DATABASE INTEGRATION: THE KEY TO DATA INTEROPERABILITY

Christine Parent
UNIL/HEC-INFORGE
CH 1015 Lausanne
christine@lbdsun.epfl.ch

Stefano Spaccapietra
Swiss Federal Institute of Technology
CH 1015 Lausanne
Stefano.Spaccapietra@epfl.ch

Abstract

Most of new databases are no more built from scratch, but re-use existing data from several autonomous data stores. To facilitate application development, the data to be re-used should preferably be redefined as a virtual database, providing for the logical unification of the underlying data sets. This unification process is called database integration. This chapter provides a global picture of the issues raised and the approaches that have been proposed to tackle the problem.

1. Introduction

Information systems for large organizations today are most frequently implemented on a distributed architecture, using a number of different computers interconnected via Intranet or Internet. Information is usually stored in various databases, managed by heterogeneous database management systems (DBMSs), or in files, spreadsheets, etc. Disadvantages of using multiple independent databases within the same organization are well known, including: high potential for incompleteness, inaccuracy and inconsistencies in data acquisition and data processing, lack of coordination resulting in duplication of efforts and of resources, and eventually conflicts in the allocation of responsibilities for data maintenance. Still, such situations are very common. For instance, different databases in different departments support applications specific to each department. Interoperability is the magic word that is expected to solve these problems, allowing heterogeneous systems to talk to each other and exchange information in a meaningful way.

Two levels of complexity may be separated in addressing interoperability. The more complex case is when information involves data sources that are not in a database format, typically local files and spreadsheet data or external data reached through Internet. In this case, understanding of unstructured (e.g., free text) or semi-structured (e.g., a web page with html incrustation) data calls for sophisticated mechanisms for extraction of semantics. Moreover, the global information system has to be able to dynamically evolve according to changes in configuration of the available data sources (new data or new sources become available or available ones temporarily or definitely disappear). The relevant sources may even not be defined a priori, but may have to be determined on the fly through one or more web searches. To implement interoperability in such a context many diverse functionalities are needed. They include [Papakonstantinou 95]:

- a communication kernel enforcing information exchange standards for data representation and exchange requests,

- a set of knowledge discovery tools supporting various forms of intelligent data browsing, semantic extraction, and learning,
- a set of tools for semantic interoperability: wrappers, to adapt local sources to specifications of the global system (typically performing schema, data and query language translations) and mediators, performing integration of data or services from the various local sources,
- a global distributed data management system extending traditional DBMS operations to the federated context: query decomposition and optimization, transaction management, concurrency and recovery.

The simpler case is when the scope of information exchange is limited to databases within the organization (e.g., a typical Intranet environment). Here existing database schemas provide basic knowledge about the semantics of data, which may be easily enhanced into data dictionaries or data warehouse formats through interviews of current users and data administrators or analysis of the documentation. Exchange standards become easier to define and enforce as part of some general policy for information technology within the organization. Hence the challenge in the design of an integrated information system is on the mediators in charge of solving discrepancies among the component systems.

Interoperability among database systems may basically be achieved in three ways, supporting different levels of integration:

- at the lowest level, i.e. no integration, the goal is nothing but to enable one DBMS to request and obtain data from another DBMS, in a typical client/server mode. Gateways, i.e. dedicated packages, support this limited functionality and are currently marketed for a number of existing DBMSs. Most well known gateways are ODBC-compliant tools, where ODBC (Open DataBase Connectivity) is an SQL-based emerging standard from Microsoft.

- at an intermediate level, the goal is to support user-driven access and/or integration of data from multiple databases. The term user-driven refers to the fact that users are given the possibility to simultaneously manipulate data from several sources in some uniform way. However, it is also user's responsibility to access and manipulate the local databases consistently. The system is not in charge of guaranteeing consistency across database boundaries. To implement such a framework, a software layer is developed, whose functionality may range from:
  - a multidatabase query language, e.g. OEM-QL [Papakonstantinou 95], providing a single SQL-like syntax that is understood by a set of translators, each one of these mapping the query to an underlying DBMS. The benefit for users is the capability to address many systems through a single language.
  - to a multidatabase system, where users are provided with a language that has full data definition and manipulation capabilities. In particular, the system supports view definition, which allows users to define their own external schema through a mapping to relations (or classes) from the different sources. MSQL [Litwin 90] is a well-known reference is this domain for relational database environments. An extension of MSQL functionality to include some conflict resolution strategies is reported in [Missier 97]. Based on user's explicit description of semantic relationships among data domains, these strategies are intended to solve, during query processing, some of the inconsistencies that may

arise among related data from different databases. Similar proposals for multidatabase systems based on some object-oriented model also exist (e.g., [Kaul 90]).

- at a higher level, the goal is to develop a global system, sitting on top of the existing systems, to provide the desired level of integration of the data sources.
  - Total integration is implied in distributed data base (DDB) management systems [Ceri 87]. All existing data are integrated into a logically unique database (the DDB), and henceforth managed in a consistent way under a single global control authority. This approach has proved not to be suited for many enterprises where the need for accessing several data sources should not interfere with the actual control of these sources by their respective owners.
  - To provide more flexible integration, researchers have lately developed specifications for federated database (FDB) systems [Sheth 90]. FDB systems aim at scalable integration, supporting a harmonious coexistence of data integration and site autonomy requirements. Site autonomy is guaranteed as local usage of the local data is preserved, schema and data evolution remains under local control, and data sharing is on a volunteer basis. Each database administrator (DBA) defines the subset of the local data, if any, which is to be made available to distant users of the federated system. The defined subset is called the local export schema. Local export schemas define the data available for integration into one (or more) virtual databases, called the FDB. Virtual, here, stands for a database that is logically defined (its schema exists) but is not directly materialized. The data described by the federated schema resides in the local databases. There is not one physical FDB somewhere, but only parts of the FDB which belong to the source databases. The FDB thus provides an integrated access without any need for data duplication. Integration, as well as import/export of data into/from the FDB, is managed by the federated system (FDBMS). FDBMS role is to enforce cooperation agreements as established by the participating DBAs (in terms of semantics of data, access rules, copy maintenance, etc.), to perform integration of data and services, as well as the traditional operations of a distributed DBMS (e.g., query processing).

While gateways and multidatabase systems do not attempt to unify the semantics of data from the various sources, distributed and federated database systems base their services on an integrated view of the data they manage. Users access the DDB or FDB like a centralized database, without having to worry about the actual physical location of data, the way the data is locally represented, or the syntax of the languages of the local DBMS. These advantages easily explain why the federated approach, in particular, is so popular today. However, before FDB systems come to reality, a number of issues have to be solved (see [Kim 95, Sheth 90] for comprehensive overviews). These include design issues, related to the establishment of a common understanding of shared data, as well as operational issues, related to adapting database techniques to the new challenges of distributed environments. The former focus on either human-centered aspects (e.g., cooperative work, autonomy enforcement, negotiation procedures) or database centered aspects (e.g., database integration, schema or database evolution). The latter investigate system interoperability mainly in terms of support of new transaction types (long transactions, nested transactions,...), new query processing algorithms, security concerns, and so on.

The kernel of design issues, and the most relevant for the topic of this book, is the database integration problem. Simply stated, database integration is the process which:
- takes as input a set of databases (schema and population), and
- produces as output a single unified description of the input schemas (the integrated schema) and the associated mapping information supporting integrated access to existing data through the integrated schema.

Database integration is a complex problem. Quite a large number of papers have investigated various facets of it, resulting in many technical contributions, a few methodologies and a few prototypes. As it is impossible to meaningfully synthesize all existing material, we apologize for incompleteness. This chapter provides a survey of the most significant trends. Our primary goal has been to draw a clear picture of what are the approaches, how far we can go with the current solutions and what remains to be achieved. The focus is on the concepts, the alternatives and the fundamentals of the solutions, not on detailed technical discussions, for which further readings are listed in the references. The presentation is organized according to the temporal sequence of actions that compose the database integration process. We identify three major steps in this process (see figure 1):

- pre-integration, a step in which input schemas are re-arranged in various ways to make them more homogenous (both syntactically and semantically);
- correspondence identification, a step devoted to the identification of related items in the input schemas and the precise description of these inter-schemas relationships;
- and integration, the final step which actually unifies corresponding items into an integrated schema and produces the associated mappings.

The last sections discuss methodological aspects and conclude on further directions of work.

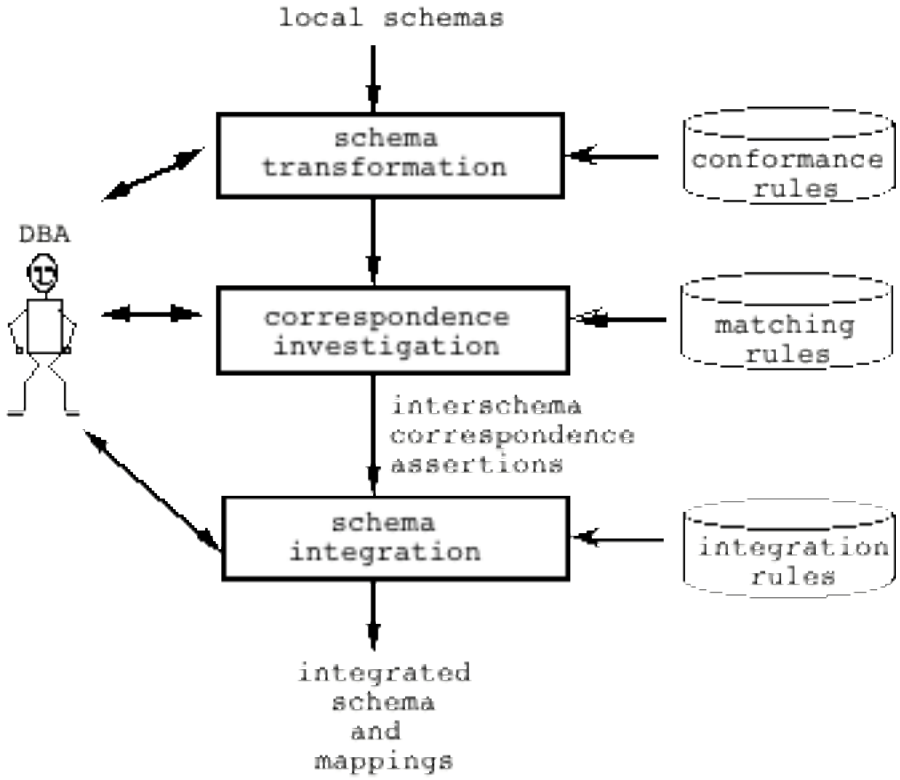

Figure 1: the global integration process

2 The Example

Discussions in the sequel will mostly be illustrated referring to the car rental example, the common case study used throughout this book. To put the example into an interoperability framework it suffices to assume that the branches of the car rental company have independently developed different databases. A company level decision to set up an integrated information system on top of existing data would lead to the database integration problem. Equivalently if we assume that different car rental companies, each one equipped with its own database, decide to merge businesses, which include merging of the information systems.

Let us consider that the databases to be integrated are described by the schemas that illustrate various chapters in this book. Having a common case study perfectly illustrates the diversity in schema designs due to different perceptions by each designer (in this case the authors of the chapters) of the same real world (the document describing the case study). Such a situation is representative of what happens in real applications.

Some differences simply stem from terminological choices. The key to identify a car, for instance, is either named "CarId", or "car_id", or "Chassis#", or "Chassis". Terminological tools will easily identify the first two and the last two as being equivalent terms. But finding the equivalence between "CarId" and "Chassis#" requires a database perspective, i.e. consideration that both serve as unique key in equivalent structures (the Car relation, the Car object type, the Car entity type) and both have the same value domain.

More differences come from the fact that designers have chosen different properties for the same object type. Car has properties <Chassis#, category> in the chapter by Gogolla, and has properties <CarId, Branch, Model, Make, Category, Year, Mileage, LastServiced> in the chapter by Jensen and Snodgrass.

Structural differences may be illustrated considering customer information. Jensen and Snodgrass propose a Customer relation which includes a "Rating" property (with value domain: "Preferred", …) to discriminate various categories of customers. Missaoui et al. Materialize the same idea adding two subtypes (Blacklist, Freq_Trav) to the Customer supertype. This difference between the two representations is mainly due to heterogeneity of the underlying data models (i.e. the relational model used by Jensen and Snodgrass does not support the generalization concept). Papazoglou and Kramer also use a data model with is-a links, but propose a different hierarchy: Customer has subtypes Person and Company, the latter with subtypes PrivateCorporation and Government. These two hierarchies are based on different specialization criteria.

Different classifications scheme, not involving is-a links, are visible when comparing the design by Missaouri et al, with the one by Gogolla. The latter includes two object types for bookings: one for current bookings (those where a specific car has been assigned), another for non-current bookings (where only a car category is specified). Missaoui's design has all bookings in a unique Rental-Booking type. Current bookings are found

by restricting Rental-Booking objects to those that are linked to a car object by the Allocate-to link.

These differences give an idea of the complexity inherent to the database integration process, which will have to sort out differences to build a consistent representation of the data. They also point at the benefit expected from the integration. In a federated car rental information system it will be possible for a user at Gogolla's site to query the model of a car, an information that is not present at that site but can be found at Jensen and Snodgrass site if the latter stores the requested car. Also, it becomes possible for a user at Papazoglou and Kramer site to know if a given customer is blacklisted or not, by looking at the related information in Missaoui's database.

3. Preparing for Integration

Generally, the databases to be integrated have been developed independently and are heterogeneous in several respects. A worthwhile first step is therefore to attempt to reduce or eliminate such discrepancies. The path from heterogeneity to homogeneity may take three complementary routes:

- syntactic rewriting. The most visible heterogeneity is when existing databases have been installed on DBMSs based on different data models (relational, CODASYL, object-oriented, …). Efficient interoperation calls for the adoption of a common data model serving as information exchange standard among participating locations. Dedicated wrappers have to be developed to enforce data model transformations between the local model and the common model [Hammer 97].
- semantic enrichment. Data model heterogeneity also induces semantic heterogeneity, in the sense that constructs in one model may provide a more accurate description of data than constructs in another model. For instance, an entity-relationship schema has different constructs for entities and associations, while some equivalent relational schema may describe the same data without making an explicit distinction between entities and associations. To compare the two schemas, one should be able to identify, in the relational schema, which relations describe entities and which relations describe associations. This is a very primitive form of semantic enrichment, i.e. the process that aims at augmenting the knowledge about the semantics of data.
- representational normalization. One more cause of heterogeneity is the non-determinism of the modeling process. Two designers representing the same real world situation with the same data model will inevitably end up with two different schemas. Enforcing modeling rules will reduce the heterogeneity of representations. This is referred to as representational normalization.

We discuss hereinafter how to cope with these three issues.

3.1 Data model heterogeneity

The issue here is how to map data structures and operations from one DBMS into data structures and operations conforming to a different DBMS. Most papers on database integration simply assume that the input schemas are all expressed in the same data

model, i.e. the so-called "common" data model, on which the integrated system is built. A data model mapping step is assumed as a pre-requisite to integration and is dealt with as a separate problem. The needed mappings are those between any local data model and the common data model. Unfortunately, the state of the art in data model mapping is poor in tools for automatic mapping (except for many CASE tools for database design, supporting entity-relationship to relational mapping). Latest developments focus on mapping between object-oriented and relational models, as part of a major effort to develop the new object-relational DBMSs, which are supposed to support both paradigms. Typically, the way the problem is addressed nowadays is by splitting the mapping task into: 1/ a series of transformations (i.e. data structure modifications within a given data model), and 2/ a translation, i.e. the rewriting of the transformed schema using the syntax of the target model. The goal of transformations is to remove from the source schema the constructs in the source data model that do not exist in the target data model. Removal is performed using alternative design strategies in the source data model [McBrien 97]. The benefit of the decomposition is to allow for the implementation of a library of schema restructuring operations (the transformations) that can be reused in different mappings [Thiran 98].

Beyond data structure transformations, some researchers have also considered the complementary problem of how to translate operations from one DBMS to another one. This is needed for a fully multilingual system, i.e. a system in which users from participating systems use the languages of the local DBMS to access the federated system. Because of their additional complexity, multilingual federations are rarely advocated in the literature. Still they offer users the substantial benefit of not having to learn new languages to interact with the FDBS.

One of the unresolved debates is the choice of the common data model. Basically, two directions have supporters. The majority favors the object-oriented approach. The argument is that it has all the semantic concepts of the other models and that methods can be used to implement specific mapping rules. An open issue is to agree on which one of the many existing object-oriented models is best in this role. A second problem is that the richest the model is in modeling concepts, the more likely it is that different designers will model the same reality using different constructs, based on their own perception of the relative importance of things. Known as semantic relativism, this flexibility makes integration more complex, as it will have to solve the many possible discrepancies dues to different modeling choices. To make integration simpler, the alternative is to adopt a data model with minimal semantics embedded, such that there is little chance of conflicts in data representation. Data representations in semantically poor models are brought down to elementary facts for which there is no modeling alternative. Binary-relationships models compete in this role with functional models [Schmitt 96].

A stream of more basic research investigates the possibility of developing a generic wrapper, capable of performing mapping between any two data models [Atzeni 97, Nicolle 96, Papazoglou 96]. In a traditional federated system, algorithms to map schemas in data model Mi into schemas in the common data model CDM (and vice versa) are explicitly implemented into the local wrappers. Adding a new data model Mj to the federation requires the development of a new wrapper supporting the two mappings: Mj to CDM, CDM to Mj. It is possible to avoid such a burden by moving from the procedural approach (defining and implementing algorithms) to a declarative

approach. The latter relies on the definition of a meta-model (i.e. a data model suited for the description of data models) [Urban 91]. The meta-model includes a number of basic modeling concepts and knows how to map each concepts into another one (for instance, how to nest flat tuples to obtain a nested tuple). Once the definitions of the local data models, in terms of the meta-model, are fed into the generic mapping tool, the tool is capable to map any source schema into any target schema. First, the source schema is mapped into the meta-model concepts. Second, restructuring rules are applied within the meta-model database to turn source concepts into target concepts, third the result is mapped into the target model. In this context, adding a new data model Mj to the federation simply calls for the definition of the Mj concepts in terms of the meta-model. Beyond data structures, [Davidson 97] extends the mapping task to include processing of some associated non-standard constraints.

3.2 Semantic enrichment

Whatever the approach, translations raise a number of difficult problems and it is unlikely that they can be fully automated. Interactions with database administrators are needed to solve ambiguities that rise because schemas only convey incomplete information on the semantics of data. On the one hand, however powerful, data models cannot express all the semantics of the real world. Limitations in the modeling concepts cannot be avoided, as the required exhaustiveness would lead to a model of unmanageable complexity. On the other hand, even augmented with integrity constraints, a schema usually relies on many implicit assumptions that form the cultural background of the organization. Implicit business rules are supposed to be known to all users, hence they need no description. But when the database enters a federation, it becomes open to users who know nothing about its implicit rules.

Semantic enrichment is basically a human decision process, used to provide more information either on how a schema maps to the real world, or on the schema itself. An example of the former is adding a definition of "Employee" as "those persons who have an employment contract with the enterprise". An example of the latter is specifying that the attribute "boss" in the "Department" relation is an external key to the "Employee" relation. Semantic enrichment is not specific to poor data models. Object-oriented schemas may also need to be augmented with information on cardinalities or on dependencies (which are not represented but are necessary, for instance, for a correct translation of multivalued attributes). As another example, reformulating an object-oriented schema may call for turning an attribute into an object: the question immediately arises whether two identical values of this attribute should be translated into only one or two objects. No automatic decision is possible, as the answer depends on the real world semantics.

Some techniques aid in acquiring additional information about the semantics of a schema. Poor data structures may be turned into richer conceptual structures using reverse engineering techniques. Such poor structures exist, for instance, in old relational systems, which only stored the description of the relations and their attributes, with no notice of primary keys, candidate keys, foreign keys, and dependencies. Moreover, relations in existing databases are not necessarily normalized. Reverse engineering relies on the analysis of whatever information is available: schema specifications, index definitions, the data in the database, queries in existing application programs. Combining inferences from these analyses (in particular, about keys and

dependencies) makes it possible to recompose complex object types from flat relations, and to identify association structures and generalization hierarchies. The result still needs confirmation by the DBA. For instance, join conditions in queries may indicate, but not assert, the existence of a foreign key; the usage of a "distinct" clause in an SQL statement may lead to the conclusion that the retrieved attribute is not a primary key [Hainaut 98, Tari 98], and so on. Similar but more complex techniques are used to reengineer existing files [Andersson 98].

Beyond data structures, when it comes to understanding the semantics of data, knowledge discovery, or knowledge elicitation, techniques are appropriate. The basic goal is to build an integrated semantic dictionary whose scope spans over all databases in the federation. Integrating ontologies, building concept hierarchies or context integration are alternative denotations for this process. Description logic is a well-known theoretical support for developing vocabulary sharing based on synonym relationships [Mena 96]. Statistical analysis of combined term occurrences may help in determining relationships among concepts [Kahng 96]. A global organization of the local knowledge is thus achieved [Castano 97]. For more enrichment, contextual information is gathered to make explicit the rules and interpretations not stated in the local schemas [Lee 96|. For instance, a salary item may be complemented with the information on the local monetary unit, not otherwise described. When the inference cannot be done using some automatic reasoning, interaction with the DBAs is necessary [Ouksel 96].

Semantic enrichment becomes an even more challenging issue when application data has to be collected dynamically from non-predefined sources available at the moment the application is run. This is the case in particular when the data is collected over the Web. Web data is typically semi-structured, and comes with little descriptions attached. Many ongoing projects address the issue of extracting semantics from a Web site data, whether on the fly during execution of a query or in a more static setting through exploration of designated Web sites (e.g., [Bayardo 97, De Rosa 98]).

3.3 Representational normalization

Modeling choices are guided by the perception of the designer and by the usage the data is for. The same real world data may thus be described using different data structures, which represent modeling alternatives supported by most data models. Support for such alternatives is known as semantic relativism. The richest a data model is in semantic expressiveness, the more it opens up to modeling alternatives. Semantic relativism is often criticized as a weakness of a model, because the designer is confronted with a non-trivial choice among alternatives. In our opinion, it should rather be considered as an advantage, as it offers flexibility to closely adjust the representation of data to the intended usage of the data.
However, undesirable discrepancies may be reduced through enforcement of rules that constrain designers to certain choices. This could be at the organizational level, by enforcing corporate modeling policies (including terminology), and/or at the technical level, by applying normalization rules. Organizational rules may range from defining the terminology to be used (i.e., names of data items) to adopting design patterns (i.e., pre-defined representation schemes) or even a complete pre-established design for the whole organization (e.g., when using the SAP product). Normalization rules are well known in the context of relational databases, but still have to be elaborated for object-

oriented databases. Two types of normalization rules can be defined. First, they may enforce design rules that command the use of a representation instead of another one. Examples are:

- if a property of an object type is only relevant for a subset of its instances (e.g., maiden name for persons), represent this using a supertype/subtype structure (e.g., a supertype Person with a subtype Wife), where the subtype bears this attribute as mandatory; do not represent this as an object type with an optional attribute. This rule allows having schemas without optional attributes.
- if a property of an object type may hold many values within the same instance (e.g., telephone number for persons), represent the property as a separate object type and a reference to it from the original type (e.g., a type Telephone and a reference to Telephone in Person); do not represent the property as a multivalued attribute in the original type. This rule allows having schemas without multivalued attributes.
- a type with an enumerated property (e.g., Person and the sex property whose domain has two predefined values) should be replaced by a supertype/subtypes structure (e.g., a Person supertype with Man and Woman subtypes).

This type of rules enforces syntactic normalization, independently of the semantics of data. Another set of normalization rules aims at conforming the schemas to the underlying dependencies. An example of a possible rule of this type is: if there is a dependency between attributes A and B of an object type, and A is not a key, replace these attributes by a composite (tuple) attribute with A and B as component attributes. This may resemble relational normalization, but differs from it on the intended purpose. Relational normal forms aim at reducing data duplication to avoid update anomalies. The object-type normal form we used as example is intended to enhance the semantics of the schema. More work on normalization is needed before normal rules for objects are agreed upon [Tari 97].

4. Identifying Interdatabase Correspondences

Once the input data sources have been rewritten and enriched into whatever level of conformance is achievable, the next step is the identification of overlapping or complementary information in different sources. Indeed, providing users with an integrated view of the available data implies that:

- at the schema level, the descriptions of related information from different sources are somehow merged to form a unique and consistent description within the integrated schema, and
- at the instance level, a mechanism is set up to link a representation within a source to related representations within the other sources. These links support integrated data access at query time.

Interdatabase correspondences are frequently found by looking for similarities in the input schemas. However, similarity between representations is not the ultimate criterion. Similarity evaluation may be misled by terminological ambiguities (homonyms and synonyms) and, more generally, by differences in the implicit contexts. Also, representations of the same data (whether real world objects, links or properties) may be completely different from one source to the other. Hence, database integration has to go beyond representations to consider what is represented rather than how it is represented. For instance, we want to know if Hans Schmidt,

represented in database A, is also represented in database B, even if the two instances have completely different sets of attributes. **Two databases are said to have something in common** if the real world subsets they represent have some common elements (i.e. a non-empty intersection) or have some elements related to each other in a way that is of interest to future applications. An example of the latter is the case where a car rental company has a database of cars in each branch, recording cars of that branch, and it is worthwhile for the company to form an integrated database showing a single object type Car that represents all cars belonging to the company.

At the instance level, **two elements** (occurrence, value, tuple, link, ...) from two databases are said to **correspond to each other** if they describe the same real world element (object, link or property). As an example, let us assume that an object type Employee, holding a Salary attribute, exists in both an Austrian database and a German database, and an employee Hans Schmidt belongs to the two databases. If Hans Schmidt in Austria is the same person than Hans Schmidt in Germany, the two database objects correspond to each other. If this correspondence is not stated, the system will assume that two persons are just sharing the same name. If there is only one person Hans Schmidt and he has only one salary, represented in marks in the German database and in shillings in the Austrian database, the two salary values correspond to each other (there exist a mapping that deduces one from the other). If the two salary values are not stated as corresponding to each other, it means that Hans Schmidt gets two salaries, independent of each other even if by chance they happen to represent the same amount.

If a correspondence can be defined such that it holds for every element in an identifiable set (e.g., the population of a type), the correspondence is stated at the schema level. This intensional definition of a correspondence is called an **interdatabase correspondence assertion** (ICA). The complete integration of existing databases requires an exhaustive identification and processing of all relevant ICAs. In an exhaustive approach, the integration process consists in finding all interdatabase correspondences and for each correspondence adding to the integrated schema an integrated description of the related elements (supporting the mapping at the instance level). Local elements with no counterpart elsewhere are directly integrated in the global schema. At the end of the process the integrated schema provides a complete and non-redundant description of all data in the FDB. The mappings between the integrated schema and the local schemas support integrated data access for users of the FDB.
Such a complete and static integration is not always possible, not even desirable. This is the case when, for instance, the number of local databases is too high, or the schemas contain too many items, or in evolvable environments where input databases may dynamically be connected and disconnected. In these cases, partial, dynamic or incremental integration strategies are advisable. Strategy issues are discussed in section 6. Whatever the strategy, ICAs will have to be found, made explicit and processed: they are a cornerstone for data interoperability. Techniques for these activities do not depend on the strategy.

The precise definition of an interdatabase correspondence assertion calls for the specification of:

- what are the related elements, both at the schema level and in terms of the population subsets that are involved in the correspondence. This information is used to build the data structure in the integrated schema;
- how to identify, for each instance involved in a correspondence, which are the corresponding instances in the other sources. This information is used for integrated access to data;
- how the representations of corresponding instances are related. This information is used to build non-redundant descriptions in the integrated schema.

We discuss below these specifications in more detail.

4.1 Relating corresponding elements

To declare a correspondence between two databases, it is desirable to identify as precisely as possible the elements that are being related. Assume, for instance, that two car rental companies decide to join their efforts and build an integrated service, hence an integrated database. Each company has its own database, say A and B, which include an object type CarModel to describe models of cars being rented. Both companies rent cars from the same manufacturers and, in particular, the same car models. They agree to keep this as a rule for future evolution of their business. In other words, they agree that updates of the available car models made by one company are also effective for the other company. In database terms, the populations of the two object types A.CarModel and B.CarModel are kept equivalent at any point in time: each object in A.CarModel always has a corresponding object in B.CarModel. Equivalence means that the car models represented in A and B are the same in the real world, although their representation in the two databases may differ (e.g., they may include different attributes). These assumptions lead to the assertion of the following ICA:

$$\text{A.}\mathit{CarModel} \equiv \text{B.}\mathit{CarModel} \ .$$

If the two companies prefer a more flexible integration, and in particular one which supports update autonomy, i.e. each company performs its own updates and no update is mandated by the other company, integration will be based on an intersection relationship:

$$\text{A.}\mathit{CarModel} \sqcap \text{B.}\mathit{CarModel} \ .$$

This instructs the system that at any time there may be in A a subset of CarModel objects which have an equivalent in the population of B.CarModel, and vice versa. Updates do not need anymore to be propagated. The correspondence rule at the instance level (see next subsection) determines which objects belong to the corresponding subsets. Assume that A.CarModel and B.CarModel are merged into a single object type I-CarModel in the integrated schema. At data access, objects in the (virtual) population of I-CarModel will show more or less information, depending on their existence in A only, in B only, or in both. In other words, attributes that exist in only one database will appear as optional attributes to the integrated user.

It may be the case that the subsets involved in the intersection are known in advance. For instance, the two car rental companies may be sharing models but only for a specific manufacturer, say BMW, for which exactly the same models are offered. In this case the ICA is stated as:

σ [manufacturer = "BMW"] A.*CarModel* $\equiv$ σ [manufacturer = "BMW"] B.*CarModel*

where σ denotes the selection operator. It is possible, in particular, to split the population of a type into subsets corresponding to populations of different types in the other database. Assume a database D1 with a type Person and another database D2 with two types, Man and Woman, which represent the same set of real world persons. It is then correct to state:

case 1: σ [sex = "male"] D1.*Person* ≡ D2.*Man*
σ [sex = "female"] D1.*Person* ≡ D2.*Woman*

This is more precise, hence preferable, than the single ICA:

case 2: D1.*Person* ≡ D2.*Man* ∪ D2.*Woman*

or the two ICAs:

case 3: D1.*Person* ⊃ D2.*Man*
D1.*Person* ⊃ D2.*Woman*

The explicit statement defining the selection criteria (case 1) allows to build more precise mappings between the integrated schema and the input schemas. For instance, if it is only known that Person is equivalent to the union of Man and Woman (case 2), a query from a federated user asking for women will be directed to D2, which knows about women. The specification of the selection criterion allows, instead, the system to either direct the query to the object type Woman in D2 or to the object type Person restricted to women in D1. This gives more power in terms of query optimization strategies. Moreover, update propagation can be supported from D1 to D2 and vice-versa, while in case 2 updates can only be propagated from D2 to D1. Allowing D1 users to update the set of persons would imply bothering the user to determine whether the person is a man or a woman.

Related sets of elements may be denoted by algebraic expressions of any complexity on each side of the correspondence. In the examples we have seen so far, the mapping at the instance level is 1:1: one person corresponds to either a man or a woman. That is not always the case. For example, a CarModel type may describe car models in one database, while in another database only parts of a car model (e.g., motor, chassis) are described in a Part type. There is a correspondence between each instance of CarModel and the set of Part instances describing this car model. This situation is referred to as a fragmentation conflict, first introduced in [Dupont 94]. Fragmentation conflicts are frequent in spatial databases, when databases at different resolution levels are interrelated [Devogele 98].

Relating elements is not sufficient to precisely capture all interrelationships among databases. Intra-database links are also important. For instance, assume two databases A and B, each one showing object types Car and Customer linked by a CC relationship. The fact that correspondences are stated between A.Car and B.Car and between A.Customer and B.Customer does not imply a correspondence between the two relationships (A.CC and B.CC). One could imagine that in database A the CC path between Car and Customer expresses the fact that a customer holds a booking for the car, while in database B CC is used to express that a customer has already rented this car in the past. In this case, assuming the integrated schema keeps the Car and Customer object types, Car and Customer will be linked by two separate relationships, image of A.CC and B.CC, each one with its specific original semantics. If in both databases the CC relationships have the same semantics, this has to be explicitly stated as a valid ICA, so that integration results in only one relationship in the integrated schema. The ICA reads, for instance:

$$A.Car\text{-}CC\text{-}Customer \equiv B.Car\text{-}CC\text{-}Customer$$

This is interpreted as the assertion that any time in database A car x is related via A.CC to customer y, in database B car x' (corresponding to x) is related via B.CC to customer y' (corresponding to y). Paths in an ICA are denoted by enumerating the elements they traverse. Path integration, first discussed in [Spaccapietra 92], has been investigated in detail in [Klas 95].

The above examples have shown relevance of the equivalence (≡) and intersection (∩) relationships in the definition of an ICA. Inclusion (⊇), or disjointedness (≠) relationships may also be used. Inclusion states that the set denoted for one database is included in the set denoted for the other database. Assume a car rental branch B only rents small or medium size cars, while another branch A from the same company rents all types of cars. A relevant ICA may be:

$$A.CarModel \supseteq B.CarModel$$

Finally, disjointedness relates sets that have no common elements, but whose integration is desired. For instance, assuming each rental branch has its own cars, the ICA

$$A.Car \neq B.Car$$

directs the integration process to merge the two car object types in the integrated schema, despite the fact that the two populations are disjoint. The virtual population of the integrated type is the union of the source populations.

4.2 How corresponding instances are identified

When federated users request a data element via the integrated schema, the federated system may find that some properties of the element exist in one database, while other properties exist in another database. To provide users with all available data, the system has to know how to find in one database the object (instance or value) corresponding to a given instance/value in another database. Assume, for instance, that CarModel in A has attributes (name, manufacturer, number of seats, trunk capacity) and corresponding CarModel in B has attributes (code, manufacturer, year, available colors). To answer a user query asking for Ford models with trunk capacity greater than 800cm^3 and available in blue or black, the federated system knows that it has to perform a join between objects in A (which hold the trunk capacity criterion) and corresponding objects in B (holding the color criterion). How does the federated system know which join criterion applies?

To solve the issue, each ICA has to include the specification of the corresponding mapping between the instances: we call this the "matching criterion" (MC) clause. If we assume that code in B is nothing but the name in A, the ICA:

$$A.CarModel \supseteq_{MC} B.CarModel \qquad MC \quad A.name = B.code$$

specifies that corresponding car model objects from the two databases share a common identifying value, value of *name* in A and value of *code* in B. The join condition discussed above is nothing but A.*name* = B.*code*.

The general MC clause involves for each database a (possibly complex) predicate specifying the corresponding instances/values. Most often value-based identifiers (e.g. primary keys in relational models) can be used to match corresponding instances. This, however, does not have to be the case, and any 1:1 mapping function is acceptable,

including user-defined functions, historical conditions, complex heuristics, and look-up tables. Materialization of matching data has been suggested as a way to reduce the cost of matching when very complex criteria have to be evaluated [Zhou 95]. This induces a maintenance problem, which also arises when updates to real world objects are captured asynchronously in the source databases. A complex probabilistic approach, using historical information in transaction logs, has been proposed to solve this update heterogeneity problem [Si 96]. A matching technique for semi-structured data has been proposed in [Papakonstantinou 96], where object identification is generated by extraction of semantics when objects are imported by the mediator. In some approaches, import of objects by the mediator comes with a virtual object identity generation mechanism [Kent 92], where virtual identities are used to denote objects at the federated level. In such a setting, the federated system has to check for the transitivity of object matching: if o1 matches o2 and o2 matches o3, then o1 matches o3. Indeed, such transitivity is not necessarily guaranteed by the object identity generation mechanism [Albert 96].
Spatial databases offer a specific alternative for identification of correlated objects: by location, i.e. through their position in space. This allows to assert that two instances are related if they are located in the same point (line, area, or volume) in space. Notice that sometimes in spatial databases there is no thematic attribute to serve as an object identifier, hence no alternative to a spatial matching [Devogele 98].

4.3 How representations are related

Back at the schema level, let us now consider representations of related elements, i.e. the set of properties attached to the corresponding elements. Properties include both attributes and methods. In order to avoid duplication of properties in the integrated schema, it is important that shared properties, beyond those used for identification (denoted in the MC clause), be identified and the mappings in between specified. To this extent a "corresponding properties" (CP) clause is added to the ICA. For instance, if the two related CarModel types both include a “maker” attribute and all other properties are different, the ICA stated in 4.2 becomes

A.*CarModel* ⊇ B.*CarModel* MC A.*name* = B.*code* CP A.*maker* = B.*maker* .

The general format for a correspondence between properties X and Y is: f(X) *rel* g(Y), where *rel* is equality (=), if X or Y is monovalued, or a set relationship (≡, ⊇, ∩, ≠) if X and Y are multivalued; f and g are two functions used, whenever needed, to solve a representation conflict. The semantics of the CP clause is that, if E and F are the database elements related by the ICA, the sets of values E.X and F.Y (possibly converted through functions f and g respectively) are related by the given set relationship. Attribute matching has been extensively analyzed in the literature [Larson 89]. Method matching is a recent issue raised by object orientation [Metais 97].

4.4 Consistency of correspondences

Given a set of ICAs between two databases, the ICAs can be checked for consistency and minimality. Assume one schema has a A is-a B construct and the other schema has a C is-a D construct. An example of inconsistent ICA specification is: A≡D, B≡C. Both cannot be true because of the acyclity property of is-a graphs. Some ICAs

are derivable from others: if A≡C is asserted, B⊆C and D⊇A may be inferred. Hence, only ICAs bearing non-derivable correspondences need to be explicitly stated.
[Klas 95] analyzed the consistency issue for path correspondences. The authors defined two sets of rules:

- rules that specify which kind of paths cannot correspond to each other, e.g. a reference link cannot be equivalent to an is-a link,
- rules that check consistency of path correspondences, e.g. if two correspondences contain the same sub-path, they are either redundant or inconsistent.

4.5 Investigation of correspondences

With real, large schemas to be integrated, the task of identifying all relevant ICAs is far from trivial. A significant amount of research has been and is being invested into tools for automated identification of plausible correspondences. Traditional approaches [Gotthard 92, Metais 97] measure the similarity between two schema elements by looking for identical or similar characteristics: names, identifiers, components, properties, attributes (name, domain, constraints), methods. Computing the ratio of similarities versus dissimilarities gives an evaluation of how plausible the correspondence is. The idea has been put to an extreme in [Clifton 98], where metadata is dumped to unformatted text on which information retrieval tools evaluate string similarity. [Garcia-Solaco 95] takes the opposite direction and proposes to enrich the schemas before comparison by extracting semantics from an analysis of data instances. In a similar attempt to limit erroneous inferences due to synonyms and homonyms, [Fankhauser 92] recommends terminological knowledge bases to explain the terms used in the application domain and the semantic links in between.
Unconventional approaches include [Li 94] and [Lu 98]. The former uses neural networks to match equivalent attributes. The latter uses knowledge discovery tools borrowed from the data mining community. [Sester 98] advocates the use of machine learning techniques for settling correspondences in spatial database integration. The complexity of spatial matching criteria makes it difficult for a designer to specify correspondences without errors or approximations. It is easier to point at specific correspondences at the object level and let the system learn from these examples until the system can propose a general expression.
Whatever the technique, it is recommended that the final step be an interaction with the DBA for validation/invalidation of the findings and provision of additional information on the ICAs (e.g., the relationship between extents).

5 Solving Conflicts

Except if data sets to be integrated originated from a previous decision to duplicate data, at least partially, (e.g., for performance reasons), it is unlikely that related elements from different databases will perfectly match. Different but related databases rather have something, not all, in common, i.e. they represent overlapping subsets of the real world. Discrepancies may arise on various respects. The common set of real world objects or links might be organized into different classification schemes. The set of properties attached to objects and links may differ. Each of these differences is seen as a conflict among existing representations (interschema conflict), due to different design choices. A different type of conflict (interdata conflict) has its source in data

acquisition errors or inaccuracies: this is when the same data in different databases has different values.

All conflicts have to be solved to provide federated users with an integrated view of all available data. Solving an interschema conflict means: 1) deciding how the related conflicting elements are going to be described in the integrated schema, and 2) defining the mappings between the chosen integrated representation and the local ones. These mappings are used by the query processor component of the FDBS to transform each federated global query into the corresponding set of local queries, which are executed by the local DBMSs to retrieve and recompose all bits and pieces of data that are needed to provide the requested data. Solutions to interdata conflicts are discussed in section 5.4.

The existence of alternatives in conflict resolution strategies has received little attention [Dupont 94]. Authors usually propose specific solutions for each conflict type, with no concern about consistency of integration choices. However, different organizational goals are possible and lead to different technical solutions (cf. figure 2):

- the goal may be simplicity (i.e. readability) of the integrated schema: the appropriate technique then is to produce a minimal number of elements (object types, attributes and links). Related representations will be merged into an integrated representation, which will hide existing differences. For instance, if one object type in one database is asserted to ntersect an object type in the other database, only the union type will be described in the integrated schema. The selection criterion that defines the input types will show up in the mapping between the integrated schema and the local database. Mappings in this merging technique need to be sophisticated enough to cope with the schema conflicts. The advantage of readability is of course in human communication and understanding;
- the goal may be completeness, in the sense that every element of an input schema appears in the integrated schema. In this case, if one object type in one database is asserted to intersect an object type in the other database, both types and their common intersection subtype will be described and linked by is-a links in the integrated schema. The advantage of completeness is that elements of input schemas can be readily identified within the integrated schema, thus helping in maintaining the integrated schema when input databases evolve. Also, mappings get close to identity functions, which simplifies query processing;
- the goal may also be exhaustiveness, i.e. having in the integrated schema all possible elements, including those who are not in input schemas but complement what is there. For the running example, this principle leads to the inclusion in the integrated schema of both input types, together with their union (one common supertype), their intersection (common subtype) and the complements of the intersection (two subtypes). In some sense, this is intended to ease future integration with new databases, as chances are higher that types found in a newly considered input schema will already be present in the IS.

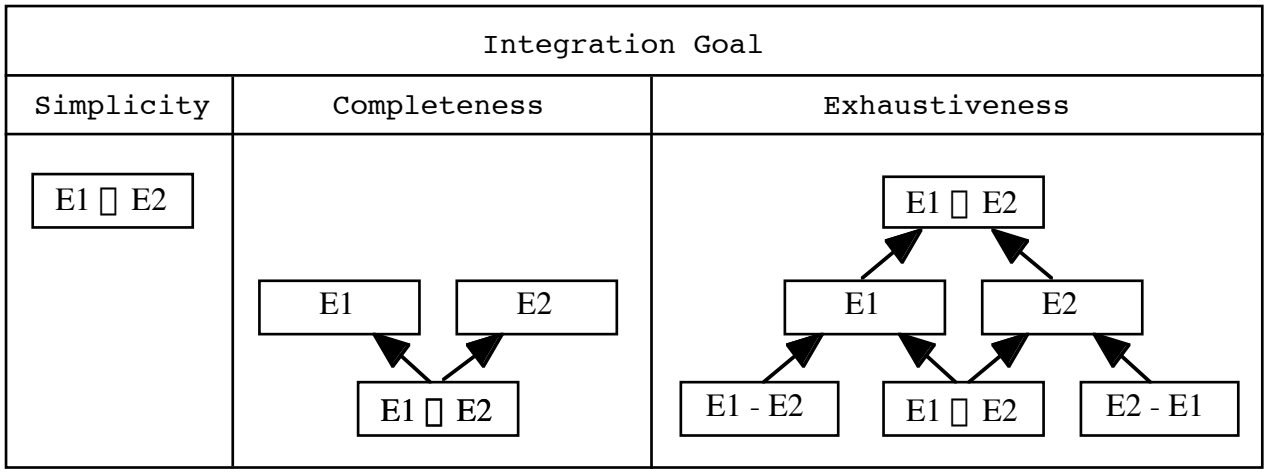

Figure 2: Alternative integrated schemas for the ICA E1 ∩ E2

For a rigorous, systematic approach to database integration, it is important that the involved DBAs agree on the integration goal. Moreover, an explicit choice allows defining the rules that mediators need to automatically perform integration. Otherwise, rules have to be defined in each mediator for each conflict type or conflict instance.

Taxonomies of conflicts abound in the literature, from very detailed ones [Sheth 92] to simpler ones [Spaccapietra 91]. Some examples of well-known conflict categories are:

- heterogeneity conflicts: different data models support the input schemas;
- generalization/specialization conflicts: related databases represent different viewpoints on the same set of objects, resulting in different generalization/ specialization hierarchies, with objects distributed according to different classification abstractions [Larson 89, Gotthard 92, Kim 93];
- description conflicts: the related types have different sets of properties and/or their corresponding properties are described in different ways [Kim 93];
- structural conflicts: the constructs used for describing the related types are different [Spaccapietra 92];
- fragmentation conflicts: the same real world objects are described through different decompositions into different component elements [Dupont 94, Devogele 98];
- metadata conflicts: the correspondence relates a type to a meta-type [Saltor 92];
- data conflicts: corresponding instances have different values for corresponding properties [Sheuermann 94, Abdelmoty 97].

In most cases, conflicts from different categories will combine to form a given correspondence. An open issue is to demonstrate if the resulting integrated schema is the same irrespectively of the order in which conflict types are addressed in a mediator. If not, the next issue is to find the best order, either in terms of quality of the result or in terms of processing time.
Detailed, and different, proposals on how to solve the above conflicts can easily be found in the literature. Despite the differences, some general principles supporting conflict resolution and integration rules may be highlighted:

- preservation of local schemas and databases: input databases should be kept as they are to preserve the investment in existing data and programs. If modifications are needed to solve conflicts and conform each input database to the integrated schema, they are only virtually performed, i.e. modifications are implemented as

part of the mappings between the integrated schema and the existing input schemas. These mappings may rely on a view mechanism;
- production of both an integrated schema and the mappings to input schemas: mappings are necessary to make integration operational;
- subsumption of input schemas by the integrated schema: the integrated schema must describe all data made available in the input databases. Hence integrated types must subsume the corresponding input types: subsume their capacity to describe information (adopting the least upper bound) and subsume the constraints which are inherent to or attached to them (adopting the greatest lower bound). Capacity denotes which combination of identity/value/links is modeled by a given construct [Spaccapietra 92]. For instance, an attribute in OO and relational approaches models either a value or a link, while in ER approaches it models only a value. A relational relation models a value and possibly links (through foreign keys), not identity. Therefore, if an ICA between relational schemas identifies a relation (value+links) as corresponding to an attribute (value or link), the integrated schema will retain the relation. The same principle dictates that every type with no counterpart elsewhere should be added to the integrated schema as it is. The least upper bound of its capacity and the greatest lower bound of its constraints define the type itself. Finally, this principle also directs the way to integrate integrity constraints: keep their greatest lower bound.

The next subsection discusses general principles that apply whenever a correspondence relates one instance in one database to one instance in the other database, i.e. there is a 1:1 mapping at the instance level. The following subsection similarly discusses n:m mappings. Third we discuss structural conflicts. Finally, interdata conflicts are considered.

5.1 One to one matching

We discuss here the situation where it is possible to identify a one to one mapping between elements of two sets of objects, one in each database. In other words, for each object from a given set in database A, there is a corresponding object in database B, and vice versa. The major conflict in this case is when objects have been classified using different schemes (generalization/specialization conflict). For instance, in database A there may be an object type Customer, with subtypes GoodCustomer and BadCustomer, while in database B there is an object type Customer with subtypes ExternalCustomer and LocalCustomer. Let us assume Hans Schmidt is a customer in both databases. He will be represented as an instance of some object type on both sides, but not the same. If the membership predicate for each object type is known, the hierarchy of customer object types in A can be mapped onto the hierarchy of customer object types in B.

Because of the 1:1 mapping at the instance level, predicates expressing the interdatabase correspondence assertions will use object-preserving operations. Algebraic expressions in the ICAs will thus include selections, unions or intersections, to recompose the distribution of objects, but no join or aggregation. They may also include projections in order to reduce the sets of properties to the common set (i.e. properties present in both databases). As an example, ICAs for the related customer hierarchies may be:
A.GoodCustomer = (SELECT B.ExternalCustomer WHERE type ="good") UNION

(SELECT B.LocalCustomer WHERE type ="good")

A.BadCustomer = (SELECT B.ExternalCustomer WHERE type ="bad") UNION
(SELECT B.LocalCustomer WHERE type ="bad")

B.LocalCustomer = (SELECT A.GoodCustomer WHERE state ="CH") UNION
(SELECT A.BadCustomer WHERE state ="CH")

B.ExternalCustomer = (SELECT A.GoodCustomer WHERE state ≠"CH") UNION
(SELECT A.BadCustomer WHERE state ≠"CH")

A strategy for integration of generalization hierarchies related by multiple ICAs is presented in [Schmitt 98]. The main problem stems from the twofold semantics of generalization/specialization links: extent inclusion and property inheritance. The two semantics tend to diverge when two somehow interrelated hierarchies with populated types are merged. When inserting a type from one hierarchy into the other hierarchy, the place determined according to the extent inclusion may differ from the place determined according to property inheritance. The algorithm in [Schmitt 98] complies with both semantics in merging two hierarchies, but to achieve this it has to split the input types so that, for each pair of corresponding types, their extents are distributed into three sets: the common extent (objects in a 1:1 correspondence) and the extents that belong to only one database (objects with no correspondence). The type of each partial extent receives all attributes of both schemas which are meaningful (i.e. valued) for the extent. In general, the integrated schema will contain new types, with smaller extents than the input ones. A refinement phase allows suppressing abstract types or types without own attributes. The approach assumes that the distribution of objects is known, i.e. specialization criteria are explicitly defined.

5.2 Many to many matching

Many to many correspondences relate a set of objects in one database to a set of objects in another database, such that there is no one to one mapping between objects in the two sets. As mentioned in section 4.1, these have been called fragmentation conflicts, which expresses that the conflict stems from a different decomposition of the real world thing being represented in the two databases. Let us recall the example we used (cf. figure 3): a CarModel type may describe car models in one database, while in another database only parts of a car model (e.g., motor, chassis) are described in a Part type. In this specific example the correspondence is 1:n, between one object (a CarModel instance) and a set of objects (the corresponding instances of the Part type).

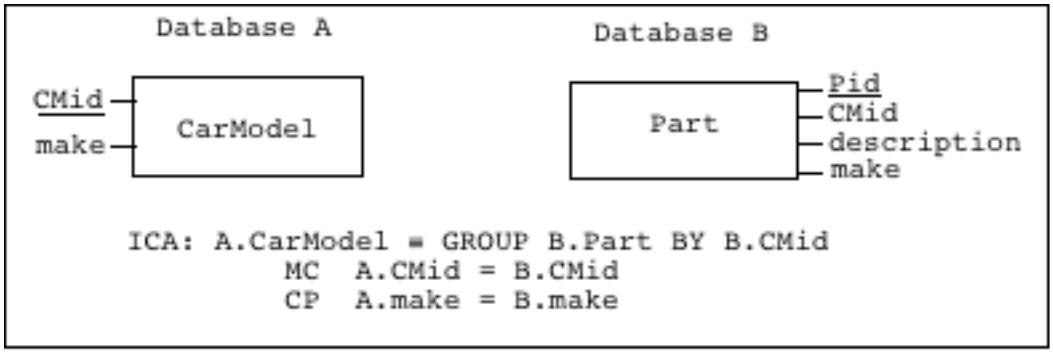

Figure 3: a fragmentation conflict

To illustrate a generic n:m correspondence let us consider two cartographic databases that include representations of buildings for map production purposes. Let us assume the two databases have different resolution (i.e. they contain information to produce maps at different scales), and there is a set of buildings, e.g. a university campus, which needs to be represented using some abstraction because the scale does not allow precise representation of each individual building. This abstraction mechanism is known as cartographic generalization. It may be the case that generalization in database A resulted in representing the campus as 8 abstract buildings, while generalization for less precise database B resulted in representing the same campus as a set of 5 buildings. When interrelating the two databases, it is correct to state that the eight abstract buildings in A represent the same thing that the five abstract buildings in B, while it would be incorrect to state a correspondence between individual abstract buildings.

There is no easy, meaningful operation to map a set of objects into another set of objects. However, fragmentation conflicts may be solved by transformation of the n:m matching into an equivalent 1:1 matching. This is done through schema enhancement and object-generating operations. Whenever a configuration of objects (i.e. a set of objects and links in between) in a database is collectively involved in a correspondence, a new type is created, whose instances represent those configurations. The new types will be linked to the existing ones by the appropriate kind of links: aggregation links, composition links, associations, etc. Once the new types are established, the correspondence may be restated as a 1:1 correspondence using the new types and object-generating operations [Devogele 98]. In the previous CarModel versus Part example, a derived CarModel object type is defined for B, such that a CarModel object is the aggregation of the Part objects that share the same value for the Cmid property (cf. figure 4). The aggregated object has derived properties Cmid and maker, "inherited" from the related Part objects. At this point, the correspondence between A and B can be stated as a 1:1 correspondence between the CarModel type in A and the derived CarModel type in B.

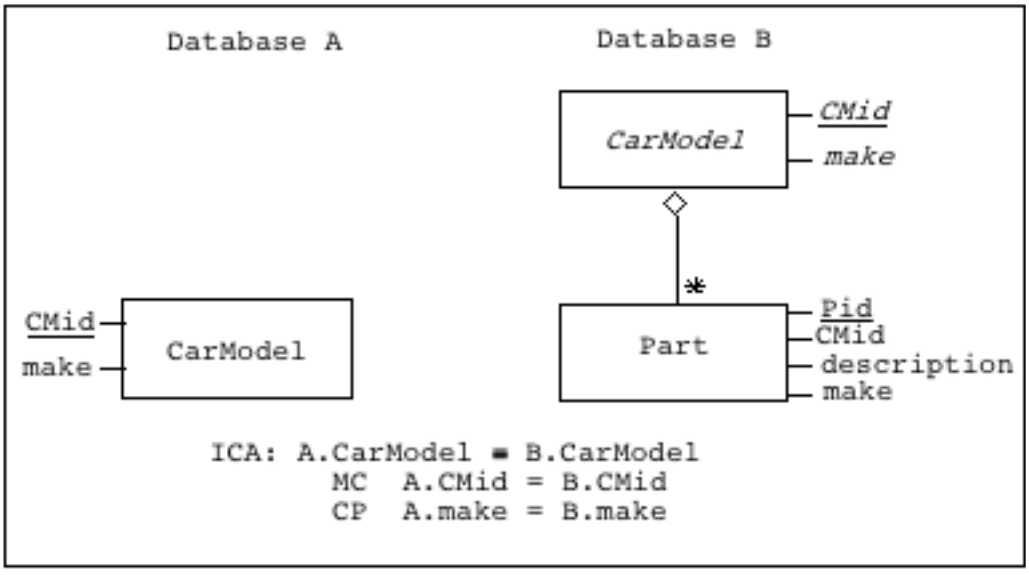

Figure 4: Transformation of a 1:n correspondence into a 1:1 correspondence

5.3 Structural conflicts

The schema restructuring principle is also used to solve structural conflicts.. These arise whenever something in the real world has been represented by different constructs, which have different representational power or different constraints: a car model, for instance, may be represented as an object class in one database and as an attribute in another database. Similarly for object types and a relationship types.
The solution of structural conflicts obeys the rule that the integrated schema must describe the populations of the two conflicting types. Hence, as stated at the beginning of section 5, the integrated type must subsume both input types in terms of information capacity and constraints. Typical constraints to be considered are cardinality constraints and existence dependencies. For instance, an attribute is existence dependent on its owner, while an object is generally not constrained by existence dependencies. If an ICA relates an object type to an attribute, the integrated schema will retain the object type (the greatest lower bound in this case is: no constraint). More about the solution of structural conflicts may be found in [Spaccapietra 92].

An extreme case of structural conflict is the so-called data/metadata conflict. Here, the decision choices that generate the conflict are the representation of the same thing as a value for some data on one hand, and as a name of some schema component on the other hand. For instance, the car model ZY-roadster may be represented by a value of an attribute car-model in a Car object type, or by an object type ZY-roadster whose instances represent such cars. Again, schema transformation operations are needed to solve the conflict, such as partitioning a class into subclasses according to the value of a specialization attribute, or creating a common superclass, with a new classifying attribute, over a set of given classes. Different variants of this solution may be found in [Saltor 92], [Lakshmanan 93] or [Miller 93].

5.4 Interdata conflicts

This type of conflict occurs at the instance level if corresponding occurrences have conflicting values for corresponding attributes. For instance, the same car is stored in two databases with different car model values. Sources for interdata conflicts include typing errors, variety of information providers, different versioning, deferred updates. Spatial databases have an even richer set of possible conflicts (nine kinds are identified in [Abdelmoty 97]).
These conflicts are normally found during query processing. The system may just report the conflict to the user, or might apply some heuristic to determine the appropriate value. Common heuristics are choosing the value from the database known as "the most reliable", or uniting conflicting values in some way (through union for sets of values, though aggregation for single values). Another possibility is to provide users with a manipulation language with facilities to manipulate sets of possible values; such a set is built as an answer to a query whenever a data conflict occurs [Tseng 93]. Similarly, [Agarwal 95] and [Dung 96] propose a flexible relational data model and algebra, which adapt the relational paradigm to inconsistent data management by making visible the inconsistency, if any, among tuples of an integrated relation (i.e. tuples with the same value for the key and different values for the same attribute).

6 Integration Strategies

Beyond the technical issues that we have surveyed, a very important open question relates to the strategy to be used to face database integration in real, quite complex environments. Complexity may be due to a huge number (hundreds or more) of databases to be integrated, as it is the case in some telecommunications businesses, or to very large schemas with hundreds of object or relationship types, or to the heterogeneity of the sources, ranging from purely unstructured to fully structured data, coupled with very little information available on the semantics of the existing data (which is the case, in particular, for data gathered via Internet).
In real applications, achieving full integration may be a very long process, which needs to be carefully planned for a step by step implementation possibly over several years. This idea of incremental integration has become very popular and most contributions today aim at providing a way to smoothly install integrated services while existing systems stay in operation. In fact, being incremental is orthogonal to the methodology, as all integration methodologies can be revisited and reformulated so that they can be applied in a stepwise way.

Incrementality may be database driven: each time an interdatabase correspondence is identified, the corresponding elements (instances or types) are integrated, either by adding the integrated element to an evolving integrated schema, or by adding logical interdatabase references at the instance level. The latter provides a direct way to navigate from an element in one database to the corresponding element in the other database [Scholl 94, Klas 95, Vermeer 96]. Another database driven technique is clustering of existing databases by areas of interest [Milliner 95].

Alternatively, incrementality may be user driven: each time a query is formulated (or a class of similar queries is identified) which calls for accessing related data in several databases, a multidatabase view is explicitly defined and implemented in an ad hoc mediator [Hohenstein 97, Li 98]. While the database driven approach aims at ultimately building a global federated system, the user driven approach trades the benefits of integration for the simplicity of multidatabase operations. It is our feeling that the user driven approach is more rewarding in the short term, as ad hoc services are easily implemented, but may in the long term result in a chaotic system with no global view of the information system and no global consistency. Notice that whatever the approach, the issues that we discussed (identification of correspondences, conflict resolution, integration rules) are relevant.

It is worthwhile mentioning that there is a basic split in the philosophy behind integration methodologies that characterizes methodologies as manual or semi-automatic. Manual strategies build on the fact that necessary knowledge of data semantics is with the DBA, not in the databases. Hence they choose to let the DBA lead the integration process. They just provide a language for schema manipulation, that the DBA uses to build (if the language is procedural) or to define (if the language is declarative) the integrated schema. Procedural languages offer schema transformation primitives which allow to restructure input schemas up to the point where they can be merged by a mere union operation into a unified schema. The system automatically maintains the mappings between input schemas and the current integrated schema [Motro 87]. Declarative, logical languages are easier to use, as the

DBA or user only has to define the rules inferring the integrated schema from the input schemas [Li 98]. Manual strategies are easier to implement, but they can only be operational if the DBA knows which integrated schema is to be installed. This may not be the case, resulting in many iterations (try and go) before a correct result is achieved. Conversely, semi-automatic strategies aim at building a tool, which automatically performs integration once the ICAs are defined. The tool also defines the mappings. The DBA keeps responsibility for the identification of the ICAs and for the choice among integration alternatives.

The issue likely to be tackled next is the heterogeneity in existing data models and DBMS. Integration methodologies generally assume that all input schemas have been translated into a common model. In fact, they only integrate schemas that are expressed in their own data model. In current terms, each participating DBMS is equipped with a wrapper that ensures this homogenization task. A different approach has been proposed in [Spaccapietra 92]. It advocates that problems and solutions for each type of conflicts are basically the same ones, irrespectively of data models. It is therefore feasible to identify the set of fundamental integration rules that are needed and to define, for any specific data model, how each rule can be applied by reformulating the rule according to the peculiarities of the model under consideration. A tool can then be built, capable of supporting direct integration of heterogeneous schemas and of producing an integrated schema in any known data model. Higher order logic has also been suggested as a formalism capable of solving all types of conflicts, including heterogeneity [Lakshmanan 93, 96]. Such a language allows users to directly define the integrated schema over heterogeneous input schemas.

Most approaches today recommend building a common ontology before integration starts, i.e. a repository of all current knowledge in the organization or beyond [Lee 96, Bressan 97]. To some extent this is similar to the data warehouse approach. The ontology describes the semantics of all concepts and the relationships in between, and is therefore capable of correctly identifying interdatabase correspondences. If a new database joins the existing federation, its schema is used to enrich the ontology with the additional knowledge it may contain. In case of conflict, the ontology dominates the new schemas [Collet 91]. The content of an ontology is not limited to existing schemas. It includes the description of the contextual knowledge that is necessary to support proper interpretation of the specific semantics of each database. For instance, it will contain a definition of a car as seen in the different databases, to make sure there is no confusion: a Car type in one database may classify a van as a car, while another database may have a specific Van type such that a van is not a car.

7 Conclusion

Integrating existing databases is a very difficult task. Still, it is something that enterprises face today and cannot avoid if they want to launch new applications or to reorganize the existing information system for better profitability.
We have discussed basic issues and solutions. We focused on the fundamental concepts and techniques, insisting on the alternatives and on criteria for choice. More details are easily found in an over-abundant literature. To the best of our knowledge, no integration tool has yet been developed as a commercial product. Some research

projects have produced significant prototypes, e.g. [Bayardo 97, Genesereth 97, Yan 97, Li 98]. Some other are on their way, e.g. [Klas 95] and [Lee 96] for relational databases. One commercial product, dbMain, intended for schema maintenance and engineering, and database reverse engineering, is being extended with capabilities for schema integration [Thiran 98].

Despite research has been active for nearly twenty years, with a significant increase in the very last years, several important problems remain to be investigated, at least to some extent. Examples of these are: integration of complex objects (as commonly found in object-oriented databases), complex correspondences (fragmentation conflicts), consideration of integrity constraints and methods, direct integration of heterogeneous databases. Theoretical work is still needed to assess integration rules and their properties (commutativity, associativity, ...), as well as heuristics in using the rules. It is therefore important that the effort to solve integration issues be continued and that proposed methodologies are evaluated through experiments with real applications.

References

[Abdelmoty 97] Abdelmoty A., Jones C.B. Towards Maintaining Consistency of Spatial Databases. In *Proceedings of the ACM International Conference on Information and Knowledge Management, CIKM'97* (November 10-14, Las Vegas, USA), 1997, pp. 293-300

[Agarwal 95] Agarwal S., Keller A.M., Wiederhold G., Saraswat S. Flexible Relation: An Approach for Integrating Data from Multiple, Possibly Inconsistent Databases. In *Proceedings of the 11th International Conference on Data Engineering* (March 6-10, Taipei, Taiwan), 1995, IEEE CS Press, pp. 495-504

[Albert 96] Albert J. Data Integration in the RODIN Multidatabase System. In *Proceedings First IFCIS International Conference on Cooperative Information Systems* (June 19-21, Brussels, Belgium), 1996, IEEE CS Press, pp. 48-57

[Andersson 98] Andersson M. Searching for semantics in COBOL legacy applications. In *Data Mining and Reverse Engineering*, Spaccapietra S., Maryanski F. (Eds.), Chapman & Hall, 1998, pp. 162-183

[Atzeni 97] Atzeni P., Torlone R. MDM: a Multiple-Data-Model Tool for the Management of Heterogeneous Database Schemes. In *Proceedings of ACM SIGMOD International Conference* (May 13-15, Tucson, AZ, USA), 1997, pp. 528-531

[Bayardo 97| Bayardo R.J. et al. Infosleuth: Agent-Based Semantic Integration of Information in Open and Dynamic Environments. In *Proceedings of ACM SIGMOD International Conference* (May 13-15, Tucson, AZ, USA), 1997, pp. 195-206

[Bressan 97] Bressan S. et al. The Context Interchange Mediator Prototype. In *Proceedings of ACM SIGMOD International Conference* (May 13-15, Tucson, AZ, USA), 1997, pp. 525-527

[Castano 97] Castano S., De Antonellis V. Semantic Dictionary Design for Database Interoperability. In *Proceedings of the 13th International Conference on Data Engineering* (April 7-11, Birmingham, UK), 1997, IEEE CS Press, pp. 43-54

[Ceri 87] Ceri S., Pelagatti G. *Distributed databases: principles & systems*. McGraw-Hill, 1987

[Clifton 98] Clifton C., Housman E., Rosenthal A. Experience with a Combined Approach to Attributed-Matching Across Heterogeneous Databases. In *Data Mining and Reverse Engineering*, Spaccapietra S., Maryanski F. (Eds.), Chapman & Hall, 1998, pp. 428-450

[Collet 91] Collet C., Huhns M.N., Shen W.-M. Resource Integration Using a Large Knowledge Base in Carnot. *Computer*, 24, 12 (December 1991), pp. 55-62

[Davidson 97] Davidson S.B., Kosky A.S. WOL: A Language for Database Transformations and Constraints. In *Proceedings of the 13th International Conference on Data Engineering* (April 7-11, Birmingham, UK), 1995, IEEE CS Press, pp. 55-65

[De Rosa 98] De Rosa M., Catarci T. Iocchi L., Nardi D., Santucci G. Materializing the Web. In *Proceedings Third IFCIS International Conference on Cooperative Information Systems* (August 20-22, New York, USA), 1998, IEEE CS Press, pp.24-31

[Devogele 98] Devogele T., Parent C., Spaccapietra S. On Spatial Database Integration. *International Journal of Geographic Information Systems*, Special Issue on System Integration, 12, 4, (June 1998), pp. 315-352

[Dung 96] Dung P.M. Integrating Data from Possibly Inconsistent Databases. In *Proceedings First IFCIS International Conference on Cooperative Information Systems* (June 19-21, Brussels, Belgium), 1996, IEEE CS Press, pp.58-65

[Dupont 94] Dupont Y. Resolving Fragmentation Conflicts in Schema Integration. In *Entity-Relationship Approach - ER'94*, P. Loucopoulos Ed., LNCS 881, Springer-Verlag, 1994, pp. 513-532

[Fankhauser 92] Fankhauser P., Neuhold E.J. Knowledge based integration of heterogeneous databases. In *Proceedings of IFIP DS-5 Conference on Semantics of Interoperable Database Systems* (November 16-20, Lorne, Australia), 1992, pp. 150-170

[Garcia-Solaco 95] Garcia-Solaco M., Saltor F., Castellanos M. A Structure Based Schema Integration Methodology. In *Proceedings of the 11th International Conference on Data Engineering* (March 6-10, Taipei, Taiwan), 1995, IEEE CS Press, pp. 505-512

[Genesereth 97] Genesereth M.R., Keller A.M., Duschka O.M. Informaster: An Information Integration System. In *Proceedings of ACM SIGMOD International Conference* (May 13-15, Tucson, AZ, USA), 1997, pp. 539-542

[Gotthard 92] Gotthard W., Lockemann P.C., Neufeld A. System-Guided View Integration for Object-Oriented Databases. *IEEE Transactions on Knowledge and Data Engineering*, 4, 1 (February 1992), pp. 1-22

[Hammer 97] Hammer J. et al. Template-Based Wrappers in the TSIMMIS System. In *Proceedings of ACM SIGMOD International Conference* (May 13-15, Tucson, AZ, USA), 1997, pp. 532-535

[Hainaut 98] Hainaut J.-L., Englebert V., Hick J.-M., Henrard J., Roland R. Contribution to the reverse engineering of OO applications: methodology and case study. In *Data Mining and Reverse Engineering*, Spaccapietra S., Maryanski F. (Eds.), Chapman & Hall, 1998, pp. 131-161

[Hohenstein 97] Hohenstein U., Plesser V. A Generative Approach to Database Federation. In *Conceptual Modeling - ER'97*, Embley D.W., Goldstein R.C. (Eds.), LNCS 1331, Springer, 1997, pp. 422-435

[Kahng 96] Kahng J., McLeod D. Dynamic Classificational Ontologies for Discovery in Cooperative Federated Databases. In *Proceedings First IFCIS International Conference on Cooperative Information Systems* (June 19-21, Brussels, Belgium), 1996, IEEE CS Press, pp. 26-35

[Kaul 90] Kaul M., Drosten K., Neuhold E.J. ViewSystem: Integrating Heterogeneous Information Bases by Object-Oriented Views. In *Proceedings of the 6th International Conference on Data Engineering* (February 5-9, Los Angeles, USA), 1990, IEEE CS Press, pp. 2-10

[Kent 92] Kent W., Ahmed R., Albert J., Ketabchi M., Shan M.-C. Object Identification in Multidatabase Systems. In *Proceedings of IFIP DS-5 Conference on Semantics of Interoperable Database Systems* (November 16-20, Lorne, Australia), 1992

[Kim 93] Kim W., Choi I., Gala S., Scheevel M. On Resolving Schematic Heterogeneity in Multidatabase Systems. *Distributed and Parallel Databases*, 1, 3, (July 1993), pp. 251-279

[Kim 95] Kim W. (Ed.) *Modern Database Systems: The Object Model, Interoperability and Beyond*, ACM Press and Addison Wesley, 1995

[Klas 95] Klas W., Fankhauser P., Muth P., Rakow T.C., Neuhold E.J. Database Integration using the Open Object-Oriented Database System VODAK. In *Object Oriented Multidatabase Systems: A Solution for Advanced Applications*, Bukhres O., Elmagarmid A.K. (Eds.), Prentice Hall, 1995

[Lakshmanan 93] Lakshmanan L.V.S., Sadri F., Subramanian I.N. On the Logical Foundation of Schema Integration and Evolution in Heterogeneous Database Systems. In *Deductive and Object-Oriented Databases*, Ceri S., Tanaka K., Tsur S. (Eds.), LNCS 760, Springer-Verlag, 1993, pp. 81-100

[Lakshmanan 96] Lakshmanan L.V.S., Sadri F., Subramanian I. SchemaSQL – A Language for Interoperability In Relational Multi-database Systems. In *Proceedings of the 22nd VLDB Conference* (September 3-6, Mumbai, India), 1996, pp. 239-250

[Larson 89] Larson J.A., Navathe S.B., Elmasri R. A Theory of Attribute Equivalence in Databases with Application to Schema Integration. *IEEE Transactions On Software Engineering*, 15, 4, (April 1989), pp. 449-463

[Lee 96| Lee J., Madnick S.E., Siegel M.D. Conceptualizing Semantic Interoperability: A Perspective from the Knowledge Level. *International Journal of Cooperative Information Systems*, 5, 4, (December 1996), pp.367-393

[Li 94] Li W.S., Clifton C. Semantic Integration in Heterogeneous Databases Using NeuralNetworks. In In *Proceedings of the 20th VLDB Conference* (Santiago, Chile), 1994, pp. 1-12

[Li 98] Li C. et al. Capability Based Mediation in TSIMMIS. In *Proceedings of the 1998 ACM SIGMOD Conference*, (June 1-4, Seattle, USA), ACM SIGMOD Record, 27, 2, (June 1998), pp.564-566

[Litwin 90] Litwin W., Mark L., Roussopoulos N. Interoperability of multiple autonomous databases. *ACM Computer Surveys*, 22, 3 (Sept. 1990), pp. 267-293

[Lu 98] Lu H., Fan W., Goh C.H., Madnick S.E., Cheung D.W. Discovering and Reconciling Semantic Conflicts: A Data Mining Perspective. In *Data Mining and Reverse Engineering*, Spaccapietra S., Maryanski F. (Eds.), Chapman & Hall, 1998, pp. 409-426

[McBrien 97] McBrien P., Poulovassilis A. A Formal Framework for ER Schema Transformation. In *Conceptual Modeling - ER'97*, Embley D.W., Goldstein R.C. (Eds.), LNCS 1331, Springer, 1997, pp. 408-421

[Mena 96] Mena E., Kashyap V., Sheth A., Illarramendi A. OBSERVER: An Approach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies. In *Proceedings First International Conference on Cooperative Information Systems* (June 19-21, Brussels, Belgium), 1996, IEEE CS Press, pp. 14-25

[Metais 97] Metais E., Kedad Z., Comyn-Wattiau I., Bouzeghoub M. Using Linguistic Knowledge in View Integration: Toward a Third Generation of Tools. Data and Knowledge Engineering, Vol. 23, No. +, June 1997, pp. 59-78

[Miller 93] Miller R.J., Ioannidis Y.E., Ramakrishnan R. Understanding Schemas. In *Proceedings of RIDE-IMS'93 Interoperability in Multidatabase Systems* (April 19-20, Vienna, Austria), 1993, pp. 170-173

[Milliner 95] Milliner S., Bouguettaya A., Papazoglou M. A Scalable Architecture for Autonomous Heterogeneous Database Interactions. In *Proceedings of the 21st VLDB Conference* (Zurich, Switzerland), 1995, pp. 515-526

[Missier 97] Missier P., Rusinkiewicz M. Extending a Multidatabase Manipulation Language to Resolve Schema and Data Conflicts. In *Database Application Semantics*, Meersamn R. and Mark L. eds., Chapman & Hall, 1997, pp. 93-115

[Motro 87] Motro A. Superviews: Virtual integration of multiple databases. *IEEE Transactions On Software Engineering*, 13, 7, (July 1987), pp. 785-798

[Nicolle 96] Nicolle C., Benslimane D., Yétongnon K. Multi-Data Models Translations In Interoperable Information Systems. In *Advanced Information Systems Engineering*, Constantopoulos P., Mylopoulos J., Vassiliou Y. (Eds.), LNCS 1080, Springer, 1996, pp. 176-192

[Ouksel 96] Ouksel A.M., Ahmed I. Coordinating Knowledge Elicitation to Support Context Construction in Cooperative Information Systems. In *Proceedings First IFCIS International Conference on Cooperative Information Systems* (June 19-21, Brussels, Belgium), 1996, IEEE CS Press, pp. 4-13

[Papakonstantinou 95] Papakonstantinou Y., Garcia-Molina H., Widom J. Object Exchange Across Heterogeneous Information Sources. In *Proceedings of the 11th International Conference on Data Engineering* (March 6-10, Taipei, Taiwan), 1995, IEEE CS Press, pp. 251-260

[Papakonstantinou 96] Papakonstantinou Y., Abiteboul S. Garcia-Molina H. Object Fusion in Mediator Systems. In *Proceedings of the 22nd VLDB Conference* (September 3-6, Mumbai, India), 1996, pp. 413-424

[Papazoglou 96] Papazoglou M., Russell N., Edmond D. A Translation Protocol Achieving Consensus of Semantics between Cooperating Heterogeneous Database Systems. In *Proceedings First International Conference on Cooperative Information Systems* (June 19-21, Brussels, Belgium), 1996, IEEE CS Press, pp. 78-89

[Saltor 92] Saltor F., Castellanos M.G., Garcia-Solaco M. Overcoming Schematic Discrepancies in Interoperable Databases. In *Proceedings of IFIP DS-5 Conference on Semantics of Interoperable Databases Systems*, (Nov. 16-20, Lorne, Australia), 1992, pp. 184-198

[Schmitt 96] Schmitt I., Saake G. Integration of Inheritance Trees as Part of View Generation for Database Federations. In *Conceptual Modeling – ER'96*, Thalheim B. (Ed.), LNCS 1157, Springer, 1996, pp. 195-210

[Schmitt 98] Schmitt I., Saake G. Merging Inheritance Hierachies for Database Integration. In *Proceedings Third IFCIS International Conference on Cooperative Information Systems* (August 20-22, New York, USA), 1998, IEEE CS Press, pp.322-331

[Scholl 94] Scholl M.H., Schek H.-J., Tresch M. Object Algebra and Views for Multi-Objectbases. In *Distributed Object Management*, Ozsu T., Dayal U., Valduriez P. (Eds.), Morgan Kaufmann, 1994, pp. 353-374

[Sester 98] Sester M. Interpretation of Spatial Data Bases using Machine Learning Techniques. In *Proceedings 8th International Symposium on Spatial Data Handling*, (July 11-15, Vancouver, Canada), 1998, IGU, pp. 88-97

[Sheth 90] Sheth A., Larson J. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computer Surveys*, 22, 3 (Sept. 1990), pp. 183-236

[Sheth 92] Sheth A., Kashyap V. So Far (Schematically) yet So Near (Semantically). In *Proceedings of IFIP DS-5 Conference on Semantics of Interoperable Databases Systems*, (Nov. 16-20, Lorne, Australia), 1992, pp. 272-301

[Sheuermann 94] Sheuermann P., Chong E.I. Role-Based Query Processing in Multidatabase Systems. In *Advances in Database Technology - EDBT'94*, Jarke M., Bubenko J., Jeffery K. (Eds.), LNCS 779, Springer-Verlag, 1994, pp. 95-108

[Si 96] Si A., Ying C., McLeod D. On Using Historical Update Information for Instance Identification in Federated Databases. In *Proceedings First IFCIS International Conference on Cooperative Information Systems* (June 19-21, Brussels, Belgium), 1996, IEEE CS Press, pp. 68-77

[Spaccapietra 91] Spaccapietra S., Parent C. Conflicts and Correspondence Assertions in Interoperable Databases, *ACM SIGMOD Record*, 20, 4, (December 1991), pp. 49-54

[Spaccapietra 92] Spaccapietra S., Parent C., Dupont Y. Model Independent Assertions for Integration of Heterogeneous Schemas. *VLDB Journal*, 1, 1 (July 1992), pp. 81-126

[Tari 97] Tari Z., Stokes J., Spaccapietra S. Object Normal Forms and Dependency Constraints for Object-Oriented Schemata. *ACM Transactions On Database Systems*, 22, 4, (December 1997), pp. 513-569

[Tari 98] Tari Z., Bukhres O., Stokes J., Hammoudi S. The reengineering of relational databases based on key and data correlation. In *Data Mining and Reverse Engineering*, Spaccapietra S., Maryanski F. (Eds.), Chapman & Hall, 1998, pp. 184-215

[Thiran 98] Thiran Ph., Hainaut J.-L., Bodart S., Deflorenne A., Hick J.-M. Interoperation of Independent, Heterogeneous and Distributed Databases. Methodology and CASE Support: the InterDB Approach. In *Proceedings Third IFCIS International Conference on Cooperative Information Systems* (August 20-22, New York, USA), 1998, IEEE CS Press, pp. 54-63

[Tseng 93] Tseng F.S.C., Chen A.L.P., Yang W.-P. Answering Heterogeneous Database Queries with Degrees of Uncertainty. *Distributed and Parallel Databases*, 1, (1993), pp. 281-302

[Urban 91] Urban S.D. A Semantic Framework for Heterogeneous Database Environments. In *Proceedings of RIDE-IMS'91 Interoperability in Multidatabase Systems* (April 7-9, Kyoto, Japan), 1991, pp. 156-163

[Vermeer 96] Vermeer M., Apers P. On the Applicability of Schema Integration Techniques to Database Interoperation. In *Conceptual Modeling – ER'96*, Thalheim B. (Ed.), LNCS 1157, Springer, 1996, pp.

[Yan 97] Yan L.L., Otsu M.T., Liu L. Accessing Heterogeneous Data Through Homogenization and Integration Mediators. In *Proceedings Second IFCIS International Conference on Cooperative Information Systems* (June 24-27, Kiawah Island, SC, USA), 1997, IEEE CS Press, pp.130-139

[Zhou 95] Zhou G., Hull R., King R., Franchitti J.-C. Using Object Matching and Materialization to Integrate Heterogeneous Databases. In *Proceedings of the Third International Conference on Cooperative Information Systems* (May 9-12, Vienna, Austria), 1995, pp. 4-18