

# **FEMUS**

## **a Federated Multilingual Database System**

**Martin Andersson, Yann Dupont, Stefano Spaccapietra, Kokou Yétongnon**

EPF Lausanne  
Department of Computer Science  
Databases Laboratory  
CH - 1015 Lausanne, Switzerland  
{martin,yann,stefano}@lbdsun.epfl.ch

**Markus Tresch\*, Haiyan Ye**

ETH Zürich  
Department of Computer Science  
Information Systems - Databases  
CH - 8092 Zürich, Switzerland  
{tresch,ye}@inf.ethz.ch

### **ABSTRACT**

This paper describes the objectives, choices, and first results of the FEMUS project in building a federated multilingual database system. The focus is on semantic issues, concentrating, in particular, on data model translations and database integration in a heterogeneous environment. Also discussed are mechanisms for exchanging data and metadata, and related implementation aspects.

---

+ This work is supported by the Schweizerischer Schulrat.

\* Present address: Department of Computer Science, University of Ulm, Germany.

## 1. INTRODUCTION

The increased availability of various databases in large corporations has created the need to federate the databases into loosely coupled collections of autonomous systems to allow controlled sharing of information and at the same time preserve the autonomy of each participant. Traditional distributed database (DDB) research has provided the earliest solutions to information sharing in distributed computing environments. The DDB approach, however, assumes that a single and integrated conceptual view of the databases must be provided to the users. Federated database (FDB) schema/system architecture design has partially benefited from this effort. For example, the design of both DDB and FDB systems includes functionalities such as: schema integration, query processing, transaction management. The DDB techniques used to provide these functionalities can be extended and applied to FDBS. The main difference between the two approaches are in the classes of users and the levels of component autonomy they support. Users of DDB systems access shared data only through the single conceptual schema of the DDB. This facilitates the enforcement of integrity constraints attached to the DDB, in much the same way (from user's point of view) as centralized DB systems. FDB systems, on the other hand, generally support two classes of users: federation users manipulate shared distributed information through one or more federated schemas; and local users, to whom the federation is transparent, access local data only. Preservation of local access to, and control over, the local database is essential for the support of component autonomy, the most salient feature of FDB systems.

Typically, federations are built upon heterogeneous systems and may include databases which support different data models: relational, CODASYL, and object-oriented data models. One of the important issues that need to be addressed is therefore the data model translation problem. Usually, to avoid a proliferation of translators, the use of a common data model within the distributed/federated system is encouraged. Semantic or object-oriented data models represent the best candidates for this role. Translations between semantic, object-oriented and relational models have largely been dealt with in the past. They have been commonly developed for two major cases: 1) to support a database design process where the database is first described in a conceptual schema which is subsequently converted into a logical schema [Bouzeghoub 91]; and 2) to support the design and operation of traditional DDB system. In the former case the translations are one-way processes which produce a single logical target schema from the source conceptual schema. As user queries are expressed and performed directly on the logical schema, there is no need for further translations. In the latter case, interoperability among data models is needed to move data in and out from the DDB. However, interoperability is usually not required among data manipulation languages, as most of the proposed DDB systems are monolingual: they support only one DML, the one associated to the common data model in use ([Demurjian 88] is an exception). The distinguishing feature of federated environments is that they have to be multilingual, because of the autonomy goal [Kim 89]. Therefore, the support of interoperability among DMLs becomes in fact the primary criterion for the specification of data model translations.

Besides the translation problem, interoperability in the federated approach necessitates a mechanism for making data available to the federation and for ensuring access to the data. This mechanism is usually called an import/export facility. Data exchanges require first that description of accessible (exported) data be available to all users of the federation or designated set of partners, and second that the information be presented to them in their local model (in a model they understand). Partners may then select (import) data of interest to them and include/integrate them into their view of the FDB. Integration facilities are the essential feature in building this unified view from the different pieces of imported data.

This paper presents the approach of an ongoing research project, FEMUS (FEderated MULtilingual System), jointly developed by the database research groups at Swiss Institutes for Technology (EPFL and ETHZ). The primary goal of the project is to provide a framework for investigating semantic related issues of interoperable database architectures. The focus of this paper is particularly on federated database construction issues. The FEMUS project itself is not discussed in detail. Instead, we examine two of its most important aspects: data model translation and integration issues. The former is dealt with in section 3, while the latter is the topic of section 4. Section 2 presents the architecture of the FEMUS prototype and the modeling paradigms on which interoperability is being experimented. Section 5 discusses semantics issues related to schema negotiation and data/metadata

exchange. Section 6 reviews implementation aspects related to the export/import mechanism. Section 7 concludes the paper.

## 2. FEMUS

The aim of the FEMUS project is to set up a framework for building a federated multilingual database system. By *federated* it is meant that the global system provides the functionalities to include, as components, different heterogeneous database systems cooperating together, and that the major goals are preserving site autonomy and supporting maximum flexibility in the interoperability mechanisms. By *multilingual* it is meant that an equally important goal is to build a system which can be accessed by different users through the local interface (data model and manipulation language) they are used to.

The initial version of the project includes two different database approaches:

- **ERC+** [Parent 92], an object-based extension of the entity-relationship model, including the specification of an ER algebra [Parent 84] and calculus. The fundamental concepts of the ERC+ model are entities, relationships, and complex objects.
- **COCOON**, an object-oriented data model with an object algebra that was developed based on the nested relational algebra [Scholl 87, 91] [Schek 86, 90]. The basic concepts are objects and functions. COCOON, like DUAL [Perl 89], promotes separation between the type and class hierarchies. This separation introduces original problems that need to be addressed by the model translation processes.

The two approaches are briefly presented in the subsections hereinafter. Studying interoperability between these two approaches is of particular interest, as they represent, as stated above, two very important and widely used families of models possible in a federation.

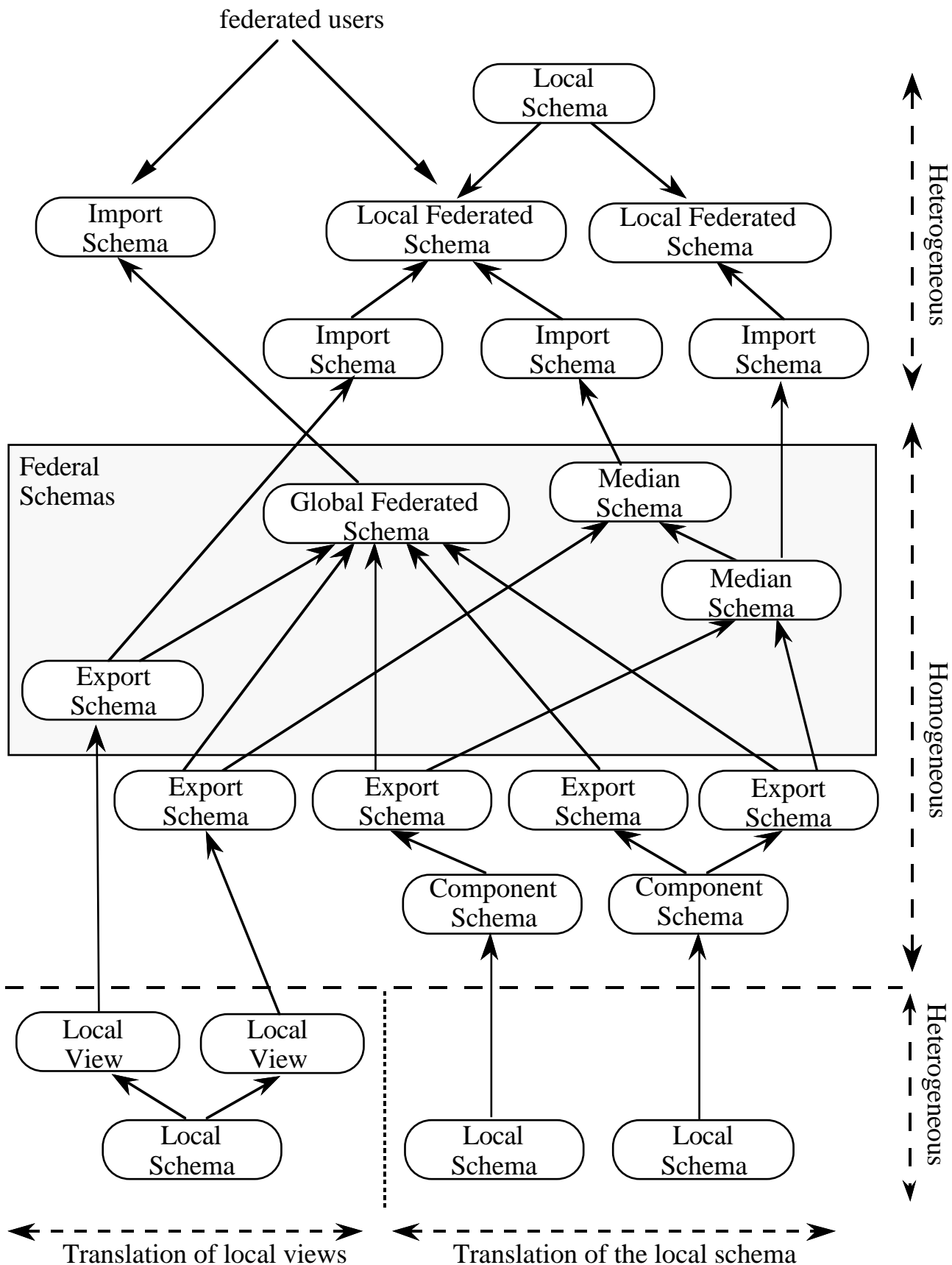
In [Sheth 90], five schema levels were proposed as a reference architecture for federated database systems, separating local pre-existing schemas, component schemas (common model counterpart of local schemas), export schemas, federated schemas (FS), and external schemas defined over federated schemas. The federated schemas hold a global dictionary with additional information about fragmentation and allocation of distributed objects. Thus, to the users of FS, both fragmentation and allocation are fully transparent. The FDBS is responsible for transforming the global queries and updates into statements for the component databases. FEMUS basically adheres to this proposal, while adding one more level to allow users to integrate various import schemas (derived from either export schemas or from federated schemas) to form their own federation. In this way ad-hoc federations may be built, enhancing the flexibility of the system from user's perspective. Figure 1 shows FEMUS six levels generic schema architecture. It includes the concept of median schema, as suggested in [Schilberschatz 90], to emphasize domain-specific federations. It also shows that export schemas may be derived as views over component schemas or directly from local views through data model translation. Finally, import schemas are derivable from any of the schemas available at the federated level: export, federated or median schemas.

A simplified architecture (Figure 2) is currently being implemented for the first exploratory prototype. It mainly includes two process types, *Translators (mappers)* and *Integrators*. The mapping processes translate schemas, integrity constraints, and language elements from one data model/language to the other (ERC+, COCOON); the integration processes combine schema and instances from the two components.

### 2.1 The ERC+ approach

A complete definition of ERC+, and a discussion of its features, may be found elsewhere [Parent 92]. Here we briefly recall the main features of ERC+:

- *entity types* bear one or more attributes. As attributes may in turn, iteratively, be composed of other attributes, the structure of an object type may be regarded as an unlimited attribute tree;
- *relationship types* may connect any number of participating entity types. As entity types, they may have attributes. They are said to be cyclic if the same entity type participates more than once in the relationship type;
- a *role* name is associated to each participation of an entity type in a relationship type. The participation is characterized by its minimum and maximum cardinalities;



**Figure 1: the generic schema architecture in FEMUS**

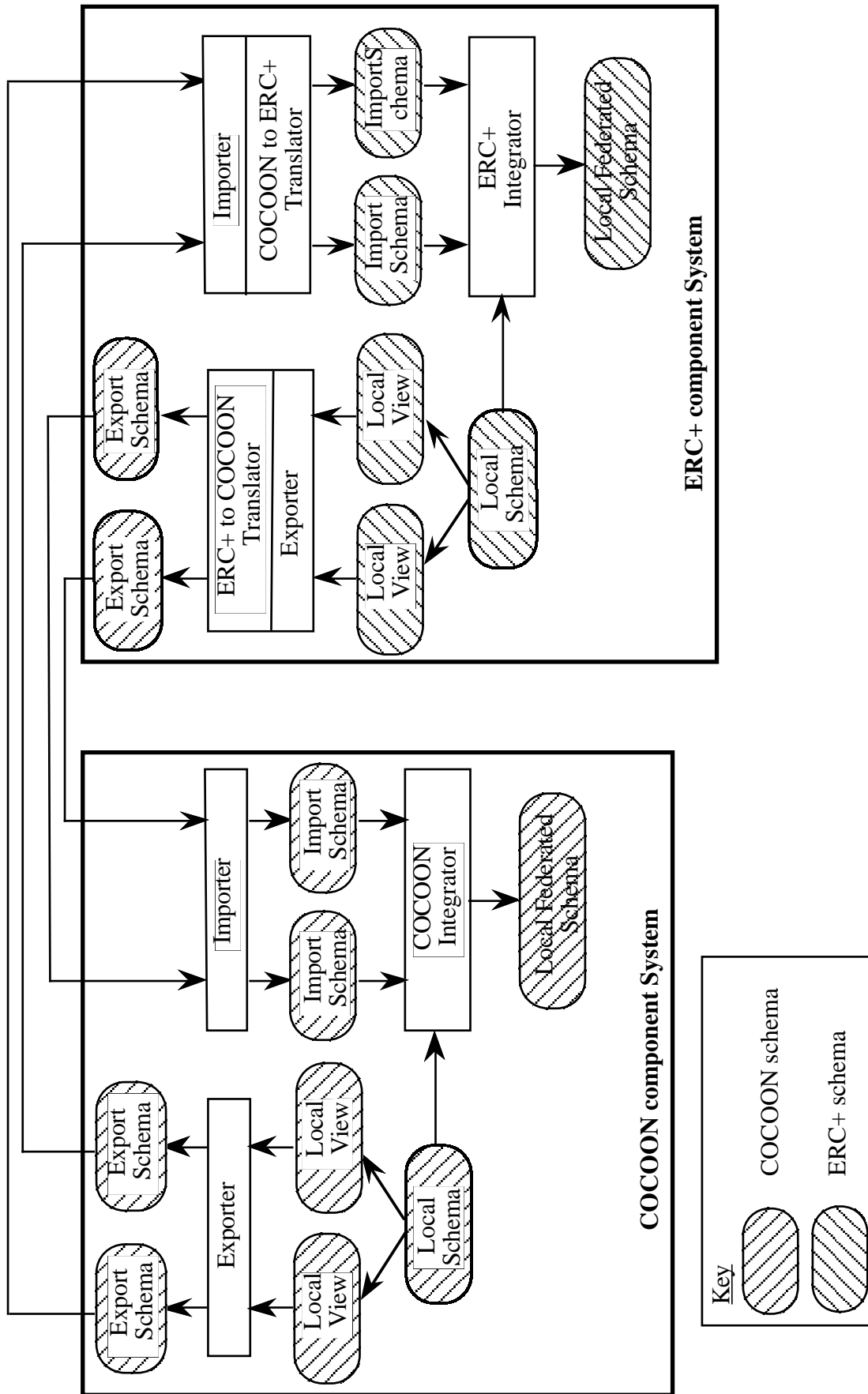
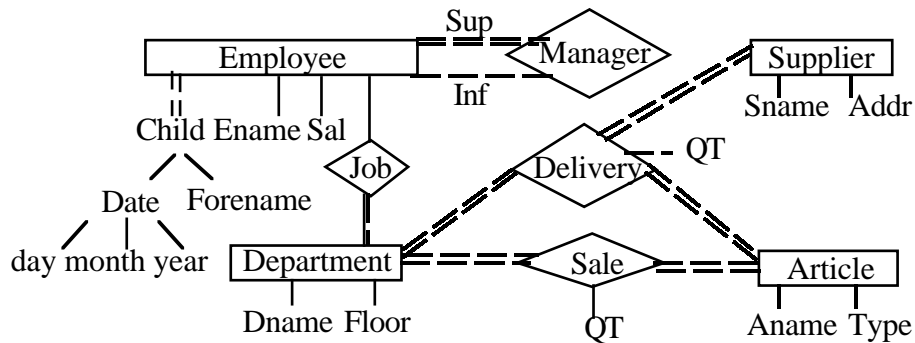


Figure 2: FEMUS prototype specific architecture

- *attributes* may be either atomic (non decomposable) or complex, i.e. decomposable into a set of component attributes, which may themselves be either atomic or complex. An attribute is also characterized by its minimum and maximum cardinalities (mandatory/optional, monovalued/multivalued). Attributes may be valued in a multiset (i.e. not excluding duplicates);
- two *generalization* relationships are supported, the classical "is-a" and an additional "may-be-a" relationships. The former corresponds to the well-known generalization/specialization construct; the latter has the same semantics, but does not require an inclusion dependency between the subtype and the type and is therefore used to describe multi-instanciation at the schema level;
- an *object identity* is associated to entities.



**Figure 3. An example ERC+ schema**

Figure 3 shows a sample ERC+ diagram. A single continuous line represents a 1:1 link (mandatory monovalued), a single dotted line represents a 0:1 link (optional monovalued), a double dotted line represents a 0:n link (optional multivalued), a double line (once dotted, once continuous) represents a 1:n link (mandatory multivalued). No generalizations appear in this particular example.

Two formal query languages, an algebra and an equivalent calculus, are associated with the ERC+ data model. The functionalities provided by the algebraic operators include: selection of entity type occurrences based on a given predicate, projection of entity type occurrences on a subset of its attributes, union of the populations of two entity types. Specific to ERC+ is the *reduction* operator, which allows the elimination of the values of an attribute which do not conform to a given predicate. Most important is the relationship-join (*r-join*, in short) operator. Let  $E_1, E_2, \dots, E_n$  be the set of entity types linked by a relationship type  $R$ , the *r-join* of  $E_1$  with  $E_2, \dots, E_n$  via  $R$  builds a new entity type (and the corresponding population) whose schema includes the schema of  $E_1$  plus an additional complex attribute, named  $R$ , whose components are the schemas of  $R, E_2, \dots, E_n$ . In some sense, this operator groups into a single entity the information scattered over entities linked by a relationship. A *spe-join* operator allows to join entity types participating into a given generalization [Spaccapietra 89], thus providing for an explicit inheritance mechanism.

Every ERC+ operation is objects preserving and creates a new derived entity type. The attributes, relationships, generalization links and population of the created type are derived from the operands types. Operations may thus be combined into expressions of arbitrary complexity.

## 2.2 The COCOON approach

COCOON is an object-function model. Its basic constituents are objects, functions, types, classes and views. The following is an excerpt from [Scholl 92]. Objects are instances of abstract types, specified by their interface operations. Functions are either retrieval functions or update methods. They are described by their name and signature. They may be multivalued. Types are described by their name and the set of functions that are applicable to their instances. A subtype hierarchy defines type inheritance relationships. Objects are instances of one or more types (multi-instanciation). Full static type-checking is supported. Classes are collection of objects (type extents). A subclass hierarchy defines class inclusion relationships. Objects are members of one or more classes. For each object class, its membertype is an associated type, and its extent denotes a set of objects of that type. Classes may be constrained by a predicate that must be satisfied by all members of the class.

A set-oriented query language, called COOL, similar to relational algebra, provides operators to build an output set of objects from input sets of objects. Query operators can be applied to extents of

classes, set-valued function results, and query results. The algebra has object preserving semantics. Queries are also used as the view definition mechanism: they introduce new virtual classes, and define their extent. Views may be defined by basic COOL operators, over other views or by composite queries.

COOL operators are: selection, projection, extend (allows the definition of new derived functions), set operators (union, difference, intersection), and generic update operators (update, insert, delete, add, remove, and set to assign return values to functions).

Figure 4 shows a COCOON type diagram describing the same universe of discourse as the ERC+ diagram in Figure 3. Arrows with single arrow head (respectively, double arrow head) represent single-valued (respectively, set-valued) functions. A function and its inverse are linked by a straight line.

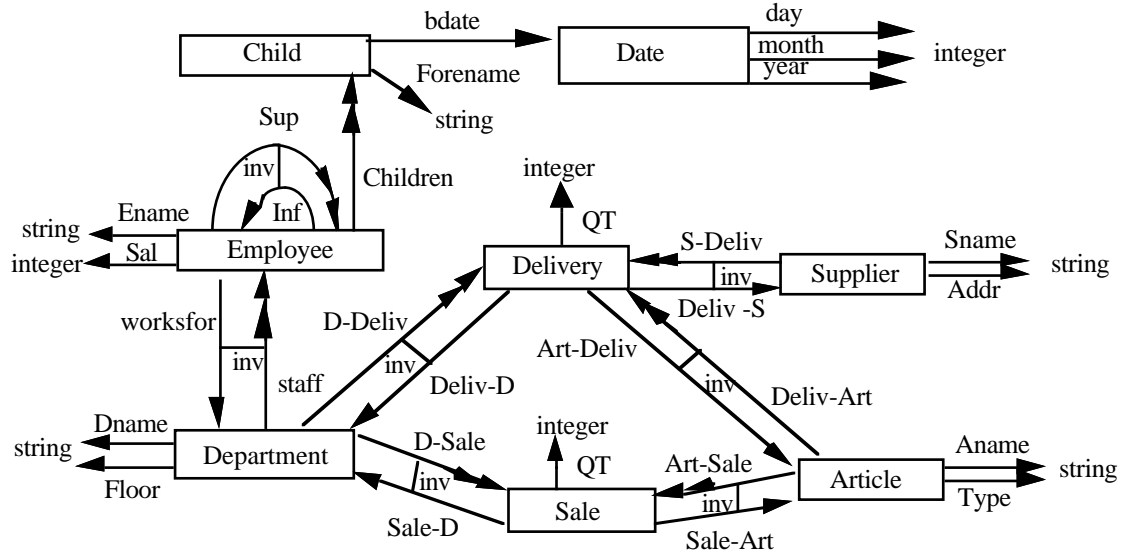


Figure 4. The COCOON equivalent to the example ERC+ schema

### 3. THE MAPPING PROCESS

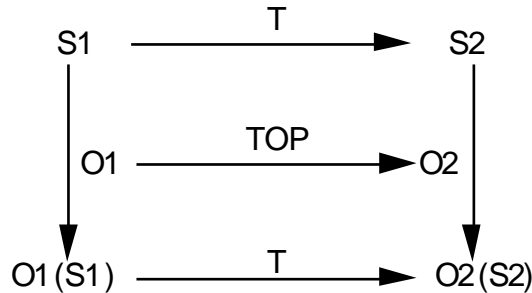
The translation process is well understood when converting from a conceptual data model to a target (logical) data model during the design of a database. In this context the main requirement is to minimize semantic loss that may occur when semantically rich modeling concepts of the source model are converted into less expressive constructs in the target model. These schema translations mainly focus on implementing data structures and access paths in the target model. By analogy to top down design methodologies, this conversion process is regarded as a "vertical translation". Since users queries during the operational phase of the database are expressed in the DML of the target model, there is generally no DML translation between the conceptual and logical models. On the contrary, cooperative DBS require bi-directional translation processes between any pair of component sites. These translations must take into consideration the reversibility and equivalence issues between two different data models of similar semantic expression power. Thus, the translations are called "horizontal and bi-directional translations". Moreover, interoperability implies that data manipulation requests will generally be issued by each component of the federation towards the other components, this also calling for a horizontal translation between the various DML in the federation.

FEMUS focuses on the horizontal translations between object-oriented and semantic data models. The following subsections discuss mappings for the ERC+/COCOON case. The discussion, however, may be easily adapted to other similar models.

As stated, interoperable mappings need to be defined in terms of both static (schemas) and dynamic (operations) aspects. Of course, these two mappings are strongly related to each other, as illustrated in Figure 5. Let *S1*, *S2* be schemas in two different data models, say model-1 and model-2. Let *O1*, *O2* denote operations of the algebras of model-1, model-2 respectively. An operation is either a single

operator or an expression. The result of an operation is both in terms of new schema elements and their instances. Let  $O1(S1)$  denote the result of applying  $O1$  to  $S1$  (similarly for  $O2(S2)$ ). Let  $T$  be a translation from model-1 into model-2 such that  $T$  transforms  $S1$  into  $S2$  and the underlying database-1 into database-2. Let  $TOP$  be a translation from model-1 into model-2 such that it transforms an operation  $O1$  into an operation  $O2$ . Then the following should hold:

$$T(S1) = S2 \wedge TOP(O1) = O2 \Rightarrow T(O1(S1)) = O2(S2)$$



**Figure 5: mappings consistency**

### 3.1 Mapping an ERC+ schema to COCOON

The major differences between the two modeling approaches are:

- ERC+ distinguishes between attributes and relationships, while COCOON treats them uniformly as functions,
- ERC+ supports the representation of a complex object as a single entity while COCOON decomposes it into a set of related objects.

Accordingly, the following schema translation rules may be stated.

#### a) Attributes

A simple attribute is mapped into a COCOON function whose domain is the COCOON object type corresponding to the attribute owner, and whose range is the COCOON primitive type corresponding to the ERC+ value domain of the attribute. The function is single-valued or set-valued depending on whether the attribute is monovalued or multivalued.

Examples:

- Ename (Figure 3) results in the Ename function: Employee  $\rightarrow$  string (Figure 4).
- day (Figure 3) results in the day function: Date  $\rightarrow$  integer (Figure 4).

A complex attribute is mapped into a new object type and its associated class. A function is added to link the owner object type to the new type.

Examples:

- Child (Figure 3) results in the Child object type. A set-valued function, named children in Figure 4, is added to link Employee to Child.
- Date (Figure 3) results in the Date object type. A single-valued function, named bdate, is added to link Child to Date.

#### b) Entity types

An entity type is mapped into an object type and its associated class. Functions are attached to the type according to attributes and relationships translation rules.

#### c) Relationship types

- 1.- Binary relationship types without attributes

A binary relationship type with no attribute is mapped into two inverse functions between the object types corresponding to the entity types participating into the relationship type. These functions are single- or set-valued depending on the cardinalities of the corresponding roles.

Example:

- Job (Figure 3) results in the two inverse functions worksfor and staff. Worksfor is single-valued as the role of Employee in Job is monovalued. Staff is set-valued as the role of Department in Job is multivalued.

## 2.- Binary relationship types with attributes and n-ary relationship types

These relationship types are mapped into a new object type and its associated class. The attributes, if any, of the relationship type are mapped into functions according to rules in a). Each role of the relationship type is mapped into a pair of inverse functions between the new type and the type corresponding to the entity type attached to the role.

Examples:

- Sale (Figure 3) results in the object type Sale (and its class), with the attached QT function: Sale  $\rightarrow$  integer, and two pairs of inverse functions, D-Sale and Sale-D, linking Sale with Department, and Art-Sale and Sale-Art linking Sale with Article.
- Delivery (Figure 3) similarly results in one new object type, Delivery, and three pairs of inverse functions.

### d) Generalization links

An is-a link between two entity types is mapped into an is-a link between the object types and classes corresponding to these entity types. May-be-a links are not mapped, as there is no such concept in COCOON at the moment.

## 3.2 Mapping a COCOON schema to ERC+

The COCOON to ERC+ mapping rises problems typical of reverse engineering situations, in which a more decomposed representation has to be translated into a more compact one. A classical example is the relational to ER translation.

A straight algorithm may be used to transform a COCOON schema into an ERC+ schema. The algorithm is driven by class and function definitions. A function from an object class to a primitive type is mapped into a simple attribute of the ERC+ element corresponding to the class. A pair of inverse functions between object classes is translated into a binary relationship between the two entity types corresponding to the related classes. A function between object classes which has no inverse is mapped as an attribute of the ERC+ element corresponding to the class where the function originates. COCOON classes which have not been translated as attributes using the above rules are mapped into ERC+ entity types, whose structure (the set of attributes) is derived from the membership type of the class. An is-a link between two object classes is mapped as an is-a link between the corresponding entity types.

The algorithm generates a valid ERC+ schema, which includes only binary relationship types with no attributes. Applied to the schema in Figure 4, it does not produce the schema in Figure 3. The consequence is that applications using the ERC+ schema will be overloaded in terms of data manipulations needed to get the desired result. For instance, assume we want a list of departments showing their deliveries in terms of quantity, supplier and article. To produce the list using the schema in Figure 3, a single r-join operation is needed. To do the same using the schema resulting from the translation of the schema in Figure 4, it will need three r-join operations (because Delivery is now an entity type linked by three binary relationship types to Department, Supplier and Article). There is no automatic solution to avoid such overloading, unless the COCOON schema is enriched with information on dependencies among its components. It is known that, if certain dependencies hold, a n-ary relationship type may be decomposed into relationship types of degree less than n. Reversing this reasoning, the translation algorithm can infer from the known dependencies if a set of pairs of inverse functions defined on the same object class can be mapped as a n-ary relationship type also representing that object class.

### 3.3 Operators Mapping

Both ERC+ and COCOON models incorporate an algebra as DML. In this section we compare the characteristics features of the two algebraic query languages in order to define a mapping (translation) of operators from one model to the other. The two algebra have in common several operations (such as selection and projection, for instance) with well known semantics. In addition, they contain operations which have no direct corresponding counterpart in the other model. Thus, an operator in one model may correspond to an algebraic expression, rather than a single operator, in the other model. The requirement for correctness of the translation is the consistency of the mappings as discussed at the beginning of section 3. The mappings relationships illustrated in Figure 5 can also be used to determine TOP. Knowing T, we can use it to translate S1 and O1(S1) into the second model, getting S2 and the result of O2(S2). We can then check how, in the second model, S2 can be mapped onto O2(S2). If there is no direct operator which does it, then we have to find the adequate expression to build the expected known result.

Hereinafter we discuss the operators mapping based on ERC+ operators.

#### Select Operator

Select operators in ERC+ and COCOON have the same semantics. They preserve the schema of their input operands. Their main difference is in the type of predicate they allow. ERC+ predicates may contain quantifiers (over set-valued attributes), whereas COCOON selections may contain nested expressions and set comparison operators.

Consider the following query:

"select all employees who earn, at least, one salary greater than 6000".

Below we give ERC+ and COCOON algebraic expressions for the queries and derive the rules for translating the operators.

#### ERC+ query

The selection operator, noted  $\sigma$  in ERC+, creates a new entity type which contains the entities (objects) that satisfy the selection predicate. The ERC+ algebraic expression corresponding to the above query is given by:

$$E1 = \sigma [\exists_{s \in \text{Sal}} (s > 6000)] \text{Employee}$$

where the existential quantifier ( $\exists$ ) is extended to apply to multisets.

#### COCOON query

In the following EmployeeC denotes the class of objects belonging to the type Employee. The above query is written in COCOON as:

$$C1 = \text{select} [\text{select} [s > 6000] (s:\text{Sal}) \neq \emptyset] (\text{EmployeeC})$$

The nested select operation implements the selection predicate "at least one salary greater than 6000" by first retrieving the set of salaries over 6000 of an employee, and then checking whether the retrieved set is empty or not.

If the above query is modified to "select all employees who have all their salaries greater than 6000", then the corresponding ERC+ algebraic expression is:

$$E2 = \sigma [\forall_{s \in \text{Sal}} (s > 6000)] \text{Employee}.$$

The universal quantifier included in the predicate can be expressed in different ways in COCOON by using nested select operations and set comparison operations. The resulting queries are:

$$C2 = \text{select} [\text{select} [s > 6000] (s:\text{Sal}) = \text{Sal}] (\text{EmployeeC}).$$

$$C2' = \text{select} [\text{select} [\text{NOT} (s > 6000)] (s:\text{Sal}) = \emptyset] (\text{EmployeeC}).$$

The second query is obtained from the first one by the application of the standard transformation of the universal quantifier into the existential one:  $\forall x (P) \Leftrightarrow \neg \exists x (\neg P)$ .

In summary the following rules can be derived for converting selection operations between ERC+ and COCOON.  $P(a)$  is a predicate on variable  $a$ , and  $a$  is a variable on a multivalued attribute  $A$ . Equivalence rules between ERC+ and COCOON selection predicates:

$$\exists a \in A (P(a)) \Leftrightarrow \text{select } [ P(a) ] (a.A) \neq \emptyset$$

$$\forall a \in A (P(a)) \Leftrightarrow \text{select } [ \text{NOT } P(a) ] (a.A) = \emptyset$$

## Project Operator

### 1. Projection over a simple attribute

Project operators in ERC+ and in COCOON have equivalent behavior when they are applied to simple attributes. In essence, they are used to drop one or more simple attributes from the schema of an entity or object type.

The translation rule in this case is trivial. For instance:

$$\begin{array}{ccc} \text{ERC+} & & \text{COCOON} \\ \pi [\text{Ename}] \text{Employee} & \Leftrightarrow & \text{project } [\text{Ename}] \text{EmployeeC} \end{array}$$

### 2. Projection over complex attributes

As discussed in section 3.1 above, the translation of ERC+ complex attributes leads to the decomposition of an object over several objects. Therefore the translation of an ERC+ projection over a component of a complex attribute requires that the corresponding COCOON object type be augmented with a derived function which links it directly to the desired attribute. Derivation is through composition of COCOON functions. For instance, a projection over the attribute Forename of the complex attribute Child (Figure 3) is mapped as follows:

$$\begin{array}{ccc} \text{ERC+} & & \text{COCOON} \\ \pi [\text{Child} \cdot \text{Forename}] \text{Employee} & \Leftrightarrow & \text{project } [\text{forename}] \\ & & (\text{extend } [\text{forename} = \text{Forename}(\text{Child})] ) \text{EmployeeC} \end{array}$$

## Product Operator

A product operator in ERC+ is useful for linking two entity types which are not linked by any relationship type. In the case where a relationship exists between the two entity types, it is simply ignored by the product operator. The operation extends every occurrence of its first operand with the set of all the occurrences of the second operand (represented in the schema of the resulting entity type as a multivalued complex attribute whose components are the attributes of the second operand). For example, a product will be applied to the entity types Employee and Supplier of Figure 3 as a first step to check whether there are employees having the same name as a supplier.

The result of this product operation is an entity type, say ES, with all the original attributes from Employee plus an additional complex attribute supplier, which is composed of Sname and Addr. The check is then done through a selection on the result of the product. The ERC+ selection predicate is:

$$\exists n \in \text{supplier} \cdot \text{Sname} (n = \text{Ename}).$$

The result of the product operation can be achieved in COCOON by extending the object type Employee with a new function to create the link to the object type Supplier.

## Relationship-Join (R-Join)

The R-join transforms a relationship type into a complex attribute structure with respect to one of its participating entity types. Consider the two entity types Department and Article, and the relationship type Sale in Figure 3. The result of applying R-join to Department through the relationship Sale is an entity type with all the attributes of Department (Dname, Floor) and an additional complex and

multivalued attribute named Sale, whose attributes are Qt and Article, the latter being a complex attribute with Aname and Type as components.

As the product operator, the relationship-join is used to merge information from various entity types into a single entity type. While product merges systematically each entity of one type with all entities of the other type, R-join uses as merging criteria the fact that the entities forming one occurrence in the result are linked by a relationship of the given type.

To some extent, we could state that R-join builds more complex entity types from the existing ones. COCOON objects are not complex objects. Because of decomposition rules of the schema mapping, the result of an R-join, translated in COCOON, would generate the same objects and functions as those already there. Therefore, the translation of an R-join is the identity operation.

#### 4. THE INTEGRATION PROCESS

Database integration is the second key feature in building integrated database services in an interoperable environment. The autonomy goal inherent to FDBS requires that new integration techniques be developed to cope with all possible discrepancies among component databases, without altering them, while providing for maximum integratability. Although schema integration has already been investigated for a long time [Batini 86], existing methodologies lack the power to integrate schemas showing structural conflicts, i.e. to solve situations where, for instance, the same real world object is represented as an attribute in one schema and as an object in another schema [Spaccapietra 91]. These methodologies need a conforming step, prior to integration, such that all structural conflicts are removed from the input schemas. The conforming process relies on schema modification, a consequence which contradicts the primary FDBS requirement: to ensure continuation of local usage of data without any user visible impact due to the new federation services.

Moreover, current methodologies do not really handle data model heterogeneity. They only propose to translate every input schema in some common model within a pre-integration step. This is consistent with the usual approach to federated or distributed heterogeneous databases, but might become a bottleneck in more flexible architectures in which multiple federations, although defined on the same component databases, are not necessarily built upon the same data model.

Integration in FDBS is a bottom-up task, that must combine databases that may have existed for a long time. These databases may already have a considerable amount of data. Thus, integration in federated databases must cover both tasks, the integration of schemas and of existing objects. This is usually called database integration, in contrast with view integration where only schemas with no associated extensions are integrated. A relatively small amount of work concerns database integration. One recent exception is the Pegasus project [Ahmed 91, Krishnamurthy 91]. Kent emphasized the separation between real world entity objects and their database counterpart, called proxy objects [Kent 91]. The latter ones represent entities in different component databases. Object integration must deal with the fact that due to historical evolution of databases, the same real world entity may be stored as different database approximations in different databases (the proxy objects). Thus, the FDB administrator must also specify what proxy objects of what component databases represent the same real world object and under what circumstances.

FDBS integration requirements also differ from DDB requirements. In DDB systems, integration is performed once, taking as input the schemas of existing databases and producing as output the global schema of the DDB. As there is no mandate to preserve site autonomy, local pre-existing databases may be modified to make integration easier or to redefine data allocation. For instance, if the same data appear in more than one local database, the DDB administrator may just keep one copy and have the other ones deleted to avoid the data replication problem. In FDBS, integration may be performed at *different levels*, depending on the organization's approach to FDBS architecture (cf. Figure 1b). As in DDBS, it might be the task of the FDB administrator (for each FDB being built). It might as well be a process performed by end users, if they are given the ability to import data from various sources (whether directly from local databases or through a FDB) to build their own, single user FDB.

Integration is a two-fold process. First, syntactic as well as semantic relationships among elements in the input schemas have to be precisely identified. This is the investigation phase. A first stream of research developed methods and tools to assist the database administrator in identifying interschema relationships [Sheth 88]. In the second phase, the integration phase, related elements have to be

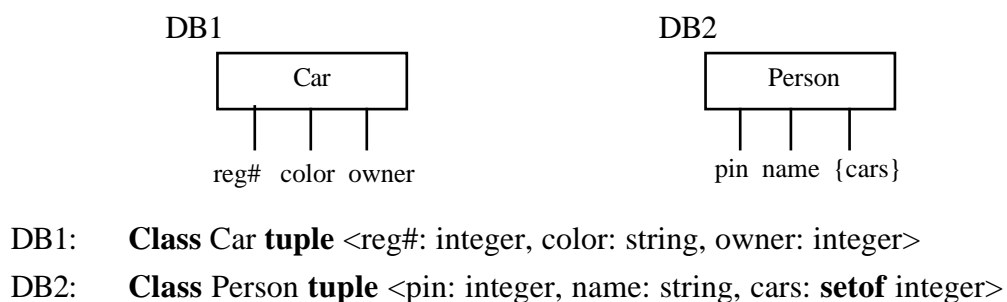
"integrated". Various techniques have been proposed for this purpose. The first approach is manual integration, where the DBA is provided with a schema restructuring language. The latter allows the DBA to direct the integration process towards the step by step construction of the integrated schema the DBA is aiming at. The integrated schema is seen as a superview defined over input schemas [Motro 81]. This approach is relatively simple to implement, as the functionality supported by the system is limited to executing the restructuring operations. It is, however, of poor user friendliness and badly suited for the non expert users which might be allowed to build federations in a flexible FDBS.

To cope with these inconveniences, more powerful assertional approaches have been proposed. They are intended to automatically perform integration from input correspondence assertions, which instruct the integrator tool on which interschema relationships exist. Assertional techniques provide a higher level of service to their users (users just have to care about existing correspondences, not about how corresponding items may be merged to form the integrated schema). They build the integrated schema (and the associated mappings to/from source schemas), using established integration rules which allow to solve all types of conflicts supported by the tool.

Finally, rather than building a new integrated schema, a third approach performs integration by extending input schemas (and databases) with the additional interschema descriptions. These either record correspondence information, or add new constructs to relate one element in a schema to another elements in another schema [Scholl 92].

FEMUS currently focuses on the integration phase. We discarded the manual superview approach as contrary to FDBS flexibility goal. As for the two other approaches - the integration assertional technique and the augmentation technique described just above - no definite assessment has evaluated and compared their pros and cons. Intuitively, building an integrated schema seems preferable if there is a heavy overlap among component schemas, with many component elements resulting in a single element after the merging. On the other hand, if the component databases have complementary content, augmenting existing schemas with interschema references is a simpler process. To know more about the comparison we have decided to investigate and experiment both techniques in parallel. They are hereinafter separately presented.

To show differences and/or similarities between these two integration techniques, we illustrate their usage on the same very simple example (Figure 6 below) modeled according to object-oriented notations. The example assumes a FDB environment with two databases, DB1 and DB2. DB1 contains information about cars (the class Car, with three attributes: the car's registration number and color, and the person identification number of the car's owner), whereas DB2 holds persons (the class Person, with three attributes: the person's identification number and name, and the registration numbers of the cars (s)he owns). Since cars are owned by persons, and persons own cars, there is some sort of inter-database correspondence. We show below how this correspondence is specified using the assertion-driven integration, and how it is done through the augmentation approach.



**Figure 6: the example integration case**

As stated, the definitions of the two classes do not bear any indication of their interrelationship. It is assumed that the FDBA has the external knowledge about the semantics of the attributes and the object classes being described. (S)he is therefore responsible for directing the integration tool through explicitation of that knowledge. In a repository environment, the definitions would be complemented with some natural language descriptive information. For instance, the owner attribute could be described as: this attribute holds the person identification number of the person who owns the car. An investigation tool could then guess that there might be some relationship between the two classes, due to the fact that the term "person" appears in both descriptions. Nevertheless, the FDBA would have to

be prompted anyway to confirm or correct the proposed correspondence. Note that the guess made by the tool is only in terms of structural correspondence, unless both databases are accessed to check the corresponding populations.

#### 4.1 Assertion-driven integration

The EPFL team has proposed a new assertional method to integrate heterogeneous source schemas [Spaccapietra 92]. The method solves structural, semantic and descriptive conflicts without changing the input schemas. It is also able to directly integrate heterogeneous schemas, without going through a preliminary translation step. To this purpose, interschema correspondence assertions, stated by the DBA, may relate an element whatsoever in one schema to an element of any type in another schema. An assertion defines the relationships between the element "types" (structural description) and the related "classes" (sets of associated instances). It also includes the necessary definition of an object mapping at the instance level, which provides for the integration of the related databases. Integration rules are generically defined on an abstract "generic data model" (which basically supports objects, value attributes and reference attributes), but their implementation in the actual integration of two input schemas is tailorable to the specific features of the input data model.

Considering the example in Figure 6, and assuming that the two schemas describe exactly the same sets of cars and persons (for every car seen by DB1, its owner is seen by DB2, and vice versa), the interrelationship between the two schema would be first described through two element correspondence assertions:

Car $\equiv$ cars	with corresponding attributes	reg# = cars
owner $\equiv$ Person	with corresponding attributes	owner = pin

The first assertion states that the set of cars described by the object type Car in DB1 (the real world state of Car) is the same ( $\equiv$ ) as the one described by the cars attributes of the Person object type in DB2. The set equality is between sets of real world objects, and holds independently of their representation in the two schemas. The "with corresponding attributes" clause describes the structural relationship between the two representations. In this case there is only one information about cars which is represented in both databases: the car's registration number. This is hold by the reg# attribute in DB1 and by the cars attribute itself in DB2. Hence the stated equality of the two attributes. The same considerations apply to the interschema relationship between DB1 car owners and DB2 persons.

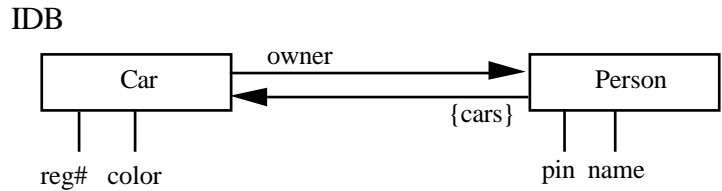
The fact that associated elements in one schema (Car and owner in DB1) are equivalent to associated elements in another schema (cars and Person in DB2) does not necessarily imply that the association has the same semantics in both schemas. It could be the case that DB1 talks about ownerships, and DB2 talks about cars being driven by persons, while still referring to the same real world state for cars and persons. Therefore, one assertion remains to be stated: a path correspondence assertion, making explicit that the link between cars and owners in DB1 (noted Car — owner) has the same semantics as the link between persons and car in DB2 (noted Person — cars). Links are bi-directional: Person — cars and cars — Person denote the same DB2 link. The path correspondence assertion for the example simply is:

Car — owner  $\equiv$  cars — Person

The integrator tool, with the two schemas and these three assertions as input, will generate the following integrated schema:

```
IDB:  Class Car tuple <reg#: integer, color: string, owner: reference Person>
      Class Person tuple <pin: integer, name: string, cars setof: reference Car>
```

The corresponding diagram may be drawn as follows, with labeled arrows representing reference attributes:



**Figure 7: the integrated schema**

IDB goes with the mapping information which states that the Car class is to be found in DB1, the Person class is to be found on DB2, and the references in between have to be evaluated through the matching criteria  $\text{Car} \cdot \text{reg\#} = \text{Person} \cdot \text{cars}$  and  $\text{Car} \cdot \text{owner} = \text{Person} \cdot \text{pin}$ . Mapping information supports transformation of user queries against IDB into queries on the underlying DB1 and DB2 databases.

## 4.2 Integration through augmentation

The ETHZ team has developed an approach to provide a flexible way to specify the correspondence between existing objects of different databases. This is achieved by defining (global) object identity in terms of algebraic (extend) views [Scholl 92]. We thereby make use of COCOON's view definition facility [Scholl 91] that was extended such that it can span over multiple databases. It includes mechanisms for linking objects across systems and to deal with semantic conflicts. The necessary view definition method is to extend the local schema by elements of the schema of another system. Consider the example in Figure 6. The DBA will start the integration process by defining a view "Cars" as an extension of Car, with an additional function owned-by that returns for each car-object the person-object of the other database, that owns that car. This definition is stated as follows:

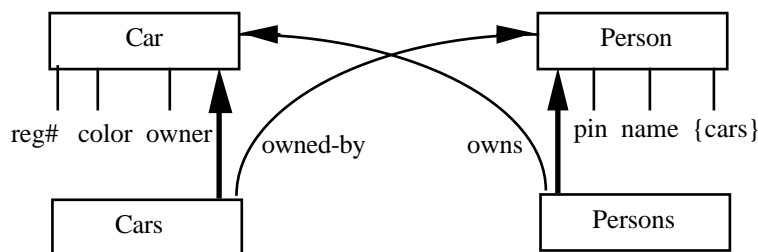
**define view Cars as extend** [ owned-by:= select [ reg#(c) ∈ cars(p) ] ( p:Person ) ] ( c:Car ).

The selection predicate in the view definition materializes DBA's knowledge of the fact that the attribute cars of class Person specifies registration numbers of cars, which are recorded as values of the reg# attribute of Car in the other database. Knowledge that, similarly, the attribute owner of Car specifies a pin of a person in the other database, leads to similar extension of the Person type in DB2:

**define view Persons as extend** [ owns:= select [ pin(p) = owner(c) ] ( c:Car ) ] ( p:Person ).

The additional function owns returns the cars owned by each person. It acts as the inverse of owned-by in DB1.

Figure 8 below shows the schema of both databases, extended with views having functions that lead from one database to the other.



**Figure 8: the augmented schemas**

The advantage of this approach is that it uses almost exclusively the expressive power of a query language, together with a view definition facility. The only needed extension is a global object identity predicate. This is necessary, since, due to demand of local autonomy, we are not allowed to make any assumption on the local internal object-identities. Consequently, no change is required in the implementation of local database systems.

While from a conceptual point of view this approach looks simple and clean, more effort remains to be spent on the implementation issues. Particularly, an efficient, but overridable, implementation of the identity test seems to require further investigations. Till now, comparing object identities is

typically hardcoded into the implementation of local object based management systems. Efficient support by indexes or some other form of replicated information is needed.

## **5. NEGOTIATION**

In a federated system, it is the responsibility of the local DBAs to decide and define which parts of their local data is available to external users (users from other sites). The decision process relies on negotiation with the other DBAs, for which specific tools may be developed within the federated system [Heimbigner 85]. Negotiation needs an understanding of the semantics of data. This is usually supported by information attached to data descriptions and stored in a repository. The repository manager provides browsing capabilities and facilities to support dialogue and explanations among DBAs. Negotiation also settles an agreement on access restrictions and the desired level of consistency between imported materialized views, and the original version in the exporting system (as discussed below).

### **5.1 Exchanging metadata**

The specification of exported data is stored in an export schema (or export view) in the exporting system. The export schema provides access to both the data definitions (description of types, classes, ...) and the corresponding instances (the objects in the database). Export schemas may be defined using different techniques: subschemas, views, virtual objects,... Usage permissions granted (read, update, ...) are also specified.

Export schemas may be made available to designated users only (a usage permission is attached to the view itself), or they may be broadcasted (available to everybody), or designed to become a component of some designated federation. It is worthwhile noting that data is exported to become part of one or more federations (thus providing for location transparency), not for direct access as in multidatabase systems [Litwin 84]. If a local system accesses a federation it contributed to, the system is importing data it exported. From the semantic perspective this is perfectly correct (similar to what happens everyday in international trade). From the performance point of view, it is expected that, as in DDBSSs, the federated query processors will be able to access local data directly, rather than access some copy elsewhere.

### **5.2 Exchanging data**

A well-known problem in moving objects around in a federated system relates to object identity. The requirement clearly is that object identity has to be preserved (to avoid semantic loss). This implies that, when an object comes back to its exporter, updated by some importer, it is recognized by the exporter so that the update can be correctly applied to the local database.

A first problem obviously comes from the fact that the federation may include value based systems (relational DBMSs), which do not know about object identity. However, an object-oriented system can also offer values to the federation, instead of objects, either because it generally does not extract object identities, or because some type of queries it accepts generate values as result. If object identities can be provided by component systems, a global identity is easily built [Heimbigner 85, Eliassen 91]. It will include the local identity, which allows to later apply external updates on the original exported object. If no local identity is available to the federation, exported values can be turned into objects by the FDBS, but this does not make sure that an external update will be correctly propagated. To that extent, restrictions have to be imposed on exported values: i.e., they must contain a key which uniquely identifies the value in the local database (as in relational databases). Otherwise, export should be restricted to read only import.

When an imported object is stored by the importer, it gets a local object identity. If the object might have to be returned to the exporter, the global object identity (which is locally meaningless) has to be stored as an additional attribute of the object (not visible to users), for future reuse in the federation layer. An object to be imported may contain object identities as references to component objects in the exporting database. If the component objects are not imported, references are just dropped before import. If a component object is imported as a component value in the importing system, the reference has to be replaced in the imported copy with the value of the corresponding object. If the component

object is imported as an object, new references, local to the importing system, should complement the imported ones (or replace them, if the importer is not allowed to update the object).

## 6. IMPLEMENTATION ISSUES

Since object identifiers in general are not available outside a system, the specification of information to import has to be done by means of algebraic expressions [Scholl 92]. Also exported information can be specified comfortably with a query expression. The query is given a name whose value is the result of the query, i.e., a data definition and a set of instances. At each use of the name the query can be re-executed in the exporting system and the result transmitted to the importer. To the user there is only one version of the information located in the exporting system, and up to date. A problem arises when the amount of data specified by the expression is large, the data transfer might take considerable time. To avoid the delay, the user might want to create a local copy, thus he explicitly stores the result of the import expression at his own site. Now, on the other hand, the importer does not know whether his local copy is up to date or not, since updates on the original version are not by default propagated to his copy. As a compromise, it would be possible to refresh the copy of the importer at some access occasions, but not all.

Another solution to the delay problem would be to keep an automatically updated local copy that the user never sees. The problem with this approach is of course how to ensure the consistency between the copy and the original. The following is a summary of techniques used to solve materialized view management problems. Materialized view update strategies can be classified according to, first, the consistency that is required between the original and the materialized view, secondly how to detect that an update to the original is relevant to a materialized view, and lastly how to actually update the view.

### 6.1. Consistency requirement

The strongest consistency requirement is *transactional consistency*, i.e., the materialized view is updated before the update transaction on the base has terminated. This implies total correspondence between the versions [Blakely 86]. The consistency requirement may be relaxed in a well-defined way [Alonso 89] where the notion of quasi-copy is introduced. A quasi-copy may deviate from the original in one of the ways:

- Time delay
- Version number
- Numeric deviation

A way to specify these deviations exactly and to calculate the export costs is proposed. An even weaker consistency requirement is proposed by [Lindsay 86], where a "snapshot" is a copy that is updated only periodically, to be used by update-insensitive applications.

### 6.2. Detection of relevant updates

An update operation on the database is made on a set of objects specified by a query. If these objects are also present in a materialized export view, the view has to be updated, otherwise not.

Thus the problem is to say whether the sets of objects specified by two queries are disjoint or not, a problem that is in general impossible to solve. It is solvable for queries involving only the project or join operators, and the select operator with a condition predicate containing only conjunctions. This is used by [Blakely 86] to obtain transaction consistency. [Elkan 89] improves the method. A different approach is made in [Lindsay 86] to update snapshots. A "snapshot" is the stored result of a view definition expression. A snapshot is read-only and is updated "periodically", i.e. all updates on the database since the last snapshot are propagated. Here a timestamp is added to every tuple when it is updated in order to be able to say if the tuple object has been updated since the last snapshot refresh.

### 6.3. Differential Refresh

When an update has been found relevant to a view, the view materialization has to be updated. One way to do this is to recompute the query specifying the view and to update the whole query. This clearly may result in the rewrite of a lot of unchanged data. It is desirable to update the view only with

the data actually changed. [Lindsay 86] proposes a differential refresh algorithm for snapshots. When the snapshot is to be updated, the tuples of the base table are traversed, and if a tuple is younger than the snapshot and satisfies the view condition it is transmitted to the snapshot. Improvements are made by discarding clusters of not interesting tuples. [Blakely 86] proposes a different method based on the application of the same update operations on the view materialization, that were applied to the base table. This method relies on the distributivity properties of set union and set difference, which do not hold in the relational model, but hold for object oriented models. The method is used to provide transaction consistency, since the view update is included last in the update operation transaction of the base table. The authors claim that their results are also valid for snapshot update, i.e., when the view update is made after the transaction updating the original table has terminated. It is not evident that the application of an update operation at a time point when the execution environment may have changed will produce the same database state as in the original environment.

Another way to improve system efficiency at update, and to avoid the exporter bottleneck problem that occurs when a lot of importers are requesting updates from the same exporter, is proposed in [Kang 91]. The idea in this method is to let the importing systems re-export information to other importing systems, thus relieving the original exporter from work.

There are cases when keeping a local copy at the importing site is necessary for performance reasons. This copy can be invisible to the user, giving the impression that there is only one version of the information. The consistency requirement varies, depending on application requirements, from periodically updated snapshots, over quasi-copies to transaction consistency. The results in the referenced papers concerns the relational model, it remains to be studied whether they are also applicable to semantic and object-oriented models. Nothing is however pointing in any other direction, on the contrary, it seems like further problems encountered in the relational model do not occur in this environment. For example, the problem caused by the fact that projection is not distributive over difference in the relational model, does not occur in a model with object identities, see [Blakely 86].

An interesting approach seems to be to combine the periodic update with the differential refresh method that applies the update operations to the view. To update the view at every modification of the original data even if the materialized view is not used at the moment seems inefficient, instead the update should be made on request from the view user, then all the updates on the base table can be applied at one time. An open question is if the update operations applied in the same order will produce the same result at the later time point. Another question is how the mechanism to update materialized views can be made symmetric, i.e., how updates on the materialized view can be propagated to the original.

## **7. CONCLUSION AND FUTURE RESEARCH**

In this paper we have presented the motivation and the key features of the ongoing FEMUS research project. The aim of the project is to explore architectural and semantic related issues of federated database systems. We have discussed, in particular, two of the most distinguishing features of the project: data model translation and database integration.

Cooperation between heterogeneous systems can involve bi-directional translations between data models of equivalent semantic power. Semantic data models and object oriented data models are more and more included in current federated database systems. In order to avoid a proliferation of translators in the federation, they are often used as common data model. We have examined in detail the characteristics of the translation process between object oriented and extended semantic data models. The translations rules between two representative models, ERC+ and COCOON, are given as an example. Their scope covered both schema and operation mapping.

Next, we have investigated database integration issues in interoperable systems. Integration techniques are used in these systems to build unified views of information imported from different databases. The main thrust of our research is to extend to federated database systems two alternative integration methodologies we have developed. The first one uses interschema correspondence assertions and integration rules to derive an integrated schema from a set of heterogeneous input

schemas. The second one uses the existing view mechanism to augment the input schemas with interschema correspondences expressed as constructs of the model (interschema functions in the case of COCOON). We presented the key features of these integration methodologies and an example to support their comparison.

Our future research effort is on:

- the specification and identification of tools to aid the translation process. The translation rules can serve as a basis for defining a rule based generator for the translation process;
- the specification and implementation of a federation server to aid federation users in sharing information, meta information, and schema construction tools (translators and integrators). The server must provide sophisticated dictionary and/or directory look up services to aid navigation and negotiation throughout the federation;
- the extension of the assertional integration technique to solve more cases of schema discrepancies and the extension of the augmentation integration technique to cope with heterogeneous input schemas;
- the extension of both integration techniques to the integration of object-oriented methods.

## ACKNOWLEDGMENTS

The authors are indebted to the following people, that contributed to the FEMUS project: Hans-Jörg Schek (ETH Zürich), Marc Scholl (University of Ulm), as well as the students Amadou Fall (EPF Lausanne) and Hans-Peter Waldegger (ETH Zürich) who did some of the prototype implementation.

## REFERENCES

- [Ahmed 91] Ahmed R., et al., "The Pegasus Heterogeneous Multidatabase System", IEEE Computer, 24(12), December 1991.
- [Alonso 89] Alonso R., Barbara D., "Negotiating Data Access in Federated Database Systems.", 5th International Conference on Data Engineering, Los Angeles, February 6-10, 1989
- [Batini 86] Batini C., Lenzerini M., Navathe S.B., "A comparative Analysis of Methodologies for database schema integration", ACM Computing Surveys, 18(4), December 1986
- [Blakely 86] Blakely J. A., Larson P-A., Tompa F. W., "Efficiently updating materialized views", ACM SIGMOD International Conference on Management of Data, May 1986
- [Bouzeghoub 91] Bouzeghoub M., Metais E., "Semantic Modelling of Object-Oriented Databases", 17th International Conference on Very Large Data Bases, Barcelona, September 3-6, 1991
- [Demurjian 88] Demurjian S., Hsiao D., "Towards a better understanding of data models through the multilingual database system", IEEE Transactions On Software Engineering, 14(7), July 1988
- [Eliassen 91] Eliassen F., Karlsen Randi., "Interoperability and Object Identity", ACM SIGMOD Record, 20(4), December 1991
- [Elkan 89] Elkan C., "A Decision Procedure for Conjunctive Query Disjointness", 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Philadelphia, March 29-31, 1989
- [Heimbigner 85] Heimbigner D., McLeod D., "A federated architecture for information management" ACM Transactions on Office Information Systems, 3(3), 1985

- [Kang 91] Kang H., Son S.H., "A Hierarchical Export/Import Scheme for Data Sharing in a Federated Distributed Database System", International Symposium on Database Systems for Advanced Applications, Tokyo, April 1991.
- [Kent 91] Kent W., "The breakdown of the information model in multi-database systems", ACM SIGMOD Record, 20(4), December 1991.
- [Krishnamurthy 1991] Krishnamurthy R., Litwin W., Kent W., "Language features for interoperability of databases with schematic discrepancies", ACM SIGMOD International Conference on Management of Data, Denver, May 29-31, 1991.
- [Kim 89] Kim W., "Research Directions for Integrating Heterogeneous Databases", Workshop on Heterogeneous Database Systems, Chicago, December 11-13, 1989
- [Lindsay 86] Lindsay B., Haas L., Mohan C., Pirahesh H., Wilms P., "A snapshot differential update algorithm", ACM SIGMOD International Conference on Management of Data, May 1986
- [Litwin 84] Litwin W., "MALPHA: A relational multidatabase manipulation language", 1st IEEE Conference on Data Engineering, Los Angeles, April 24-27, 1984
- [Motro 81] Motro A., Buneman P., "Constructing Superviews", ACM-SIGMOD International Conference on Management of Data, Ann Arbor, April 29-May 1, 1981
- [Parent 84] Parent C., Spaccapietra S., "An entity-relationship algebra", 1st IEEE International Conference on Data Engineering, Los Angeles, April 1984.
- [Parent 92] Parent C., Spaccapietra S., "ERC+: an object based entity-relationship approach", in Conceptual Modelling, Databases and CASE: An Integrated View of Information Systems Development, P.Loucopoulos, R.Zicari Eds., John Wiley, 1992
- [Perl 89] Perl Y., Geller J., Neuhold E.J., Turau V., "The Dual Model for Object-Oriented Databases", Research Report CIS-91-30, New Jersey Institute for Technology
- [Schek 86] Schek H.-J., Scholl M.H., "The relational model with relation-valued attributes", Information Systems, 11(2), 1986.
- [Schek 90] Schek H.-J., Paul H.-B., Weikum G., "The DASDBS project: Objectives, experiences, and future prospects", IEEE Transactions on Knowledge and Data Engineering, 2(1), 1990
- [Scholl 87] Scholl M.H., Paul H.-B., Schek H.-J., "Supporting flat relations by nested relational kernel", 13th International Conference on Very Large Data Bases, Brighton, September 1987
- [Scholl 91] Scholl M.H., Laasch C., Tresh M., "Updatable Views in Object-Oriented Databases", 2nd International Conference on Deductive and Object-Oriented Databases, Munich, December 1991
- [Scholl 92] Scholl M.H., Schek H.-J., Tresch M., "Object Algebra and Views for Multi-Objectbases", International Workshop on Distributed Object Management, Edmonton, Canada, August 1992
- [Sheth 88] Sheth A., Larson J., Cornelio A., Navathe S., "A tool for integrating conceptual schemas and user views", 4th IEEE International Conference on Data Engineering, Los Angeles, February 1-5, 1988
- [Sheth 90] Sheth A., Larson J., "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases", ACM Computing Surveys, 22(3), September 1990

[Schilberschatz 90] Schilberschatz , Stonebraker M., Ullman J.D., "Database Systems: Achievements and Opportunities", ACM SIGMOD Record, 19(4), December 1990

[Spaccapietra 89] Spaccapietra S., Parent C., Yétongnon K., AbaidiM.S., "Generalizations: a Formal and Flexible Approach", 1st Conference on Management of Data, Hyderabad, India, November 23-25, 1989

[Spaccapietra 91] Spaccapietra S., Parent C., "Conflicts an Correspondence Assertions in Interoperable Databases", ACM SIGMOD Record, 20(4), December 1991

[Spaccapietra 92] Spaccapietra S., Parent C., Dupont Y., "Model-Independent Assertions for Integration of Heterogeneous Schemas", Very Large Data Bases Journal, 1(1), July 1992