VLDB Journal, 1, 81-126 (1992) Joachim Schmidt, Editor

Model Independent Assertions for Integration of Heterogeneous Schemas

Stefano Spaccapietra, Christine Parent, and Yann Dupont

Received February 15, 1991; revised version received October 9, 1991; accepted December 4, 1991.

Abstract. Due to the proliferation of database applications, the integration of existing databases into a distributed or federated system is one of the major challenges in responding to enterprises' information requirements. Some proposed integration techniques aim at providing database administrators (DBAs) with a view definition language they can use to build the desired integrated schema. These techniques leave to the DBA the responsibility of appropriately restructuring schema elements from existing local schemas and of solving inter-schema conflicts. This paper investigates the *assertion-based* approach, in which the DBA's action is limited to pointing out corresponding elements in the schemas and to defining the nature of the correspondence in between. This methodology is capable of: ensuring better integration by taking into account additional semantic information (assertions about links); automatically solving structural conflicts; building the integrated schema without requiring conforming of initial schemas; applying integration rules to a variety of data models; and performing view as well as database integration. This paper presents the basic ideas underlying our approach and focuses on resolution of structural conflicts.

Keywords. Database design and integration, distributed databases, federated databases, heterogeneous databases, schema integration, conceptual modeling.

1. Introduction

Interoperability is becoming one of the most critical issues for medium to large size enterprises. Due to the complexity and worldwide span of economy today, enterprise management often needs access to several local as well as remote information resources. Moreover, "even a single enterprise may have heterogeneous information

Stefano Spaccapietra is Professor, and Yann Dupont is Research Assistant, Ecole Polytechnique Fédérale de Lausanne, DI-Laboratoire Bases de Données, IN-Ecublens, 1015 Lausanne, Switzerland. Christine Parent is Professor, Université de Bourgogne, Département Informatique, B.P. 138–21004 Dijon Cedex, France. (Reprint requests to Prof. Spaccapietra, Département d'Informatique, Laboratoire de Base de Données, EPFL-Ecublens, CH-1015, Lausanne, Switzerland.)

bases for reasons of history or departmental autonomy"(Kaul, 1990). The user community is therefore experiencing the need for interconnection of existing, possibly heterogeneous, databases, to provide more data to their applications, as well as the need for enforcing a better consistency among databases containing related information (for instance, a manufacturing database including data on employee's activities and a personnel database).

A variety of approaches to interoperability have been proposed, aiming at different levels of integration (Sheth, 1990). The loosest degree of integration characterizes the multidatabase[1] approach (Litwin, 1984, 1990), in which users can query different databases with a single request, but have to specify where data are located. Federated approaches (Landers, 1982; Larson, 1989) support location transparency (users may ignore the actual location of data and nevertheless query distributed data). The major goal is site autonomy: each site controls the evolution of its local database (updates, restructuring) as well as the usage of its data from other sites (usually through the definition of export schemas). In such a system, one or more global views are built on each site, each one extending the local database with selected parts of external databases.

Conversely, in integrated approaches, local databases are integrated into a single distributed database (DDB). A distributed DBMS (DDBMS) is built to manage the DDB. Most of DDBMS are monolingual: they support a single data manipulation language, which has to be used to query and update the DDB (Ferrier, 1982; Stocker, 1984) Any existing program on a local database has to be rewritten using the distributed DDL/DML. To avoid this burden, some authors advocated that integrated systems should be multilingual, i.e., should offer at each participating site the ability to access the DDB through the particular DDL and DML in use at that site (Spaccapietra, 1982; Demurjian, 1988; Kim, 1989). Besides user friendliness, this allows existing programs to continue operating after the DDB has been installed. Of course, the DDBMS has to be complemented with a set of DDL/DML translators.

Except in the multidatabase case, a mechanism is needed to derive a new schema (whether a global view or the DDB schema) from existing specifications. This process is called *database integration.* It is similar to *view integration,* a process in classical database design deriving an integrated schema from a set of user views. Because of similarities, a generic term, *schema integration* (Batini, 1986), has been used to indistinctly refer to both processes. However, there are differences which might preclude from using a view integration technique for database integration, and vice versa. These differences may be briefly sketched as follows.

---

1. As there is no standard taxonomy, terms like multidatabase, federated, or integrated receive different definitions depending on the author.

View integration methodologies cope with a situation in which views:

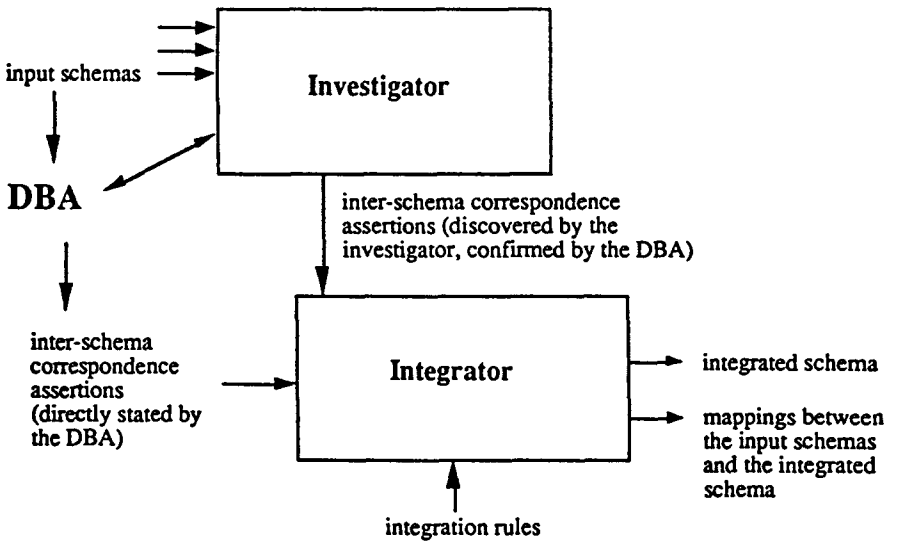
- are usually homogeneous, i.e., based on the same data model;
- have no associated recorded extension (it still is in the real world);
- have no coded program using them;
- are not necessarily implementable per se; they might therefore be expressed using any desirable conceptual model, regardless of any future transformation to make them acceptable by some DBMS.

On the contrary, in database integration, the initial (local) schemas:

- may be based on different data models;
- have an associated extension, i.e., they describe data which are actually stored in a database;
- support a number of application programs (whether directly or through some view);
- are implemented in an existing DBMS.

Current database integration methodologies do not really cope with heterogeneity. They assume that, before integration starts, all existing schemas are translated into equivalent schemas based on a unique data model. The choice of the unique data model varies, of course, from one proposal to the other. As it deals with existing databases, database integration may be limited to support mainly modeling concepts from current DBMSs. However, in order to ensure proper understanding of the semantics of the input schemas, additional information has to be provided by the DBA (Data Base Administrator) before integration is performed (Biskup, 1986; Templeton, 1987; Siegel, 1989). Finally, existence of extensions and programs is the most important peculiarity of database integration versus view integration. Because of this, and its economical impact, database integration methodologies should carefully avoid modification of existing schemas (more precisely, avoid modification of existing views on these schemas). This fundamental requirement makes that most of view integration methodologies do not apply here, as they use schema modification to solve conflicts among initial specifications.

Figure 1. The generic framework for schema integration



Generally speaking, schema integration is a two phases process (see Figure 1). First, commonalities and discrepancies among input schemas have to be determined. We call this the investigation phase. In traditional DB design methodology, this phase is manual. The DBA examines input schemas and defines the applicable set of inter-schema correspondences. Alternatively, automated reasoning may be used to discover correspondences. The basic idea is to evaluate some degree of similarity between two descriptions, mainly based on matching of names, structures and constraints (Navathe, 1982; Batini, 1984; Sheth, 1988; Bouzeghoub, 1990; Hayne, 1990; Siegel, 1991). The extent to which these CASE tools are effective depends on the amount of available knowledge about the semantics of the input schemas. The investigator's goal is to identify plausible correspondences and prompt the DBA for confirmation or denial of the findings.

Second, *integration* is performed. The integrated schema is built semi-automatically, according to the inter-schema correspondences and available integration rules. Interaction with the DBA is required to solve conflicts among input schemas each time the integrator does not have the knowledge to do it. Conflicts arise whenever corresponding concepts are modeled with different representations. The extent to which conflicts are solved automatically is a measure of the power the integration methodology.

This paper is a contribution towards more powerful methodologies for the integration phase. No hypothesis is made about how the investigation phase is performed. The methodology simply assumes, as an initial point, that the correspondences are defined. The distinguishing features of our proposal are:

- it extends the scope of automatic integration by: 1) solving new cases of conflict, such as integration of object types and attributes, 2) integrating not only elements (object types, attributes) but also links between elements. This is achieved by defining appropriate integration rules;
- as a consequence of the former, it performs integration without requiring initial schemas to be modified. This also relies on adequate mapping functionalities;
- it supports heterogeneity of input schemas through a data-model-independent description of inter-schema correspondences and generic integration rules. For instance, a relational schema may be directly compared with an object-oriented schema. This requires the DBA to master various modeling techniques, a reasonable requirement compared to the advantage of avoiding systematic homogenization of the input schemas;
- it may be easily tailored to either view or database integration.

The paper is organized as follows. The next section briefly reviews past work on database integration. Section 3 defines a taxonomy of conflicts between schemas. As we focus on structural conflicts, this section includes examples of such conflicts in current data modeling techniques. Section 4 introduces our approach and sets the formal framework for integration. Integration rules are discussed and illustrated on various data models through Sections 5 to 9. Finally, the conclusion points out ongoing or future work we plan on this topic.

2. Review of Past Work

There has been a large amount of work in the integration area: a detailed survey by Batini et al. (1986) discussed twelve methodologies for view or database integration (or both), and new contributions continuously appear in the literature (deSouza, 1986; Deen, 1987; Civelek, 1988; Fankhauser, 1988; Sheth, 1988, 1989; DeMichiel, 1989; Diet, 1989; Jardine, 1989; Larson, 1989; Siegel, 1989, 1991; Bouzeghoub, 1990; Hayne, 1990; Kaul, 1990; Kent, 1991) and many more in (Kambayashi, 1991).

An analysis of current methodologies shows a prevalent dichotomy of approach. Most of view integration papers attempt to establish a semi-automated technique for

deriving an integrated schema from a set of integration assertions relating corresponding objects in the views. We qualify these approaches as semi-automatic. They aim at building an integrator, as discussed in section 1.

On the contrary, database integration methodologies aim at providing a tool allowing the DBA to build, by himself, the integrated schema, as a view over the initial schemas. A restructuring manipulation language is defined, whose functionalities allow selection and restructuring of schema elements from existing local schemas. We qualify these approaches as manual. As the integrated schema is implemented as a view over the initial schemas, manual approaches do not apply to view integration.

2.1 Manual integration methodologies. A manual database integration methodology was first developed by Motro and Buneman (1981). The integrated schema is built as a view, called "superview," over existing schemas of local databases. Both the input schemas and the superview are described using a functional model augmented with generalization. A superview results from a DBA-driven schema editing process. This process defines a sequence of operations (a program), each one performing a modification or a restructuring transformation to be applied to initial schemas. Basically, the restructuring operators allow to build or modify an object hierarchy (introducing either a supertype or a subtype common to two existing types, merging a subtype and its supertype), as well as to modify the attribute structure (introducing new aggregations or removing existing ones).

A later paper (Motro, 1987) shows how "a mapping of the superview into the individual databases is derived from the editing process and stored with the superview as a virtual database." This mapping "is used to decompose each query into a set of queries against the individual databases, and recompose the answers to form an answer to the original query." Finally, Motro suggests that heterogeneity be dealt with by translating all existing schemas into his functional model, as a pre-integration step.

The MULTIBASE approach (Landers, 1982; Hwang, 1984) also features all usual integration steps: homogenization of local schemas, building of the integrated schema, automatic derivation of mappings, use of these mappings for automatic query modification. MULTIBASE also assumes a functional model, making extensive usage of generalizations to build the integrated schema. A QUEL-like language is defined for superview definition and as user query language. The authors advocate that this choice results in a more powerful and more versatile integration methodology.

A similar approach may be found in the PRECI* distributed database system (Deen, 1987). Its authors are concerned with integration of relational databases. An algebraic restructuring language is proposed, resulting in mappings expressed as alge-

braic transformations. In the authors' opinion, this greatly facilitates query modification.

Integration of relational databases also is the goal of the MERMAID system (Templeton, 1987). Their approach is still mainly manual, although the DBA is somehow assisted by the system. In a first step, the DBA has to extend existing schemas with the definition of all underlying semantic domains. This allows the system to determine which relations share some semantic domain: these relations are presented to the DBA. Confronted with such a set of relations, the DBA chooses which relations and which attributes have to be included into the superview.

More general and more powerful than MERMAID semantic domains, abstract data types have been proposed (Siegel, 1989) to add semantics to a relational schema. The authors also discuss how these abstract data types are used for a domain matching process which includes both static and behavioral aspects of data. More about domain matching may be found in (DeMichiel, 1989; Larson, 1989; Sheth, 1989).

Finally, the recent converging of the programming languages and databases paradigms has generated several efforts to develop superview definition languages using an object-oriented approach (Fankhauser, 1988; Kaul, 1990; Bertino, 1991; Czejdo, 1991).

2.2 Semi-automatic integration methodologies. A second stream of research has investigated the feasibility of automating database integration. In fact, these methodologies are proposed for both view and database integration. They use assertions to state correspondences between objects in different schemas. To each type of assertion corresponds an integration rule, so that the system knows what to do to build the integrated schema from the initial schemas. Interaction with the DBA is invoked only if unresolvable conflicts are detected (then the DBA instructs the system on how to solve the conflict).

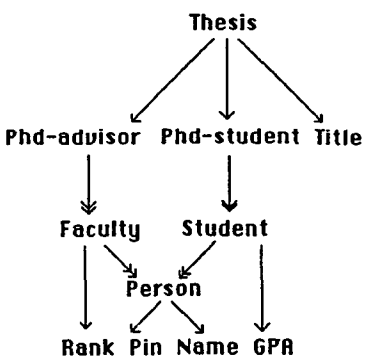
This basic framework may be found, for instance, in (Mannino, 1984). The authors introduce matching techniques for both object types and attributes. Object types integration builds various generalization hierarchies based on which set relationship (equality, inclusion, intersection, disjointedness) holds between the extensions of the related object types. Attribute integration is mainly based on semantic equivalences defined by the DBA. An algorithm to check the consistency and completeness of attribute assertions is also provided.

Concepts for matching of objects, relationships and attributes have been refined and formally defined in a series of papers by Navathe, Elmasri and Larson, who may be regarded as the major contributors to this field (with papers from Navathe, 1982 to Larson, 1989, based on the ECR entity relationship model defined in Elmasri, 1985).

Implementation of integration tools, based on their methodology, has been reported in (Hayne, 1990) and (Sheth, 1988). Larson (1989) presents a detailed analysis of possible attribute equivalences. These are the basis on which object and relationships equivalences may be stated. Attribute integration strategies are discussed, as well as their application to both object and relationship integration. Ultimately, an attempt is made to integrate an object type with a relationship type, showing a first concern to solve structural differences. Additional criteria for attribute integration have later been proposed (Sheth, 1989).

2.3 A comparison of the approaches. To illustrate the difference between the two approaches, let us consider an integration example proposed in (Motro, 1987). The schemas to be integrated are as follows (an arrow represents an attribute function, a double-headed arrow represents an is-a link):

The integrated schema is:



Motro's approach needs a 14-operations program (using 6 operators) to produce the final result. An assertion-based approach, as proposed in this paper, only needs two statements, one to assert that Phd advisors are faculty members, the second one to assert that Phd students are students.

The manual approach leaves to the user the responsibility (and the burden!) of solving structural conflicts. This, we believe, is fairly unsatisfactory. First, it means that we are only able to provide users with a toolkit (the restructuring manipulation language), up to them to use it properly. Second, although most of the conflicts may be similar in nature, users will have to "program" again and again the appropriate

restructuring (no powerful macros are available at the moment). Examples from the literature show that these "programs" may be fairly complex. Third, the proposed language may be inappropriate if a different data model is adopted (portability of the integration technique has not been investigated).

For these reasons we give our preference to the semi-automatic approach. Its current state of the art shows that only some types of conflicts have been tackled. The taxonomy in the next section identifies which conflicts remain to be supported.

3. Conflicts Between Schemas

Conflicting representations are a major challenge for integration methodologies. Two designers modeling the same universe of discourse, or two overlapping universes of discourse, will probably describe the common real-world objects in different ways. Designers might have different perceptions, different information needs or use different tools to express their perception of the universe of discourse. At the moment, there is no standard classification of the possible types of conflicts between two schemas, and the terminology is somewhat confusing. The following subsection defines the terms we use to discuss conflicts.

3.1 A Taxonomy of Conflicts. We emphasize four reasons leading to the design of different representations for common real-world objects.

1. The two designers do not perceive exactly the same set of real world objects, but instead they visualize overlapping sets (included or intersecting sets). For instance, a "Student" object class may appear in one schema, while a more restrictive "CS-Student" object class (grouping students majoring in computer science) is in another schema.

   This is the first kind of conflicts, called *semantic conflicts*. The generalization concept has been extensively used as a solution to semantic conflicts (Mannino, 1984) (except in works based on the relational model). For instance, the "CS-Student" class will be integrated as a subclass of the "Student" class.

2. When describing related sets of real-world objects, two designers do not perceive exactly the same set of properties. For instance, let us assume two relational schemas, S1 and S2, describing the same set of expensive car models:

   - S1: Expensive_car (modelname, manufacturer, maximumspeed, price)
   - S2: Car_model (name, horsepower, fuelconsumption, price)

Designers of S1 and S2 recorded different items, because of their different interest in the many available pieces of information on car models in the real world (one designer may have to keep data for advertisements in a fine arts journal, while the other is concerned with advertisements in a technical journal, for instance).

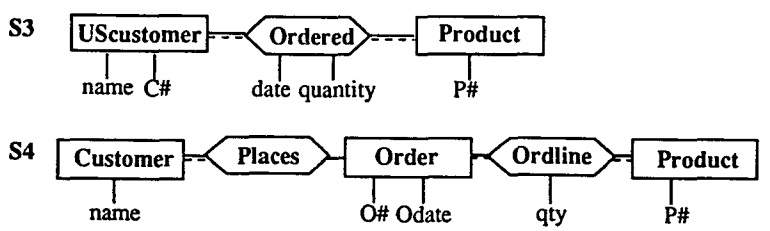
We call this second kind of conflicts, *descriptive conflicts.* Descriptive conflicts include naming conflicts due to homonyms and synonyms (Navathe, 1982; Batini, 1984), attribute domain, scale, constraints, operations, et cetera (Larson, 1989).

3. The designers use different data models, for example a relational one and an object-oriented one. This is called a *heterogeneity conflict.*

4. Lastly, even if they use the same data model, they can choose different constructs to represent common real-world objects. For instance, in object-oriented models when a designer describes a component of an object type O, (s)he has to choose between creating a new object type or adding an attribute to O. We call this kind of conflict a *structural conflict.*

   The extent to which structural conflicts may arise is related to the semantic relativism of the data model in use, i.e., to its ability to support different, although equivalent, representations of the same reality. Semantic and object-oriented models have more semantic relativism than the relational model. Therefore, the ability to solve structural conflicts is likely to be of ever-increasing importance for integration methodologies.

These conflicts are orthogonal and can be cumulative. Discrepancies between schemas usually show a mix of conflict types. Different data models, or different sets of real-world objects generate different structures. As an example, consider the following entity relationship diagrams[2] which represent related universes of discourse:



2. A single plain line denotes a 1:1 cardinality of the entity type in the relationship type. Combination of a plain and a dotted line denotes a 1:N cardinality.

Both represent some information about customers who order products from an enterprise. S4 includes all customers, while S3 only considers a subset of the customers (semantic conflict). Customer's attributes differ between S3 and S4 (descriptive conflict). The information about an order is given in S3 as a direct relationship between a customer and a product (s)he ordered. S4 favored a more detailed representation based on order's materialization as an Order entity type (structural conflict).

The first two kinds of conflicts have been dealt with in earlier research. This paper concentrates on the last two kinds of conflicts. For heterogeneity, we simply allow heterogeneous schemas to be directly related to each other, as in the homogeneous case. A data-model-independent formalism is used to express inter-schema correspondences. This formalism and integration rules are defined through the use of a generic data model. Additional rules define how integration is tailored to the specific data models underlying input schemas. Such a customization is described in this paper.

Structural conflicts, although well known in literature (Elmasri, 1979), have received little attention. No automated strategy for their resolution has been proposed. Existing methodologies rely on the DBA (purposely in Navathe, 1986) for conforming of schemas, a process in which views are modified by forcing related concepts to be represented by the same structural construct. Input modification is not suitable for database integration, where existing schemas continue to be in use after integration.

3.2 Examples of Structural Conflicts. Our main contribution, in this paper, is devoted to the automatic resolution of structural conflicts. This section gives some examples which show that such structural conflicts may arise irrespective of the data model which is used for data description. To emphasize structural conflicts, the examples show only structural conflicts. They all represent the same universe of discourse: persons, cars and ownerships. The first ones, S5 and S6, illustrate conflicting relational databases.

- S5: Car (*registration#,* color, horsepower, owner ID)

  Person (*pin,* name, sex, birthdate)

  Inclusion dependency: Person.pin $\supseteq$ Car.owner ID

- S6: Car (*registration#,* horsepower, color)

  Carownership (*registration#,* pin)

  Person (*pin,* name, sex, birthdate)

Inclusion dependencies:

Person.pin $\supseteq$ Carownership.pin

Car.registration# $\supseteq$ Carownership.registration#

S5 uses one relation to hold information on cars and their ownerships, while S6 has split this information over two relations constrained by an inclusion dependency.

Let us consider an object-oriented model. Car ownership may now be represented in six different ways, S7 to S12, which correspond to six different ways of defining a relationship in an object-oriented model.

S7: **Class** Car **tuple** <
registration#: · · ·
color: · · ·
horsepower: · · ·
owner: **tuple** < pin: · · ·, name: · · ·, sex: · · · , birthdate: · · · >
>

S8: **Class** Person **tuple** <
pin: · · ·
name: · · ·
sex: · · ·
birthdate: · · ·
cars: **setof tuple** < registration#: · · ·, color: · · ·, horsepower: · · · >
>

S9: **Class** Person **tuple** <
pin: · · ·
name: · · ·
sex: · · ·
birthdate: · · ·
>

Class Car **tuple** <
registration#: · · ·
color: · · ·
horsepower: · · ·
>

Class Carownership **tuple** <
car: Car
owner: Person
>

S10: Class Person tuple <
pin: · · ·
name: · · ·
sex: · · ·
birthdate: · · ·
cars: **setof** Car
>

Class Car tuple <
registration#: · · ·
color: · · ·
horsepower: · · ·
>

S11: **Class** Person **tuple** <
pin: · · ·
name: · · ·
sex: · · ·
birthdate: · · ·
>

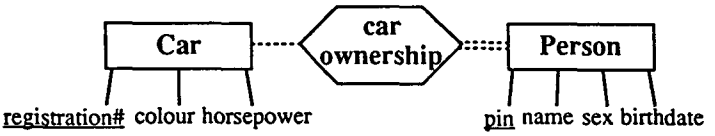
Class Car **tuple** <
registration#: · · ·
color: · · ·
horsepower: · · ·
owner: Person
>

S12: **Class** Person **tuple** <
pin: · · ·
name: · · ·
sex: · · ·
birthdate: · · ·
cars: **setof** Car
>

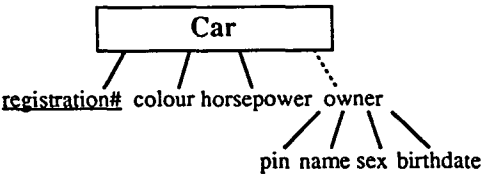
Class Car **tuple** <
registration#: · · ·
color: · · ·
horsepower: · · ·
owner: Person
>

As in the relational case, we are confronted with different structures expressing equivalent schemas. These six possibilities are due to the object-oriented flexibility in implementing links through references. They reduce to three if we consider a semantic modeling approach, as, for instance, an extended entity relationship (ER) model. The following diagrams illustrate the three equivalent viewpoints. An extended ER model, supporting complex objects, is assumed (Parent, 1985, 1992). Lines are drawn according to minimum and maximum cardinalities of the link: a single dotted line stands for 0:1, a single plain line stands for 1:1, a double dotted line stands for 0:n, and a double line, one plain, one dotted, stands for 1:n.

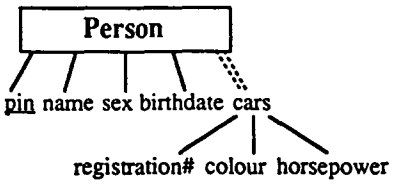
S13:



S14:



S15:



In these car ownership examples, the conflicts are due to different representations of the real-world associations (the car ownership links) between persons and cars. Indeed, an association may be represented:

i as a nesting of the linked objects (S7, S8, S14, S15). This is not possible in the relational approach, due to normalization rules (but is possible in non-first normal form approaches (Schek, 1986);

ii as a reference from one of the linked objects to the other one (S5, S10, S11, S12) for data models supporting this kind of reference concept;

iii as an additional object-bearing references on the linked objects (S6, S9, S13).

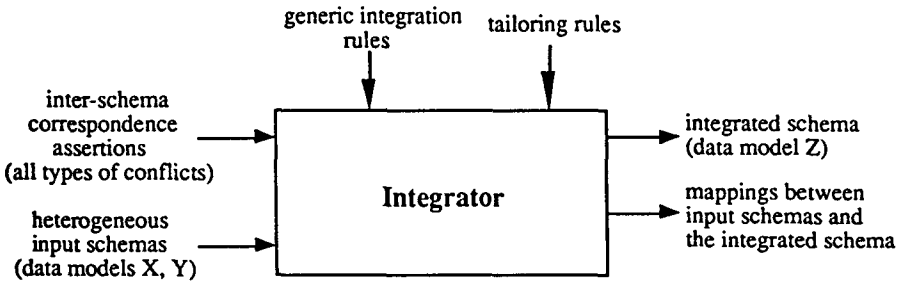
The relational model supports representations ii) and iii); ER models support i) and iii); object-oriented models support i), ii) and iii).

4. A Generic Description of Inter-Schema Correspondences

Basically, our approach to schema integration relies on the idea of moving knowledge from outside the integrator into the integrator. Current integrators do not know how to map schemas to each other if structural conflicts are involved, hence, they call onto the DBA for conforming of input schemas. Putting knowledge about schema transformations into the integrator allows it to take care of schema conforming and to solve structural conflicts. Similarly, current integrators have no knowledge of different data models. Hence, the need to transform all input schemas into the integrator's data model before integration can start. Putting knowledge about data models into the integrator allows it to manage correspondences between constructs from different data models.

In the approach that we propose, the core of the integration methodology (description of commonalities between schemas, integration rules) is defined in terms of a few generic concepts, abstracting from any specific data model. In addition to this abstract framework, our integrator knows how to tailor the framework to cope with the

Figure 2. Framework for proposed integration methodology



constraints inherent to specific data models. In other words, it knows how to deal with a statement about commonalities depending on the particular data model of the input schemas. It also knows how to explicit the resulting integrated schema in different data models. Moreover, integration rules have been defined to deal with correspondences between concepts with different structural behavior. This supports resolution of structural conflicts. The framework specific to our methodology is illustrated in Figure 2.

Our approach starts with the hypothesis that commonalities between input schemas have been identified and checked with the DBA and the users. These commonalities are defined using inter-schema correspondence assertions. An *inter-schema correspondence assertion* is a declarative statement asserting that something in one schema is somehow related to something in another schema. Assertions precisely identify, if applicable, which semantic, descriptive, and structural conflicts exist within the correspondence. Hereinafter the term assertion is used to denote an inter-schema correspondence assertion.

The integrator receives as input two (or more) schemas and the assertions in between[3]. The set of assertions is scanned and ordered for processing. Each assertion is then considered and the appropriate integration rule is applied, taking into account the data models of the input schemas. Integration rules define which constructs have to be built into the integrated schema and how these constructs are mapped to the corresponding constructs in the input schemas.

---

3. In this paper we do not discuss the problem of checking the consistency of the set of assertions relating two schemas.

Generic data modeling concepts we use are defined in Section 4.1. Section 4.2. defines their real-world counterpart, a concept needed to define the semantics of assertions. Assertions are then discussed in section 4.3.

4.1 The Generic Data Model (GDM). GDM is a set of modeling concepts which allows us to reason about integration of conflicting schemas. It is a tool to define the generic assertions and integration rules. The GDM discussed in this paper is a basic one, showing the major concepts: objects, value attributes, and reference attributes (defined below). Completeness of the approach would require additional generic concepts (generalization, for instance), but this is beyond the scope of this paper. GDM is able to model complex objects, i.e., objects with a complex data structure, possibly including other objects as their components. This simplifies the expression of assertions. It also bears the potential for structural conflicts we are interested in, as what is one component in one schema may be considered as a self-standing object in another schema. GDM offers three modeling concepts: objects, value attributes, and reference attributes.[4]

A GDM *object* is an object identity complemented with a data structure consisting of a tuple of attributes. For each attribute its minimum and maximum cardinalities define the number of values it may bear (at least, at most): zero, one or more. An attribute is either atomic or complex. A complex attribute is a tuple of attributes. Atomic attributes are either *value attributes* (the associated domain is a value domain, like integer, characters or date) or *reference attributes* (the associated domain is an object type). Reference attributes are regarded as bidirectional: adding to an object type O1 a reference attribute pointing at object type O2 is equivalent to adding to O2 a reference attribute pointing at O1. In the sequel, we use the term element to indistinctly refer to objects and attributes.

GDM is structurally object oriented, but it is not identical to the object-oriented model used in examples S7 to S12. The main difference is that GDM abstracts from access paths: GDM reference attributes are non-directed binary relationships between two objects. On the contrary, in most object-oriented models, as in the one of the examples, links between a compound object (for instance *Car* in S11) and its component objects (*Person* in S11) are one way links from the compound object to its components. Another, minor, difference between GDM and object-oriented models is that, in GDM, cardinalities of attributes are precisely defined, in order to be used during integration of links.

---

4. This is similar to the notions of object, own attribute, and ref attribute (Carey, 1988).

Besides modeling concepts, we also identify an additional concept: the link. We use the term *link* to denote any direct connection between two elements: attribute link between an element and one of its value or complex attributes, reference link between an element and an object through a reference attribute. We do not discuss generalization links in this paper. The two kinds of links that we consider are defined below.

Definition D1: Link

let X and Y be object types, value or complex attributes, then X—Y (also noted Y—X) is a link if:

- either Y is a value or complex attribute of X; then X—Y is called an *attribute link*,
- or X holds a reference attribute, named r, pointed at object type Y; then $X \overset{r}{—} Y$ is called a reference link. If there is no ambiguity (only one reference exists between X and Y) $X \overset{r}{—} Y$ may be simply denoted X—Y. □

Cardinalities of links are used in the integration process. Minimum/maximum cardinalities of X in the X—Y link are the minimum/maximum number of $y \in Y$ which may be reached from a $x \in X$ through the X—Y link. Conversely for cardinalities of Y in the X—Y link. X—Y cardinalities are hereinafter denoted as: min(X):max(X), min(Y):max(Y). These cardinalities are as follows:

min(X):max(X) =
minimum: maximum cardinality of the Y attribute, if X—Y
of the r attribute, if $X \overset{r}{—} Y$

min(Y):max(Y) = 1:n, if X—Y (n = 1 if Y is an identifier of X)
0:n, if $X \overset{r}{—} Y$

Examples:

S5: Car $\overset{\text{ownerID}}{———}$ Person is a reference link, with cardinalities 0:1,0:n.

S6: Car—color and sex—Person are attribute links.
Car—color has cardinalities 0:1,1:n. Sex—Person has cardinalities 1:n,0:1.
Carownership $\overset{\text{registration\#}}{———}$ Car, Carownership $\overset{\text{pin}}{—}$ Person are reference links.

S9: Carownership $\overset{\text{car}}{—}$ Car, Carownership $\overset{\text{owner}}{———}$ Person are reference links.

S13: Car—Carownership and Carownership—Person are reference links.

Two elements in a schema may be bound directly by a link, or indirectly by a composition of links, called path. For example, Car and Person in S6 are bound through the path: Car—Carownership—Person.

Definition D2: Path

let $X_1, X_2,..., X_n$ be elements (object types, complex or value attributes) in a schema such that $\forall$ i$\in$1,2,...,n-1, X_i is linked to X_{i+1}, either by an attribute link or by a reference link, then X_1—X_2...X_n is a path. □

Cardinalities of the X_1—X_2—...—X_n path equal the product (Π) of the corresponding cardinalities in the component links:

minimum cardinality of X_1 = $\Pi_{i\in[1:n-1]}$ minimum cardinality of X_i in X_i—X_{i+1}
maximum cardinality of X_1 = $\Pi_{i\in[1:n-1]}$ maximum cardinality of X_i in X_i—X_{i+1}
minimum cardinality of X_n = $\Pi_{i\in[2:n]}$ minimum cardinality of X_i in X_{i-1}—X_i
maximum cardinality of X_n = $\Pi_{i\in[2:n]}$ maximum cardinality of X_i in X_{i-1}—X_i

Example from S6:
sex—Person—Carownership—Car—color is a path associating the sex of a person to the color of the car the person owns. Cardinalities of this path are 0:n, 0:n.

4.2. Real world states. As previously stated, the semantics of correspondence assertions is defined by referring to the real-world counterpart of the involved elements. Larson (1989) introduced the "real-world state" of an object class O, RWS(O), as the set of real-world instances of object class O at a given moment in time. We extend this RWS concept to attributes, links and paths, in order to deal with every concept of GDM (and of any data model). This will allow us to establish the meaning of correspondence assertions relating elements of different types (object types and attributes), or paths and links, in the next section.

The RWS of a complex or value attribute A may be defined, like the RWS of an object type, as the set of real-world objects that the set of the present values of A represents. In the case of a multi-valued attribute, the set of its present values is made up of single values, not of sets of values. For instance, if color was a multi-valued attribute of an entity type Car which contained two cars, a green one and a black-yellow one, its RWS would be:

RWS(color) = {green, black, yellow}

One may wonder about the real objects described by a color or horsepower attribute. One could think of a lexical RWS, where the perception is related to attribute values, and a non-lexical RWS, where the perception abstracts from the present values to refer to objects implicitly referenced through those values. For instance, refer

to the simple value attribute registration# of the class Person in S8. Its RWS can be viewed either as the set of string characters which represent the plate numbers, or as the set of real cars which are owned by a person. However, when an attribute is stated as corresponding to an object type, this always refers to the non-lexical RWS of the attribute.

Definition D3: Real world state of an element (object type, complex or value attribute)

The RWS of an object type O (respectively a complex or value attribute A) is the set of real-world objects that the set of the present occurrences of O (respectively values of A) represents. □

There is a one-to-one mapping between the RWS and the set of the present occurrences (or values) of the object type (or attribute). An attribute may have the same value in several different objects of the database. Nevertheless, when one looks at the set of the present values of an attribute, abstracting from duplicates, each value describes exactly one real object of its RWS.

In the previous definition of RWS we did not deal with reference attributes. Reference attributes do not bear values, and thus they cannot be perceived as corresponding to objects, but to other links or paths. For this reason we are not interested in their RWS as elements, but in the RWS of the link they express. The latter is defined below.

A link X—Y, or generally a path X—...—Y, is a connection between two object types, X and Y. Its RWS is made up of pairs of real objects, one described by X and one by Y, such that these two real objects are in the real world bound by an association which the link (or path) represents.

Definition D4: Real world state of a path.

The real-world state of path X_1—X_2-...-X_n, RWS(X_1-X_2-...-X_n), is the bag of real-world object pairs $\langle o_1, o_n \rangle$, such that $o_1 \in$RWS(X_1) and $o_n \in$RWS(X_n), There exist objects $o_2, o_3, ..., o_{n-1}$ such that $\forall i \in 1,2,...,n\text{-}1$, $o_i \in$RWS(X_i), with o_i and o_{i+1} linked by the real-world association represented by the X_i—X_{i+1} link. □

Example from S6:
RWS(sex—Person—Carownership—Car—color) is a bag of pairs of type ⟨sex, color⟩, associating, for each person in RWS(Person), his/her sex to the color of one of his/her cars.

4.3. Correspondence assertions. There are two types of correspondence assertions: those relating two elements, and those relating two paths or links. Assertions between elements identify the semantic, descriptive and structural conflicts, if any, between the two related elements:

i) if one of the elements is an object type and the other one is an attribute, this denotes a structural conflict;

ii) depending on the set relationship which relates the RWS of the elements, there is no semantic conflict in between ($\equiv$), or there is one ($\supseteq, \cap, \neq$);

iii) an additional clause in the assertion specifies if and how the attributes of the two elements are related to each other. A descriptive conflict appears if there is either at least one attribute in one element with no corresponding attribute in the other element, or at least one pair of related, but not equal, attributes.

4.3.1 Element correspondence assertions. We first illustrate the four possible set relationships between the RWS of corresponding elements. The formal definition is given next to the examples. Let us consider an enterprise with several local databases operating in different departments that have to be integrated. Various local databases may include the product catalog, describing all products sold by the enterprise. The catalog is the same for all departments: an equivalence assertion will relate product catalogs together. Suppose each local database maintains a file of department employees, using the same format, and suppose an employee works in only one department. Employee object types will be asserted as corresponding but disjointed. Suppose now that each department maintains a customer file. Different departments may share some customers: Customer object types will be related by an intersection correspondence. Finally, suppose each department has its suppliers, but suppliers have to be chosen from a global file maintained at the head office. Local Supplier object types will intersect each other, but they will be asserted as inclusion correspondences with respect to Supplier in the head office database.

Definition D5: Element correspondence assertions

Let X_1, X_2, be two elements (object types, complex or value attributes), X_1 from schema S_1, X_2 from schema S_2. A correspondence between X_1 and X_2 may be asserted as one of the following:

- X_1 and X_2 are equivalent, expressed as: $X_1 \equiv X_2$

  which states that at any time $RWS(X_1) = RWS(X_2)$;

- X_1 contains X_2, expressed as: $X_1 \supseteq X_2$

which states that at any time $RWS(X_1) \supseteq RWS(X_2)$;

- X_1 and X_2 intersect, expressed as: $X_1 \cap X_2$

  which states that at some time $RWS(X_1) \cap RWS(X_2) \neq \emptyset$;

- X_1 and X_2 are disjoint, expressed as: $X_1 \neq X_2$

  which states that at any time $RWS(X_1) \cap RWS(X_2) = \emptyset$. □

This last assertion means that, although disjoint, their semantics is related, and the DBA wants to merge them into a more generic element in the integrated schema.

4.3.2 Corresponding attributes assertions. Whenever two elements are asserted as corresponding, complementary assertions about attribute correspondences are needed to direct the integrator towards the production of the integrated structure, i.e., what are its attributes. In our model, these assertions about corresponding attributes of corresponding elements X and Y are stated for reference attributes as path assertions, and for value and complex attributes as part of the correspondence assertion between X and Y, using a "with corresponding attributes" clause. This clause defines the descriptive conflict, if any.

Similar to element correspondences, the set relationship between the sets of values of two attributes is one of the following:

i $=$ the attributes have the same values;

ii $\supseteq$ the value(s) of one attribute include the value(s) of the other attribute. If both attributes are monovalued, either they both have the same value, or the included attribute has a null value;

iii $\cap$ the two attributes are multi-valued and their sets of values intersect;

iv $\neq$ the values of the attributes are always different, but they are related. The DBA wants to merge them into a broader one, union of the two attributes.

The "with corresponding attributes" clause defines for each attribute correspondence, which is the set relationship, and, if any, the function mapping one domain into the other. Different types of functions may be involved:

- a 1:1 mapping defining a translation of the domains. A matching table may be used for this purpose. For example, Swiss francs may be converted into US dollars.

- an aggregate function defining the value of a mono-valued attribute as the result of the aggregation of the set of values of a multi-valued attribute. For example, an attribute children—number is equal to the count of values of an attribute children of another database.

- a tuple function defining the value of an attribute as the result of the Cartesian product of several attributes. For example, an attribute address is equal to the Cartesian product of attributes number, street and city of another database.

More facets to be considered have been suggested (Larson, 1989): integrity and security constraints, allowable operations. We will not discuss these additional facets, as they do not change the nature of the problem. The reader interested in attribute matching may also refer to DeMichiel (1989) or Sheth (1989), who specifically deal with this topic.

Because our aim is not to analyze those many facets, Definition D6 deals only with the set relationships between the value sets of two attributes. Each A_{1i} may be replaced by $f(A_{1i})$ (translation of domains or aggregation function), by $f(A_{1i1},A_{1i2},...,A_{1ip})$ (tuple function) or by any composition of these functions, and similarly for A_{2i}.

Definition D6: Corresponding value attributes assertions.

Let $X_1 \langle \text{ cor } \rangle$[5] X_2 be an element correspondence assertion. Let $A_{11}, A_{12},...,A_{1n}$ be value attributes of X_1, and $A_{21}, A_{22},...,A_{2n}$ be value attributes of X_2 (if X_1 or X_2 is an atomic attribute, it is implicitly considered here as having itself as unique component).

Let us call o any element common to both X_1 and X_2 real-world states, $o \in RWS(X_1) \cap RWS(X_2)$[6], and e_1, e_2 be the occurrences representing o in the databases described by S_1, S_2. Then:

$X_1 \langle \text{ cor } \rangle X_2$ with corresponding attributes:

$$attcor_1(A_{11}, A_{21}), attcor_2(A_{12}, A_{22}),...,attcor_i(A_{1n}, A_{2n})$$

is also a correspondence assertion which states that:

$X_1 \langle \text{ cor } \rangle X_2$ is true, and for each $attcor_i(A_{1i}, A_{2i})$:

- if $attcor_i(A_{1i}, A_{2i})$ is $A_{1i} = A_{2i}$

  then at any time, for any $o \in RWS(X_1) \cap RWS(X_2)$: $e_1.A_{1i} = e_2.A_{2i}$;

---

5. $\langle \text{cor} \rangle ::= \equiv | \supseteq | \cap | \neq$

6. If X_1 and X_2 are disjointed, we consider o to be a hypothetical element contradicting the disjointedness (i.e., if such a o would exist, $o \in RWS(X_1) \cap RWS(X_2)$, then ...)

- if $attcor_i(A_{1i}, A_{2i})$ is $A_{1i} \supseteq A_{2i}$

  then at any time, for any $o \in RWS(X_1) \cap RWS(X_2)$: $e_1.A_{1i} \supseteq e_2.A_{2i}$;

- if $attcor_i(A_{1i}, A_{2i})$ is $A_{1i} \cap A_{2i}$

  then it is possible that for some $o \in RWS(X_1) \cap RWS(X_2)$: $e_1.A_{1i} \cap e_2.A_{2i} \neq \emptyset$

- if $attcor_i(A_{1i}, A_{2i})$ is $A_{1i} \neq A_{2i}$

  then at any time, for any $o \in RWS(X_1) \cap RWS(X_2)$: $e_1.A_{1i} \cap e_2.A_{2i} = \emptyset$,

  but the two attributes are semantically related and the DBA wants to merge them into a broader one, a union of the two. □

Attribute correspondences should not contradict correspondences asserted for their parent elements (Larson, 1989). Each element correspondence assertion involving a mapping of occurrences of different databases, i.e., element equivalence, inclusion or intersection assertion, must contain a 1 to 1 attribute equality assertion relating identifiers:

$$A_{1i} = A_{2i} \text{ or } A_{2i} = \text{bijective function } (A_{1i}).$$

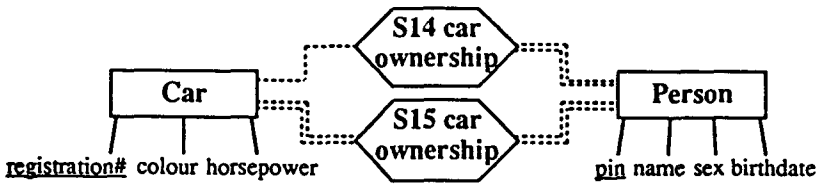
4.3.3 Path correspondence assertions. The analysis of the inter-schema relationships also calls for the identification of the correspondences among paths. Refer to S14 and S15. If we suppose that the two schemas see exactly the same objects (cars and persons), the element correspondence assertions between S14 and S15 are:

Car≡cars with corresponding attributes:
registration# = registration#, color = color,
horsepower = horsepower

owner≡Person with corresponding attributes:
pin = pin, name = name, sex = sex, birthdate = birthdate

These two assertions will generate two entity types, Car and Person, in the integrated schema. Nothing in the above assertions states that the real-world associations between cars and persons, described by S14 and S15, are one and the same ownership association. Consequently, the integrator will generate in the integrated schema two

relationship types between Car and Person, one to express the Car-owner S14 link and the other to express the Person-cars S15 link:



In order to allow the integrator to integrate those two links into a unique relationship type (producing S13 as integrated schema), the DBA has to state that both links have the same semantics. In our methodology, the DBA will define the following path correspondence assertion (which is explained below):

Car—owner ≡ cars—Person.

Two paths or links may be asserted as corresponding only if they relate corresponding elements. That is why the definition of path assertions refers not to the whole RWS of the corresponding elements, but to the subset of this RWS which involves only the objects that have a corresponding object in the other database. In definition D7, we call this subset RWS′.

Definition D7: Path equivalence assertion.

Let X_1—X_2—...—X_n be a path in schema S, and Y_1—Y_2—...—Y_p be a path in schema S′, such that there is a correspondence assertion relating X_1 to Y_1 and an assertion relating X_n to Y_p.

Let RWS′(X_1) be the subset of RWS(X_1) defined by its restriction to X_1 objects which are involved in the asserted correspondence with Y_1 objects. Let RWS′(Y_1), RWS′(X_n) and RWS′(Y_p) be similar restrictions of the corresponding RWS.

Let RWS′(X_1—X_2—...—X_n) be the subbag of RWS(X_1—X_2—...—X_n) defined by its restriction to object pairs in RWS′(X_1) × RWS′(X_n), and similarly for RWS′(Y_1—Y_2—...—Y_p).

The assertion that the two paths are equivalent, expressed by the statement:

$$X_1—X_2—...—X_n \equiv Y_1—Y_2—...—Y_p$$

states that at any time:

$RWS'(X_1—X_2—...—X_n) = RWS'(Y_1 Y_2—...—Y_p)$. □

The other assertions, path inclusion, intersection and exclusion, may be defined in the same way as for element assertions.

Example relating S3 and S4:

USCustomer $\subseteq$ Customer with corresponding attributes: name=name
Ordered $\equiv$ Ordline with corresponding attributes: quantity=qty
Product $\equiv$ Product with corresponding attributes: P#=P#
Ordered · date $\equiv$ Order · Odate
Ordered—date $\equiv$ Ordline—Order—Odate
Customer—Ordered $\equiv$ Customer—Places—Order—Ordline
Product—Ordered $\equiv$ Product—Ordline

The last three assertions are path assertions which respectively state that:

- the date of Ordered in S3 is the same as the date of the Order linked to the corresponding Ordline of S4;
- the Ordered occurrences which link a customer in S3 are equivalent to the Ordline occurrences which link the corresponding customer of S4 through Order and Places;
- the Ordered occurrences which link a product in S3 are equivalent to the Ordline occurrences which link the corresponding product of S4.

4.3.4 Conclusion. Until now we have defined two kinds of correspondence assertions: between elements with their corresponding attributes, and between paths. They cover all the concepts of GDM, and can describe most of the current inter-schema correspondences involving one occurrence or value of each database. Other correspondence assertions have to be defined when a set of occurrences (or values) is corresponding to an occurrence (or value), as in the famous convoy of ships example. Even in the one to one mapping, full integration of complex attributes at any depth will require more precise correspondence assertions.

As we have seen, correspondence assertions can be used with any kind of data model (object-oriented, relational, ER). Moreover, they can state correspondences between two heterogeneous schemas, as in the following example.

Example relating the relational schema S5 and the object-oriented schema S9.

Car ≡ Car with corresponding attributes:
registration# = registration#, color = color, horsepower=horsepower
Person ≡ Person with corresponding attributes:
pin = pin, name = name, sex = sex, birthdate = birthdate
Car—Person ≡ Car—Carownership—Person

5. Schema Integration

This section and the following ones discuss the integration rules which govern the definition of the integrated schema from the initial schemas and the correspondence assertions among them. Not to overload the paper and the reader, we only consider hereinafter the equivalence correspondence assertions[7]. We also restrict the discussion to atomic attributes and complex attributes with only atomic component attributes. Finally, our rules assume a binary integration strategy (integration of two schemas at a time). However, their extension to a n-ary integration strategy (integration of several schemas in one step) is rather straightforward.

Each integration rule is first stated according to our generic model GDM. Customization of the rule is then shown on relational, entity relationship and object-oriented models. When applying a generic rule to a particular model, constraints which are specific to this model are taken into account. Those constraints are:

- existence dependencies,
- for most of object-oriented models, the fact that reference attributes are directed: a link A→B and a link A←B do not provide the same facilities to the users,
- for ER models, the fact that reference attributes are mandatory and monovalued: they cannot have a null value, they must always point at some unique, existing object.

For each initial schema element and link, the database integration process has to:

- define what elements have to be inserted into the resulting schema,

7. Those interested in how inclusion, intersection, and exclusion assertions behave with respect to equivalence assertions may refer to Larson (1989), Mannino (1984), and a different viewpoint in Jardine (1989).

- define the distribution information attached to these elements, showing on which local database which subset of the corresponding population may be found,

- define the mappings between the initial schemas and the integrated schema.[8] These mappings support the translation of global queries on the integrated schema, into local queries on the local schemas. In our approach, the mapping definitions are based on the ERC+ algebra (Parent, 1985). This algebra extends the relational algebra to deal with entity types, relationship types and complex attribute structures.

Different algorithms may be designed to perform the integration process. The choice depends on whether integration adopts a binary strategy or a more efficient n-ary strategy. The algorithm also depends on the degree of interaction with the DBA. On one extreme, the DBA states all assertions and the integration process integrates the input schemas in one shot, showing the final result. On the other extreme, integration may proceed one assertion at a time, for instance with the DBA pointing on the two corresponding elements on a screen and describing the correspondence, and the integrator immediately displaying the result of processing that particular assertion. For this reason, this paper does not suggest any particular algorithm[9].

This section first explains the principles governing our integration rules. Next, it introduces how two value attributes are merged, which is needed for the definition of an extended join operator, which we call integrate-join, used in the integration rules to build the structure resulting from the merging of two corresponding structures. Finally, the very first integration rule is stated, governing integration of elements which are local to one of the input schemas.

5.1 Integration principles. The rule definitions are based on two basic principles, which are model independent:

1. The scope of integration rules has to include both elements and links integration,
2. Whenever there is a structural conflict between two schemas, the integrated schema will hold the more unconstrained structure: the one which has less existence dependencies.

---

8. When n schemas are integrated using a binary strategy, the intermediate integrated schemas are used as input to the next integration step. In this case it is preferable to generate correspondence assertions (instead of mappings) to meet initial conditions for the next step (i.e., two schemas and the correspondence assertions in between).

9. An algorithm for integration of ER views is proposed in Spaccapietra (1992).

The rationale for the first principle has been given in section 4.3.3. The rationale for the second principle is that the integrated schema has to support queries and updates on all underlying databases. If different constraints hold on related data in different databases, the integrated constraint has to be the weakest one, so that no access request is unduly rejected at the integrated level. For instance, if the age of persons is limited to the 20–50 range in database DB1, and database DB2 has more persons than DB1, with age ranging from 20 to 65, the integrated schema will hold a 20–65 age range constraint. The 20–50 restriction will be enforced on DB1 through the mapping between the integrated schema and DB1. The same reasoning holds for data structures. The identification of the most unconstrained structure depends on the data model in use, as follows.

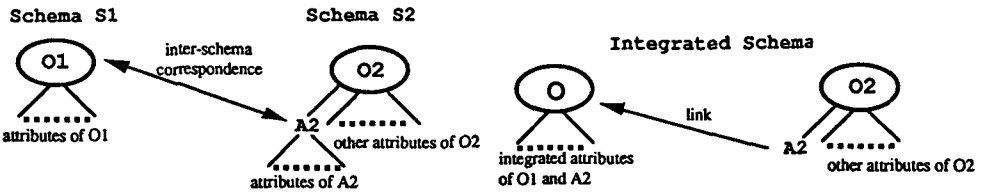
GDM and object-oriented models: GDM element types are object types and attributes. They differ in two aspects: object types have an identity, attributes don't. Object types have no existence dependency: they can be linked to other object types and to attributes, but this is not mandatory. On the other hand, attributes have to be linked to one and only one other element, their parent element (object type or complex attribute). These dependencies can be summarized in the following table.

Existence dependencies in GDM and object-oriented models:

	object type		attribute
object type	0:n		0:n
attribute	1:1	or	1:1

where n:p in line i and column j means that any element of type i must be linked to at least n and at most p elements of type j. The two 1:1 figures in the attribute line are linked by an exclusive or (an attribute is either attached to an object type or to another attribute).

The most unconstrained structure is the object type. When an object type O1 of schema S1 is corresponding to a value or complex attribute A2 of schema S2, an object type O will be put into the integrated schema and a link will be added to bind O to the integrated parent element of A2, as is shown in the following figure.



Relational model.

Existence dependencies

	relation	attribute
relation	0:n	1:n
attribute	1:1	0:0

Moreover, both attributes and relations have no identity. The most unconstrained structure is the relation.

Entity relationship models: Here, we refer to entity relationship models, as ERC+, where entity types and relationship types have an identity while attributes don't.

Existence dependencies

	entity type		relationship type		attribute
entity type	0:0		0:n		0:n
relationship type	2:n		0:0		0:n
attribute	1:1	or	1:1	or	1:1

When an entity type of schema S1 corresponds to an attribute or to a relationship type of schema S2, the more unconstrained structure is the entity type. In the integrated schema, an entity type is generated together with the relationships which express the attribute link or role links of S2.

When a relationship type and an attribute correspond, their existence dependencies are not compatible: a relationship type has to be linked to at least two entity types, an attribute to at most one entity type. They will be integrated into an entity type plus relationship types expressing the attribute and role links.

5.2 Basic definitions for merging attribute structures. We first define how two atomic value attributes are merged to produce an integrated attribute.

Definition D8: Integration of two corresponding value attributes of two equivalent elements.

Let E1, E2 be two corresponding elements (object types or complex attributes) of schemas S1 and S2. Let A1 and A2 be atomic value attributes of E1 and E2 respectively. If it is asserted that A1 and A2 correspond to each other:

E1 $\equiv$ E2 with corresponding attributes: attcor(A1, A2)

then integration of A1 and A2 is defined as a simple attribute A, such that:

- its name is A1, except if the DBA chooses another one[10],
- its domain is defined as follows:

  if attcor(A1, A2) is A1=A2 or A1$\supseteq$A2 then domain(A1)

  if attcor(A1, A2) is A1$\cap$A2 or A1$\neq$A2 then domain(A1)$\cup$domain(A2)

- its cardinalities are defined as follows:

  if attcor(A1, A2) is A1=A2 or A1$\supseteq$A2 or A2 = (A1)

  then cardmin(A)=cardmin(A1), cardmax(A)=cardmax(A1)

  if attcor(A1, A2) is A1$\cap$A2

  then cardmin(A)=Max(cardmin(A1),cardmin(A2))

  cardmax(A)=cardmax(A1)+cardmax(A2)

  if attcor(A1, A2) is A1=A2

  then cardmin(A)=cardmin(A1)+cardmin(A2)

  cardmax(A)=cardmax(A1)+cardmax(A2) □

We can now define the integrate-join operator, merging two composite elements, object types, or complex attributes.

Definition D9: Integrate-join.

Let E1, with value attributes (A_{11},...,A_{1j}, B_1,...B_k), and E2, with value attributes (A_{21},..., A_{2j}, C_1, ...C_h), be two elements (object types or complex attributes) of databases S1 and S2, asserted to be equivalent to each other:

E1 $\equiv$ E2 with corresponding attributes:

$attcor_1(A_{11}, A_{21})$, $attcor_2(A_{12}, A_{22})$,..., $attcor_j(A_{1j}, A_{2j})$.

Let $attcor_1(A_{11}, A_{21})$ be the assertion which specifies the 1:1 mapping between the identifiers of E1 and E2. Then the operation:

E:= integratejoin (E1, E2, $attcor_1(A_{11}, A_{21})$, $attcor_2(A_{12}, A_{22})$,..., $attcor_j(A_{1j}, A_{2j})$)

10. For DBA users of the future global database, a relevant criterion may be understandability. Which linguistic and ergonomic considerations would be useful is beyond the scope of this paper.

creates a new object type E defined as follows:

- its structure consists of the union of E1 and E2 attributes, defined as:
  1. an attribute B'_i for each attribute B_i of E1 which has no counterpart in E2; its domain and cardinalities are equal to those of B_i
  2. an attribute C'_i for each attribute C_i of E2 which has no counterpart in E1; its domain and cardinalities are equal to those of C_i
  3. an attribute A_i for each $attcor_i(A_{1i}, A_{2i})$; A_i is the integration of A_{1i} and A_{2i}
- its population contains one occurrence, e, for each real-world object of the RWS of E1 and E2. The value of e is defined as the merging of the values of the E1 and E2 occurrences describing this real-world object and which are linked by the 1:1 mapping $attcor_1(A_{11}, A_{21})$:

  for each attribute B'_i: $e.B'_i = e1.B_i$

  for each attribute C'_i: $e.C'_i = e2.C_i$

  for each attribute A_i

  if $attcor_i(A_{1i}, A_{2i})$ is $A_{1i}=A_{2i}$ or $A_{1i} \supseteq A_{2i}$

  then $e.A_i = e1.A_{1i}$

  if $attcor_i(A_{1i}, A_{2i})$ is $A_{1i} \cap A_{2i}$ or $A_{1i} \neq A_{2i}$

  then $e.A_i = e1.A_{1i} \cup e2.A_{2i}$ □

5.3 Integration of local elements and links. A first, quite evident, model independent integration rule applies to elements and links which appear in only one of the schemas to be integrated.

Integration rule 1:
Each element X1, of schema S1, which has no counterpart in the other schema, is added, as element X, to the integrated schema. The type of X is the same as the one of X1.

Correspondence assertion: X≡X1.
Mapping: X := X1.
Distribution: X is X1 on database S1.

Each link, X1—Y1, of schema S1, which has no counterpart in the other schema, is added, as link X—Y, to the integrated schema, where X and Y are the integrated

elements corresponding to X1 and Y1. The type of XY depends on the types of X and Y (see the discussion in section 7).

Correspondence assertion: X—Y≡X1—Y1.
Mapping: X—Y := rename [X1—Y1].
Distribution: X—Y is on database S1. □

6. Integration of Two Object Types

This section only considers value attributes within the object types. Reference attributes participate in link or path correspondence assertions. Their integration is therefore discussed in the next sections.

Integration rule 2:

Let X1, with value attributes (A_{11},..., A_{1j}, B_1,...B_k), and X_2, with value attributes (A_{21},..., A_{2j}, C_1,...C_h) be two object types in two schemas, $X_1 \in S_1$, $X_2 \in S_2$, such that:

$X_1 \equiv X_2$ with corresponding attributes:

$\text{attcor}_1(A_{11}, A_{21})$, $\text{attcor}_2(A_{12}, A_{22})$,..., $\text{attcor}_j(A_{1j}, A_{2j})$

the element in the integrated schema resulting from the integration of X1 and X2 is an object type X, such that:

- its name is X_1's one, except if the DBA chooses another one;
- its structure consists of the union of X1 and X2 attributes, as defined by the integrate-join of X1 and X2.

Correspondence assertions relating X to X_1 and X_2 are obvious:

$X \equiv X_1$ with corresponding attributes:

$\text{attcor}_1(A_1, A_{11})$, $\text{attcor}_2(A_2, A_{12})$,..., $\text{attcor}_j(A_j, A_{1j})$

$\text{attcor}_1(B'_1, B_1)$, $\text{attcor}_2(B'_2, B_2)$,..., $\text{attcor}_k(B'_k, B_k)$

$X \equiv X_2$ with corresponding attributes:

$\text{attcor}_1(A_1, A_{21})$, $\text{attcor}_2(A_2, A_{22})$,..., $\text{attcor}_j(A_j, A_{2j})$

$\text{attcor}_1(C'_1, C_1)$, $\text{attcor}_2(C'_2, C_2)$,..., $\text{attcor}_h(C'_h, C_h)$

Mappings between X, X_1 and X_2 may be defined as:

X:= integratejoin (X_1, X_2, $\text{attcor}_1(A_{11}, A_{21})$, $\text{attcor}_2(A_{12}, A_{22})$,..., $\text{attcor}_j(A_{1j}$, $A_{2j})$)

X_1:= project [A_1,..., A_j, B'_1,...B'_k] X.

X_2:= rename [A_1:A_{21},..., A_j:A_{2j}] project [A_1,..., A_j, C'_1, ...C'_h] X.

Description of actual *distribution* records that X is stored in both databases, S1 and S2. More precisely, X is split into fragments (vertical partition):

project $[B'_1,..., B'_k]$ X is in database S1
project $[C'_1,..., C'_h]$ X is in database S2
for each A_i, if $attcor_i(A_{1i}, A_{2i})$ is:

- $A_{1i} = A_{2i}$ then Ai is duplicated in both databases,
- $A_{1i} \supseteq A_{2i}$ then A_i is on site S1, and some values are duplicated in database S_2,
- $A_{1i} \cap A_{2i}$ then A_i is partially duplicated,
- $A_{1i} \neq A_{2i}$ then A_i is distributed between databases S_1 and S_2. □

Example: Referring to schemas S1 and S2 (section 3), let us suppose that the two databases describe the same set of cars. The correspondence assertions would be:

Expensive_car≡Car_model with corresponding attributes:
modelname = name, price = price

The integrated schema would be:

Expensive_Car (modelname, manufacturer, maximumspeed, price, horsepower, fuelconsumption)

Indeed, integration rule 2 can be applied directly:

- for relational models, to integration of relations without any external key,
- for ER models, to integration of entity types,
- for object-oriented models, to integration of object types without any reference attribute.

More integration rules, dealing with links and paths, have to be defined in order to be able to integrate ER relationship types, object-oriented reference attributes, and relational relations with external keys.

7. Integration of Two Links

Integration rule 3 deals with elementary links (attribute and reference links), and allows the integration of two equivalent links which bind equivalent elements. Rule 4 deals with paths which are composed of several links.

Integration rule 3:
Let A_1 and B_1 be two linked elements (object types, value or complex attributes) in schema S1, A2 and B2 be two linked elements (object types, value or complex attributes) in schema S2 , with the following correspondence assertions:

$A_1 \equiv A_2$
$B_1 \equiv B_2$
$A_1—B_1 \equiv A_2—B_2$

Let A be the integrated element in the integrated schema corresponding to A_1 and A_2, let B be the integrated element in the integrated schema corresponding to B_1 and B_2. The integration of $A_1—B_1$ and $A_2—B_2$ links is then a link A—B. The type of the link depends upon those of A and B:

- if A or B is a value attribute then A—B is an attribute link
- if A and B are object types then A—B is a reference link: a reference attribute, named B is added to A, or vice-versa.

As the three correspondence assertions are equivalence ones, the cardinalities of the two links, $A_1—B_1$ and $A_2—B_2$, are necessarily the same, and the cardinalities of the integrated link, A—B, are also the same.

Correspondence assertions are obvious:
$A—B \equiv A_1—B_1$
$A—B \equiv A_2—B_2$

Mappings:
A—B: = rename $[A_1—B_1]$
A—B: = rename $[A_2—B_2]$

Distribution: the A—B link is duplicated: it is stored in both databases, S1 and S2. □

Let us now discuss how rule 3 applies to the different models.

Relational model:
As the relational model has no existence constraint on its reference attributes (a relation may have zero, one or several external keys), rule 3 applies without modification.

Example:

S18:	S19:
Man (*manID*, name, address, wife)	Man (*manID*, name, address)
Woman (*womanID*, name, address)	Woman (*womanID*, name, address, husband)
Woman.womanID $\supseteq$ Man.wife	Man.manID $\supseteq$ Woman.husband

Correspondence assertions between S18 and S19:

Man ≡ Man with corresponding attributes:

manID = manID, name = name, address = address

Woman ≡ Woman with corresponding attributes:

womanID = womanID, name = name, address = address

Man—Woman ≡ Man—Woman

The integrated schema is S18 or S19, plus an integrity constraint which defines the cardinalities of the integrated link. These cardinalities are deduced from those of the two corresponding links. In S18, the link Man—Woman has cardinalities 0:1, 0:n (wife is an external key which must either reference one Woman, or bear a null value). In S19, the link Man—Woman has the inverse cardinalities: 0:n, 0:1. Therefore, the integrated link will have cardinalities 0:1, 0:1. These cannot be expressed in the relational model. An integrity constraint is needed. For instance, the integrated schema will be S18 plus the following integrity constraint: *there are no two men with the same wife* (i.e., there are no two Man tuples with the same wife value). Without the link correspondence assertion, integration of S18 and S19 would have generated two redundant external keys, wife and husband.

Object oriented models:

Integration rule 3 has to be modified in order to deal with the directed links of most object-oriented models. Suppose an object-oriented model where reference attributes, as value attributes, are directed. By "directed" we mean that a reference attribute allows direct access only from the parent element to the referenced object, and not in the other way around. Rule 3 is adjusted as follows.

Object oriented rule 3: Integration of two equivalent directed links:

$A_1 \rightarrow B_1 \equiv A_2 \rightarrow B_2$

generates a directed link A→B. If B is a value attribute, A→B is an attribute link. If B is an object type, A→B is a reference link: A holds a reference attribute pointing at B. □

Opposite directed links, like $A_1 \rightarrow B_1$ versus $B_2 \rightarrow A_2$, cannot be asserted as corresponding. Therefore, rule 3 will not apply and the two links are integrated through rule 1. As local elements, both will be added to the integrated schema.

Example: refer to schemas S10 and S11 of section 3.

Correspondence assertions between S10 and S11 are:

Car ≡ Car with corresponding attributes:

registration# = registration#, color = color, horsepower = horsepower

Person ≡ Person with corresponding attributes:

pin = pin, name = name, sex = sex, birthdate = birthdate

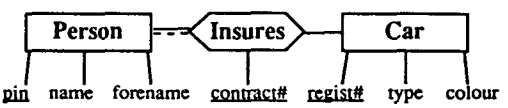
Integration only calls for rules 1 and 2, and results in schema S12.

Entity relationship models:
Integration rule 3 may be directly applied to ER models in order to integrate role links between entity types and relationship types. Using both rules 3 and 1 allows us to define deduced rules integrating object types with reference attributes, i.e., relationship types. Equivalent reference links are integrated[11], and local reference links are added. Applying these two rules to ER models, consistently with the basic principle of choosing the more unconstrained structure (the entity type) in case of structural conflict, produces the following rule:

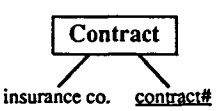
ER rule 1+3: Integration of an entity type E1 and an equivalent n-ary relationship type R2 is an entity type E and n binary relationship types linking E and the entity types that R2 links. Integration of two equivalent relationship types, R1 and R2, is a relationship type which links all the entity types that R_1 or R_2 link. □

Example: integration of an entity type and a relationship type

S20:

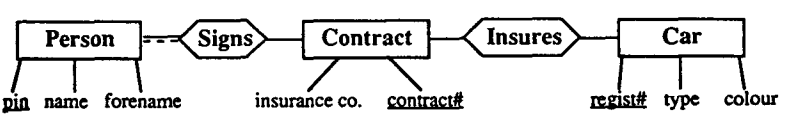


S21:



Correspondence assertion between S20 and S21: Insures ≡ Contract with corresponding attributes: contract# ≡ contract#

The integrated schema is[12]:

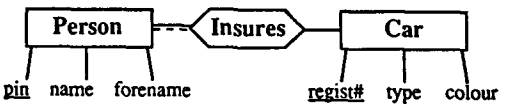


11. The limited cardinalities of role links (each role link of any relationship must always point at exactly one existing entity) must be taken into account when choosing the type of integrated link.

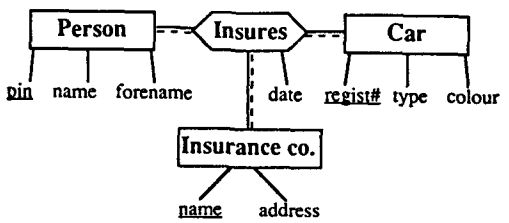
12. This schema could be simplified (by merging Signs+Contract+Insures into one relationship type) if no other relationship comes on Contract. This would be done in a final refinement step, not discussed here.

Example: integration of two relationship types

S22:



S23:



Correspondence assertions between S22 and S23:

Person ≡ Person with corresponding attributes:

pin ≡ pin, name ≡ name, forename ≡ forename

Car ≡ Car with corresponding attributes:

registration# ≡ registration#, type ≡ type, color ≡ color

Insures ≡ Insures

Person—Insures ≡ Person—Insures

Insures—Car ≡ Insures—Car

The integrated schema is S23.

8. Integration of Links and paths

The integration process should not generate redundant information in the integrated schema. When integrating links and paths, we have to know if each one bears independent information, or if one can be deduced from the other. Two cases may happen:

- a link A_1—B_1 is equivalent to a path A_2—...—B_2; therefore keeping in the integrated schema only A_2—...—B_2 is enough. The direct link will be deduced by composition of the components links of A_2—...—B_2.

- two paths A_1—...—B_1 and A_2—...—B_2 are equivalent, then both paths must be kept in the integrated schema. Deleting one path would delete all its component links which are not equivalent to any other link or path. An integrity constraint stating that the two paths are equivalent is added to the integrated schema.

Integration rule 4: links and paths integration rule:
Let $A_1, B_1, \ldots D_1$ be elements in schema S1, let $A_2, B_2, \ldots D_2$ be elements in schema S2, with the two correspondence assertions:

$A_1 \equiv A_2, D_1 \equiv D_2$.

Let A (respectively D) be the integrated element in the integrated schema corresponding to A_1 and A_2 (respectively to D_1 and D_2), then:

1. the correspondence assertion between a link and a path:

   $A_1—D_1 \equiv A_2—B_2—\ldots—D_2$

   generates in the integrated schema the path $A—B_2'—\ldots—D$ (where B_2' is the integrated element corresponding to B_2);

2. the correspondence assertion between two paths:

   $A_1—B1—\ldots—D_1 \equiv A_2—B_2—\ldots—D_2$

   generates in the integrated schema the two paths:

   $A_1—B_1'—\ldots—D_1$ and $A_2—B_2'—\ldots—D_2$

   (where B_1' and B_2' are the integrated elements corresponding to B_1 and B_2), and an integrity constraint which states that the two paths link the same occurrences.

In both cases, the generated paths are created according to the modeling concepts of the linked elements, as in rule 3. □

Integration rule 4 includes integration rule 3.

The schema will be integrated to the extent that the DBA describes the correspondence assertions. For example, let two schemas be related by the following correspondence assertions:

$A_1 \equiv A_2, C_1 \equiv C_2, F_1 \equiv F_2$
$A_1—C_1 \equiv A_2—\ldots—C_2$
$C_1—F_1 \equiv C_2—\ldots—F_2$

If, instead of stating the two path assertions, the DBA only asserts:

$A_1—\ldots—C_1—\ldots—F_1 \equiv A_2—\ldots—C_2—\ldots—F_2$

then less knowledge is given to the integrator and the integration will be less advanced.

Relational model:
Integration rule 4, as rule 3, applies without modification.

Example: refer to schemas S5 and S6, with the following correspondence assertions in between:

Car ≡ Car with corresponding attributes:
registration# = registration#, color = color, horsepower = horsepower

Person ≡ Person with corresponding attributes:
pin = pin, name = name, sex = sex, birthdate = birthdate
Car—Person ≡ Car—Carownership—Person

The integrated schema is S6.

Object-oriented models:
Integration rule 4, as rule 3, has to be adjusted in order to integrate links and paths which are oriented in the same direction.

Entity relationship models:
Integration rule 4, as rule 3, applies without modification.

9. Integration of an object type and an attribute

One of our basic integration principles is that whenever conflicting descriptions exist in different views, the integrated schema will hold the more unconstrained representation in order to be able to derive the other descriptions through restrictive mappings. Integration of an object type O and a value or complex attribute A produces an object type whose structure results from the merging of the structures of O and A as in rule 2. The distribution and mappings are also similar to those of rule 2. The main difference is that the integrated object is linked via a reference attribute to the parent element of A.

Integration rule 5: integration of an object type and a value or complex attribute:

Let X_1, with value attributes (A_{11},..., A_{1j}, B_1,..., B_k), be an object type of schema S1; let X_2 be a complex attribute of element E_2 of schema S_2 with component value attributes (A_{21},..., A_{2j}, C_1,..., C_h), or an atomic-value attribute. In this latter case, we consider that X2 has itself as component attribute; let the correspondence assertion be:

$X_1 \equiv X_2$ with corresponding attributes:
$\text{attcor}_1(A_{11}, A_{21})$, $\text{attcor}_2(A_{12}, A_{22})$, ..., $\text{attcor}_j(A_{1j}, A_{2j})$.

Let E be the element corresponding to E_2 in the integrated schema, the elements in the integrated schema resulting from the integration of X_1 and X_2 are an object type X, and a reference link between E and X, such that:

- the attribute X_2 of E_2 is transformed into a reference attribute X'_2 referencing X; cardinalities of X'_2 are equal to those of X_2,
- the name of X is the same as the one of X_1, unless the DBA chooses another one,
- the structure of X consists of the union of the attributes of X_1 and X_2, as defined by the integrate-join of X_1 and X_2.

Correspondence assertions relating X to X_1 and X_2 are obvious:

$X \equiv X_1$ with corresponding attributes:

$attcor_1(A_1, A_{11})$, $attcor_2(A_2, A_{12})$,..., $attcor_j(A_j, A_{1j})$

$attcor_1(B'_1, B_1)$, $attcor_2(B'_2, B_2)$, ..., $attcor_k(B'_k, B_k)$

$X \equiv X_2$ with corresponding attributes:

$attcor_1(A_1, A_{21})$, $attcor_2(A_2, A_{22})$,..., $attcor_j(A_j, A_{2j})$

$attcor_1(C'_1, C_1)$, $attcor_2(C'_2, C_2)$, ..., $attcor_h(C'_h, C_h)$

$E—X \equiv E_2—X_2$

Mappings between the integrated schema, S1 and S2 may be defined as:

X:= integrate-join (X_1, X_2, $attcor_1(A_{11}, A_{21})$, $attcor_2(A_{12}, A_{22})$,..., $attcor_j(A_{1j}$, A_{2j}))

E—X:= rename[$E_2—X_2$]

X_1:= project [A_1,..., A_j, B'_1,...B'_k] X

X_2:= rename [A_1:A_{21},..., A_j:A_{2j}] project [A_1,..., A_j, C'_1,...C'_h] X

Distribution:

X is stored in both databases, S_1 and S_2; it is split into fragments as in rule 2. The E—X link is on database S_2 only. □

If X_1 and/or X_2 have reference attributes, rule 1 or 3 is activated in order to add or integrate those reference links.

Relational model:

Rule 5 applies as follows.

Relational rule 5:

Integration of a relation R1 of schema S1 and a value attribute A2 of relation R2 of schema S2, generates in the integrated schema a relation R with the attributes of R1 and a reference integrity constraint binding the relation R2′ (the integrated relation corresponding to R2) to R. □

Example:

S24: Car (*registration#*, color, power, owner#)

S25: Person (*pin*, name, address)

Correspondence assertion between S24 and S25:

owner# ≡ Person with corresponding attributes: owner# = pin

Under these assumptions, the integrated schema is:

Car (*registration#*, color, power, owner#)

Person (*pin*, name, address)

Person.pin ⊇ Car.owner#

Relational rule 5 transforms the value attribute owner# of S24 into a reference attribute; i.e., owner# in the integrated schema is an external key referencing Person.

Entity relationship models:
Rule 5, as rule 3, has to be adjusted to entity relationship models as follows.

Entity relationship rule 5:
Integration of an entity type X1 of database S1 and an attribute X2 (whose parent element is entity type E2) of database S2 generates an object type X and a link E—X (where E is the entity type corresponding to E2 in the integrated schema). As entity type X1 may be bound by a relationship in S1, X must be an entity type, and the E—X link is a binary relationship binding entity types E and X. □

Examples:
Rules 5 and 3 (link integration rule) allow us to integrate S14 and S15 of section 3. If the two databases are equivalent, and if the equivalence of the two links, Car—owner and cars—Person, is asserted, the integrated schema is S13.

In the same way, integrating S13 and S14 also generates S13.

Object-oriented models:
Rule 5 applies without modification.

Example: refer to schemas S7 and S11 of section 3.

Correspondence assertions between S7 and S11 are:

Car ≡ Car with corresponding attributes:
registration# = registration#, color = color, horsepower = horsepower
owner ≡ Person with corresponding attributes:
pin = pin, name = name, sex = sex, birthdate = birthdate
Car→owner ≡ Car→Person

The integrated schema is S11.

10. Conclusion and Future Work

There is an ever-increasing need for building integrated or federated systems from various heterogeneous database systems that are already in operation. A semi-automatic database integration methodology would significantly alleviate the integration task, which is presently a manual task. From an economic perspective, this integration, while opening the way to new federated database services, should also allow the continued usage of existing databases and application programs.

This paper introduces a database integration methodology, designed to meet the above objectives. Our approach is based on the following major features:

- consideration of links in the integration process,
- automatic resolution of structural conflicts (arising because of different representations of the same real-world objects),
- conflict resolution performed without modification of initial views,
- applicability to a variety of data models.

The first three features are distinguishing. The semantics of links is out of the scope of current methodologies. Structural conflicts necessarily arise from user requirements, based on the different needs that exist in the real world. Instead of forcing schemas to conform to a unique representation, as presently required, our approach relies on the idea that the complexity inherent to structural conflicts should be supported by establishing appropriate, powerful mapping facilities among initial schemas and the integrated schema.

The fourth feature is of special interest when heterogeneous databases have to be integrated. To that extent, we defined data model independent integration rules, which are customized to the various classical data models. This approach ensures that the integration strategy is consistent over the various models. Moreover, it is feasible to allow the description of inter-schema correspondences directly on the existing schemas, i.e., without requiring a preliminary step to translate all existing schemas into their equivalent version based on some common model. The whole integration process becomes much simpler for the DBA. The resulting system can be made user-friendly through the support of a multimodel interface, allowing each user to interact with the new DBMS through his/her preferred data model.

Irrespective of the data model, the schemas can be interpreted as graphs, i.e., sets of nodes and edges. We focused on defining integration rules for these two sorts, which we called elements and links.

Additional features are common with other existing approaches:

- use of a formal declarative approach for the definition of inter-schema correspondences,
- automatic generation of structural and operational mappings between the initial schemas and the integrated schema. Operational mappings provide support to allow users to query and update the database through their own view.

To implement a formal declarative approach, we defined a model for describing correspondence assertions. These assertions instruct the integrator tool about similarities in the semantics of the schemas. For each assertion, formal rules state how to derive the constructs which are to be inserted into the integrated schema and the mappings between the integrated schema and the initial ones.

Finally, the methodology proposed in this paper is applicable to the view integration process, a crucial step in classical database design. Our scheme allows the users to state their views without being constrained by requirements from other users, and without being forced to modify their definition if a conflict arises with some other view.

In the future our research will be devoted to:

- the integration of inclusion, intersection, and exclusion assertions. We intend to analyze when and how it is appropriate to build generalization hierarchies in the integrated schema,
- consideration of generalization links in correspondence assertions and integration rules,
- detailed analysis of the integration of complex attributes,
- integration of 1:n correspondences, in which one object in one view/schema corresponds to a set of objects in the other view/schema.

As far as view integration is concerned, our plans include the specification and implementation of an intelligent view definition facility, so that most of the integration problems in an actual situation are solved at view-definition time, rather than at times when views are to be integrated.

Acknowledgements

The authors are indebted to Prof. Bharat Bhargava for many helpful suggestions. Thanks should be extended to the anonymous reviewers, who did an excellent job in carefully analyzing the paper. Their criticisms were a major source for improving its quality. This research is supported by the Fonds National de la Recherche Scientifique Suisse (Projet SUPER), and by INRIA, under the auspices of the French national database research project (PRC BD3: Programme de Recherches Coordonnées Bases de données de 3ème génération).

References

Batini, C., Lenzerini, M. A methodology for data schema integration in the entity-relationship model, *IEEE Transactions On Software Engineering*, 10:650–664, 1984.

Batini, C., Lenzerini, M., and Navathe, S.B. A comparative analysis of methodologies for database schema integration, *ACM Computing Surveys,* 15:323–363, 1986.

Bertino, E. Integration of heterogeneous data repositories by using object-oriented views, *First International Workshop on Interoperability in Multidatabase Systems,* Kyoto, 1991.

Biskup, J., Convent, B. A formal view integration method, *ACM SIGMOD International Conference on Management of Data,* Washington, 1986.

Bouzeghoub, M., Comyn-Wattiau, I. View integration by semantic unification and transformation of data structures, *Ninth International Conference on Entity-Relationship Approach,* Lausanne, 1990.

Carey, M., DeWitt, D. and Vandenberg, S. A data model and query language for EXODUS, *ACM SIGMOD International Conference on Management of Data,* Chicago, 1988.

Civelek, F., Dogac, A., and Spaccapietra, S. An expert system approach to view definition and integration, *Seventh International Conference on Entity-Relationship Approach,* Rome, 1988.

Czejdo, B., Taylor, M. Integration of database systems using an object-oriented approach, *First International Workshop on Interoperability in Multidatabase Systems,* Kyoto, 1991.

Deen, S., Amin, R., and Taylor, M. Data integration in distributed databases, *IEEE Transactions On Software Engineering,* 13:860–864, 1987.

DeMichiel, L. Resolving database incompatibility: An approach to performing relational operations over mismatched domains, *IEEE Transactions on Knowledge and Data Engineering,* 1:485–493, 1989.

Demurjian, S. and Hsiao, D. Towards a better understanding of data models through the multilingual database system, *IEEE Transactions On Software Engineering,* 14:946–958, 1988.

Diet, J. and Lochovsky, F. Interactive specification and integration of user views using forms, *Eighth International Conference on Entity-Relationship Approach,* Toronto, 1989.

Elmasri, R. and Wiederhold, G. Data model integration using the structural model, *ACM-SIGMOD International Conference on Management of Data,* Boston, 1979.

Elmasri, R., Weeldreyer, J., and Hevner, A. The category concept: an extension to the entity-relationship model, *Data and Knowledge Engineering,* 1:75–116, 1985.

Fankhauser, P., Litwin, W., Neuhold, E., and Schrefl, M. Global view definition and multidatabase languages—Two approaches to database integration. In: Speth, R., ed. *Research into Networks and Distributed Applications,* Amsterdam: North-Holland, 1988, pp. 1069–1082.

Ferrier, A. and Stangret, C. Heterogeneity in the distributed database management system SIRIUS-DELTA, *Eighth International Conference on Very Large Data Bases,* Mexico City, 1982.

Hayne, S. and Ram. S. Multi-User view integration system (MUVIS): An expert system for view integration, *IEEE Sixth International Conference on Data Engineering,* Los Angeles, 1990.

Hwang, H. and Dayal, U. View definition and generalization for database integration in a multibase system, *IEEE Transactions On Software Engineering,* 10:628–645, 1984.

Jardine, D. and Yazid, S. Integration of information submodels. In: Falkenberg, E.D. and Lindgreen, P., eds. *Information Systems Concepts: An In-Depth Analysis,* Amsterdam: North-Holland, 1989, pp. 247-267.

Kambayashi, Y., Rusinkiewicz, M., and Sheth, A. *First International Workshop on Interoperability in Multidatabase Systems,* Kyoto, 1991.

Kaul, M., Drosten, K., and Neuhold, E. ViewSystem: integrating heterogeneous information bases by object-oriented views, *IEEE Sixth International Conference on Data Engineering,* Los Angeles, 1990.

Kent, W. Solving domain mismatch and schema mismatch problems with an object-oriented database programming language, *Seventeenth International Conference on Very Large Data Bases,* Barcelona, 1991.

Kim, W. Research directions for integrating heterogeneous databases, *Workshop on Heterogeneous Database Systems,* Chicago, 1989.

Landers, T. and Rosenberg, R. An overview of multibase. In: Schneider, H.-J., ed. *Distributed Data Bases,* Amsterdam: North-Holland, 1982, 153–184.

Larson, J., Navathe, S., and Elmrasri, R. A theory of attribute equivalence in databases with application to schema integration, *IEEE Transactions on Software Engineering,* 15:449–463, 1989.

Litwin, W. MALPHA: A relational multidatabase manipulation language, *IEEE First International Conference on Data Engineering,* Los Angeles, 1984.

Litwin, W., Mark, L., and Roussopoulos, N. Interoperability of multiple autonomous databases, *ACM Computing Surveys,* 22:267–293, 1990.

Mannino, M. and Effelsberg, W. Matching techniques in global schema design, *IEEE First International Conference on Data Engineering,* Los Angeles, 1984.

Motro, A. Superviews: Virtual integration of multiple databases, *IEEE Transactions On Software Engineering,* 13:785–798, 1987.

Motro, A. and Buneman, P. Constructing superviews, *ACM-SIGMOD International Conference on Management of Data,* Ann Arbor, 1981.

Navathe, S. and Gadgil, S. A methodology for view integration in logical database design, *Eighth International Conference on Very Large Data Bases,* Mexico City, 1982.

Navathe, S., Elmrasi, R., and Larson, J. Integrating user views in database design, *IEEE Computer,* 19:50–62, 1986.

Parent, C. and Spaccapietra, S. An algebra for a general entity-relationship model, *IEEE Transactions On Software Engineering,* 11:634–643, 1985.

Parent, C. and Spaccapietra, S. ERC+: an object based entity-relationship approach. In: Loucopoulos, P. and Zicari, R., eds. *Conceptual Modelling, Databases and CASE: An Integrated View of Information Systems Development,* London: John Wiley, 1992, to appear.

Schek, H.-J. and Scholl, M. The relational model with relation-valued attributes, *Information Systems,* 11:137–147, 1986.

Sheth, A., Larson, J., Cornelio, A., and Navathe, S. A tool for integrating conceptual schemas and user views, *IEEE Fourth International Conference on Data Engineering,* Los Angeles, 1988.

Sheth, A. and Gala, S. Attribute relationships: An impediment in automating schema integration, *Workshop on Heterogeneous Database Systems,* Chicago, 1989.

Sheth, A. and Larson, J. Federated database systems for managing distributed, heterogeneous, and autonomous databases, *ACM Computing Surveys,* 22:183–236, 1990.

Siegel, M. and Madnick, S. Schema integration using metadata, *Workshop on Heterogeneous Database Systems,* Chicago, 1989.

Siegel, M. and Madnick, S. A metadata approach to resolving semantic conflicts, *Seventeenth International Conference on Very Large Data Bases,* Barcelona, 1991.

de Souza, J. SIS - A schema integration system, *Fifth British National Conference on Databases,* Canterbury, England, 1986.

Spaccapietra, S., Demo, S., DiLeva, A., Parent, C., Celis, C., and Belfar, K. An approach to effective heterogeneous database cooperation. In: van de Riet, R. and Litwin, W., eds. *Distributed Data Sharing Systems,* Amsterdam: North-Holland, 1982, pp. 209–218.

Spaccapietra, S. and Parent, C. View integration: a step forward in solving structural conflicts, *IEEE Transactions on Data and Knowledge Engineering,* due to appear October 1992.

Stocker, P., Atkinson, M., Gray, P., Gray, W., Oxborrow, E., Shave, M., and Johnson, R. Proteus: A heterogeneous distributed data-base project. In: Gray, P. and Atkinson, M., eds. *Databases: Role and Structure,* Cambridge: Cambridge University Press, 1984, 125-150.

Templeton, M., Brill, D., Chen, A., Dao, S., Lund, E., McGregor, R., and Ward, P. Mermaid: A front end to distributed heterogeneous databases, *Proceedings of the IEEE,* 75:695–708, 1987.