oriented langauage called TM ([BBZ93], specifically tailored for object-oriented analysis and design of

databases. In [BS98] this approach using TM was further pursued, providing a basis for automatic verification of transaction correctness in OO databases.

5. THE INTEGRATED DATABASE

We now offer our construction of a virtual database EX-DBINT, represented in terms of a derived class in UML/OCL. The database we describe below, intends to capture the integrated meaning of the features found in the component frame described earlier



We have introduced a superclass PERS with two subclasses CRM and SLS. A subclass inherits all of the attributes of a superclass, and possibly also includes (e.g., the class SLS) attributes of its own. Furthermore, this database has the following integrity 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.oclIsTypeOf(SLS) implies (p.sal >= 1000.convertTo$ and
part=5)) and (p.oclIsTypeOf(CRM) implies p.sal <= 1500 ))
```

```
tel.size <= 16

context CLNT inv:
Clnt.allInstances --> isUnique (c: CLNT | c.clno)

context ORD inv:
Order.allInstances --> isUnique (o: ORD | o.ordno)
ord-manager.func = 'Sales'
```

Informally, the integrity constraints on the PERS-class state that

- PERS-objects have unique values for the *combination* of attributes (`dep,pno`)
- If a PERS-object is also a SLS-object, then its salary is at least the dollar-equivalent of 1000 euros
- If a PERS-object is also a CRM-object, then its salary is at most 1500 dollars
- Its telephone number consists of strings of at most 16 characters

In order to control the quality, completeness, and consistency of the data, we shall now carefully analyze the specification of this (integrated) database EX-DBINT, and check whether it captures the intended meaning of integrating the classes in the component frame EX-CF and resolves potential integration conflicts.

Conflict 1: Classes Emp and Pers in EX-CF have partially overlapping attributes, but Emp has no attribute `part` yet, and one still needs to discriminate between the two kinds of class objects (due to specific integrity constraints pertaining to the classes Emp and Pers). Our solution in DBINT is based on applying **syn + def + diff + sub**: map to common class name (`PERS`); add a default value (to the attribute `part`); add an extra discriminating attribute (`dep`); introduce suitable subclasses (`CRM` and `SLS`).

Conflict 2: Attributes `prsno` and `eno` intend to have the same meaning (a *key constraint,* entailing uniquely identifying values for employees, for Emp- and Pers-objects). Our solution in DBINT is therefore based on applying **syn + diff** (map to common attribute name (`pno`); introduce extra discriminating attribute (`dep`)) and enforce uniqueness of the value combination of the attributes `pno` and `dep`.

Conflict 3: Attributes `sal` (in Pers) and `sal` (in Emp) partially have the same meaning (salaries), but the currency values are different. Our solution is based on applying **conv** (convert to a common value).

Conflict 4: The attribute combination of `street` and `hnr` (in Pers) partially has the same meaning as `addr` in Emp (both indicating address values), but the domain values are differently formatted. Our solution is therefore based on applying **syn + conv** (map to common attribute name and convert to common value).

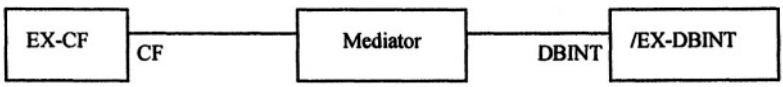
Conflict 5: Attributes `telint` (internal telephone number) and `tel` (general telephone number) partially have the same meaning, but the domain values are differently formatted. Our solution is therefore based on applying **syn + conv** (map to common attribute name and convert to a common format).

6. INTEGRATING BY MEDIATION

Our strategy to integrate a collection of legacy databases is based on the following principle:

> *An integrated database is intended to hold exactly the "union" of the data in the source databases in the component frame CF*

This principle is also known as the Closed World Assumption for integrated databases (**CWA-INT**, cf. [Hull97]), and is a direct extension of the CWA first introduced in [Rei84] for monolithic databases. In [Bal03] we offer a mathematical basis for CWA-INT, by introducing a so-called *integration isomorphism,* which aims at correctly mapping elements of the component frame CF to the integrated database DBINT. In [Bal03] we furthermore offer an elaborate description of a UML/OCL model containing a class, called the *mediator,* explicitly relating CF to DBINT, as depicted below



Our construction of the mediator class closely reflects the basic ideas of mediation as first described in [Wie95]. Further elaboration on our construction of such a mediator class (and the accompanying integration isomorphism) is beyond the scope of this paper; the interested reader is refered to [Bal03] for more details.

7. INTER-DATABASE (OR COMPONENT-FRAME) CONSTRAINTS

Additional information analysis might reveal the following two wishes regarding the consistency of data in the component frame EX-CF:

(1) Nobody is registered as working for both the CRM and Sales department; i.e., these departments have no employees in common
(2) Client numbers in the Sales database should also be present in the CRM database

This entails that certain integrity constraints should be added, and in this case on the level of the class EX-CF, since these integrity constraints hold between two *databases* DB1 and DB2. Such integrity constraints are called *inter-database constraints,* or *component-frame constraints.* We now offer a specification of the two inter-database constraints mentioned above. We first offer some appropriate abbreviations (using a so-called **let-**construct), and then offer the two constraint specifications.

The following constraint states that there are no common `prsno-` and `eno`-values, and that each `clno`-value in the Order class corresponds to some `clno`-value in the Client class in the context of the class EX-CF

```
context  EX-CF  inv:
let  Pers-nrs = ((self.CRM.Pers.allInstances     ->
                 collect (p:Pers | p.prsno))     -> asSet)
let  Emp-nrs  = ((self.Sales.Emp.allInstances    ->
                 collect (e:Emp | e.eno))        -> asSet)
let  Cl-nrs   = ((self.CRM.Client.allInstances   ->
                 collect (c:Client | c.clno))    -> asSet)
let  Ord-nrs  = ((self.Sales.Order.allInstances  ->
                 collect (o:Order | o.clno))     -> asSet)
in
(Pers-nrs->Intersect(Emp-nrs)) -> isEmpty  and                (1)
 Ord-nrs  ->  forall(o:Integer  |  (Cl-nrs  ->  exists(c:Integer  |
o=c)))                                                        (2)
```

We are now, of course, also faced with the obligation to suitably introduce these inter-database constraints in the integrated database DBINT. This can now be done in a straightforward manner, as illustrated below for the first constraint

```
context  EX-DBINT  inv:
let  X =  (self.CRM.allInstances -> collect(c:CRM | c.pno))
                                 -> asSet
let  Y =  (self.SLS.allInstances -> collect(s:SLS| s.pno))
                                 -> asSet
in
 (X->Intersect(Y)) -> isEmpty
```

Informally, this constraint states that there are no `pno`-values common to the CRM-class and the SLS-class.

As the examples offered above illustrate, OCL offers a powerful means to specify inter-database constraints in a very general manner in the context of our approach.

8. HEURISTICS: FROM SPECIFIC EXAMPLES TO A GENERAL APPROACH

This section concerns a discussion on methodology, in which we attempt to move from specific examples to a general approach in constructing a database federation from a collection of legacy databases, in order to maintain quality, completeness, and consistency of the data.

We adopt the following strategy to integrate a collection of legacy databases (collected in a component frame) into a virtual integrated database: (1) create a tightly-coupled architecture of the federated system, and (2) abide to the principle of the Closed World Assumption of Database Integration (CWA-INT). CWA-INT is established by supplying an *integration isomorphism,* mapping from the component frame to the virtual integrated database ([Bal03]).

In practice, fulfilling this strategy can often be a challenging demand, but without eventually realizing both aspects, the resulting federated database will fall short due to incorrect query results and inadequate constraint integration (resulting in loss of completeness and/or consistency).

We now offer some heuristics concerning the realization of the isomorphic mapping from component frame to integrated database. The construction of this isomorphic mapping from the component frame to the virtual integrated database cannot –in principle- be determined beforehand in algorithmic terms. By this we aim to say that given some set of conflicts in moving from the components to the integrated federated schema, it is usually an illusion to state that there exists an indisputable *algorithm* determining how those conflicts are resolved. On the contrary, establishing the

specification of a suitable integration isomorphism usually, in terms of a formal specification (cf. [Bal03]), demonstrates the mostly *ad hoc* nature of resolving the conflicts at hand, reflecting the need for a *business semantics* to reach an eventual solution. For example, the resolution of the conflict of the currencies dollar ($ in DB1) and euro (€ in DB2), deciding which currency is to be taken on the common integrated level is basically *ad hoc,* and has to be offered by the business. In general, the process of constructing the formal specification of an isomorphism between the component frame and the virtual integrated database constantly has to be guided by knowledge of relevant business semantics.

Equipped with knowledge of relevant business semantics, we can proceed by following a short step-by-step guideline (constituting a heuristics, *not* an algorithm) for constructing a virtual integrated database from a collection of legacy databases, as described below:

1. Specify the details of the Component Frame **CF**
2. Analyse semantic heterogeneity: detect conflicts due to *Homonyms, Synonyms, Different representations, Default Values, Missing Attributes,* and *Subclassing*
3. Construct an integrated schema DBINT (applying the principles of **syn, hom, conv, def, diff,** and **sub)**
4. Introduce a **mediator class**
5. Enforce ***CWA-INT,*** by constructing an ***integration isomorphism*** (via the mediator class) between CF and DBINT
6. Add possible **inter-database constraints** in CF, and incorporate them into DBINT

This guideline –though systematic- is not algorithmic in nature. Applying the principles set out in this guideline will often demand the necessary creativity from the database modeler in close cooperation with a business expert, who has sufficient knowledge of the specific business domain. Apart from these challenges (which apply to most modeling methodologies), our guideline can offer a powerful methodology in moving from a collection of legacy systems to a correctly integrated database system maintaining quality, completeness, and consistency of the involved data.

REFERENCES

[AB01] Akehurst, D.H., Bordbar, B.; On Querying UML data models with OCL; «UML» 2001 4th Int. Conf., LNCS 2185, Springer, 2001

[Bal03] Balsters, H.; Object-oriented modeling and design of database federations; SOM Report, University of Groningen, 2003

[Bal03b] Balsters, H. ; Modelling database views with derived classes in the UML/OCL-framework ; 6th UML-conference, San Francisco, LNCS 2863, Springer, 2003

[BB01] Balsters, H., de Brock, E.O.; Towards a general semantic framework for design of federated database systems; SOM Report 01A26, University of Groningen, 2001

[BBZ93] Balsters, H., de By, R.A., Zicari, R.; Sets and constraints in an object-oriented data model; Proc. 7th ECOOP, LNCS 707, Springer 1993.

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

[BS98] Balsters, H., Spelt, D.; Automatic verification of transactions on an object-oriented database, 6th Int. Workshop on Database Programming Languages, LNCS 1369, Springer, 1998.

[DH99] Demuth, B., Hussmann, H.; Using UML/OCL constraints for relational database design; «UML»'99: 2nd Int. Conf., LNCS 1723, Springer, 1999

[DHL01] Demuth, B., Hussmann, H., Loecher, S.; OCL as a specification language for business rules in database applications; «UML» 2001, 4th Int. Conf., LNCS 2185, Springer, 2001

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

[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.

[OCL2.0] Response to the UML 2.0 OCL RfP, Revised Submission, Version 1.6, January 6, 2003

[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 and heterogeneous and autonomous databases; ACM Computing surveys 22,1990

[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 systems FIP Data Semantics (DS-6), 1995

[WK03] Warmer, J.B., Kleppe, A.G.; The object constraint language, 2nd ed.; Addison Wesley, 2003