

# Interoperation of Independent, Heterogeneous and Distributed Databases. Methodology and CASE Support: the InterDB Approach

Ph. Thiran, J-L. Hainaut, S. Bodart, A. Deflorenne, J-M. Hick

University of Namur, Institut d'Informatique, rue Grandgagnage, 21, 5000 Namur, Belgium  
jlh@info.fundp.ac.be

## Abstract

*Accessing and managing data from several existing independent databases pose complex problems that can be classified into platform, DMS, location and semantic levels. This paper describes a general architecture, a methodology and a CASE environment intended to address the problem of providing users and programmers with an abstract interface to independent heterogeneous and distributed databases. The architecture comprises a hierarchy of mediators that dynamically transform actual data into a virtual homogeneous database. Each layer provides a certain kind of independence. The InterDB approach provides a complete methodology to define this architecture, including schema recovery through reverse engineering, database integration and mapping building. The methodology is supported by the DB-MAIN CASE tool that helps to generate the mediators and their repository.*

## 1. Introduction

Most large organizations maintain their data in many distinct independent databases that have been developed at different times on different platforms and DMS (Data Management Systems).

The new economic challenges force enterprises to integrate their functions and therefore their information systems and the databases they are based on. In most case, these databases cannot be replaced by a unique system, nor even reengineered due to the high financial and organizational costs of such a restructuration. Hence the need for interoperation frameworks that allow the database to be accessed by users and application programs as if they were a unique homogeneous and consistent database.

Accessing and managing data from such heterogeneous databases pose complex problems that can be classified into platform, DMS, location and semantics level [2]. The platform level copes with the fact that databases reside on different brands of hardware, under different operating systems, and interacting through various network protocols. Leveling these differences leads to platform independence. DMS level independence allows programmers to ignore the technical detail of data implementation in a definite family of models. Location independence isolates the user from knowing where the data reside. Finally, semantic level independence solves the problem of multiple, replicated and conflicting representations of similar facts.

Current technologies such as de facto standards (e.g. ODBC and JDBC), or formal bodies proposals (e.g. CORBA), now ensure a high level of platform independence at a reasonable cost [12], so that this level can be ignored from now on. DMS level independence is effective for some families of DBMS (e.g. through ODBC or JDBC for RDB), but the general problem still is unsolved when several DMS models have to cooperate. Location independence is addressed either by specific DBMS (e.g. distributed RDBMS) or through distributed object managers such as CORBA middleware products [3].

Despite much effort spent by the scientific community, semantic independence still is an open and largely unsolved problem except in simple situations.

This paper describes a general architecture, a methodology and a CASE environment intended to address the problem of providing users and programmers with a abstract interface to independent, heterogeneous and distributed databases. These

components are being developed as part of the InterDB project<sup>1</sup>.

The paper is organized as follows. Section 2 develops a small case study that allows us to identify some of the main problems to be solved. Section 3 presents the main aspects of the InterDB architecture for federated databases. Section 4 proposes a general methodology that helps developers to build the components of the architecture. In section 5, we discuss the role of CASE technology to support that methodology. Section 6 concludes the paper.

## 2. Problem statement

The general architecture of integrated databases, generally known as *federated database*, has been described in, e.g., [14]. It consists of a hierarchy of data descriptions that ensure independence according to different dimensions of heterogeneity: location, technology and semantics for example. According to this framework, each local database is described by its own local schema, from which a subset, called export schema, is extracted. The latter are merged into the global schema. Each application interacts with the local data through a view derived from this global schema. The mappings between adjacent levels ensure the various kinds of independence. The global schema, and possibly the export schemas, are expressed into some kind of pivot data model, generally independent of the underlying technologies (except when the latter all belong to the same family of models).

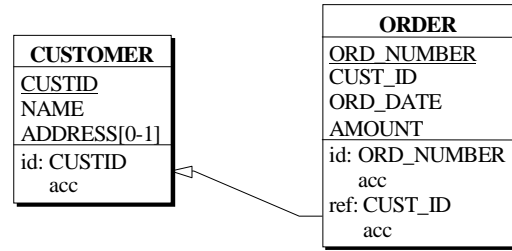
In this section we develop a small example that illustrates the problems we intend to address in the InterDB approach.

The database integration example comprises two independent heterogeneous database both describing aspects of a bookshop, and that are required to interoperate. The first one is made up of two COBOL files and the second one includes two relational tables. The integration process will be carried out in three steps, namely *conceptual schema recovery*, *conceptual integration* and *mapping definition*.

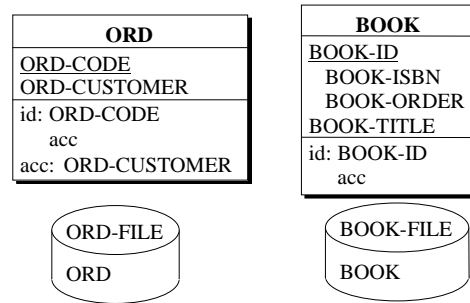
### 2.1. Conceptual schema recovery

By analyzing the COBOL programs and SQL DDL scripts, we can extract the local physical schema (LPS)

of each database. Figures 1a and 1b show the extracted schemas according a common abstract physical model.



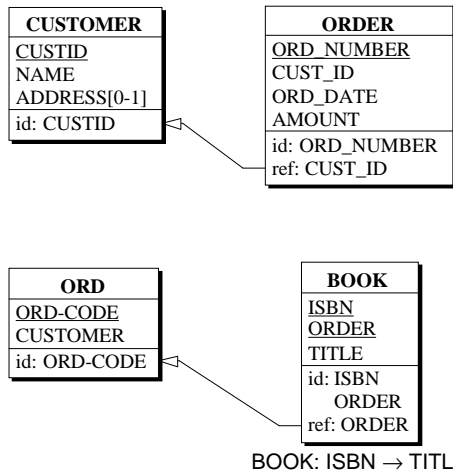
**Figure 1a. The relational local physical schema.** The DB comprises tables CUSTOMER and ORDER. ADDRESS is an optional column (cardinality [0-1]). Unique keys are represented by the **id**(entifier) construct, foreign keys by the **ref**(erence) construct and index by the **acc**(ess key) construct.



**Figure 1b. The COBOL local physical schema.** It is made of two files and two record types (ORD and BOOK). BOOK-ID is a compound field. ORD-CUSTOMER is a non-unique alternate record key.

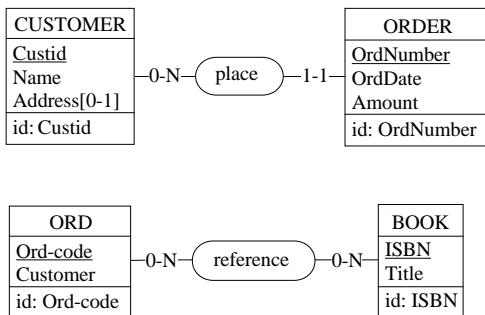
Based on the analysis of declarative code fragments or data dictionary contents, this process is fairly straightforward. However, it recovers explicit constructs only, ignoring all the implicit structures and constraints that may be buried into the procedural code of the programs or of the user interface. Hence the need for a refinement process that cleans the physical schema and enriches it with implicit constraints elicited by such techniques as program analysis and data analysis. The schemas are refined through in-depth inspection of the way in which the COBOL program and SQL DML use and manage the data in order to detect the record structures declared in the program sources. Moreover, names have been reworked (semantics-less prefix removed) and physical constructs are discarded (e.g., files and access keys). We therefore obtain the local logical schemas (LLS) of Figure 2. They express the data structures in a form that is close to the DMS model, enriched with semantic constraints.

<sup>1</sup> The InterDB project [1995-2000] is supported by the Belgian Région Wallonne.



**Figure 2.** The local logical schemas of the relational database (top) and of the COBOL files (bottom). We observe the elicitation of an implicit foreign key and of a functional dependency in the COBOL database.

The next phase consists in interpreting the logical structures by extracting their underlying semantics. The resulting conceptual schemas are expressed in a simplified variant of the Entity/Object-relationship model (Figure 3).

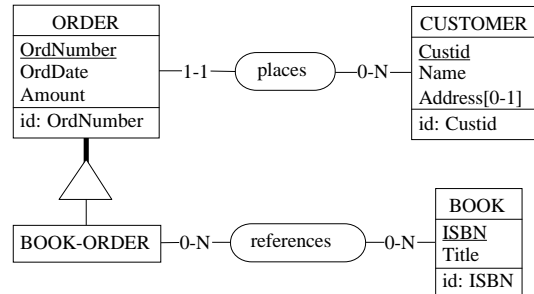


**Figure 3.** The local conceptual schemas (LCS) of the relational database (top) and of the COBOL files (bottom). The relational foreign keys have been transformed into relationship types. The BOOK record type has been normalized by splitting it into BOOK and REFERENCE, the latter being transformed into a many-to-many relationship type.

## 2.2. Conceptual integration

The global schema must include the semantics of these two local schemas. It appears that the ORD and ORDER entity types are *similar*. Data analysis shows that all the ORD.Ord-code values are in the ORDER.OrdNumber value set. Therefore, this

*similarity* has to be interpreted as a supertype/subtype relation: ORD (renamed as BOOK-ORDER) is a subtype of ORDER. Besides this reasoning, the integration process is fairly standard. Hence the schema of Figure 4.



**Figure 4.** The global conceptual schema (GCS). Some names have been changed for readability.

## 2.3. Mapping definition

Once the global schema has been built, we have to state how global retrieval and update queries can be mapped onto physical data. In the processes above, the transitions between two schema levels can be expressed as formal transformations, such as *normalization* or translating *foreign key into relationship type*. The local physical/conceptual mappings can then be constructed by interpreting the reverse engineering transformations as two-ways data conversion functions. Global/local mappings are based on the conceptual integration process. For instance, the following global conceptual query

```
select C.Custid,C.Name
from   BOOK B,ORDER O,CUSTOMER C
where  B.ISBN = '2-02-025247-3'
and    O references B and C places O
```

will be broken down into two local conceptual queries to be processed by the local servers:

```
select O.Ord-code,B.ISBN,B.Title
into   :OID,:BISBN,:BTITLE
from   BOOK B,ORD O
where  B.ISBN = '2-02-025247-3'
and    O reference B

select C.Custid,C.Name into :CID,:CNAM
from   ORDER O,CUSTOMER C
where  O.OrdNumber = :OID
and    C place O
```

Despite its small size, this example emphasizes some difficulties of database integration. The problem is specially complex when old, ill-designed and poorly documented applications are addressed. Hence the need for methodological and CASE support that will be discussed in this paper.

### 3. Architecture

The InterDB architecture is shown in Figure 5. The architecture comprises a hierarchy of mediators, namely local servers dedicated to each database and a global server based on the GCS. These mediators dynamically transform actual data into a virtual homogeneous database, and conversely, transform global conceptual queries into local physical queries. To simplify the discussion, export and view schemas have been ignored.

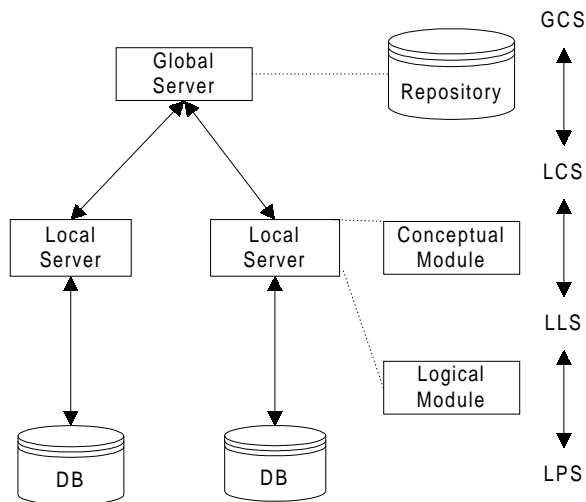


Figure 5. The InterDB architecture of a federated database.

#### 3.1 Local servers

The local server is in charge of managing the conceptual/physical conversion of each local database. It comprises two components, namely the logical module and the conceptual module. Though they can be implemented as a unique software component, they will be described distinctly.

The *logical module* hides the syntactic idiosyncrasies and the technical details of the DMS of a given model family. In addition, it makes the implicit constructs and constraints explicitly available. For instance, relational databases and flat COBOL files appear as similar logical structures. A logical module dynamically transforms queries (top-down) and data (bottom-up) from this logical model to the actual physical model. In particular, it emulates implicit constructs such as foreign keys in COBOL files or multivalued fields in relational DB.

All logical modules offer a common interface and present the physical data according to a common data model called the *generic logical model*. This model encompasses not only the current and legacy technologies (RDB, standard files, CODASYL, IMS, etc.), but also emerging ones, such as object-oriented models (ODMG).

The *conceptual module* provides a conceptual view of a local logical database. It hides the specific aspects of a family of models as well as the technical and optimization-dependent constructs of the actual database.

To summarize, each *local server* appears as a conceptual database manager that offers a unique abstract interface to application programs. It can translate local client queries into physical DML primitives (COBOL file primitives, SQL, DBTG DML, etc.). It can also compose physical data to form conceptual data aggregates requested by client applications. For performance reason, we have decided to develop the local servers as program components dedicated to a local database. In particular, the logical/physical and conceptual/logical mapping rules are hardcoded in the modules rather than interpreted from mapping tables.

To illustrate the translation mechanism, the local conceptual query

```
select C.Custid,C.Name into :CID,:CNAM
from   ORDER O,CUSTOMER C
where  O.OrdNumber = :OID
and    C.place O
```

will be translated by the local server into the physical query

```
insert into T0453(CID,CNAM)
select C.CUST_ID,C.NAME
from   ORDER O,CUSTOMER C
where  O.ORD_NUMBER = :OID
and    O.CUST_ID = C.CUSTID
```

This translation is based on the transformations (considered the reverse way) used to produce the conceptual schema (Fig. 3) from the physical one (Fig. 2), namely *FK-to-relationship-type* and *name-conversion*.

#### 3.2 The global server

The global server offers a conceptual interface based on the GCS. It processes the global queries, that is, queries addressing the data independently of their distribution across the different sites. For flexibility reason, the module is based on a repository that describes the global conceptual schema, the local conceptual schema of each local server, its location,

and the relationships between local and global schemas. Information concerning data replication, semantic conflicts and data heterogeneity allows the server to interpret and distribute the global queries, and to collect and integrate the results sent back by the local servers.

### 3.3 Heterogeneity issues

The architecture model depicted in Figure 5 provides an adequate framework for solving the heterogeneity issues discussed above. DMS and local semantic independence is guaranteed by the local servers. Location and global semantic independence is provided by the global server. This module provides data global access irrespective of their location and resolves semantic conflicts. Finally, platform independence is ensured by both the local servers and ad hoc middleware such as commercial ORB.

### 3.4 The pivot model

The pivot model is an abstract formalism intended to express data structures independently of the implementation technologies. For methodological reason, we propose a unique generic model from which several abstract submodels can be derived by specialization. This approach provides an elegant way to unify the multiple interfaces and mapping description of the architecture. For instance this object-relationship model is used to describe logical structures such as IMS, CODASYL, file, RDB and OO data structures (see Fig. 2), as well as ERA and OO conceptual structures, both at the local and global levels (Fig. 3 and 4). As illustrated in Section 2.3, queries are expressed into an OO SQL language that degenerates into a subset of SQL-2 for pure flat structures. These interfaces therefore comply with the current standards in distributed object management.

## 4. The InterDB methodology

Considering a collection of independent databases, building the infrastructure that integrates them is a complex engineering activity. As suggested by the small case study of Section 2 and the architecture of Section 3, the methodology comprises three major steps:

- recovery of the local logical and conceptual schemas,
- integration of the local schemas,
- building the mappings.

Though these processes have been described in some way in the literature, we will discuss them, particularly as far as non trivial issues are concerned.

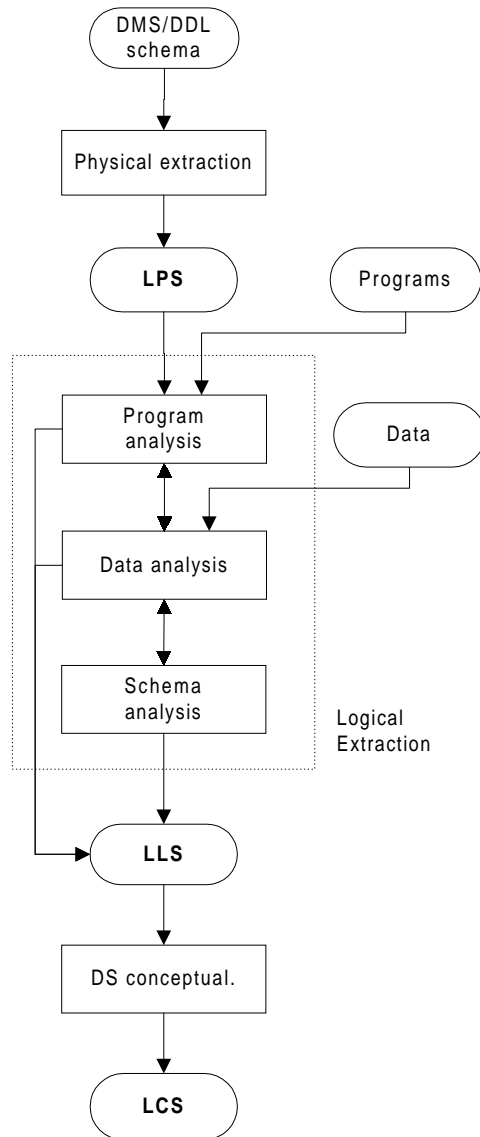
### 4.1. Local schema recovery

Recovering the logical and conceptual schemas of an existing local database is the main goal of Database Reverse Engineering (DBRE). The InterDB approach relies on the general DBRE methodology that has been developed in the DB-MAIN project [6]. The general DBRE architecture is outlined in Figure 6. It shows clearly the three main processes that will be described in Sections 4.1.1 to 4.1.3.

This methodology can be specialized according to the various data models which most legacy systems are based on, such as standard files, or CODASYL, IMS and relational databases. Since this has been presented in former papers (see [5] and [9]), we will only recall those of its processes that are relevant to our problem.

#### 4.1.1. Physical Extraction

This phase consists in recovering the local physical schema (LPS) made of all the declared, i.e. explicit, structures and constraints. Databases systems generally provide a description of this schema (catalogue, data dictionary contents, DLL texts, file sections, etc.). The process consists in analyzing the data structures declaration statements (in the specific DLL) or the contents of these sources. It produces the LPS. The process is more complex for file systems, since the only formal descriptions available are declaration fragments spread throughout the application programs. This process is easy to automate since it can be carried out by a simple parser which analyses the DMS-DDL texts, extracts the data structures and expresses them as the LPS. For instance, several popular CASE tools include some sort of extractors, generally limited to RDB, but sometimes extended to COBOL files and IMS databases.



**Figure 6. A generic DBRE methodology.** The main processes extract the physical schema (LPS), refine it to produce the logical schema (LLS) and interpret the latter as a conceptual schema (LCS).

#### 4.1.2. Logical Extraction

The LPS is a rich starting point that can be refined through the analysis of the other components of the applications (views, subschemas, screen and report layouts, programs, fragments of documentation, program execution, data, etc.). This schema is then refined through specific analysis techniques [9] that search non declarative sources of information for evidences of implicit constructs and constraints. This schema is finally cleaned of its non-logical structures.

In this phase, three processes are of particular importance:

1. *Program analysis.* This process consists in analyzing parts of the application programs (the procedural sections, for instance) in order to detect evidences of additional data structures and integrity constraints.
2. *Data analysis.* This refinement process examines the contents of the files and databases in order (1) to detect data structures and properties (e.g. to find unique fields or functional dependencies in a file), and (2) to test hypotheses (e.g. *Could this field be a foreign key to this file?*).
3. *Schema analysis.* This process consists in eliciting implicit constructs (e.g. foreign keys) from structural evidences, in detecting and discarding non-logical structures (e.g. files and access keys), in translating names to make them more meaningful, and in restructuring some parts of the schema.

The end product of this phase is the local logical schema (LLS).

#### 4.1.3. Data Structure Conceptualization

The Data Structure Conceptualization Process addresses the semantic interpretation of the LLS. It consists for instance in detecting and transforming, or discarding, non-conceptual structures. The main objective is to extract all the relevant semantic concepts underlying the logical schema. Two different problems have to be solved through specific techniques and reasoning: the identification and the replacement of DMS constructs with the conceptual constructs they are intended to translate and the elimination of optimized structures. The result of this process is the local conceptual schema (LCS). This process has been described in, e.g., [7].

#### 4.2. Local schemas integration

Integration is the process of identifying the objects in different local conceptual schemas which are related to one another, identifying and solving the conflicts of these schemas, and finally, merging local conceptual schemas into a global one.

##### 4.2.1. Conflicts identification

Identifying the syntactic and semantic conflicts of independent schemas has long been studied in the database realm [1, 17]. Conflicts occur in three possible ways : syntactic, semantic and instance.

Besides the usual conflicts related to synonyms and homonyms, a syntactic conflict occurs when the same concept is presented by different object types in local schemas. For instance, an *OrderDetail* can be represented by an entity, by an attribute value and by a relationship. A semantic conflict appears when a contradiction appears between two representations A and B of the same application domain concept or between two integrity constraints. Solving such conflicts uses reconciliation techniques, generally based on the identification of set-theoretic relationships between these representations:  $A = B$ ,  $A$  in  $B$ ,  $A$  and  $B$  in  $AB$ , etc. It is based on set-theoretic relations existing among the instances of data types, and that may conflict with semantic reconciliation. Instance conflicts are specific to existing data. Though their schemas agree, the instances of the databases may conflict. As an example, common knowledge suggests that *USER* be a subtype of *EMPLOYEE*. However, data analysis shows that  $\text{inst}(\text{EMPLOYEE}) \subseteq \text{inst}(\text{USER})$ , where  $\text{inst}(A)$  denotes the set of instances of data type  $A$ . This problem has been discussed in [16]. This process is highly knowledge-based and cannot be performed automatically.

#### 4.2.2. Conflicts resolution

Solving the conflicts occurring in heterogeneous databases has been studied in numerous references, by e.g. [1], [15] and [16]. It is important to note that most conflicts can be solved through three main techniques that are used to rework the local schemas before their integration: renaming, transforming and discarding. Heuristics exist to cope with this problem [15].

#### 4.2.3. LCSs merging

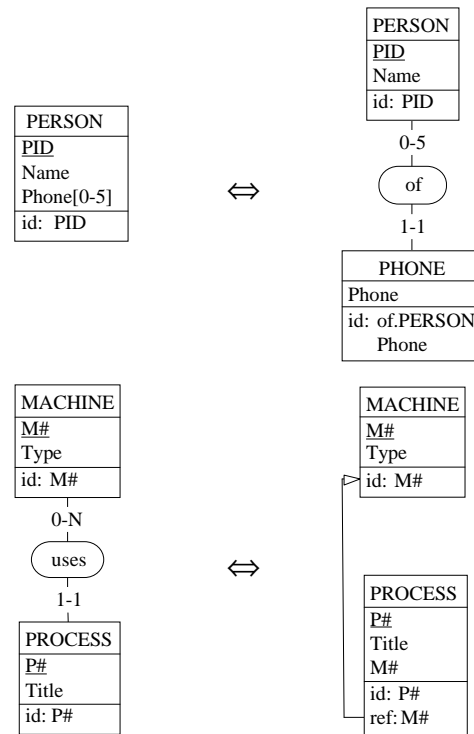
Since the syntactic, semantic and instance conflicts have been resolved by restructuring the local schemas, merging the latter is fairly straightforward, and can be automated to a large extent.

### 4.3. Building the inter-schema mappings

A careful analysis of the processes that have been described in 4.1 and 4.2 shows that deriving a schema from another one is performed through techniques such as renaming, translating, conceptualizing, solving conflicts, which basically are *schema transformations*. As suggested in [4], most database engineering processes can be formalized as a chain of schema

transformations. This is the case for reverse engineering and integration, as demonstrated in [7].

Roughly speaking, a schema transformation consists in deriving a target schema  $S'$  from a source schema  $S$  by replacing construct  $C$  (possibly empty) in  $S$  with a new construct  $C'$  (possibly empty). Adding an attribute to an entity type, replacing a relationship type with an equivalent entity type or with a foreign key are three examples of schema transformations.



**Figure 7. Two sample semantics-preserving schema transformations.** The first example explain how a multivalued attribute can be expressed as an external entity type. The second one shows the transformation of a relationship type into a foreign key. Only the structural mappings of both transformations are shown.

More formally, a transformation  $T$  is defined as a couple of mappings  $\langle T, t \rangle$  such that:

$$C' = T(C)$$

$$\text{inst}(C') = t(\text{inst}(C))$$

Structural mapping  $T$  explains how to modify the schema while instance mapping  $t$  states how to compute the instance set of  $C'$  from the instances of  $C$ .

Any transformation  $T$  can be given an inverse transformation  $T_i$ , such that  $T_i(T(C)) = C$ . If, in addition, we also have:  $t_i(t(c)) = c$ , then  $T$  is declared *semantics-preserving*. A compound transformation  $T_{12} = T_2 \circ T_1$  is obtained by applying  $T_2$  on the

database that results from the application of **T1**. These concepts and properties are analyzed in [4].

An important conclusion of the transformation-based analysis of database engineering processes is that most of them can be modeled through semantics-preserving transformations. For instance, conceptualizing logical schema LLS into conceptual schema LCS (Section 4.1.3) can be modeled as a compound semantics-preserving transformation **L-to-C** =  $\langle \mathbf{L-to-C}, \mathbf{l-to-c} \rangle$  in such a way that:

$$\text{LCS} = \mathbf{L-to-C}(\text{LLS})$$

This transformation has an inverse: **C-to-L** =  $\langle \mathbf{C-to-L}, \mathbf{c-to-l} \rangle$  such that:

$$\text{LLS} = \mathbf{C-to-L}(\text{LCS})$$

Fig. 7 illustrates two semantics-preserving transformations that are commonly used in reverse engineering and in schema integration.

Finally, the conceptual/logical mappings between the *logical data* and the *conceptual data* can be derived easily from the instance mappings:

$$\text{inst}(\text{LCS}) = \mathbf{l-to-c}(\text{inst}(\text{LLS}))$$

$$\text{inst}(\text{LLS}) = \mathbf{c-to-l}(\text{inst}(\text{LCS}))$$

Now, we are able to describe the physical/conceptual mappings for local databases. In addition to the conceptual/logical mappings mentioned above, we consider the logical/physical mappings **P-to-L** =  $\langle \mathbf{P-to-L}, \mathbf{p-to-l} \rangle$  and its inverse **L-to-P** =  $\langle \mathbf{L-to-P}, \mathbf{l-to-p} \rangle$ . A local server will rely on compound mapping  $[\mathbf{C-to-L} \circ \mathbf{L-to-P}]$  to translate queries, and on compound mapping  $[\mathbf{p-to-l} \circ \mathbf{l-to-c}]$  to form the result instances.

Of course, these mappings appear as pure functions that cannot be immediately translated into executable procedures in 3GL. However, it is fairly easy to produce procedural data conversion programs as shown in [10].

## 5. CASE support

Deriving a common, abstract and conflict-free image of independent databases, defining the mappings between the specification layers, building the local servers and the global repository are complex and time consuming tasks that are out of scope of most developers.

These processes are supported by an extension of the DB-MAIN CASE tool<sup>2</sup>. This graphical, repository-based, software engineering environment is dedicated to *database applications engineering*. Besides standard functions such as specification entry, examination and management, it includes advanced processors such as transformation toolboxes, reverse engineering processors and schema analysis tools. It also provides powerful assistant to help developers and analysts to carry out complex and tedious task in a reliable way. The assistant offer scripting facilities through which method fragments can be developed and reused.

One of the main feature of DB-MAIN is the *Meta-CASE layer*, which allows method engineers to customize the tool and to add new concepts, functions, models and even new methods. In particular, DB-MAIN offers a complete development language, Voyager 2, through which new functions and processors can be developed and seamlessly integrated into the tool. The InterDB CASE tool is built around the DB-MAIN tool by adding concepts and processors that support the specific InterDB architecture and methodology described in Sections 3 and 4.

### 5.2 Reverse engineering support

DB-MAIN offers functions and processors that are specific to DBRE [11]. The *physical extraction* process is carried out by a series of processors that automatically extract the data structures declared into a source text. These processors identify and parse the declaration part of the source texts, or analyze catalog tables, and create corresponding abstractions in the repository. Extractors have been developed for SQL, COBOL, CODASYL, IMS and RPG data structures. Additional extractors can be developed easily thanks to the Voyager 2 environment.

The *logical extraction* process is supported by a collection of processors such as:

- a pattern matching engine that can search a source text for definite patterns such as programming *clichés* (program analysis);
- a variable dependency analyzer that detects and displays the dependencies between the objects (variables, constants, records) of a program (program analysis);
- a program slicing tool that computes the set of program statements that contribute to the state of a

<sup>2</sup> An Education version of the DB-MAIN CASE environment as well as various materials of the DB-MAIN laboratory, e.g. reference [8], can be obtained at <http://www.info.fundp.ac.be/~dbm>

selected variable at a selected point (line) of a source program program analysis.

- a facility to quickly develop Voyager 2 generators of program/queries to examine data (data analysis);
- a foreign key assistant that helps find the possible foreign keys of a schema through structural analysis (schema analysis);

The *conceptualization* process can be performed in a reliable way thanks to a rich semantics-preserving transformation toolset. Transformation scripts that implement specific heuristics can be quickly developed. A programmable schema analysis processor can be used to detect structural patterns and problematic constructs to be further analyzed.

### 5.3 Schema integration support

Schema integration occurs mainly when merging the local conceptual schemas into the global schema. It also appears in reverse engineering to merge multiple descriptions into a unique logical schema. In addition, several strategies can be applied, depending on the complexity and the heterogeneity of the source databases and on skill of the analyst. As a consequence, DB-MAIN offers a toolbox for schema integration instead of a unique, automated, schema integrator. Together with the transformation toolbox, the integration toolbox allows manual, semi-automatic and fully automatic integration.

### 5.4 Mapping building

This function is not a specific component of DB-MAIN, but rather is a by-product of all the engineering activities. Indeed, DB-MAIN automatically generates and maintains a history log (say  $\mathbf{h}$ ) of all the activities that are carried out when the developer derives a schema B from schema A. This history is completely formalized in such a way that it can be replayed, analyzed and transformed. For example, any history  $\mathbf{h}$  can be inverted into history  $\mathbf{h}'$ . Histories must be normalized to remove useless sequences and dead-end exploratory branches. History  $\mathbf{h}$  of the reverse engineering process that produces the conceptual schema LCS of a local database from its physical schema LPS can be considered as a *structural mapping*. In other words,  $\mathbf{h} \equiv [\mathbf{P-to-L} \circ \mathbf{L-to-C}]$ , according to the notation of Section 4.3.

History processing, including inversion, has been described in [10].

### 5.5 Local server generation

The inverted history yields a fictive history of how the database could have been produced. In fact  $\mathbf{h}' \equiv [\mathbf{C-to-L} \circ \mathbf{L-to-P}]$ . Let us now consider  $\mathbf{t}$ , the *instance mapping* of  $\mathbf{h}$ , that is,  $\mathbf{t} = [\mathbf{p-to-l} \circ \mathbf{l-to-c}]$ . Formally,  $\{\mathbf{h}', \mathbf{t}\}$  is the functional specification of the local servers. Therefore, history  $\mathbf{h}$  can be used to generate the local servers components.

Generators have been developed in Voyager 2 to produce the logical and physical modules. At the current time, generators for COBOL files and RDB data structures are available.

### 5.6 The global repository

The global server is not developed yet. It will rely on a global repository which will be a simplified and autonomous, variant of the repository of the DB-MAIN CASE tool.

This repository is an OO database managed by a custom-made, high performance, DBMS which has been developed as a C++ platform-independent module.

## 6. Conclusions

Database interoperability has been studied for years, leading to architectures that can be considered standard. The InterDB contribution is on the methodological level. It proposes a comprehensive process that integrates known techniques such as database reverse engineering, schema integration and query decomposition, with new ones, such as mapping building and component generation. This approach is supported by a CASE tool in which standard functions are augmented with specific functions, assistants and processors dedicated to reverse engineering, schema integration, history processing and architecture component generation.

It appears that processes such as reverse engineering and schema integration require strong skills from the analyst. Therefore, CASE technology can only help him carry out these processes, but cannot automate them, except in very simple and academic situations.

One of the most challenging issue was building the inter-schema mappings. Thanks to a formal transformational approach to schema engineering, it was possible to build the mappings that drive the query decomposition and the data recomposition required by the hierarchy of schemas.

Though important issues have not been tackled so far, such as global query processing and transaction management, the architecture as well as the methods and engineering tools we have developed cope in an elegant way with all the heterogeneity dimensions that appear when one integrates existing independently developed databases.

A short analysis of the market shows that no current CASE tools can help either in reverse engineering complex databases, or in building the components of the architecture. Hence the importance of the DB-MAIN CASE tool, particularly of its Voyager 2 meta-language that allowed us to develop repository-based analyzers and local servers generators. An interesting feature of the architecture is that the global repository of the global server is the repository of the CASE tool as well.

## 7. References

- [1] C. Batini, M. Lenserini, and S.B. Navathe, "A Comparative Analysis of Methodologies for Database Schema Integration", *ACM Computing Surveys*, 18(4), Dec. 1986, pp. 323-364.
- [2] A. Dogac, C. Dengi, E. Kilic, G. Ozhan, F. Ozcan, S. Nural, C. Evrendelek, B. Halici, B. Arpinar, P. Koksals, N. Kesim and S. Mancuhan, "METU Interoperable Database System", *SIGMOD RECORD*, Vol. 24(3), 1995, pp. 56-61.
- [3] A. Dogac, C. Dengi, E. Kilic, G. Ozhan, F. Ozcan, S. Nural, C. Evrendelek, B. Halici, P. Koksals, and S. Mancuhan, "A Multidatabase System Implementation on CORBA", in *Proc of 6th Intl. Workshop on Research Issues in Data Engineering (RIDE-NDS'96)*, New Orleans, 1996.
- [4] J-L. Hainaut, "Specification preservation in schema transformations - Application to semantics and statistics", *Data & Knowledge Engineering*, Elsevier Science Publish, 16(1), 1996.
- [5] J-L. Hainaut, M. Chandelon, C. Tonneau, M. Joris, "Contribution to a Theory of Database Reverse Engineering", in *Proc. of the IEEE Working Conf. on Reverse Engineering*, IEEE Computer Society Press, Baltimore, May 1993.
- [6] J-L. Hainaut, V. Englebert, J. Henrard, L-M. Hick, D. Roland, "Evolution of database Applications: the DB-MAIN Approach", in *Proc. of the 13th Int. Conf. on ER Approach*, Springer-Verlag, Manchester, 1994.
- [7] J-L. Hainaut, C. Tonneau, M. Joris, M. Chandelon, "Schema Transformation Techniques for Database Reverse Engineering", in *Proc. of the 12th Int. Conf. on ER Approach*, E/R Institute, Arlington-Dallas, 1993.
- [8] J-M. Hick, V. Englebert, J. Henrard, D. Roland, J-L. Hainaut, "The DB-MAIN Database Engineering CASE Tool (version 3) - Functions Overview", *DB-MAIN Technical manual*, Institut d'informatique, University of Namur, November 1997.
- [9] J-L. Hainaut, J. Henrard, D. Roland, V. Englebert, and J-M. Hick, "Structure Elicitation in Database Reverse Engineering", in *Proc. Of the 3<sup>rd</sup> IEEE Working Conf. On Reverse Engineering*, IEEE Computer Society Press, Monteney, Nov. 1996.
- [10] J-L. Hainaut, J. Henrard, J-M. Hick, D. Roland, and V. Englebert, "Database Design Recovery", in *Proc. of the 8th Conf. on Advanced Information Systems Engineering (CAISE' 96)* Springer-Verlag, 1996.
- [11] J-L. Hainaut, D. Roland, J-M. Hick, J. Henrard, and V. Englebert, "Database Reverse Engineering: from Requirements to CARE tools", *Journal of Automated Software Engineering*, 3(1), 1996.
- [12] E. Kilic et al, "Experiences in Using CORBA for a Multidatabase Implementation", in *Proc. of 6th Intl. Workshop on Database and Expert System Applications*, London, Sept. 1995.
- [13] A., Lefebvre, P., Bernus, Topor, R., "Query Transformation for Accessing Heterogenous Databases", *Workshop on Deductive Databases*, Washington DC, Nov. 1992.
- [14] M.T. Özsu, P. Valduriez, *Principles of Distributed Database Systems*, Prentice Hall, New Jersey, 1991.
- [15] S. Spaccapietra and C. Parent, "Conflicts and correspondence assertions in interoperable databases", *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 20(4), pp. 49-54, December 1991
- [16] M.W.W. Vermeer and P.M.G Apers, "On the Applicability of Schema Integration Techniques to Database Interoperation", in *Proc. Of 15th Int. Conf. On Conceptual Modelling*, ER'96, Cottbus, Oct. 1996, pp. 179-194.