Issues and Approaches of Database Integration

Christine Parent and Stefano Spaccapietra

In many large companies the widespread usage of computers has led a number of different application-specific databases to be installed. As company structures evolve, boundaries between departments move, creating new business units. Their new applications will use existing data from various data stores, rather than new data entering the organization. Henceforth, the ability to make data stores interoperable becomes a crucial factor for the development of new information systems.

Data interoperability may come in various degrees. At the lowest level, commercial gateways connect specific pairs of database management systems (DBMSs). Software providing facilities for defining persistent views over different databases [6] simplifies access to distant data but does not support automatic enforcement of consistency constraints among different databases. Full interoperability is achieved by distributed or federated database systems, which support integration of existing data into virtual databases (i.e. databases which are logically defined but not physically materialized). The latter allow existing databases to remain under control of their respective owners, thus supporting a harmonious coexistence of scalable data integration and site autonomy requirements [9]. Federated systems are very popular today. However, before they become marketable, many issues remain to be solved. Design issues focus on either human-centered aspects (cooperative work, including autonomy issues and negotiation procedures) or database-centered aspects (data integration, schema/database evolution). Operational issues investigate system interoperability mainly in terms of support of new transaction types, new query processing algorithms, security concerns, etc. General overviews may be found elsewhere [4, 9]. This paper is devoted to database integration, possibly the most critical issue. Simply stated, database integration is the process which takes as input a set of databases, 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. As such, database integration is also used in the process of re-engineering an existing legacy system.

Database integration has attracted many diverse and diverging contributions. The purpose, and the main intended contribution of this article is to provide a clear picture of what are the approaches and the current solutions and what remains to be achieved. We focus on fundamental concepts and alternatives. We first give an example to provide a context for the rest of the paper. We then present the steps involved in developing an integrated schema (see Figure 1):

- Pre-integration, where input schemas are transformed to make them more homogenous (both syntactically and semantically); homogenous (both syntactically and semantically);
- Correspondence identification, devoted to the identification and description of interschema relationships; and
- Integration, the final step which solves interschema conflicts and unifies corresponding items into an integrated schema.

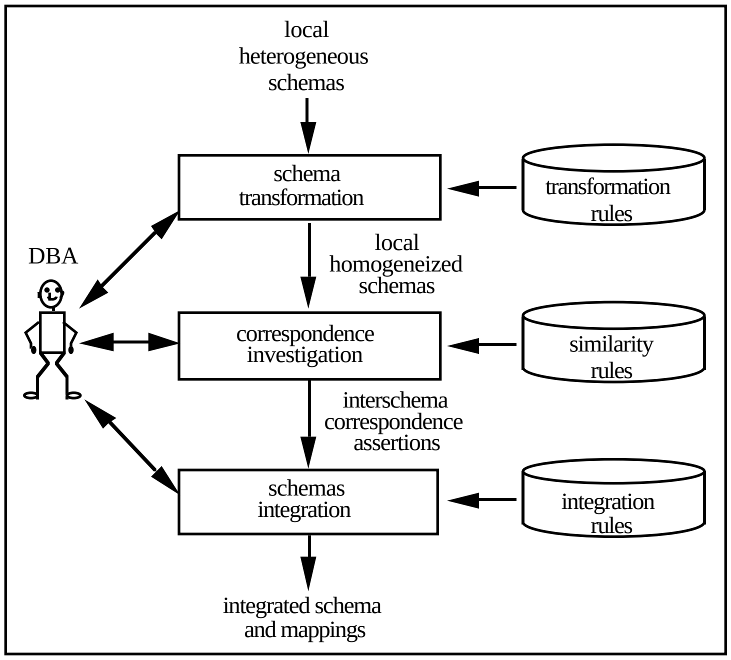

local heterogeneous schemas
schema transformation
transformation rules
DBA
local homogeneized schemas
correspondence investigation
similarity rules
interschema correspondence assertions
schemas integration
integration rules
integrated schema and mappings

Figure 1. The global integration process

Our example shows records for two computer science libraries. The first librarian uses an extended entity-relationship model (schema S1). This database is about journal and conference papers. A generic object "publication" represents either a journal issue (such as *Communications*, April 1994) or a conference proceedings (such as SIGMOD 1993). Each publication is described by generic properties: ISN (International Standard Number), name, year. Conference proceedings have location, editors and number (20th, 21st, ...) as additional information. Each publication contains at least one paper, and each recorded paper belongs to at least one publication.

The second librarian (schema S2) uses an object-oriented (OO) model and is only interested in conference papers. There is one object per conference type (SIGMOD). Proceedings of actual conferences (SIGMOD 1993, SIGMOD 1994, ...) are represented as a multivalued attribute (one value per venue), where each value contains a set of references to the papers which appeared that year in that conference.

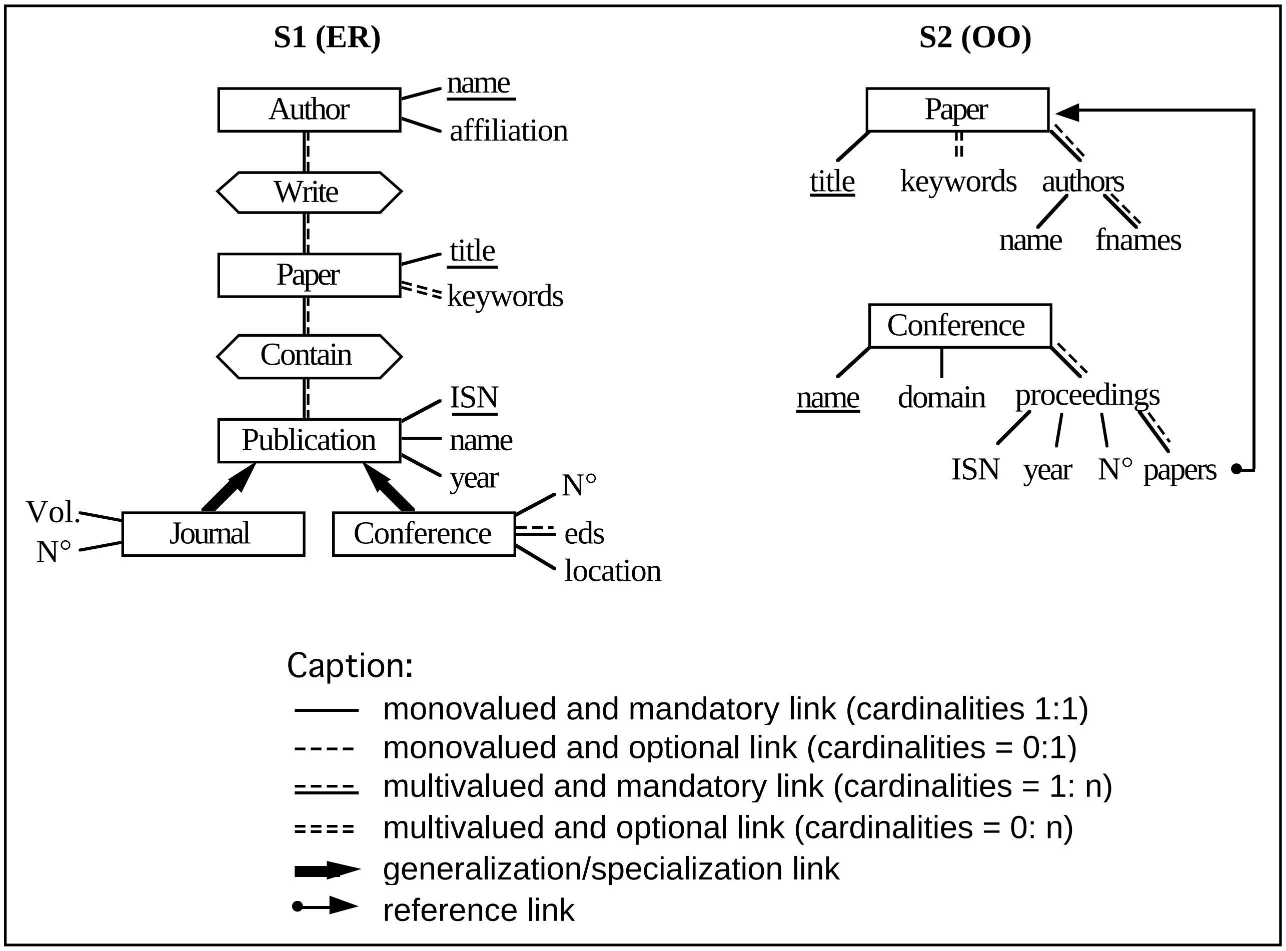

S1 (ER)
S2 (OO)
Author
name
affiliation
Write
Paper
title
keywords
Contain
Publication
ISN
name
year
Journal
Vol.
N°
Conference
N°
eds
location
Paper
title
keywords
authors
name
fnames
Conference
name
domain
proceedings
ISN
year
N°
papers
Caption:
monovalued and mandatory link (cardinalities 1:1)
monovalued and optional link (cardinalities = 0:1)
multivalued and mandatory link (cardinalities = 1: n)
multivalued and optional link (cardinalities = 0: n)
generalization/specialization link
reference link

Figure 2. The publication example

The Pre-integration Step

Establishing a common understanding of existing data is a pre-requisite to successful database integration. To this aim, input schemas are usually transformed to make them as homogeneous as possible.

Researchers in database integration generally assume that input schemas are all expressed in the same data model, the so-called "common" data model (CDM). A translation step becomes a pre-requisite to integration and is dealt with as a separate problem. Unfortunately, the state of the art in data model translation is poor in tools for automatic translation. Current developments focus on translations between OO and relational models. Some researchers have also considered translating operations, which is needed for a multilingual system allowing every partner to speak his own language.

One of the unresolved debates is the choice of the CDM. Most researchers favor the OO model. The argument is that it has all concepts of the other models and that methods can be used to implement specific mapping rules. But the richer the model is, the more complex the integration process is going to be, as many discrepancies will arise due to different modeling choices by the various designers. To simplify integration, the alternative is a CDM with minimal semantics, where data representations are brought down to elementary facts for which there is no modeling alternative, as in binary-relationship models.

Instead of dealing with specific translations, e.g. from model X to model Y, recent works look for generic techniques. Generic translators use a metamodel, i.e. a data model capable of describing data models, to acquire knowledge about data models. Translations are then performed as a sequence of data restructuring process within the metamodel database (for instance, nesting flat tuples to obtain a nested tuple).

Data models cannot express all the semantics of the real world. The incompleteness of their specifications leads to ambiguities in the interpretation of a schema. Semantic enrichment is the process of acquiring additional information to solve such ambiguities. The hardest case is when data resides in files, but understanding unnormalized and poorly documented relational databases also sets a serious challenge. Reverse-engineering techniques are being developed to recompose object types from relations, and to identify association structures and generalization hierarchies. These techniques are based on an analysis of the schema, indexes, queries, and instances [1].

Semantic enrichment provides information either on how a schema maps to the real world, or on the schema itself. An example of the former is adding to the schema of S1 a definition of "Conference" as "any annual venue of a computer science meeting whose proceedings are stored in the library". An example of the latter is specifying in the description of Conference of schema S1 the functional dependency:

(name, year) $\rightarrow$ ISN.

OO schemas also need to be augmented with information on cardinalities, on dependencies (for a correct translation of multivalued attributes), on the semantics of duplicate values, ...

Modeling is not a deterministic process. Semantic relativism (schemas depending on the designer's perspective) is the price to pay for semantic richness of the data model. Discrepancies may be reduced, at the organizational level by enforcing corporate modeling policies, and at the technical level by applying normalization rules. Syntactic and semantic normalization rules can be defined. The former enforce design rules. Examples are: a type with an optional attribute should be replaced by a supertype/subtype structure where the attribute is mandatory in the subtype; a multivalued attribute should be replaced by an object class and a reference to it. The latter conform the schema with underlying dependencies. An example is: if there is a dependency A->B between attributes A and B of an object type, and A is not a key for the object type, replace these attributes by a tuple attribute with components A and B. Note that these normal forms aim at enhancing the semantics of the schema, not at avoiding update anomalies.

The Correspondence Identification Step

The next step is the identification of commonalties. Databases contain representations of real world facts (objects, links or properties). Database integration goes beyond representations to first look for what is represented rather than how it is represented. Therefore, **two databases are said to have something in common** if the real world facts they represent have some common elements or are otherwise interrelated. An example of the latter is: two separate libraries, one specializing in scientific disciplines and the other one specializing in social sciences, willing to build an integrated database. It would be worthwhile to build integrated object types, like Author (respectively Paper) representing all the authors (respectively all the papers) in either library.

We say that **two elements** (occurrence, value, tuple, ...) from two databases **correspond to each other** if they describe the same real world fact (object, link or property). Rather than defining correspondences extensionally, between instances, correspondences are defined intensionally, between types. Example: every S2 *paper* corresponds to the S1 *paper* such that the value for *title* in S1 is equal to the value for *title* in S2. The intensional definition is called an **interschema correspondence assertion** (ICA). The integration process consists in determining these ICAs and for each one providing federated users with a global description with all the data available on the related elements. The federated system stores the global description in the integrated schema (IS), and the definition of the mappings between the IS and the local schemas. Complete integration of existing databases requires an exhaustive identification and processing of all relevant ICAs. It is nevertheless possible to adopt an incremental approach, where the IS is smoothly enriched as new correspondences are progressively identified.

The following specifications have to be given to fully define an ICA:

1/ What are the related schema elements

In simple cases, related elements are just denoted by their qualified names. For instance, an ICA relates S1.*Author* to S2.*Paper.authors*, as both elements represent persons who authored a paper. Links are denoted by enumerating the elements they traverse: an ICA relates S1.*Author-Write-Paper* to S2.*authors-Paper* to specify that these two links have the same semantics. Simply stated, if author X has written paper Y in S1, and X and Y also belong to S2, then in S2 X appears in the *authors* attribute of *Paper* instance Y.

In complex cases, algebraic expressions specify the set of instances involved in the correspondence. In our example, only S1 conference papers may appear in S2. The appropriate ICA is:

SELECTION ["is a conference paper"] S1.Paper $\cap$ S2.Paper

Expressions may include more sophisticated operations (unions, joins, ...) [2].

2/ How their potential extensions are related

To build the integrated schema, the relationship in the real world between the extensions of the related elements has to be known. Type extensions are seen as sets of real world elements. Each ICA specifies which of the usual set relationships holds: equivalence ($\equiv$), inclusion ($\supseteq$), intersection ($\cap$) or disjointedness ($\neq$).

Example:

S1.Paper $\cap$ S2.Paper

asserts that our two librarians sometimes record papers from the same conferences.

3/ How corresponding instances are identified

When users request a data element via the integrated schema, the system may have to access a few properties of the element in one database, and other properties in another database. The system has to know how to find in one database the object (instance or value) corresponding to a given object in another database. Therefore, each ICA has to specify the mapping between the corresponding instances: this may be done using a "with corresponding identifiers" (WCI) clause. In the example, assuming there are no homonyms in the titles of papers:

S1.Paper $\cap$ S2.Paper **WCI** title

specifies that papers in the two databases have a common identifier, *title*. This is the simplest case. The general case requires for each database a predicate specifying the corresponding instances/values.

4/ How representations are related

Representations of related elements may show common properties, beyond those used for identification. S1.*Conference* and S2.*Conference.proceedings*, for instance, share the properties *year* and *N°*, plus the *ISN* identifier. To avoid duplication in the integrated schema, ICAs specify such correspondences among properties using a "with corresponding attributes" (WCA) clause. Hence, the following ICA holds:

S1.Conference $\cap$
S2.Conference.proceedings **WCI** ISN **WCA** year, N°

Correspondences between properties may be very complex, due to differences in coding, scales, units, extents, etc. Ad hoc mapping functions may be needed [5].

The set of stated ICAs should 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 $\equiv$ D, B $\equiv$ C. Both cannot be true. Moreover, if A $\equiv$ C is asserted, B $\supseteq$ C and D $\supseteq$ A may be inferred. Only ICAs bearing non-derivable correspondences need to be explicitly stated.

An appropriate set of ICAs for the publication example is given in Figure 3. Figure 4 illustrates an integrated schema built according to these ICAs.

1 S1.SELECTION ["is a conference author"]Author $\cap$ S2.authors **WCI** name
2 S1.SELECTION ["is a conference paper"]Paper $\cap$ S2.Paper **WCI** title **WCA** keywords
3 S1.Conference $\cap$ S2.proceedings **WCI** ISN **WCA** year, N°
4 S1.Conference.name $\cap$ S2.Conference **WCI** name
5 S1.Author—Write—Paper $\equiv$ S2.authors—Paper
6 S1.Conference—Publication—Contain—Paper $\equiv$ S2.proceedings—papers:Paper
7 S1.Conference.name—Conference $\equiv$ S2.Conference—proceedings

Figure 3. A set of ICAs for the example

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 [3]. These tools measure the similarity between two schemas elements by looking for identical or similar characteristics: names, identifiers, components, properties, attributes (name, domain, constraints), methods. A final interaction with the DBA (in)validates the findings and provides additional information (namely, the relationship between extents).

Most tools only analyze syntactic and structural aspects, thus facing the usual issues of synonyms and homonyms leading to wrong inferences. To overcome this, the description of the types can be augmented with a description of their semantics [8], or a terminological knowledge base can be used to explain the terms of the application domain and how they interrelate.

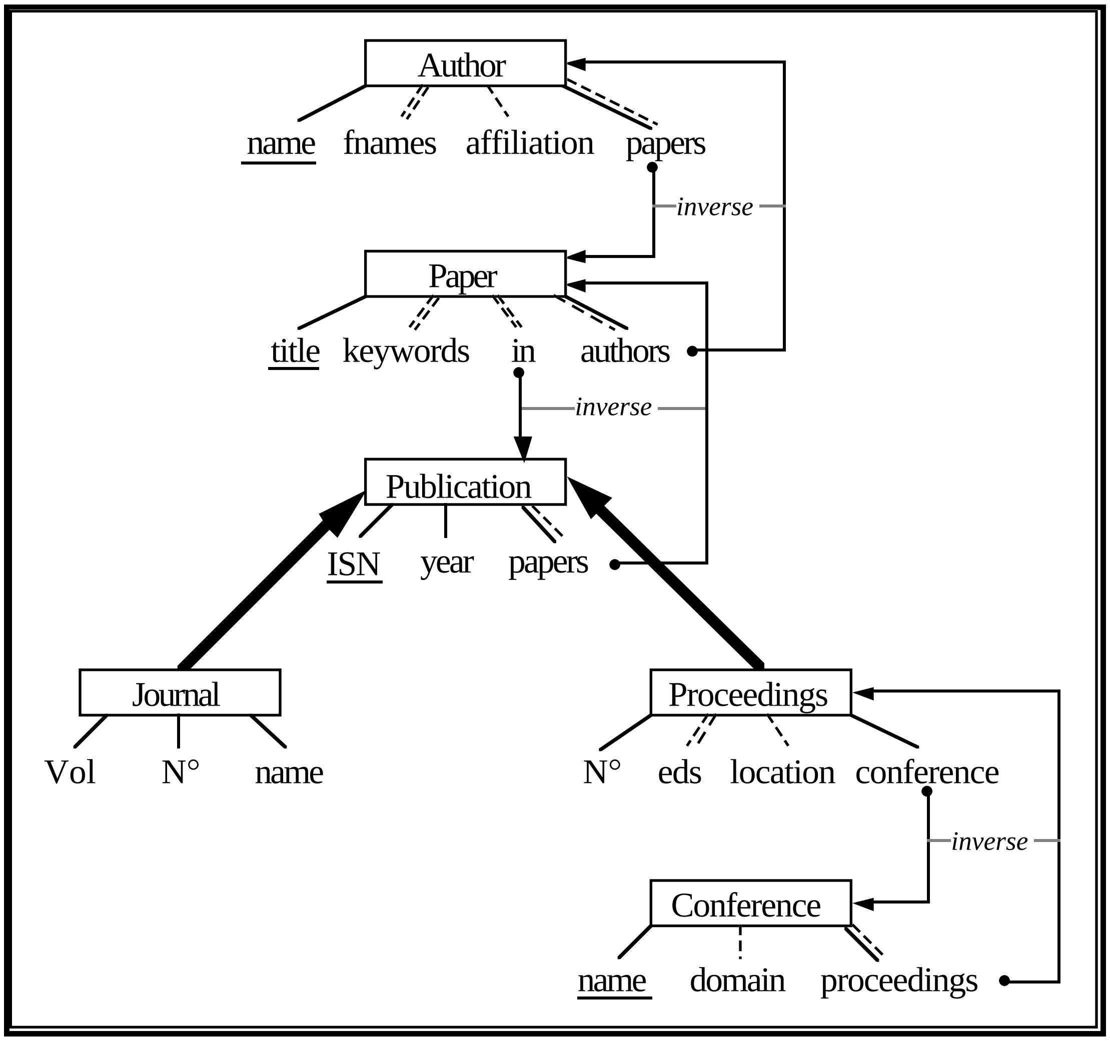


Figure 4. An OO integrated schema for the example

Conflicts: Taxonomy and Solutions

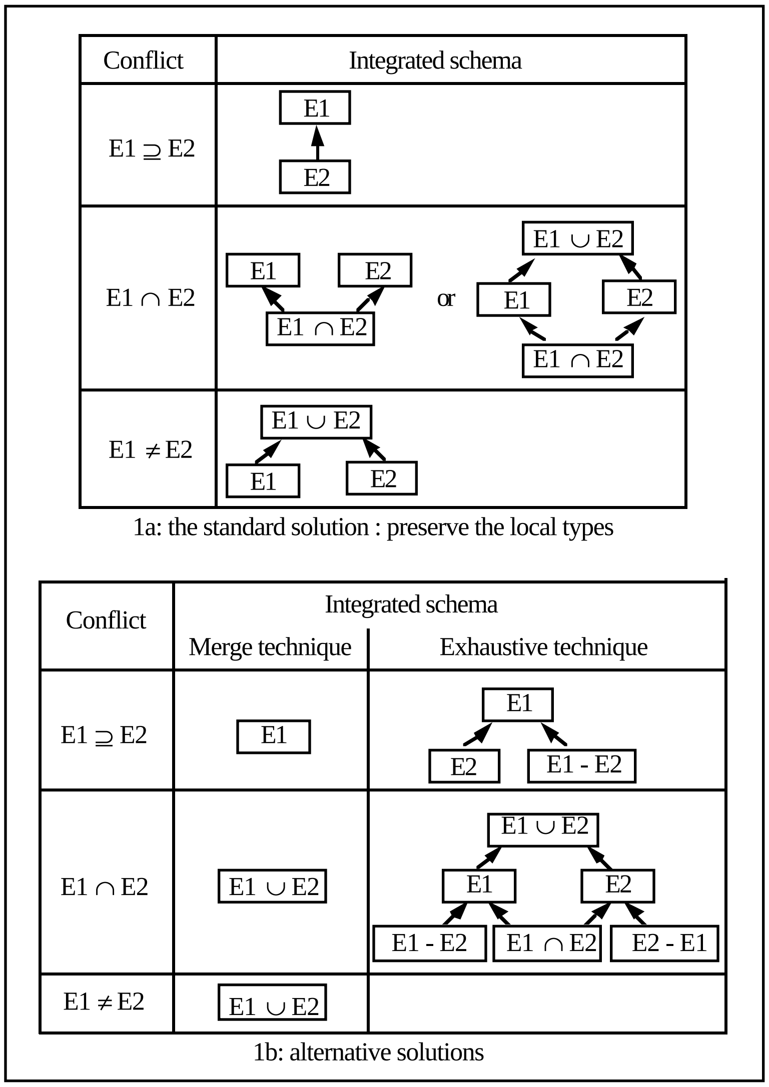
Once correspondences have been described, the integration process comes to the point where actual integration is performed. Each ICA is analyzed to decide which representation of the related elements is to be included in the integrated schema (IS) and to define the mappings between the IS and the input schemas. Integration may be performed manually by the federated DBA, using some procedural [7], declarative or logical manipulation language. The system assists the DBA by generating the mappings. Conversely, integration may be performed semi-automatically by a tool [12]. The DBA in this case is responsible for the ICAs and for the choice of integration alternatives (see Table 1).

When an ICA describes the corresponding types as identical (same representation and equivalent extent), their integration is straightforward: the integrated type equals the input types, and the mapping is the identity. Most frequently, however, the corresponding types will show differences in extent or in representation. This situation is called a conflict. A detailed taxonomy of conflicts can be found in [10]. For the purpose of this survey, a simplified one will suffice. Hereinafter, we survey the various conflicts, indicating existing solutions and open problems.

The literature is rich in proposals for solving classification and description conflicts, but only few proposals exist for metadata conflicts, and even less for structural, heterogeneity [12] and data conflicts [11]. No integration method is complete. The precise **goal of integration** is rarely made explicit. Indeed, one may look for simplicity and readability, therefore producing a minimal number of classes. One may look for completeness, so that every element of an input schema appears in the IS, thus helping in maintaining the IS when input databases evolve. One may also look for exhaustiveness: having in the IS all possible classes, including those who are not in input schemas but complement what is there, might ease future integrations.

A *classification conflict* arises whenever the corresponding types describe different sets of real world elements. In the example, S1 describes authors of journal and conference papers, while S2 describes authors of conference papers only. Classification conflicts are materialized in the ICA by a $\supseteq$, $\cap$ or $\neq$ set relationship.

Based on the completeness goal, a standard solution for such conflicts is to build in the IS the appropriate generalization hierarchy [3, 5], as shown is table 1a. If needed, a common subtype (supertype) is added to achieve connectivity of the generalization hierarchy. Mappings between the IS and input databases reduce to identity functions.



1a: the standard solution : preserve the local types

1b: alternative solutions

Table 1. Solving classification conflicts

Alternatively, the simplicity principle calls for the insertion into the IS of a unique type which describes the union of the extents ("merge technique" in Table 1b). Mappings will then use selection operators to relate the input populations to the integrated type. Finally, the exhaustiveness principle leads to the inclusion into the IS of both input types, together with their union (one supertype) and their intersection plus the complements of the intersection (three subtypes).

In the integrated schema of Figure 4, the merge technique has been applied. For instance, the S1.*Paper* type has been merged with the S2.*Paper* type.

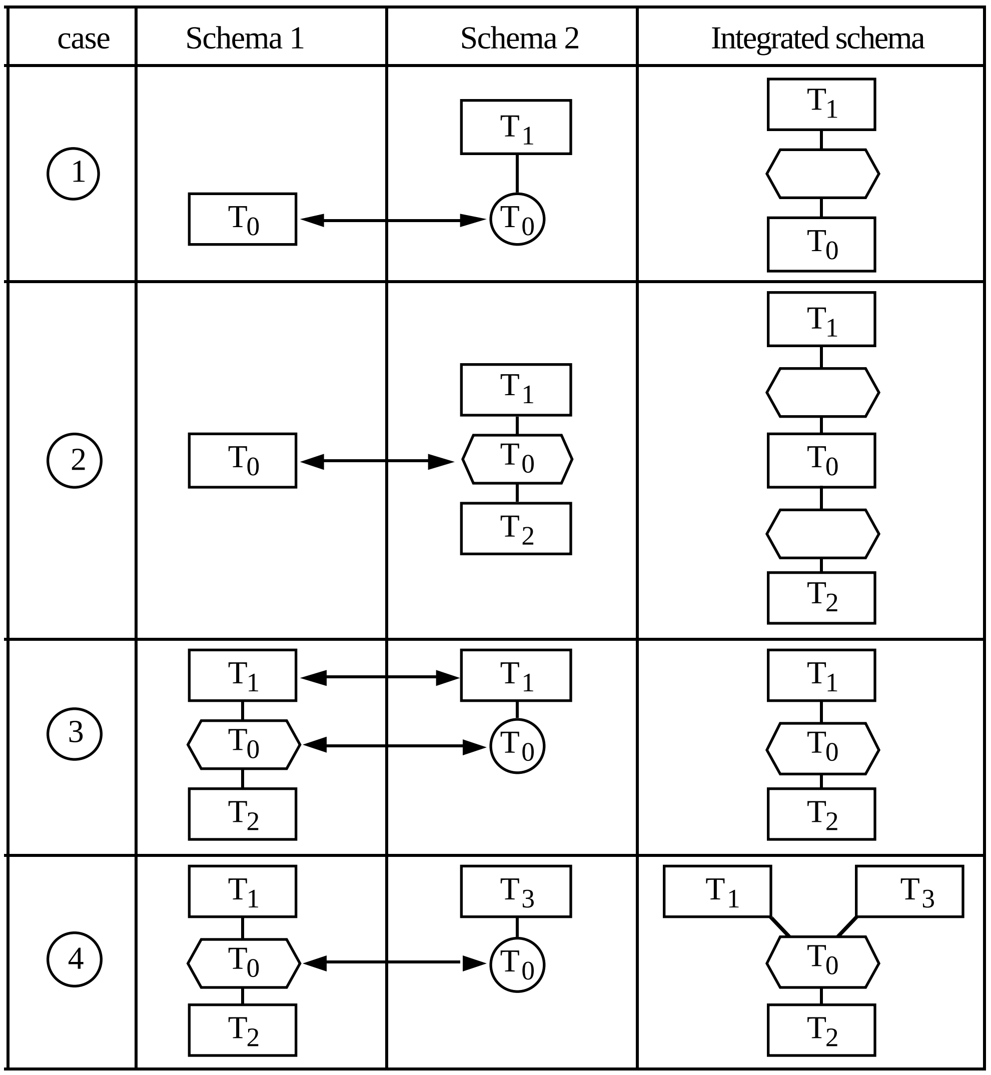
Strategies for classification conflicts have been extended to the integration of **generalization hierarchies** related by multiple ICAs. The hierarchies are merged by taking each class from one hierarchy and determining the place where this class fits in the other hierarchy. Placement is based on class inclusion semantics.

A *structural conflict* arises whenever corresponding types are described with constructs which have different representational power: an object class and an attribute, for instance, or an entity type and a relationship type. In the publication example, ICAs #1 and #4 about authors and about conferences identify structural conflicts. Few contributions discuss this kind of conflict, mostly in the scope of a particular data model [4].

The integrated schema must describe the populations of the two conflicting types [12]. Hence, the integrated type must subsume both input types: subsume their capacity to describe information (adopting the least upper bound) and subsume the attached constraints (adopting the greatest lower bound). Capacity denotes which combination of identity/value/link a construct expresses. For instance, an ER attribute holds a value, while an OO or relational attribute holds either a value or a link. A relational relation holds 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 IS will retain the relation.

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. Considering the ICA relating the S1.*Author* entity type to the S2.*Paper.authors* attribute, the IS will retain the object class. The greatest lower bound in this case is: no constraint.

Table 2 shows which integrated construct may be chosen for each structural conflict. In the example, ICAs #1 and #4 denote case ① conflicts: the S2.*Paper.authors* and S1.*Conference.name* attributes are changed into classes in the IS.



These diagrams are in terms of a generic data model where :

▭ represents an object type (object class, entity type, relation or record type)

⬡ represents a relationship between object types (reference attribute, relationship type, external key or Codasyl set)

○ represents an attribute.

Types with the same name are assumed equivalent : $T_0 \equiv T_0$
(and in case ③ $T_1 \equiv T_1$)

Table 2. Solutions for structural conflicts

More complex structural conflicts (involving for instance join expressions) are possible [2, 4]. A general solution remains to be found.

A *descriptive conflict* arises whenever there is some difference between properties of corresponding types. These conflicts have been widely discussed [4, 5]. Table 3 summarizes many existing taxonomies.

1. Corresponding object types may differ with respect to their:

- names:	homonyms and synonyms
- keys:	different attributes/alternative keys
- attributes:	different sets of attributes conflicting corresponding attributes (refer to point 2 below)
- methods:	different sets of methods conflicting corresponding methods (refer to point 3 below)
- integrity constraints	
- access rights	

2. Corresponding attributes may differ with respect to their:

- names:	homonyms and synonyms
- scopes:	local to one database or global (i.e., meaningful over the integrated database)
- structures:	- simple versus complex (i.e. composed of other attributes) - cardinalities: monovalued versus multivalued, optional versus mandatory - different constructors (set, bag, list, array)
- data types:	different scales, units, default values, operations
- integrity constraints	
- access rights	

3. Corresponding methods may differ with respect to their:

- names:	homonyms and synonyms
- signatures:	different sets of parameters, types of parameters, results

Table 3. Descriptive conflicts

Conflict		Solution
names	homonyms	prefix the names
	synonyms	provide aliases
different keys		use a conversion function or a correspondence table
different sets of attributes		do the union of the sets of attributes (following the least upper bound approach)
conflicting corresponding attributes		use a conversion function or a correspondence table
integrity constraints		take the greatest lower bound of the integrity constraints

Table 4. Classical solutions for descriptive conflicts

Table 4 shows traditional solutions for most frequently discussed conflicts. A global solution has been proposed in [8] by attaching to each attribute a context which describes its semantics. But, up to now, the attribute structure conflict has no real solution. Existing methodologies deal with simple monovalued attributes, rarely with multivalued attributes [5], and ignore complex attributes (attributes composed of other attributes).

In the publication example:

• A naming conflict appears in ICA #3: S1.*Conference* has been renamed *Proceedings* in the IS.

• Different sets of attributes occur in ICAs #1, 3 and 4. In each case the integrated type bears all the attributes of the two corresponding types.

Current integration methodologies only integrate schemas which are expressed in their own data model. Schemas which are not expressed in this data model have to be translated during the pre-integration step.

We advocate that problems and solutions for conflicts are basically independent of data models. It should therefore be feasible to identify the set of fundamental integration rules which 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 [12]. A tool can then be built, capable of supporting direct integration of heterogeneous schemas and of producing an integrated schema in any data model.

Another proposal suggests higher order logic to allow users to directly define the integrated schema over heterogeneous input schemas.

Data/metadata conflicts arise when data (values) in one database correspond to metadata (type names) in the other database. Discussions of these conflicts have been illustrated using a relational Stock example (Figure 5), where a value for stockcode in DB1 corresponds to an attribute name in DB2 and to a relation name in DB3.

DB1 :

Stock (date ,	**stockcode ,**	**price)**
951001	IBM	77.72
951002	IBM	79.23
....	....	....
951001	HP	60.02
951002	HP	61.45
....	....	....

DB2 :

Stock (date ,	**IBM ,**	**HP ,**	**...)**
951001	77.72	60.02	...
951002	79.23	61.45	...
....	....	....	...

DB3 :

IBM (date ,	**price)**
951001	77.72
951002	79.23
....	....

HP (date ,	**price)**
951001	60.02
951002	61.45
....	....

....

Figure 5. An example of data/metadata conflict

Normal relational languages cannot turn an attribute or relation name into a value, nor vice versa. It is not possible to map DB2 or DB3 into DB1. These conflicts can be solved introducing mapping languages which support simultaneous manipulation of both data and metadata. Examples of such languages are an extended relational algebra and a higher order logic.

Alternatively, it is possible to get rid of these conflicts by using a data model inhibiting any usage of meaningful type names. More likely, OO approaches will provide schema transformation operations to perform all needed mappings:

namely, partitioning a class into subclasses according to the value of a specialization attribute (mapping DB1 into DB3), creating a common superclass, with a new classifying attribute, over a set of given classes (mapping DB3 into DB1) ...

Data conflict occurs at the instance level if corresponding occurrences have conflicting values for corresponding attributes. For instance, the same paper is stored in the two publications databases with different keywords. Sources for data conflicts may be: typing errors, variety of information sources, different versioning, deferred updates ...

Data conflicts are normally found during query processing. The system may just report the conflict to the user, or may apply some heuristic to determine the appropriate value. Common heuristics are: choosing the value from "the most reliable" database, 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 the set of possible tuples or values generated by data conflicts.

Conclusion

Integrating existing databases is certainly not an easy task. Still, it is something that enterprises probably cannot avoid if they want to launch new applications or to reorganize the existing information system for a better profit.

We have shown in this article that a basic understanding of the issues and of the solutions is available. We focused on the fundamental concepts and techniques, insisting on the alternatives and on criteria for choice. More details are easily found in the literature.

Several important problems remain to be investigated. Examples are: integration of complex objects, n-n correspondences (fragmentation conflicts), integration of integrity constraints and methods, integration of heterogeneous databases. Theoretical work is needed to assess integration rules and their behavior (commutativity, associativity, ...). It is therefore important that the effort to solve integration issues be continued and that proposed methodologies be evaluated through experiments with real applications.

References

1. Brodie M.L., Stonebraker M. Migrating Legacy Systems: Gateways, Interfaces & The Incremental Approach. Morgan Kaufmann, 1995.

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

3. Gotthard W., Lockemann P.C., Neufeld A. System-guided view integration for object-oriented databases. IEEE Trans. Knowl.Data Eng. 4 , 1 (Feb. 1992), pp. 1–22.

4. Kim W. (Ed.) Modern Database Systems: The Object Model, Interoperability and Beyond. ACM Press and Addison Wesley, Reading, Mass.,1995.

5. Larson J.A., Navathe S.B., Elmasri R. A Theory of attribute equivalence in databases with application to schema integration. IEEE Trans. Softw. Eng. 15, 4 (Apr. 1989), pp. 449–463.

6. Litwin W., Mark L., Roussopoulos N. Interoperability of multiple autonomous databases. ACM Comput. Surveys 22, 3 (Sept.1990), pp. 267–293.

7. Motro A. Superviews: Virtual integration of multiple databases. IEEE Trans. Softw. Eng. 13, 7 (July 1987), pp. 785–798.

8. Sciore E., Siegel M., Rosenthal A. Using semantic values to facilitate interoperability among heterogeneous information systems. ACM TODS 19, 2 (June 1994), pp. 254–290.

9. Sheth A., Larson J. Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Comput. Sur. 22, 3 (Sept. 1990), pp. 183–236.

10. Sheth A., Kashyap V. So Far (Schematically) yet So Near (Semantically). In Interoperable Database Systems (DS-5). D.K. Hsiao, E.J. Neuhold, and R. Sacks-Davis< Eds. IFIP Trans. A-25, North-Holland, 1993, pp. 283–312.

11. Scheuermann P., Chong E.I. Role-Based Query Processing in Multidatabase Systems, In Advances in Database Technology - EDBT'94. M. Jarke, J. Bubenko, and K. Jeffery, Eds. LNCS 779, Springer-Verlag, Germany, 1994, pp. 95–108.

12. Spaccapietra S., Parent C., Dupont Y. Model independent assertions for integration of heterogeneous schemas. VLDB J. 1, 1 (July 1992), pp. 81–126.

CHRISTINE PARENT (parent@di.epfl.ch) is a professor in the computer science department at the University of Burgundy in Dijon, France. She is currently leading a research project on spatial database modeling at the Swiss Federal Institute of Technology in Lausanne, Switzerland.

STEFANO SPACCAPIETRA (spaccapietra@di.epfl.ch) a professor in the computer science department at the Swiss Federal Institute of Technology in Lausanne, Switzerland. He is currently involved in the development of visual user interfaces and cooperative design methodologies.