View Integration: A Step Forward in Solving Structural Conflicts

Stefano Spaccapietra, *Senior Member, IEEE*, and Christine Parent

***Abstract*—Thanks to the development of the federated systems approach on the one hand and the emphasis on user involvement in database design on the other, the interest in schema integration techniques is significantly increasing. Theories, methods and sometime tools have been proposed. Conflict resolution is the key issue. Different perceptions by schema designers may lead to different representations. A way must be found to support these different representations within a single system.**

Most current integration methodologies rely on modification of initial schemas to solve the conflicts. This approach needs a strong interaction with the database administrator, who has authority to modify the initial schemas.

This paper presents an approach to view integration specifically intended to support the coexistence of different representations of the same real-world objects. The main characteristics of this approach are the following:

- **automatic resolution of structural conflicts,**
- **conflict resolution performed without modification of initial views,**
- **use of a formal declarative approach for user (or database administrator) definition of interviews correspondences,**
- **applicability to a variety of data models, and**
- **automatic generation of structural and operational mappings between the views and the integrated schema.**

Allowing users' views to be kept unchanged should result in improved user satisfaction. Each user will be able to define his own view of the database, without having to conform to some other user's view.

Moreover, such a feature is essential in database integration if existing programs are to be preserved.

***Index Terms*—Database design, database management systems, data structures, schema integration, data models, view integration, entity-relationship modelling.**

I. Introduction

CONSIDERING the actual trend emphasizing user involvement in application development, cooperative work, and user-oriented data models, view integration is likely to become a key issue for future database design methodologies.

Since 1978, view integration has been identified as the design step aimed at producing a global conceptual schema of a database from a set of formally defined users' views [1]. At first, integration was a completely manual process. A superman called the database administrator (DBA), who was able to seize the whole complexity of data in the enterprise, would produce the correct global image of the data structure. Views, at that time, were mainly considered to be an aid to the DBA to let him know about user requirements, or as a filter to ensure privacy of data.

Nowadays, it is clear that database design is too complicated a task to be performed in a centralized way. Thus, a more reasonable approach is to first let the different components in the enterprise build their own view of the database (which should be a manageable task), and second to integrate these views in a global schema using an automated tool directed by the DBA. Current view integration methodologies intend to rule this second process.

There also is an increasing interest in federated architectures, where existing databases are integrated into a single distributed database. Assuming the schemas of the existing databases to be views over the future distributed database, an integration methodology should lead to the automatic design of the global schema. This is usually called database integration, to emphasize that the integration process has to cope with the integration of existing data (and programs), not just with metadata in the views.

There has been a large amount of work in the integration area: a detailed survey by Batini *et al.* [2] discusses 12 methodologies for view or database integration (or both), and new contributions continuously appear in the literature [3]–[10].

As view integration is a process in which knowledge of the semantics of data is needed, it is no surprise that most of the current methodologies rely on a semantic data modelling approach. In particular, the entity-relationship (ER) approach, in its various extended forms, represents the majority choice. This is consistent with the actual state of affairs, in which the ER model acts as an almost *de facto* standard in the area of conceptual design methods and tools.

When using a semantic data model, it is possible that different designers (users, in our case) model the same piece of reality in different ways. This might happen either because the data model supports equivalent constructs, or because designers have different perceptions of that reality. Multiplicity of possible representations of a given real world is called *semantic relativism.*

In the ER approach, for instance, the designer is left with the responsibility to decide whether a real-world object should be represented as an entity, a relationship, or an attribute.

Supporting semantic relativism is the key to achieving the goal of enabling each user to construct his own view, accord-

Manuscript received August 1990. This work was supported by INRIA under the auspices of the French national research project Programme de Recherches Coordonnées Bases de données de 3ème génération (PRC BD3), as well as by the Fonds National de la Recherche Scientifique Suisse.

S. Spaccapietra is with the Computer Science Department, Ecole Polytechnique Fédérale, 1015 Lausanne, Switzerland.

C. Parent is with the Computer Science Department, Université de Bourgogne, 21004 Dijon Cedex, France.

IEEE Log Number 9211303.

ing solely to his perception of the world and independently from other users [7]. To fully support semantic relativism, a view integration methodology should cope with all possible conflicting representations that may be found among various views of a database.

Conflicting representations have always been a challenge for integration methodologies. Naming conflicts (due to homonyms and synonyms) have been dealt with from the beginning (see, for instance, [11] and [12]). The difficulty here lies in conflict identification (how to find out that there is a conflict), rather than in conflict resolution (usually, one view is modified to remove the naming conflict).

Once the existence of similar object classes in two views has been identified (and the possible name conflict resolved), a comparison of their semantics may lead to another conflict if the classes are not equivalent. For instance, a Student class in one view may be found similar to a CS-Student class (grouping students majoring in computer science) in another view. In this case, view integration should make explicit the fact that one class, CS-Student, is a subclass of the other one.

Mannino and Effelsberg [13] first focused on this type of conflict, which subsequently became the main topic in most later works on view integration. The generalization concept has been extensively used as a solution to such conflicts (except in works based on the relational model).

Finally, a structural conflict arises whenever parts of the same reality are represented in different views using different structural constructs. Typically, the same set of real-world objects may be represented as an entity type in one view and as an attribute of an entity type in another view. Although structural conflicts were the topic of what was possibly the first paper on view integration [14], little effort has been put into the search of automated strategies to solve this type of conflict. Existing methodologies rely on the DBA (purposely in [15]) for conforming of schemas, a process in which views are modified by forcing related concepts to be represented by the same structural construct. For instance, whenever an attribute A, in view $V1$, corresponds to an entity type E, in view $V2$, the DBA must transform the attribute A into an entity type, say EA. After view modification, the correspondence will be between EA and E: The structural conflict has been removed and integration of EA with E can proceed.

Consequently, current ER methodologies are restricted to integration of entities with entities, relationships with relationships, and attributes with attributes. This is similar to what a relational approach may achieve using various interrelational dependencies [16]–[18]. One limited exception may be found in [8], where the authors integrate an entity type with a relationship type, under the very restrictive hypothesis that the key of the entity type is the aggregate of the keys of the entity types participating in the relationship.

This paper proposes an integration methodology, designed to be able to do the following:

- automatically integrate views with (or without) structural differences among corresponding objects,
- perform such an integration without requiring initial views to be modified,
- provide a formal definition of interviews correspondences and of integration rules,
- offer an algorithm to perform integration in all conceivable cases, and
- be model independent.

Allowing users' views to be kept unchanged should result in improved user satisfaction. Moreover, such a feature is essential in database integration, as it allows existing programs to continue running after the local databases have been integrated into a single, distributed database. Our methodology should successfully apply to database integration.

Finally, we aim at establishing fundamentals of an integration methodology, so that the same approach can be used whatever data model is used for view definition. To that purpose, our rules basically support integration of objects and of links, two concepts that can be regarded as underlying every conceptual data model. In this paper, we use an ER-like model to illustrate our arguments. Corresponding rules for object-oriented models are described in [19]. The methodology we propose starts with a manual input from the DBA. This input carries the formal description of interrelationships among the views, in terms of correspondences between types as well as between populations. The methodology automatically produces the schema integrating the views, and the mappings between the final schema and each of the initial views.

The next section briefly overviews the ER-like data model we use in this paper to support analysis of view integration. A few integration examples are then introduced, in Section III, using this model.

The description of our integration methodology follows from the next section on. The initial step in any integration methodology is the acquisition of knowledge about correspondences that exist among the views. Section IV proposes a formal model to describe that knowledge. Section V defines integration rules, used to build the integrated schema according to known correspondences. An integration algorithm, enforcing these rules, is described in Section VI and illustrated in three examples in Section VII. Finally, the conclusion points out ongoing or future work we plan on this topic.

II. The ERC+ Model

Hereinafter, we illustrate our ideas using an extended ER model we have defined to support complex object description and manipulation. This model is called ERC+: ER for complex objects (the + denotes the enrichment of the basic ERC model [20], [21] to include generalizations [22]).

The provision for complex object modelling and management is one of the major goals of data models today. By *complex object* we mean an object represented by a collection of informations, its components, such that each of these components, in its turn, may be represented by a collection of informations, and so on. Supporting complex objects does not contradict the basic distinction the ER approach makes between entities and attributes: That distinction is based on semantic considerations (which are the primary objects of interest), not on syntactic properties (i.e., being atomic or not).

ERC+ specifically allows for this iterating description of an object, up to an arbitrary number of levels. The resulting structure is an attribute tree, whose root is the object. Moreover, any node in the tree may carry a unique attribute value, or a multiset (bag) of attribute values.

Let us briefly outline ERC+ features.

1) The structure of an entity type consists of a set of one or more attributes.
2) Relationship types may have attributes as well.
3) Relationship types may connect any number of participating entity types; they are said to be cyclic if the same entity type participates more than once in the relationship type.
4) A role name is associated with each participation of an entity type into a relationship type. It is characterized by its minimum and maximum cardinalities, specifying whether it is a 0–1, 0–n, 1–1, or 1–n link from the entity type to the relationship type.
5) Attributes may be
   a) either mandatory or optional. An instance of an optional attribute may be empty (no value); for a mandatory attribute a value must be defined in each instance of the attribute.
   b) either monovalued or multivalued. An instance of a multivalued attribute may include several (possibly duplicate) values, while an instance of a monovalued attribute is made up of a single value.
   c) either atomic or complex. If atomic, the attribute is nondecomposable; its values are atomic. If complex, the attribute is made up of a set of other attributes, which are said to be the components of that attribute. Component attributes may be atomic or complex. This nesting can proceed to any number of levels (refer to Fig. 2).
6) Entity types and relationship types may have zero, one, or more sets of attributes serving as identifiers. If no identifier is known, the respective population may include duplicates (different occurrences with the same value). In particular, two or more relationships may connect the same entities and have same values for their attributes.
7) Two generalizations are supported, the "is-a" and the "may-be-a" generalizations. The former corresponds to the well-known generalization concept; the latter has similar semantics but does not require an inclusion dependency between the subtype and the type. No automatic inheritance is implicitly built in the querying mechanism, but an explicit operator provides for the desired inheritance effects [22].

Fig. 1 shows a simple diagram of a hypothetical ERC+ schema (identifiers and some role names are not shown). An example of a more complex object type is given in Fig. 2.

In these figures, a single continuous line represents a 1:1 link (mandatory monovalued); a single dotted line represents a 0:1 link (optional monovalued); a double dotted line represents a 0:n link (optional multivalued); and a double, dotted plus continuous line represents a 1:n link (mandatory multivalued).

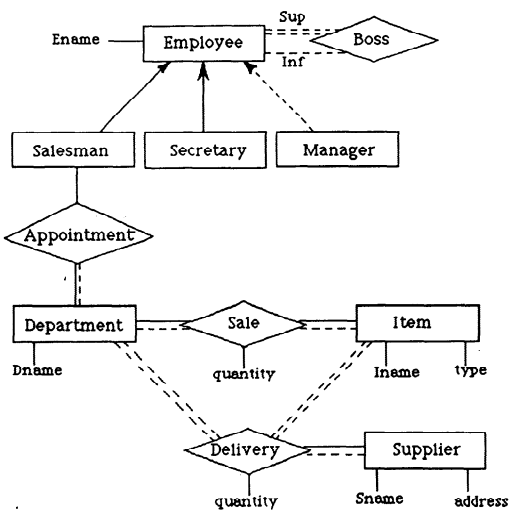

Fig. 1. An example of an ERC+ diagram.

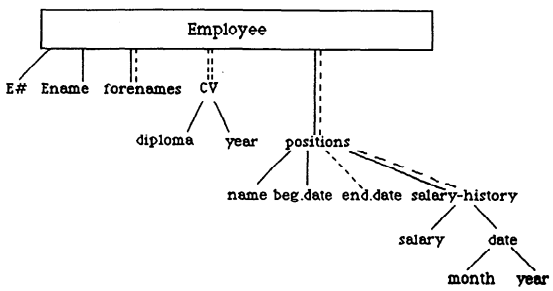

Fig. 2. An ERC+ diagram for a complex entity.

For instance, the entity type Employee in Fig. 1 is linked to itself by a cyclic, binary relationship type, Boss, whose semantics is "the employee in the Sup role is the boss of the employee in the Inf role" (or, equivalently, "the employee in the Inf role is subordinate to the employee in the Sup role"). Optionality of Sup and Inf roles states that an employee may have no subordinates, and an employee may have no boss. The Inf role is monovalued (an employee has at most one boss), while the Sup role is multivalued (a boss may have more than one subordinate). All salesmen and secretaries are employees. Managers may be employees or not.

In Fig. 2, an employee is mandatorily described by an employee number, a name, a forename and a position. Optionally, the employee record may hold more forenames, more (past) positions, and one or more pairs of ⟨diploma, year⟩ information. Each position is described by a set of component attributes, and so on.

The ERC+ model is complemented with the definition of formal manipulation languages: an associated algebra and an equivalent calculus [23] for querying an ERC+ database. In the integration methodology, these languages form the basis for building the operational mappings between the views and the conceptual schema. These mappings will transform user requests on the views into the equivalent requests on the conceptual schema.

More complete and formal presentations of ERC+ and its algebra may be found in [21] and [24]. For a discussion of

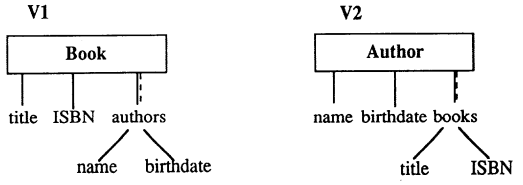

Fig. 3.

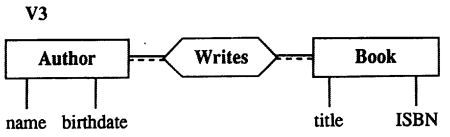

Fig. 4.

the ERC+ approach compared to object-oriented modelling, the reader is referred to [25].

III. Integration Examples

A. Example 1: Library Information System

Let us consider some library information system, for which the two views below, $V1$ and $V2$, have been defined by different users. These views correspond to the classical manual files maintained in libraries. $V1$ presents the world of interest as composed of books, where each book is described by its title, its ISBN number and its authors' name and birth date (one or several authors). $V2$ presents the world of interest as composed of authors, where each author is described by its name, its birth date, and its books' title and ISBN number (one or several books). (See Fig. 3.)

Clearly, these are two alternative representations of the same universe. They convey compatible semantics but show conflicting perceptions of data structure. The integration of the two views first needs an explicit statement, from the DBA, that Book+authors in $V1$, and Author+books in $V2$, represent the same real-world objects and links in between. Afterwards, what one would expect from a clever integration methodology is to be able to produce the integrated schema, $V3$, obvious to a human DBA (see Fig. 4.)

Current integration methodologies fail to build $V3$. They only integrate similar objects types, based on object identifiers. Here, there is no way the population of books (identifier: ISBN) can be directly compared with the population of authors (identifier: name). As a book and an author are not compatible objects, existing methodologies cannot do anything but carry over the two entity types into the integrated schema, which will thus be highly redundant (bearing twice the information on books, authors, and the link between a book and its authors). Alternatively, they will call on the DBA for him to modify both views and make them look like $V3$ (modification of only one view would lead to an incorrect result, as seen below). Thus, the modified views will no longer show any conflict, and the integrator will be able to build the integrated schema. In other terms, integration has been done by the DBA and the initial views have not been supported.

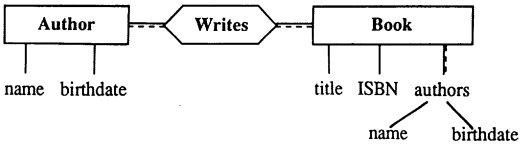

Fig. 5.

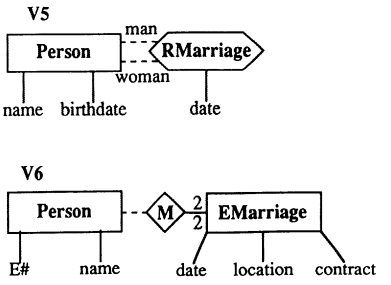

Fig. 6.

Current methodologies would also fail if the views to be integrated would be $V1$-$V3$ (or $V2$-$V3$). Directed to integrate $V1$ with $V3$, they would find out that the two Book entity types represent the same object class, and therefore merge these two into a single entity type. The resulting schema would be as shown in Fig. 5.

This schema clearly shows an unacceptable redundancy (with regard to authors information, which exists twice, as well as the author–book link). The redundancy is because the integration of the two Book entity types ignores the equivalence between the authors attribute in $V1$ and the Writes relationship in $V3$.

B. Example 2: Marriages

The library example showed a conflict based on the representation of the same concept (book, author) as an entity type in one view and as an attribute in the other view. A marriage example is now used to show a conflict involving representation as an entity type versus representation as a relationship type.

Let us consider again two possible views over the same universe of discourse (see Fig. 6).

View $V5$ has been defined by a user mainly interested in persons, but who wants also to know about marriages in between. Consequently, $V5$ describes an entity type Person and relates a person (man role) to another person (woman role) through the RMarriage relationship.

View $V6$, instead, comes from the lawyer's office. It deals with marriages as notarized acts, considered as entities (EMarriage). It also considers persons as entities (Person), to which marriages are mandatorily related twice for each marriage occurrence (via the relationship type M).

If the two views see the same set of persons, it is obvious that RMarriage and EMarriage describe the same set of real-world objects. A smart integrator would then determine that $V5$ must be conformed to V6, and produce an integrated schema equal to $V6$. Again, existing methodologies would integrate the two Person entity types and produce the redundant schema illustrated in Fig. 7.

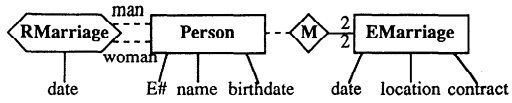

Fig. 7.

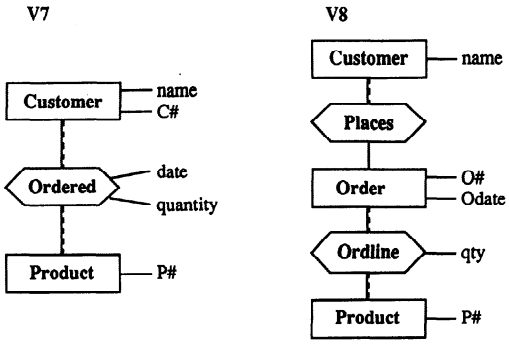

Fig. 8.

C. Example 3: Customers' Orders

Another limitation in current methodologies which provide for relationship integration is that integration is often performed only if the relationships have the same multiset of participating entity types (which implies that they have the same arity). This is too strong a restriction, as seen in the next example (Fig. 8).

Here, ordered and ordline are two equivalent relationships, linking a product to its destination. In $V7$ the latter is the customer, while in $V8$ it is the order placed by the customer. Section VII will show how our methodology integrates the two relationships despite the difference in the participating entity types.

IV. A Model for the Description of View Correspondences

Only users and the DBA definitely "know" whether and to what extent views are (partially) describing the same real-world objects, and how to match the descriptions. Of course, it is possible to build automated tools to help in identifying similarities, based on matching of names and descriptions. Simple tools use a data dictionary to find homonyms and synonyms. More sophisticated tools analyze names and structures to evaluate a degree of similarity, from which they derive their conclusions [12], [11], [9]. In any case, the DBA has to confirm, or deny, the correspondences proposed by the tool.

We assume that knowledge of view correspondences is provided by users or the DBA to the view integration tool. That knowledge may be expressed using either declarative or procedural statements. The latter approach is found in [26] and [27], in the context of database integration. Following Motro, the user specifies his global schema by defining how it is built from the set of schemas of existing databases. The mapping specification uses a set of restructuring primitives (later called schema editing operations). This approach puts the burden of mapping definitions on users, and it is far from evident that they will be able to master the complexity of programming mappings with these operators.

We prefer to limit the users' task to point, in each view, at elements that describe the same real-world objects, and to identify what correspondences hold in between (in terms of their structure and instances). The declarative approach allows this. Consequently, the mapping between views is described by users as a set of declarative statements, hereinafter called correspondence assertions.

A correspondence assertion is a declarative statement, provided by users, asserting that the semantics of some piece of data structure in one view is somehow related to the semantics of some piece of data structure in another view.

There will be several kinds of correspondence assertions, depending on involved data structures and on the nature of the correspondence. Their definitions are given in this section. All of them express the knowledge "this is related to this." Before we go into details about assertions, we must precisely define what is meant by the semantics of some piece of data structure.

A. About the Underlying Semantics of Data Constructs

In the ER approach, representation of a real-world object depends on the level of perception one has in considering the object: Basically, it will be modeled as an entity if perceived as self existing, as a relationship if perceived as a link between entities, and as an attribute if perceived as a property of some other object. Correspondence assertions that have been used so far in the ER framework are limited to the definition of interrelationships between populations (or domains) of object types at the same level of perception. These assertions relate entity (relationship) types to entity (relationship) types in terms of the object classes they describe, and attributes to attributes in terms of their domains and instantiation. In [8], for instance, attribute comparisons are deeply investigated. The same paper also introduces the term *real-world state* of an object class A, denoted RWS(A), defined as the set of real-world instances of object class A at a given moment in time.

We also use the RWS idea to define the semantics of an entity type, of a relationship type and of an attribute. However, our definitions will allow us to compare object types at different levels of perception (entity type versus attribute, for instance). Moreover, we extend the scope of the RWS idea to also apply to paths, a construct implicit in the ER approach that has not received the attention it deserves, as shown below.

Real-World State of an Entity Type (Definition D1): Let E be an entity type; the real-world state of E, RWS(E), is the set of real-world objects that occurrences of E represent.

There is a 1:1 mapping between the population of E and RWS(E).

Real-World State of a Relationship Type (Definition D2): Let R be a relationship type, linking entity types $E_1, E_2, \cdots, E_n$; the real-world state of R, RWS(R), is the bag of real-world object tuples $[\langle o_1, o_2, \cdots, o_n \rangle]$, such that $\forall_i, o_i \in$ RWS(E_i) and the objects in a tuple are linked by a real-world association represented by R.

Square brackets denote multisets.

In definition D2, we are concerned with multisets of tuples, not sets, as the ERC+ model allows the same set of entities

to participate in several occurrences of the same relationship type. There is a 1:1 mapping between the population of R and RWS(R).

Real-World State of an Attribute (Definition D3): Let A be an attribute; the real-world state of A, RWS(A), is the set of real-world objects that the values of A represent.

RWS(A) is not necessarily a deterministic concept. For instance, let us consider a "month" attribute. There are several possible interpretations of RWS(month): either as the set of integers $\{1, \cdots, 12\}$, or as the set of character strings {"January",$\cdots$,"December"}, or as the set of months in the common language sense, irrespective of the coding of the associated value. On the other hand, a complex attribute describing the manager of a project will have its RWS interpreted as the set of persons who are actually managing some project.

Generally speaking, one could think of a lexical RWS, where the interpretation is related to the coding of the attribute values, and a nonlexical RWS, where the interpretation abstracts from the actual coding.

The multiplicity of RWS interpretations is not a problem insofar as, once the concept is used in an correspondence assertion, its meaning is uniquely determined within that context.

There is a total surjective function from the values of A to RWS(A).

As stated above, we shall now define the concept of a path, needed for proper view integration, and the associated real-world state. As a path is a sequence of links, we first define links.

Link (Definition D4): Let X and Y be elements in a schema (entity type, relationship type, and/or attribute), then X–Y is a link if

Y is an attribute of X (or vice-versa); then X-Y is called an attribute link;
or X is an entity type bound by the relationship type Y (or vice-versa); then X–Y is called a role link. In case of a cyclic relationship type, the name of the role is required and written above the hyphen.

Path (Definition D5): Let $X_1, X_2, \cdots, X_n$ be elements in a schema (entity types, relationship types, and/or attributes) such that $\forall_i \in \{1, 2, \cdots, n-1\}$, X_i is linked to X_{i+1}, either by an attribute link or by a role link, then X_1–X_2–$\cdots$–X_n is a path.

Real-World State of a Path (Definition D6): The real-world state of the X_1–X_2–$\cdots$–X_n path, RWS(X_1–X_2-$\cdots$–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), and there exist objects $o_2, o_3, \cdots, o_{n-1}$ such that $\forall i \in \{1, 2, \cdots, n-1\}$, $o_i \in$ 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.

B. Element Correspondence Assertions

The RWS concept allows for definition of correspondence assertions. The basic ones, given below, deal with object classes and refer to usual set relationships (equivalence, inclusion, intersection, and exclusion).

Let X_1, X_2 be two elements (entity type, relationship type, or attribute), X_1 from view V_1, X_2 from view V_2, and o_1, o_2 two real-world objects, such that $o_1 \in$RWS(X_1), $o_2 \in$ RWS(X_2).

Element Equivalence Assertion (Definition D7): The assertion that X_1 and X_2 are equivalent, expressed by the statement

$$X_1 \equiv X_2$$

states that at any time either

1) RWS(X_1) = RWS(X_2), or
2) there is a total bijective function f: RWS(X_1) $\leftrightarrow$ RWS(X_2), such that $o_2 = f(o_1)$ iff o_1 and o_2 have the same semantics.

Element Inclusion Assertion (Definition D8): The assertion that X_1 contains X_2, expressed by the statement

$$X_1 \supseteq X_2$$

states that at any time either

1) RWS(X_1) $\supseteq$ RWS(X_2) is true, or
2) there is a total injective function f: RWS(X_2) $\rightarrow$ RWS(X_1), such that $o_2 = f(o_1)$ iff o_1 and o_2 have the same semantics.

Element Intersection Assertion (Definition D9): The assertion that X_1 and X_2 intersect, expressed by the statement

$$X_1 \cap X2,$$

states that at some time either

1) RWS(X_1)$\cap$ RWS(X_2) $\neq \emptyset$ is true, or
2) there is a partial injective function f: RWS(X_2) $\rightarrow$ RWS(X_1), such that $o_2 = f(o_1)$ iff o_1 and o_2 have the same semantics. □

Element Exclusion Assertion (Definition D10): The assertion that X_1 and X_2 are disjoint, expressed by the statement

$$X_1 \neq X_2,$$

states that at any time

1) RWS(X_1)$\cap$ RWS(X_2) = $\emptyset$ is true, and
2) there is no function f: RWS(X_2) $\rightarrow$ RWS(X_1), such that $o_2 = f(o_1)$ iff o_1 and o_2 have the same semantics. □

This last assertion is meaningful (useful) iff there is a corresponding element X_3, in some other view V_3, such that $X_3 \supseteq X_1$ and $X_3 \supseteq X_2$.

In each case, condition a) is intended to cope with the situation in which X_1 and X_2 are at the same level of perception (both entity types, or both relationship types, or both attributes). It is worthwhile noting that we do not require that corresponding elements have the same identifiers (as in [8], for instance). This is consistent with the structural object orientation of the ERC+ model and will apply as well for object-oriented models.

Conditions b) are intended to extend the scope of integration to heterogeneous perceptions of the same objects, allowing for comparisons of attributes with entity types, and so on.

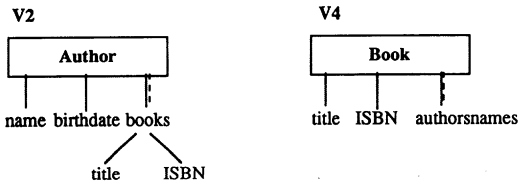

Fig. 9.

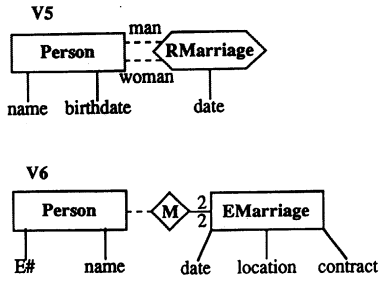

Fig. 10.

Example 1: Entity/Attribute Correspondence: Our first example to illustrate the usage of correspondence assertions refers to the library information system discussed in Section III.

Suppose the diagrams shown in Fig. 9 are part of two views to be integrated:
RWS(Author) is the set of authors in the library. Clearly, from a lexical point of view, RWS(authorsnames) is equal to RWS(name), the set of all authors' names. However, it is equally true to state that, from a nonlexical point of view, RWS(authorsnames) is equal to RWS(Author): the names of the authors, authorsnames, is nothing but a designation of the corresponding authors. Therefore, the following assertions are both valid:

-Author•name ≡ Book•authorsnames (thanks to condition a)
-Author ≡ Book•authorsnames (thanks to condition b).

The first assertion obviously compares lexical objects (character strings), while the second one compares nonlexical objects. Which interpretation of RWS(authorsnames) applies is determined by the other term in the assertion.

Example 2: Entity/relationship correspondence: Our second example refers to the marriage views as discussed in Section III (see Fig. 10).

If the two views see the same set of persons, it is obvious that the RWS of Person in the two views are equal. For the RWS of RMarriage and EMarriage, it is a bit less evident: RWS(RMarriage) is the multiset of married couples and RWS(EMarriage) might be something else—for instance, the set of marriage contracts. Nevertheless, there is a 1:1 mapping between the two RWS's. Thanks to condition b), it is therefore correct to assert: RMarriage ≡ EMarriage.

C. Correspondences Between Attributes of Corresponding Elements

The above correspondences between RWS are asserted independently from what information the views keep on the objects: There is no obligation for the views to describe corresponding objects with the same set of properties (attributes, in ER terms). However, if it is the case that corresponding elements are described, in the different views, with the same, or similar, attributes, the integrator should know about these similarities. This knowledge is needed to produce a nonredundant integrated schema, in which common attributes are integrated into a single one.

To that purpose, we introduce assertions about attribute correspondences, to be used within the context of an element correspondence assertion, as defined below.

Example: Consider the above Marriage views ($V5$–$V6$). Name of Person and date of Marriage will be identified in the two views as corresponding attributes. Equivalence assertions (in which the left term refers to $V5$, the right one to $V6$)

Person•name = Person•name
RMarriage•date = EMarriagedate

will specify that, whenever two corresponding persons (marriages) are considered, the name (date) attribute in $V5$ holds the same value as the name (date) attribute in $V6$.

Value equality is not the only possible case. Two attributes may correspond and nevertheless represent the same property in different ways.

- The coding scheme may differ: salaries in US$ versus the same salaries in another currency.
- One view may carry only a partial record of the property, or a résumé (like a statistical aggregate). For instance, salary in one view versus salary of less than 100 000 in the other view (where perception of salaries may be restricted for privacy purposes). Children (multivalued in one view) versus number_of_children in the other view is an example of résumé.
- Both views may carry only a partial record of the property, with the partial records overlapping or being disjoint: children in both views, but restricted to boys on one side and to girls on the other side, is an example of disjointness.

The notion of partial record for a property refers to the fact that only some of the possible values of this property are of interest to the user. Applied to a monovalued attribute, partial recording implies that the attribute is described as optional (to accept "no value" when the actual value is outside the domain of interest). More facets to be considered have been suggested in [8], i.e., integrity and security constraints, and allowable operations. We will not discuss these additional facets, as they do not change the nature of the problem.

In our model, assertions about corresponding attributes of corresponding elements X and Y are stated as part of a correspondence assertion between X and Y, using a "with corresponding attributes" (WCA) clause. This WCA clause is defined below. As in [8], the notation Values(A), where A is an attribute, is used to refer to the set of values of A in all the actual occurrences (or values) of the parent element. Values(A) is a subset of the domain associated with A.

Corresponding Attributes Assertions (Definition D11): Let X_1 ⟨cor⟩ X_2, be an element correspondence assertion, with (o_1, o_2) being a pair of corresponding objects: $o_1 \in \text{RWS}(X_1), o_2 \in \text{RWS}(X_2)$.

Let e_1, e_2 be the occurrences representing o_1 and o_2 in the database. Finally, let $A_{11}, A_{12}, \cdots, A_{1n}$ be attributes of X_1, and $A_{21}, A_{22}, \cdots, A_{2n}$ be attributes of X_2 (if X_1 or X_2 is a simple attribute, it is implicitly considered here as having itself as unique component). Then,

$X_1\langle\text{cor}\rangle X_2$ with corresponding attributes
$\text{attcor}_1(A_{11}, A_{21})$, $\text{attcor}_2(A_{12}, A_{22}), \cdots,$
$\text{attcor}_n(A_{1n}, A_{2n})$

is also a correspondence assertion that states that $X_1\langle\text{cor}\rangle X_2$ is true, and for each $\text{attcor}_i(A_{1i}, A_{2i})$,

- If $\text{attcor}_i(A_{1i}, A_{2i})$ is $A_{1i} = A_{2i}$, then for any o_1, o_2 pair, $e_1 \bullet A_{1i} = e_2 \bullet A_{2i}$;
- If $\text{attcor}_i(A_{1i}, A_{2i})$ is $A_{2i} = f_i(A_{1i})$, then f_i is a surjective function (explicitly defined by the DBA) from Values(A_{1i}) onto Values(A_{2i}), such that at any time for any o_1, o_2 pair, $e_2 \bullet A_{2i} = f_i(e_1 \bullet A_{1i})$. In particular, the f_i function may be bijective, or it may be a subsetting function such that $A_{1i} \supseteq A_{2i}$. Its explicit definition is required for its implementation in the views to integrated schema mappings.
- If $\text{attcor}_i(A_{1i}, A_{2i})$ is $A_{1i} \cup A_{2i}$, then it is possible that for some o_1, o_2 pair, $e_1 \bullet A_{1i} \cap e_2 \bullet A_{2i} \neq \emptyset$ (for a monovalued attribute this reduces to $e_1 \bullet A_{1i} = e_2 \bullet A_{2i}$).
- If $\text{attcor}_i(A_{1i}, A_{2i})$ is $A_{1i} \neq A_{2i}$, then for any o_1, o_2 pair, $e_1 \bullet A_{1i} \cap e_2 \bullet A_{2i} = \emptyset$, but it is possible to define a domain D such that at any time $D \supseteq$ Values(A_{1i}) and $D \supseteq$ Values(A_{2i}). □

Examples: Assume there are two Person entity types asserted to be equivalent:

$\text{Person}_1 \equiv \text{Person}_2$ with corresponding attribute
name = name.

Other assertions might hold between attributes of Person_1 (name, salary_in_US$, number_of_children, children_under_16, boys) and attributes of Person_2 (name, salary_in_FF, children, girls):

- salary_in_US$ = FF_to_US$_exchange (salary_in_FF) FF_to_US$_exchange is a function transforming an amount expressed in French francs into the equivalent (at some specified day) amount in US dollars. The inverse function may also be specified: US$_to_FF_exchange (salary_in_US$) = salary_in_FF;
- number_of_children = count(children), where children is a multivalued attribute;
- children_under_16 = select16(children) Both are multivalued attributes, but only children under 16 are recorded in the first view; select16 is a function extracting from children only those values where age is under 16;
- boys ≠ girls both are multivalued attributes and have the same structure, but boys records only male children while girls records only female children;

Attribute correspondences should not contradict correspondences asserted for their parent elements [8].

D. Path Correspondence Assertions

The analysis of views interrelationships also calls for the identification of correspondences among paths in the views. Refer to the first two views, $V1$ and $V2$, of the library information system. If we suppose that the two views see exactly the same objects (books and authors), the element correspondence assertions between $V1$ and $V2$ are (in each assertion, the left term refers to $V2$, the right term to $V2$):

Book ≡ Author • books with corresponding attributes:
title = title, ISBN =ISBN.
Book•authors ≡ Author with corresponding attributes:
name = name, birthdate = birthdate.

These two assertions will generate in the integrated schema (IS) two entity types, Book and Author. Nothing in the above correspondence assertions states the fact—obvious to a human but not to a program—that each time a Book **b** has an author **a** in $V1$, the corresponding **a** Author has this **b** Book in $V2$. The integrator will generate in IS two different relationship types between Author and Book, each one expressing one of the two links: Book-authors in $V1$ and Author-books in $V2$.

In order to allow the integrator to integrate those two links into a unique relationship type, the DBA has to state that each link is the reverse of the other. In our methodology, the DBA will define the following path correspondence assertion (which is explained below):

$$\text{Book} - \text{authors} \equiv \text{books} - \text{Author}.$$

Path Equivalence Assertion (Definition D12): Let E_1–E_2–$\cdots$–E_n be a path within view V, and F_1–F_2–$\cdots$–F_p be a path within view V', such that there is an assertion relating E_1 to F_1 (via a function f: RWS(E_1) → RWS(F_1)) and an assertion relating E_n to F_p (via a function f': RWS(E_n) → RWS(F_p)).

Let RWS'(E_1) be the subset of RWS(E_1) defined by its restriction to E_1 objects, which are involved in the asserted correspondence with F_1 objects. Let RWS'(F_1), RWS'(E_n) and RWS'(F_p) be similar restrictions of the corresponding RWS.

Let RWS'(E_1-E_2-$\cdots$-E_n) be the subbag of RWS(E_1-E_2-$\cdots$-E_n), defined by its restriction to object pairs in RWS'(E_1)× RWS'(E_n), and similarly for RWS'(F_1–F_2–$\cdots$–F_p). Finally, let $\langle e_1, e_n\rangle \in$ RWS'(E_1–E_2–$\cdots$–E_n) and $\langle f_1, f_p\rangle \in$ RWS'(F_1–F_2–$\cdots$–F_p).

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

$$E_1\text{-}E_2\text{-}\cdots\text{-}E_n \equiv F_1\text{-}F_2\text{-}\cdots\text{-}F_p$$

states that at any time, either

1) RWS'(E_1–E_2–$\cdots$–E_n) = RWS'(F_1–F_2–$\cdots$–F_p), or
2) there is a total bijective function g RWS'(E_1–E_2–$\cdots$–E_n) ↔ RWS'(F_1–F_2–$\cdots$–F_p) such that $\langle F_1, f_p\rangle = g(\langle e_1, e_n\rangle)$ iff $f_1 = f(e_1)$, $f_p = f'(E_n)$ and the F_1–f_p and the e_1–e_n paths have the same semantics. □

The other assertions are as follows:

- E_1–E_2–$\cdots$–$E_n \supseteq F_1$–F_2–$\cdots$–F_p.
- E_1–E_2–$\cdots$–$E_n \cap F_1$–F_2–$\cdots$–F_p.
- E_1–E_2–$\cdots$–$E_n \neq F_1$–F_2–$\cdots$–F_p.

These may be readily defined in the same way as element assertions.

The assertions we have defined above are not the only ones that may relate two views. As an example, assertions may involve aggregation functions: An attribute in one view may represent an average over several attributes in another view; an entity may correspond to a set of entities (like in the well-known convoy example, where a convoy is a set of ships). This type of correspondence will be dealt in future work. We also defer dealing with generalization links.

V. Rules for View Integration

Once correspondence assertions between views have been stated, the integration algorithm can proceed. Analyzing each correspondence assertion, it generates one or several constructs in the integrated schema, in accordance with the appropriate integration rule. These rules are described hereinafter, while the algorithm is described in the next section.

The background to our rule definitions consists of two basic principles, which are model independent:

1) The scope of integration rules should cover both objects and links integration. As these two concepts are present in any data model (semantic, functional, ER, object-oriented, etc.), their integration rules should be translatable from one model to another.
2) Whenever conflicting representations exist in different views, the integrated schema will hold the least restrictive representation. This allows us to support all other representations, which can be derived through restrictive mappings.

Conflict resolution is not just a matter of schema conforming. Usage of different modeling concepts usually implies different constraints on objects or links in the schema. In the ERC+ framework, entities, relationships, and attributes bear different existence dependencies. Entities are free from such dependencies (unless constrained by attached roles). Relationships depend on participating entities (two or more), and attributes depend on their (unique) parent element. Therefore, entity type is the preferred integrated concept in case of conflict. Additional links, and the associated cardinalities, will bear the needed existence dependencies.

Example: Views $V1$ and $V2$ (library information system) carry an entity type/attribute conflict on representation of books. Integration generates an entity type Book in the integrated schema ($V3$), which is complemented with a relationship type to link it to Author. This new relationship type represents the two links, Book-authors ($V1$) and Author-books ($V2$). Both views state that there does not exist any book without at least one author. In $V1$, this is because authors is a mandatory attribute of Book. In $V2$, books, as an attribute, exists only if its parent element, Author, exists. This existence dependency of book is expressed in $V3$ by making mandatory the role of Book.

Applied to the ERC+ model, the above two principles led us to define the following six integration rules:

- The first one governs integration of element correspondence assertions, except for the case where the two corresponding elements are both attributes.
- The second and third ones cope with path correspondence assertions.
- The fourth and fifth ones govern the case of correspondence assertions between attributes.
- The last rule applies to integration of elements that are not involved in correspondence assertions.

A special case when two attributes are asserted to correspond is made, due to the fact that the ER approach does not allow for attribute sharing: An attribute belongs to one and only one parent element. Applying the first rule to the assertion "attribute A of X is the same as attribute B of Y" (without X and Y corresponding to each other) would result in generating an attribute AB, son of both X and Y, which contradicts model rules.

A. Integration of Element Correspondence Assertions

Our first rule applies to correspondence assertions between elements as described in Definition D7. The elements may be entity types, relationship types or attributes, but not two attributes. This last case is discussed in Section V–C. Following our basic principles, integration rule 1 states that:

1) corresponding elements of the same modeling concept (two entity types or two relationship types) are integrated into a similar element (respectively, an entity type or a relationship type), and
2) corresponding elements of different modeling concepts (entity types, relationship types, and attributes) are integrated into an entity type.

If the existence of one (or both) of the elements to be integrated depends upon other elements, integration rule 2 or 6 will generate, for each existence dependency, a link whose cardinalities will express that dependency.

Integration Rule 1: Elements Integration Rule: Let X_1, X_2 be two elements in two views, $X_1 \in V_1, X_2 \in V_2$, such that $X_1 \equiv X_2$.

If we denote by X the element in the integrated schema resulting from the integration of X_1, X_2, then:

- If X_1 and X_2 are not of the same type, X is an entity type;
- If X_1 and X_2 are of the same type but are not attributes, X is of the same type as X_1, X_2. □

This rule considers only equivalence assertions. A detailed analysis of how the other element assertions are dealt with (compared to equivalence) may be found in [7] or [8]. Our approach would be similar to Jardine's one.

B. Integration of Links

Entity types are the only elements whose integration may not require additional integration of links. On the contrary, relationship type and attributes integration calls for integration of role links and attribute links, which express their existence dependencies. Correspondences between links are asserted as path correspondences, with each path consisting of only two elements. According to Definition D12, these elements are asserted as corresponding. Obviously, integration of the links in between generates a path connecting the IS elements that

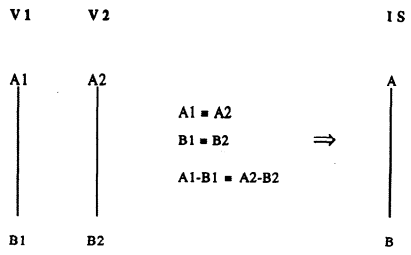

Fig. 11.

result from integration of the link ends in the views illustrated in Fig. 11.

The IS path is built to conform to the following ER axioms:

- If A or B is an attribute, A–B is an attribute link.
- If A is an entity type and B a relationship type, A–B is a role link.
- If A and B are two entity types, a new relationship type—say, X—is inserted, and the A–B link becomes an A–X–B path. X acts as a pure link relationship: It has no attributes, no significant name, and no other semantics that "link A and B." This case occurs when A (or B) results from the integration of elements of different type (Rule 1).

A and B cannot both be relationship types (integration rule 1). As elements and links are equivalent, cardinalities of the two links are necessarily the same, and cardinalities of the integrated link are set to the same values.

Integration Rule 2: Links Integration Rule: Let A_1 and B_1 be two linked elements in view V_1, A_2, and B_2 be two linked elements in view V_2, and the following correspondence assertions:

$$A_1 \equiv A_2$$
$$B_1 \equiv B_2$$
$$A_1\text{-}B_1 \equiv A_2\text{-}B_2.$$

Let A be the integrated element in IS corresponding to A_1 and A_2, let B be the integrated element in IS corresponding to B_1 and B_2, then the integration of A_1–B_1 and A_2–B_2 links is:

- a role link, if A and B are an entity type and a relationship type;
- an attribute link, if A and B are any element and an attribute;
- a link relationship type with its two roles (standard name, no attribute) if A and B are two entity types.

The cardinalities of the integrated link, or path, are:

$$\text{cardmin}(A) = \text{cardmin}(A_1) = \text{cardmin}(A_2)$$
$$\text{cardmax}(A) = \text{cardmax}(A_1) = \text{cardmax}(A_2)$$
$$\text{cardmin}(B) = \text{cardmin}(B_1) = \text{cardmin}(B_2)$$
$$\text{cardmax}(B) = \text{cardmax}(B_1) = \text{cardmax}(B_2).$$

□

C. Integration of Composite Paths

Two different cases may involve a composite path in a correspondence assertion, i.e.,

- when a path corresponds with a link, or
- when two paths, neither one a direct link, correspond with each other.

Let us first consider the case where one of the two paths is a direct link. For instance,

$$E_1\text{-}E_n \equiv F_1\text{-}F_2\text{-}\cdots\text{-}F_{p-1}\text{-}F_p.$$

In this case, the composite path will be integrated into IS, but not the direct link, which would be redundant as it is deducible from the composite path.

Let us assume that $F_1, F_2, \cdots, F_{p-1}, F_p$ elements have been integrated as $F'_1, F'_2, \cdots, F'_{p-1}, F'_p$ into IS.

The integration of the composite path proceeds step by step, considering each link in the path. For each $i = 1, 2, \cdots, p-1$ either a role link, an attribute link, or a link relationship type with its two roles is created in IS between F'_i and F'_{i+1}, according to the concepts modeling F'_i and F'_{i+1}, as in Rule 2.

Cardinalities of the integrated path are kept unchanged.

Let us now consider the case where neither of the two paths is a direct one (nor may they be decomposed into corresponding direct links). This means that the two views record alternative ways to go from the same (or corresponding) source information to the same (or corresponding) target information.

If the two paths do not have common elements, which is the case here, they both have to be recorded in IS. The only thing the integrator can do to preserve database consistency would be to add an integrity constraint to IS, stating that, for the same source element, the element reached through one path must be the same as the element reached through the other path.

Integration Rule 3: Paths Integration Rule: Let E_1, E_2, $\cdots$ E_n be elements in view V_1.

Let $F_1, F_2, \cdots F_p$ be elements in view V_2, with the following correspondence assertion:

$$E_1 \equiv F_1.$$

Let G_1 be the integrated element in IS corresponding to E_1 and F_1. Then:

- The correspondence assertion between a link and a path,

$$E_1\text{-}E_2 \equiv F_1\text{-}F_2\text{-}\cdots\text{-}F_p$$

  with $E_2 \equiv F_p$ generating G_p in IS, generates in IS a path G_1–F'_2–$\cdots$–F'_{p-1}–G_p, where $F'_2, \cdots, F'_{p-1}$ are elements of IS corresponding to $F_2, \cdots, F_{p-1}$, and each link of the path is created according to the concepts modeling the linked elements, as in Rule 2.
- The correspondence assertion between two composite paths,

$$E_1\text{-}E_2\text{-}\cdots\text{-}E_n \equiv F_1\text{-}F_2\text{-}\cdots\text{-}F_p \quad n > 2, p > 2,$$

  with $E_n \equiv F_p$ generating G_p in IS generates in IS two paths and an integrity constraint. The two paths are:

$$G_1\text{-}E'_2\text{-}\cdots\text{-}E'_{n-1}\text{-}G_p$$
$$G_1\text{-}F'_2\text{-}\cdots\text{-}F'_{p-1}\text{-}G_p,$$

where $E'_2, \cdots, E'_{n-1}$ are elements of IS corresponding to $E_2, \cdots, E_{n-1}$, and $F'_2, \cdots, F'_{p-1}$ are elements corresponding to $F_2, \cdots, F_{p-1}$, and each link of the paths is created according to the modeling concepts of the linked elements, as in rule 2.
The integrity constraint states that the two paths link the same occurrences. □

D. Integration of Attributes

Attribute integration, as relationship type integration, requires the integration of elements and links. As we have seen above, integration of two corresponding attributes is not managed by Rule 1 because of the peculiarity of the existence dependency of attributes: They depend on exactly one object.

Attribute integration is easily managed if the corresponding attributes belong to objects that are themselves involved in a correspondence assertion, i.e. the attribute and the attribute link correspondence assertions are described by a WCA clause (Definition D11).

The opposite situation is that of two corresponding attributes that are not related to any couple of corresponding elements in a path correspondence assertion. There is only one correspondence assertion $A \equiv B$ involving those attributes. In that case, A and B cannot be integrated into an attribute in the integrated schema. This attribute correspondence only expresses a constraint on the domains of the two attributes, without any immediate impact on the process of building the integrated schema. The integrator will just keep track of this kind of correspondence, as it might become useful in a subsequent integration step, where these attributes would correspond to an entity or relationship type in some other view. If no more views are to be integrated, the only thing to do is to generate in the integrated schema an integrity constraint stating the correspondence between the set of values of A and B.

An intermediate situation is when corresponding attributes relate to some corresponding objects, different from their direct parents (otherwise, it would be part of a WCA clause). In other words, there is a path correspondence, where the paths connect the corresponding attributes to some elements asserted to correspond to each other. In this case, the correspondence assertions between the attributes and the paths are handled together with the same technique as for path integration. One or two attributes are created in the integrated schema, according to the fact that one path can be deduced from the other or not.

Rules for managing the two cases where integration is possible are formally defined below.

Integration of Attributes of Corresponding Elements: An equivalence between two elements generates an element. The set of its attributes in the integrated schema will consist of one attribute for each pair of corresponding attributes, plus one attribute for each attribute belonging to only one view (with no corresponding attribute in any other view). As entities are equivalent, cardinalities of integrated attributes are the strongest.

Integration Rule 4: Integration of Attributes of Corresponding Elements: Let E_1 be an element in view V_1, E_2 an element in view V_2, with the following correspondence assertion:

$$E_1 \equiv E_2,$$

with corresponding attributes

$$A_{11} = A_{21}, A_{12} = A_{22}, \cdots, A_{1n} = A_{2n}.$$

Then, the integrated element E in IS corresponding to E_1 and E_2 will have:

- an attribute A_i for each attribute correspondence $A_{1i} = A_{2i}$.
  A_i's domain and cardinalities are equal to those of A_{1i} and A_{2i}, i.e.,
  $$\text{cardmin}(A_i) = \text{cardmin}(A_{1i}) = \text{cardmin}(A_{2i})$$
  $$\text{cardmax}(A_i) = \text{cardmax}(A_{1i}) = \text{cardmax}(A_{2i}).$$
- an attribute B'_j for each attribute B_j of E_1 (or of E_2) that has no correspondent.
  $B'j$'s domain and cardinalities are equal to Bj's ones:
  $$\text{cardmin}(B'_j) = \text{cardmin}(B_j)$$
  $$\text{cardmax}(B'_j) = \text{cardmax}(B_j).$$

Integration of Attributes of Noncorresponding Elements: The last case to be considered is the one where two corresponding attributes terminate corresponding paths. The difference with Rule 3 about paths integration is that the attributes have not yet been incorporated into the integrated schema (as stated in Section V–A), which makes the above rule not applicable.

The purpose of the path correspondence, combined with the attribute correspondence, is to avoid attribute duplication in IS. Indeed, the semantics of these correspondences states that two different paths lead to the same information. Therefore, only one of the two paths should be implemented in IS, and the attribute inserted at the end of this path. Which path is transferred into IS is determined with the same approach as for path integration. If one of the paths is a direct link, the other path is chosen. If both are composite paths, the two have to be kept, with their terminal attributes, and it is not possible for the integrator to avoid the redundancy. An integrity constraint is added to the integrated schema.

Integration Rule 5: Attributes with Path Integration Rule: Let $E_1, E_2, \cdots, E_n$ be elements and A an attribute in view V_1; and let $F_1, F_2, \cdots, F_p$ be elements and B an attribute in view V_2, with the following correspondence assertions:

$$E_1 \equiv F_1$$
$$A \equiv B$$

Let G_1 be the integrated element in IS corresponding to E_1 and F_1; then:

- The correspondence assertion
  $$E_1\text{-}A \equiv F_1\text{-}F_2 \cdots\text{-}F_p\text{-}B$$
  generates in IS an attribute B', which is the attribute of F'_p where F'_p is the element corresponding to F_p. The domain and cardinalities of B' are the same as those of B.

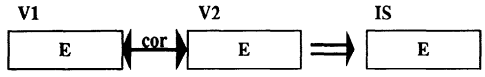

Fig. 12.

- The correspondence assertion

$$E_1\text{-}E_2\text{-}\cdots\text{-}E_n\text{-}A \equiv F_1\text{-}F_2\text{-}\cdots\text{-}Fp\text{-}B \quad n \geq 2, p \geq 2$$

generates in IS two attributes and an integrity constraint. The attributes are: A', which is an attribute of E'_n, where E'_n is the element corresponding to E_n; and B', which is an attribute of F'_p, where F'_p is the element corresponding to F_p.

Domains and cardinalities of A' and B' are, respectively, the same as those of A and B.

The integrity constraint states that the two paths link the same values.

E. Add Rule for Elements and Links without Correspondent

This last rule takes care of elements and links that exist in only one view. It has already been implicitly used in Rule 4, where attributes that are defined in only one view are added to the integrated schema.

Integration Rule 6: Add Rule: Any element (entity type or relationship type) that exists in a view and has no corresponding element in any other view is added to the integrated schema with all its attributes without modification.

Let X_1–Y_1 be a link (role or attribute link) that exists in view $V1$ and has no corresponding link nor path in view $V2$. Let X and Y be the elements of IS corresponding to X_1 and Y_1. Then, a link or a link relationship type X–Y is added to IS, according to the modeling concepts of X and Y.

Cardinalities of X–Y are defined as follows:

If X_1 is equivalent to $X (X_1 \equiv X)$, then

$$\text{cardmin}(X) = \text{cardmin}(X_1)$$
$$\text{cardmax}(X) = \text{cardmax}(X_1).$$

If not (X_1 is only a subset of X), then

$$\text{cardmin}(X) = 0$$
$$\text{cardmax}(X) = \text{cardmax}(X_1).$$

Cardinalities of Y are defined in the same way.

F. Examples

The following diagrams sketch how integration rules are applied to usual cases. Cardinalities of links are not shown.

Case 1) Equivalence of Two Entity Types: An equivalence between two entity types generates an entity type (Rule 1 plus Rule 4 for attributes). (See Fig. 12.)

In every case, each time Rule 1 is run, Rule 4 is also activated in order to add attributes to integrated elements. Hereinafter we will not repeat it.

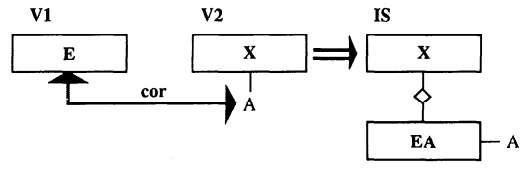

Fig. 13.

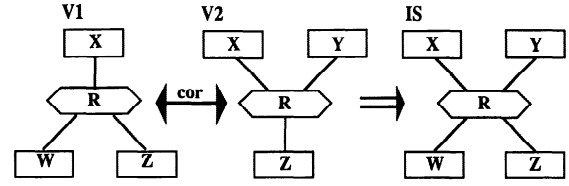

Fig. 14.

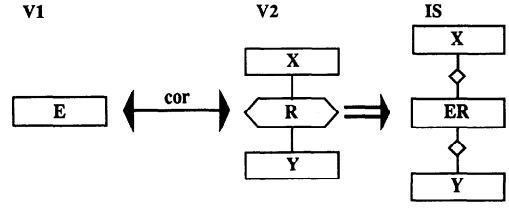

Fig. 15.

Case 2) Equivalence of an Entity Type and an Attribute: An equivalence between an entity type E and an attribute A, of some element X, generates an entity type, say EA (Rule 1). Rule 6 generates for the X–A attribute link a relationship type linking EA to the entity type representing the element to which the original attribute is attached in the view (X in our example). The name for the generated relationship type may be automatically generated by the integrator, or be specified by the DBA. (See Fig. 13.)

Case 3) Equivalence of Two Relationship Types: In this case, the correspondence assertions are

$$R \equiv R,\ X \equiv X,\ Z \equiv Z$$
$$X\text{-}R \equiv X\text{-}R,\ Z\text{-}R \equiv Z\text{-}R.$$

(See Fig. 14.)

An equivalence between relationship types generates a relationship type (rule 1). Link integration (Rule 2) generates one role link for the two X–R links and one for the two Z–R links. Rule 6 adds role links without correspondent: Y–R and W–R. The integrated relationship type will therefore link all entity types resulting from the integration of the participating entity types in the views.

Case 4) Equivalence of an Entity Type and a RelationshipType: An equivalence between an entity type E and a relationship type R generates an entity type, say, ER (Rule 1). Rule 6 generates, for each role of R, a relationship type linking ER with the entity types resulting from the integration of the entity types participating in R (here, X and Y). (See Fig. 15.)

Case 5) Equivalence of a Relationship Type and an Attribute: An equivalence between an attribute A and a relationship type R generates an entity type, say, RA (Rule 1). Rule 6 generates

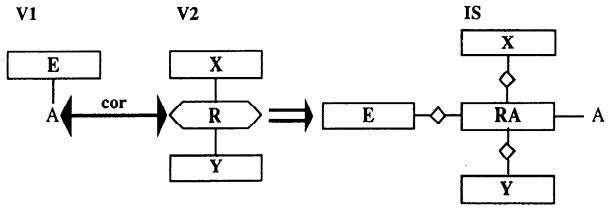

Fig. 16.

the same substitution for R as in the immediately previous case, plus a relationship type linking E and RA. This last relationship type results from adding the E–A attribute link of view $V1$ to the integrated schema.

(See Fig. 16.)

Other examples of more complex situations are described in Section VII. They will show how paths are integrated.

VI. View Integration Algorithm

Different algorithms may be designed to perform view integration based on the six rules we have defined in Section V. Certainly, algorithms will vary depending on which integration processing strategy is chosen: n-ary versus binary. N-ary strategies integrate n views in one shot. In this case, the integrator will have to sum up all correspondences involving the same object through all views. With that global knowledge, it will be able to decide which constructs to build in the integrated schema, and to generate the appropriate mappings between each view and the integrated schema.

Binary strategies integrate two views at a time. They are simpler but must be iterated over the remaining views before the integration process terminates.

Our integration rules implicitly assumed a binary strategy (they may easily be extended to n views at a time). Accordingly, in this section we propose a straightforward algorithm that integrates two views. The algorithm is split into several steps, shown in Sections A through D below. These steps first integrate corresponding elements that are not attributes. Then, corresponding paths are integrated. Finally, corresponding attributes are processed.

Once all the views have been integrated, one may want to refine the resulting integrated schema. Indeed, integration rules may have created constructs that can be simplified without semantic loss. Refinement may be performed either automatically or interacting with the DBA. Automatic use of a refinement rule is shown in Algorithm 2.

Comments are written in italics.

Algorithm 1) Integration of Two Views ($V1$ and $V2$): Input: $V1$, $V2$, and the correspondence assertions in between.

Output: an integrated schema IS and the $V1$–IS, $V2$–IS correspondences.

A. Deferring Attribute Correspondences

/ As we have seen in the previous section, attribute correspondences, other than those included in a "with corresponding attributes" clause of corresponding elements, must be processed after integration of the other elements and of the paths. The initial step of the algorithm is intended to isolate correspondences whose integration is to be deferred. /

Remove from the set of element correspondence assertions all attribute correspondences and put them aside; remove these attributes from the views (temporarily).

Remove from the set of path correspondence assertions all correspondences where the paths terminate on corresponding attributes, and put them aside.

B. Elements Integration

/ Phase 1 (Integrate Corresponding elements /):* For each element correspondence assertion $X_1\langle\text{cor}\rangle X_2$, do:

- Execute Rule 1 (element integration rule).
- Execute Rule 4 (integration of attributes) for the integrated element.
- Mark as already processed, in $V1$ and in $V2$, X_1, X_2, their attributes and their attribute links.
- Generate the correspondence assertions $X_1\langle\text{cor}\rangle X$ in $V1$–IS and $X_2\langle\text{cor}\rangle X$ in $V2$–IS.

enddo

/ Phase 2 (Add Noncorresponding Elements /):* For each $V1$ element and for each $V2$ element that has not been marked as processed in Phase 1, do:

- Execute Rule 6 (add rule).
- Mark this element as processed, and its attributes and its attribute links (in $V1$ or in $V2$).
- Generate the correspondence assertions in $V1$–IS or in $V2$–IS.

enddo

C. Path Integration

/ Phase 1 (Add Path Correspondence Assertions for Roles of Equivalent Relationship Types): When the DBA states that two relationship types are equivalent, and these relationship types link equivalent entity types, this implicitly defines equivalence between corresponding role links in the two views. This phase explicitly declares these implicit correspondences, so that they may be taken into account during the next phase /*

For each pair of corresponding equivalent relationship types, R_1 in $V1$, R_2 in $V2$, such that

$$R_1 \text{ links } E_1, F_1, \cdots, G_1, \quad R2 \text{ links } E_2, F_2, \cdots, H_2,$$
$$R_1 \equiv R_2, \quad E_1 \equiv E_2, \quad F_1 \equiv F_2, \quad \cdots.$$

do:

- Add to the set of path correspondence assertions the following assertions:

$$E_1\text{-}R_1 \equiv E_2\text{-}R_2, \; F_1\text{-}R_1 \equiv F_2\text{-}R_2, \; \cdots..$$

enddo

/ Phase 2 (Integrate Corresponding Links and Paths /):* For each path correspondence assertion $X_1\text{-}\cdots\text{-}Z_1\langle\text{cor}\rangle X_2\text{-}\cdots\text{-}Z_2$, do:

- Execute the appropriate integration rule: Rule 2 if two links are involved, Rule 3 if composite paths are involved.

- Mark as processed, in $V1$ and in $V2$, the corresponding links.
- Generate the path correspondence assertions in $V1$–IS and in $V2$–IS.

enddo

*/ *Phase 3 (Add Noncorresponding Links /*):* For each $V1$ link and for each $V2$ link that has not been marked as processed, do:

- Execute Rule 6 (add rule).
- Mark this link as processed (in $V2$ or in $V2$).
- Generate the path correspondence assertion in $V1$–IS or in $V2$–IS.

enddo

D. Integration of Attribute Correspondences

/ We now consider attribute and path correspondence assertions that have been put aside in Section VI-A /

For each attribute correspondence assertion, $A_1\langle \text{cor} \rangle A_2$, do:

- If there is a path correspondence assertion involving A_1 and A_2 then execute Rule 5 (attribute with path integration rule) else add both attributes, A_1 and A_2, to IS.
  endif
- Generate the path correspondence assertions in $V1$–IS and/or in $V2$–IS.
  enddo

end of Algorithm 1

Algorithm 2) Refinement of an Integrated Schema: Input: an integrated schema IS.

Output: an equivalent integrated schema IS′ and the IS–IS′correspondences.

/ Replace entity types, which are only bound to link relationship types, by ERC+ relationship types /

For each entity type E in IS such that all its roles are bound to link relationship types $R1, R2, \cdots, Rn$, whose cardinalities are 1:1, do:

- Substitute a new relationship type (say, R) for E together with all its roles and related relationship types. R links all entity types that were bound by link relationship types to E. R's attributes are the attributes of E.
- Generate the following correspondence assertions in IS–IS′:

$$E \equiv R, R1 \equiv R, R2 \equiv R, \cdots, Rn \equiv R.$$

enddo

end of Algorithm 2

VII. Examples Illustrating the View Integration Algorithm

This section is intended to illustrate the main aspects of our algorithm. The library information system shows how different modeling constructs (entity types and attributes) are integrated and the importance of links integration. The second one, about customers' orders, requires integration of a composite path and of corresponding attributes of noncorresponding elements. Last, we discuss a new example, taken from [8], which involves integration of an entity type and a relationship type, and refinement of the integrated schema.

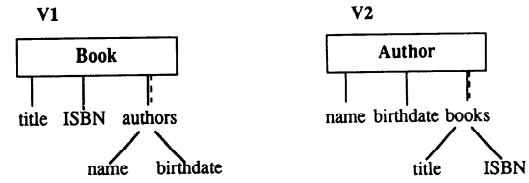

Fig. 17.

A. Example 1: Library Information System

Let us consider the views $V1$, $V2$ from Section III (see Fig. 17).

The set of correspondence assertions between $V1$ and $V2$ consists of two assertions about elements and one about paths:

Book ≡ Author • books — with corresponding attributes: title = title, ISBN = ISBN

Book •authors ≡ Author — with corresponding attributes: name = name, birthdate = birthdate

Book-authors ≡ books-Author

Step 6.1 of the integration algorithm is not required: There is no attribute correspondence assertion.

Step 6.2, phase 1, will start building the integrated schema by considering the first assertion: Book ≡ Author•books. As the two elements are not of the same type, an entity type, say Book, is generated in IS. Book has title and ISBN as attributes. The following correspondences are generated:

a) $V1$–IS: Book ≡ Book with corresponding attributes: title = title, ISBN = ISBN.
b) $V2$–IS: Author•books ≡ Book with corresponding attributes: title = title, ISBN = ISBN.

Similarly, the next correspondence will be dealt with: Book•authors ≡ Author. An entity type, say Author, is generated in IS, with name and birthdate as attributes. The following correspondences are generated:

a) $V1$–IS: Book•authors ≡ Author with corresponding attributes: name = name, birthdate = birthdate.
b) $V2$–IS: Author ≡ Author with corresponding attributes: name = name, birthdate = birthdate.

Step 6.2, phase 2: This phase is not needed. There is no noncorresponding element.

Step 6.3, phase 1: This phase is not needed. There is no relationship type.

Step 6.3, phase 2: Deals with the unique path correspondence:

Book-authors ≡ books-Author

Both Book-authors ($V1$) and books-Author ($V2$), are direct links. According to Rule 2, their integration consists in inserting a link in IS between Book and Author. As these are two entity types, the new link will conform to the following pattern: role-relationship type-role. Assuming the new relationship type

Fig. 18.

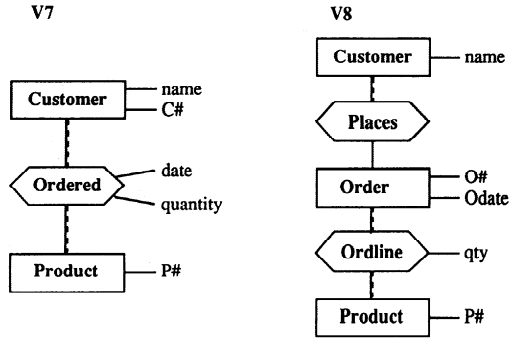

Fig. 19.

is named *BA*, two more correspondences are generated:

-$V1$-IS : Book - authors ≡ Book - BA - Author

-$V2$-IS : Author - books ≡ Author - BA - Book

Nothing else is left to be done. The integration has produced the integrated schema shown in Fig. 18.

B. Example 2: Customers' Orders

Let us now consider the customers' orders example. The views to be integrated are illustrated in Fig. 19.

The set of correspondence assertions between $V7$ and $V8$ consists of the following assertions:

Customer ≡ Customer with corresponding attributes: name=name

ordered ≡ ordline with corresponding attributes: quantity=qty

Product ≡ Product with corresponding attributes: $P\# = P\#$

ordered-date ≡ Order-Odate

ordered-date ≡ ordline-Order-Odate

Customer-ordered ≡ Customer-places-Order- ordline

Step 6.1 puts aside the two correspondences involving attributes date and Odate, and removes date and Odate from the views.

Phase 1 of step 6.2 then proceeds with the first three assertions, inserting into IS:

- an entity type Customer, with name and $C\#$ attributes,
- a relationship type, say oline, with an attribute quantity,
- an entity type Product, with the P# attribute,

and generating the appropriate correspondence assertions $V7$–IS and $V8$–IS.

Phase 2 of step 6.2 adds to IS the places and Order elements from $V8$:

- a relationship type places, without attributes,
- an entity type Order, with the $O\#$ attribute,

and generates additional correspondence assertions $V7$–IS and $V8$–IS.

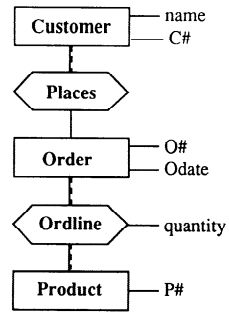

Fig. 20.

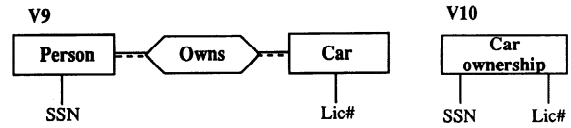

Fig. 21.

Phase 1 of step 6.3 adds the following path correspondence assertion between $V7$ and $V8$:

Product-ordered ≡ Product-ordline.

Phase 2 of step 6.3 integrates the following paths:

Customer-ordered ≡ Customer-places-Order-ordline

and generates in IS three links for the composite path: Customer-places, places-Order, Order-oline.

Then, the paths

Product-ordered ≡ Product-ordline.

are integrated, generating in IS the Product-oline link.

Adequate correspondences are generated for $V7$–IS and $V8$–IS.

The next step, 6.4, deals with the correspondences involving the date and Odate attributes (which were put aside by step 6.1):

ordered•date ≡ Order•Odate

ordered-ordered•date ≡ ordline - Order-Order•Odate.

The path correspondence includes a direct link: ordered-date. Therefore, the other link is chosen for integration in IS. The ordline-Order link already is in IS. The algorithm has only to add the Order-Odate link, which implies creating Odate in IS as attribute of Order. Some more assertions go into $V7$–IS and $V8$–IS.

As no refinement is needed, the final integrated schema is as shown in Fig. 20.

C. Example 3: Cars' Ownerships

Our last example was proposed by [8] to show a case of entity-type/relationship-type integration. It is based on the views illustrated in Fig. 21.

Correspondence assertions are:

- Person ≡ Carownership•SSN with corresponding attributes: SSN = SSN.
- Car ≡ Carownership•Lic# with corresponding attributes: Lic# = Lic#.
- owns ≡ Carownership.
- owns-Person ≡ Carownership-SSN.
- owns-Car ≡ Carownership-Lic#.

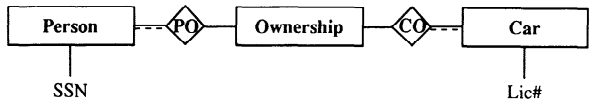

Fig. 22.

Fig. 23.

Integration starts with step 6.2, which generates three entity types in IS:

- Person (integration of Person and SSN), with attribute SSN.
- Car (integration of Car and Lic#), with attribute Lic#.
- Ownership (integration of owns and Carownership).

Path integration, step 6.3, adds the Person-Ownership and Car-Ownership links. This process implies that two new relationships are added—one, say PO, between Person and Ownership and one, say CO, between Car and Ownership. Cardinalities for the PO–Ownership and CO–Ownership roles are 1:1.

After this step, the integrated schema is as shown in Fig. 22.

In this case, if the refinement algorithm is run, the rule applies and the PO–Ownership–CO structure is replaced by a simpler Ownership relationship type (see Fig. 23), which is the final expected result.

VIII. Conclusion and Future Work

A powerful schema integration methodology is the key to successful database design and federated systems. For design purposes, it should allow users to build their view of the database independently of other users' views. For federated environment, it should support reuse of existing databases and existing application programs, without contradicting the launching of new federated database services.

We propose an integration methodology, designed to meet the above objectives. To that purpose, our approach is based on the following major features:

- automatic resolution of structural conflicts (arising because of different representations of the same real-world objects),
- conflict resolution performed without modification of initial views,
- use of a formal declarative approach for user (or DBA) definition of interviews correspondences,
- applicability to a variety of data models, and
- automatic generation of structural and operational mappings between the views and the integrated schema. Operational mappings provide support to allow users to query and update the database through their own view.

The first two features are essential with respect to the goals. They contrast with current methodologies, in which views are modified to conform to each other. Our approach relies on the idea that the complexity inherent in structural conflicts should be supported by establishing appropriate, powerful mapping facilities, to map initial views into the integrated schema. In other words, instead of forcing users to agree on a unique representation, we want to support their views as they are, and automatically build the underlying schema from which all views can be mapped in some way.

We did not discuss how mappings are built. In the ER context chosen for this paper, they will basically use the functionalities of the ERC+ algebra to modify queries and restructure their results before they are delivered to users.

To implement a formal declarative approach, we defined a model for describing correspondence assertions. Driven by users' assertions, the integrator tool acquires the necessary knowledge about similarities in the semantics of the views. For each assertion, formal rules state how to derive the constructs that are to be inserted into the integrated schema. Finally, we proposed an integration algorithm and examples which show how the algorithm achieves the desired result.

Last of all, in designing our methodology we had a strong concern with identifying the fundamentals of integration, so that the methodology could be transposed to major data models (semantic, object-oriented, functional, etc.). Whatever the data model, 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. This paper showed the interpretation of the approach in terms of an ER-like data model. ER elements are entity types, relationship types, and attributes. ER links we considered here are connections between a relationship type and the participating entity types (role links), as well as connections between attributes and the element they relate to (attribute links). In case of an object-oriented model, elements would be object classes and attributes, while links would be connections between objects and their components.

This paper highlighted the first three features in the above list, with the aim of putting forward the essential ideas of the proposed methodology. We plan to discuss the other features in forthcoming papers.

Future work will be devoted to:

- 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 integration of corresponding attributes in corresponding elements;
- integration of $1 : n$ correspondences, in which one instance or value in one view corresponds to a set of instances or values in the other view.

Our plans also include 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 once the views are defined.

Acknowledgment

The authors thank Prof. B. Bhargava for helpful suggestions that significantly aided in improving this paper.

REFERENCES

[1] V. Y. Lum *et al.*, "1978 New Orleans data base design working report," *5th Int. Conf. Very Large Data Bases*, Rio de Janeiro, Oct. 1979, pp. 328–339.

[2] C. Batini, M. Lenzerini, and S. B. Navathe, "A comparative analysis of methodologies for database schema integration," *ACM Computing Surveys*, vol. 15, no. 4, pp. 323–364, Dec. 1986.

[3] J. de Souza, "SIS—a schema integration system," in *Proc. BNCOD5 Conf.*, 1986.

[4] F. N. Civelek, A. Dogac, and S. Spaccapietra, "An expert system approach to view definition and integration," in *Proc. 7th Int. Conf. Entity-Relationship Approach*, Rome, Nov. 16-18, 1988, pp. 97–117.

[5] A. P. Sheth, J. A. Larson, A. Cornelio, and S. B. Navathe, "A tool for integrating conceptual schemas and user views," in *Proc. IEEE 4th Int. Conf. Data Engineering*, Los Angeles, Feb. 1–5, 1988, pp. 176–183.

[6] J. Diet and F. H. Lochovsky, "Interactive specification and integration of user views using forms," in *Proc. 8th Int. Conf. Entity-Relationship Approach*, Toronto, Oct. 18–20, 1989, pp. 376–390.

[7] D. A. Jardine and S. Yazid, "Integration of information submodels," in *Information Systems Concepts: An In-Depth Analysis*, E. D. Falkenberg and P. Lindgreen Eds. Amsterdam: North-Holland, 1989, pp. 247–267.

[8] J. A. Larson, S. B. Navathe, and R. Elmasri, "A theory of attribute equivalence in databases with application to schema integration," *IEEE Trans. Software Eng.*, vol. 15, no. 4, Apr. 1989.

[9] M. Bouzeghoub and I. Comyn-Wattiau, "View integration by semantic unification and transformation of data structures," in *Proc. 9th Int. Conf. Entity-Relationship Approach*, Lausanne, Oct. 8–10, 1990.

[10] S. Hayes and S. Ram, "Multi-user view integration system (MUVIS): An expert system for view integration," in *Proc. IEEE 6th Int. Conf. Data Engineering*, Los Angeles, Feb. 1990.

[11] S. B. Navathe and S. G. Gadgil, "A methodology for view integration in logical database design," in *Proc. 8th Int. Conf. Very Large Data Bases*, Mexico City, Sept. 8–10, 1982, pp. 142–164.

[12] C. Batini and M. Lenzerini, "A methodology for data schema integration in the entity-relationship model," *IEEE Trans. Software Eng.*, vol. SE-10, no. 6, pp. 650–664, Nov. 1984.

[13] M. V. Mannino and W. Effelsberg, "Matching techniques in global schema design," *IEEE Int. Conf. Data Engineering*, Los Angeles, Apr. 24–27, 1984, pp. 418–425.

[14] R. Elmasri and G. Wiederhold, "Data model integration using the structural model," in *Proc. ACM-SIGMOD Int. Conf. Management of Data*, Boston, June 1979, pp. 191–202.

[15] S. B. Navathe, R. Elmasri, and J. A. Larson, "Integrating user views in database design," *IEEE Comput.*, vol. 19, no. 1, Jan. 1986, pp. 50–62.

[16] M. A. Casanova and V. M. P. Vidal, "Towards a sound view integration methodology," in *Proc. 2nd ACM SIGMOD-SIGACT Symp. Principles Of Database Systems*, Atlanta, March 21–23, 1983, pp. 36–47.

[17] J. Biskup and B. Convent, "A formal view integration method," in *Proc. ACM-SIGMOD Int. Conf. Management of Data*, Washington, DC, May 28–30, 1986, pp. 398–407.

[18] B. Convent, "Unsolvable problems related to the view integration approach," in *Proc. Int. Conf. Database Theory*, Rome, Sept. 8–10, 1986, pp. 141–156.

[19] S. Spaccapietra, C. Parent, and Y. Dupont, "Automating heterogeneous schema integration," *VLDB J*, vol. 1, no. 1, pp. 81–126, Aug. 1992.

[20] C. Parent and S. Spaccapietra, "An entity-relationship algebra," in *Proc. IEEE Int. Conf. Data Engineering*, Los Angeles, Apr. 24–27, 1984, pp. 500–507.

[21] ——, "An algebra for a general entity-relationship model," *IEEE Trans. Software Eng.*, vol. SE-11, no. 7, July 1985, pp. 634–643.

[22] S. Spaccapietra, C. Parent, K. Yétongnon, and M. S. Abaidi, "Generalizations: A formal and flexible approach," in *Management of Data*, N. Prakash, Ed.. New York: McGraw-Hill, 1989, pp. 100–117.

[23] C. Parent, H. Rolin, K. Yétongnon, and S. Spaccapietra, "An ER calculus for the entity-relationship complex model," in *Proc. 8th Int. Conf. Entity-Relationship Approach*, Toronto, Oct. 18–20, 1989, pp. 75–98.

[24] C. Parent and S. Spaccapietra, "A model and an algebra for entity-relation type databases," *Technol. Science Informatics, Special Issue: Databases*, vol. 6, no. 8, Nov. 1987, pp. 623–642.

[25] C. Parent and S. Spaccapietra, "About entities, complex objects and object-oriented data models," in *Information System Concepts: An In-depth Analysis*, E. D. Falkenberg and P. Lindgreen Eds. Amsterdam: North-Holland, 1989, pp. 193–223.

[26] A. Motro and P. Buneman, "Constructing Superviews," in *Proc. ACM-SIGMOD Int. Conf. Management of Data*, Ann Arbor, MI, Apr. 29–May 1, 1981, pp. 56–64.

[27] A. Motro, "Superviews: Virtual integration of multiple databases," *IEEE Trans. Software Eng.*, vol. SE-13, no. 7, July 1987, pp. 785–798.

Stefano Spaccapietra (SM'93) received the Doctorat d'état from the University of Paris VI in 1978.

He has been an Assistant Professor at the University of Paris VI from 1969 to 1983. From 1983 to 1988, he was a full Professor at the Institute of Technology, University of Burgundy, Dijon, France. Since 1988, he is a full Professor at the Computer Science Department, Swiss Federal Institute of Technology, Lausanne, Switzerland, where he chairs the database laboratory. His first interest was in the area of database programming languages and data modeling, but moved in 1975 to system architectures for distributed database management. Heterogeneous databases lead him back to modeling aspects. Together with C. Parent, he developed the ERC+ approach, an entity-relationship model with object-oriented capabilities. His main research topics now include federated databases and visual interfaces. He has more than 100 publications in the area of distributed databases and conceptual modeling. Dr. Spaccapietra chaired for six years the Database Reference Model ISO working group, and for five years, the distributed database working group with AFCET, the French Computer Society. He chairs the Steering Committee for Entity-Relationship Conferences, is Secretary of the IFIP Working Group on Databases, and is a member of the executive committee of the Database Group of the Swiss Computer Society.

Christine Parent received the Doctoral degree in computer science in 1975 and the Doctorat d'état in 1987, both from the University of Paris VI.

She was an Assistant Professor at the University of Paris VI from 1970 to 1984. Since 1984, she has been a full Professor at the Computer Science Department, University of Burgundy, Dijon, France, where she chairs the Database Laboratory. She has been teaching and researching in data management since 1970. Her first research was directed towards distributed database management. Since 1983, she was deeply involved in the development of an extended ER model and of the associated algebraic and calculus-like manipulation language.

Dr. Parent has published extensively in technical journals and conferences on distributed databases, data modeling issues, and data manipulation languages. Since 1990, her research has focused on schema integration.