Resolving Structural Conflicts in the Integration of Entity-Relationship Schemas

Mong Li LEE Tok Wang LING

Department of Information Systems & Computer Science
National University of Singapore
{leeml, lingtw}@iscs.nus.sg

Abstract. Schema integration is essential to define a global schema that describes all the data in existing databases participating in a distributed or federated database management system. This paper describes a different approach to integrate two Entity-Relationship (ER) schemas. We focus on the resolution of structural conflicts, that is, when related real world concepts are modelled using different constructs in different schemas. Unlike previous works, our approach only needs to resolve the structural conflict between an entity type in one schema and an attribute in another schema and the other structural conflicts are automatically resolved. We have an algorithm to transform an attribute in one schema into an equivalent entity type in another schema without any loss of semantics or functional dependencies which previous approaches have not considered.

1. Introduction

Schema integration involves merging several schemas into one integrated schema. Recently, with the research into heterogenous database, this process finds an important role in integrating export schemas into a federated schema in a federated database system. There has been a large amount of work in the integration area: a comprehensive and detailed survey by [2] discussed twelve methodologies for view or database integration (or both), and new contributions continously appear in the literature [3, 6, 8, 11, 13, 16, 17] and many more in [10]. Most of these approaches do not provide an algorithmic specification of the integration activities. They provide only general guidelines and concepts on the integration process.

The schema integration process can be divided into five steps: preintegration, comparison, conformation, merging, and restructuring. The schema comparison step involves analysing and comparing schemas to determine correspondences among objects and detect possible conflicts in the representation of the same objects in different schemas. In the integration of ER schemas, we need to resolve the following conflicts: (1) naming conflict, (2) structural conflict, (3) identifier (primary key) conflict, (4) cardinality conflict, (5) domain mismatch or scale differences.

In this paper, we will focus on the resolution of structural conflicts in the integration of two ER schemas. Structural conflict occurs when related real world concepts are modelled using different constructs in the different schemas. We take a different approach to resolve such conflicts. Previous approaches [1, 13, 18] suggest the following types of structural conflicts in the integration of ER schemas:

(1) an entity type in one schema is modeled as an attribute of an entity type or a relationship set in another schema,
(2) an entity type in one schema is modeled as a relationship set in another schema,
(3) a relationship set in one schema is modeled as an attribute of an entity type or a relationship set in another schema,
(4) an attribute of a relationship set is modeled as an attribute of an entity type.

We find that only the first type of conflicts is meaningful. We have a precise algorithm to transform an attribute in one schema into an equivalent entity type in another schema without any loss of semantics or functional dependencies [15]. We advocate and show in this paper that the rest of the conflicts are automatically resolved after we have resolved the first type of conflict. This paper is organized as follows. Section 2 describes the ER model. Section 3 explains the terms and concepts we use in the integration of ER schemas. Section 4 presents our approach to resolve structural conflicts. Section 5 compares our approach with related works.

2. The Entity-Relationship Approach

The ER model, first introduced by Chen [4] incorporates the concepts of *entity type* and *relationship set*. An entity type or relationship set has *attributes* which represent its structural properties. An attribute can be *single-valued* (cardinality 1-1 or m-1) or *multivalued* (cardinality 1-m or m-n), or *composite*. A minimal set of attributes of an entity type E which uniquely identifies E is called a *key* of E. An entity type may have more than one key and we designate one of them as the *identifier* of the entity type. A minimal set of identifiers of some entity types participating in a relationship set R which uniquely identifies R is called a key of R. A relationship set may have more than one key and we designate one of them as the identifier of the relationship set. If the existence of an entity in one entity type depends upon the existence of a specific entity in another entity type, then such a relationship set and entity type are called *existence dependent relationship set* and *weak entity type*. A special case of existence dependent relationship occurs if the entities in an entity type cannot be identified by the values of their own attributes, but has to be identified by their relationship with other entities. Such a relationship set is called *identifier dependent relationship set*. A relationship set which involves weak entity type(s) is called a *weak relationship set*. An entity type which is not a weak entity type is called a *regular entity type*. *Recursive relationship sets* and special relationship sets such as *ISA, UNION, INTERSECT* etc, are allowed. A relationship set which is not weak or special is called a *regular relationship set*. For more details, see [14].

3. Schema Integration

Different approaches to the schema integration problem have been chosen in view integration and database integration. The majority of the view integration papers attempt to establish a semi-automated technique for deriving an integrated schema from a set of integration assertions relating equivalent constructs in the views [7, 18]. On the other hand, database integration methodologies aim at providing a tool allowing the DBA to build manually, by himself, the integrated schema, as a view over the initial schemas [6, 16]. In our integration of ER schemas, we will use the semi-automatic assertion based approach to resolve structural conflicts.

Two entity types E_1 and E_2 from two different schemas are equivalent, denoted by $E_1 \equiv E_2$, if they model the same real world concept. Real world concepts may be involved in a variety of associations called relationships. Two relationship sets R_1 and R_2 from two different schemas are equivalent, denoted by $R_1 \equiv R_2$, if they model the same relationship involving the same real world concepts. Both R_1 and R_2 have the same number of entity types and each of the participating entity type in R_1 has a corresponding equivalent participating entity type in R_2. Two attributes A_1 and A_2

from two different schemas are equivalent, denoted by $A_1 \equiv A_2$, if they model the same property of some real world concept or relationship.

A declarative statement asserting that a modelling construct in one schema is somehow related to modelling construct in another schema is called an *inter-schema correspondence assertion* (*integration assertion* for short). To ensure uniqueness of names, we adopt a full naming convention
`schemaname.objectname` for objects (entity types or relationship sets) and
`schemaname.objectname.attributename` for attributes.

There are three types of integration assertions:
(1) **Object equivalence**: These are assertions which state that two entity types, two relationship sets or two attributes are equivalent.
(2) **Structure equivalence**: These are assertions which state that a real world concept in one modelling construct is equivalent to a real world concept in a different modelling construct.
(3) **Generalization**: These are assertions which state that an entity type in one schema is a generalization of an entity type in the other schema.

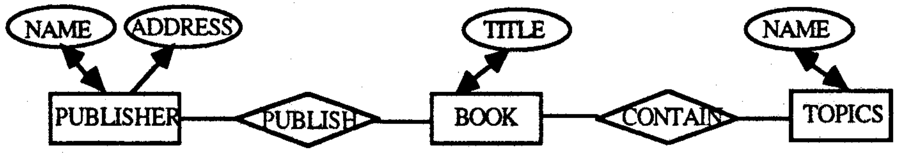


Figure 1a: Schema S1

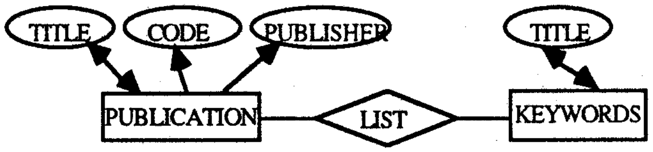


Figure 1b: Schema S2

Example 1 Consider the two schemas S1 and S2 in Figure 1. The DBA can declare that entity types `TOPICS` and `KEYWORDS` are equivalent as they model the same real world concept: `S1.TOPICS ≡ S2.KEYWORDS`
Theit attributes `NAME` and `TITLE` are equivalent since they model the same property of the same real world concept: `S1.TOPICS.NAME ≡ S2.KEYWORDS.TITLE`
The DBA can assert that entity type `PUBLISHER` in S1 is equivalent to the attribute `PUBLISHER` in S2, that is, the concept publisher in the real world has been modelled as an entity type `PUBLISHER` in one schema and as an attribute `PUBLISHER` in another schema: `S1.PUBLISHER ≡ S2.PUBLICATION.PUBLISHER`
The DBA can declare that `PUBLICATION` in S1 is a more general concept than `BOOK` in S2: `S1.BOOK` ISA `S2.PUBLICATION`
Finally, if we have `S1.BOOK.TITLE ≡ S2.PUBLICATION.TITLE`, then the attribute `TITLE` of the entity type `BOOK` is an **inherited** attribute.

The following example explains why an attribute of a relationship set in one schema cannot be equivalent to an attribute of an entity type in another schema unless one of them is a derived attribute of the other schema.

Example 2 Consider the two schemas shown in Figure 2. The entity type EMPLOYEE in Figure 2a is equivalent to the entity type EMPLOYEE in Figure 2b.

DEPT in Figure 2a is equivalent to DEPT in Figure 2b. The relationship set WORKFOR in Figure 2a is equivalent to WORKFOR in Figure 2b. The attributes EMPNO, NAME, SALARY and DNAME in Figure 2a are equivalent to EMPNO, NAME, SALARY and DNAME in Figure 2b.

If the attribute JOINDATE in Figure 2a is equivalent to the attribute JOINDATE in Figure 2b, then one of them is a derived attribute. Suppose JOINDATE in Figure 2a is a derived attribute, it is obtained from the schema in Figure 2b by joining the entity type EMPLOYEE with the relationship set WORKFOR over the attribute EMPNO. On the other hand, if JOINDATE in Figure 2a is not equivalent to JOINDATE in Figure 2b, then they are homonyms. We could have JOINDATE in Figure 2a model the date which an employee joins a department while JOINDATE in Figure 2b model the date an employee joins the organization. This is a naming conflict which can be easily resolved by renaming one of the attributes.

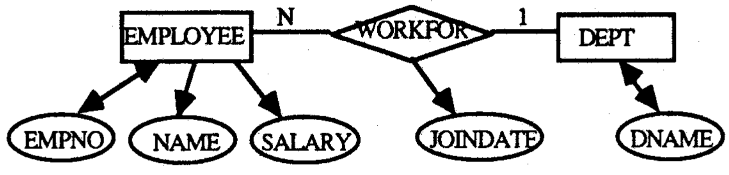


Figure 2a: JOINDATE is an attribute of relationship set WORKFOR

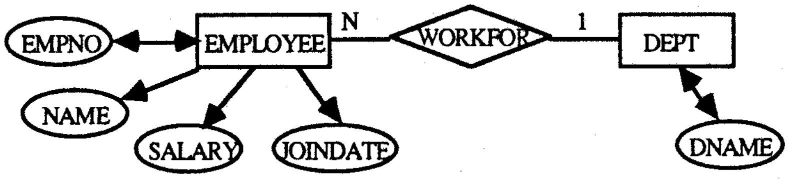


Figure 2b: JOINDATE is an attribute of entity type EMPLOYEE

We advocate that an attribute of a relationship set in one schema cannot be equivalent to an attribute of an entity type in another schema unless one of them is a derived attribute of the other schema. This is because conceptually, the semantics of these two attributes are inherently different. An attribute of an entity type is a property of the entity type. For example, EMPNO, ADDRESS and SALARY are properties of the entity type EMPLOYEE. These attributes are not related in any way to the other entity types or relationship sets in the schema. On the other hand, an attribute of a relationship set is meaningful only when it is associated with all the participating entity types in the relationship set. For example, suppose neither one of the attributes JOINDATE in Figure 2 are derived attributes. In Figure 2a, JOINDATE is meaningful only when it is associated with EMPNO and DNAME together since it models the date on which an employee assumes duty in a department of a company. This is the actual meaning of the relationship construct in the ER approach. The value of JOINDATE does not make sense with just the value of either EMPNO or DNAME only. Furthermore, the value of JOINDATE will need to be updated whenever the value of DNAME of a particular employee is updated. This is not so for JOINDATE in Figure 2b as JOINDATE is a property of the entity type EMPLOYEE and models the date an employee joins the company. The value of JOINDATE in Figure 2b need not be updated if an employee is transfered from one department to another. Therefore, if neither one of the attributes JOINDATE is a derived attribute and both of them are supposed to have the same meaning, then one of them has been designed wrongly.

4. Resolution of Structural Conflicts

In this section, we will discuss our approach to resolve structural conflicts in the integration of two ER schemas. We assume that all naming conflicts have been resolved.

First, we resolve the structural conflicts involving an entity type and an attribute. We have an algorithm to transform an attribute in one schema into an equivalent entity type in another schema [15]. This algorithm takes into consideration the cardinality of the attribute and therefore the transformation is done without any loss of semantics or functional dependencies. Next, we obtain a first-cut integrated schema by merging the two schemas together. Common entity types and relationship sets are superimposed. After the schemas are merged, we look for cycles in the merged ER schema. We interact with the DBA to determine which relationship sets in the cycle are redundant and can be removed from the merged schema. Finally, we build ISA hierarchies based on the generalization assertions given.

We now describe how we resolve the structural conflict where an entity type in one schema is modeled as an attribute in another schema. An example of this is given in Figure 1 where the concept publisher in the real world has been modelled as an entity type in schema S1 but is modelled as an attribute in schema S2. One reason why different structures has been used is because the designers of the two schemas are interested in different levels of details of the concept publisher. In schema S1, the database designer is interested in not only the name, but also the address of the various publishers. Therefore, he models this concept as an entity type PUBLISHER with attributes NAME and ADDRESS. On the other hand, the database designer of schema S2 is only interested in the publishers (names of publishers) of the various publication. Therefore, he models the publisher concept as an attribute of the entity type PUBLICATION. We resolve this structural conflict by transforming the attribute into an entity type. Our transformation takes into consideration the following factors:
(1) whether the attribute belongs to an entity type or a relationship set,
(2) whether the attribute is part of an identifier, a key or a composite attribute,
(3) the cardinality of an attribute, and
(4) the cardinality of the new entity type in the relationship sets it participates in.

These factors are not taken into consideration in previous approaches. It can be easily shown that our transformation is semantics-preserving and dependency-preserving [5, 9, 12]. Note that the cardinalities of the participating entity types in a relationship set R indicates the functional dependencies in R. Suppose R is a ternary relationship set with participating entity types A, B and C with identifiers A#, B# and C# respectively. If the cardinalities of A, B and C in R are n, m, 1 respectively, then we have the functional dependency A#, B# $\rightarrow$ C# in R. That is, if the identifier of a participating entity type of a relationship set R appears on the left hand side of a functional dependency of R, then it has a cardinality of n in R. Otherwise, it has a cardinality of 1 in R. However, note that if all the participating entity types of a relationship set have cardinality 1, then their identifiers are functionally equivalent.

We will give a sketch of our transformation algorithm here. The details of the algorithm can be found in [15]. The algorithm first considers if an attribute A belongs to an entity type E, then there are four basic scenarios when we transform A to an entity type E_A with identifier A:

Case 1: A is not part of a key and not part of a composite attribute.
E_A is connected to E by a new relationship set R.

Case 2: A is part of the identifier of E and there is no other key.
E becomes a weak entity type which is identifier dependent on E_A.

Case 3: $A \cup B$ is a key of E and there is another key (or identifier), or
$A \cup B$ is a composite multivalued attribute, where B is a set of attributes.
Transform B to an entity type E_B with identifier B.
E_A and E_B are connected to E by a new relationship set R.

Case 4: $A \cup B$ is a composite m-1 attribute, B is a set of attributes.
E_A is connected to E by a new relationship set R.
B becomes a m-1 attribute of R.

These various scenarios and their respective transformations are shown in Figure 3. Note that the relationship set R is not restricted to binary relationships only.

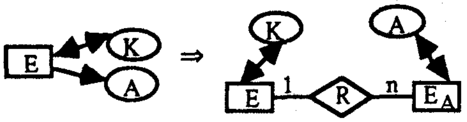


Figure 3a: Case 1 - A is not part of a key & not part of a composite attribute. Entity type E_A is connected to E by R.

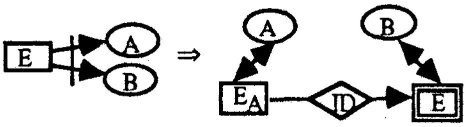


Figure 3b: Case 2 - A is part of the identifier and there is no other key. E becomes a weak entity type which is identifier dependent on E_A.

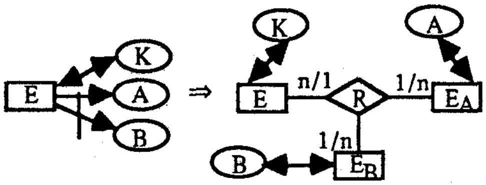


Figure 3c: Case 3 - A is part of a key and there is another key (or identifier). New entity types E_A and E_B are connected to E by R.
$K \rightarrow A, B$ & $A, B \rightarrow K$

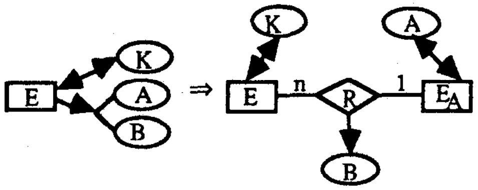


Figure 3d: Case 4 - A is part of a m:1 composite attribute. New entity type E_A is connected to E by R. B becomes an attribute of R.

Figure 3: Transformations of an attribute A to an entity type E_A.
A belongs to an entity type E.

On the other hand, if an attribute A belongs to a relationship set R, then there are three basic scenarios when we transform A to an entity type E_A with identifier A:

Case 1: A is not part of a composite attribute of R.
E_A becomes a participating entity type of R.

Case 2: $A \cup B$ is a composite multivalued or 1-1 attribute of R, B is a set of attributes.
Transform B to an entity type E_B.
E_A and E_B become participating entity types of R.

Case 3: $A \cup B$ is a composite m-1 attribute of R, B is a set of attributes.
E_A becomes a participating entity type of R.
B becomes a m-1 attribute of R.

These various scenarios and their respective transformations are shown in Figure 4.

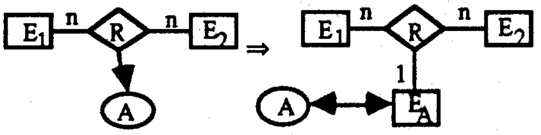


Figure 4a: Case 1 - A is not part of a composite attribute of R. E_A becomes a participating entity type of R.

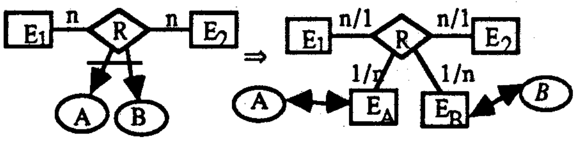


Figure 4b: Case 2 - A is a part of a 1:1 composite attribute of R. E_A & E_B become participating entity types of R. E_1 , E_2 $\rightarrow$ A, B & A, B $\rightarrow$ E_1 , E_2

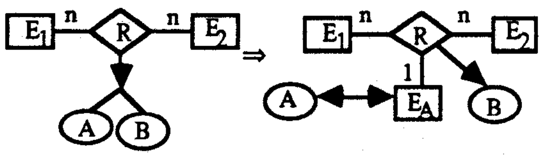


Figure 4c: Case 3 - A is part of a m:1 composite attribute of R. E_A becomes a participating entity type of R. B becomes a m:1 attribute of R.

Figure 4: Transformations of an attribute A into an entity type E_A A belongs to a relationship set R.

5. Comparison with Related Works

In this section, we will compare our approach with previous works [1, 13, 19]. We will also show that the following types of structural conflicts are automatically resolved after we resolve the structural conflict between an attribute and an entity type:
(1) an entity type in one schema is modeled as a relationship set in another schema,
(2) a relationship set in one schema is modeled as an attribute of an entity type or a relationship set in another schema.

Note that the structural conflict between a relationship set in one schema and an attribute of an entity type or a relationship set can be reduced to the structural conflict between a relationship set and an entity type by transforming the attribute into an equivalent entity type.

[1] resolves conflict involving relationship sets by transforming the relationship set into an entity type. However, this introduces a prolifieration of entity types and relationship sets which increases the complexity of integration. The approach in [19] is not satisfactory because it will result in loss of semantics in the integrated schema. Our investigation into the resolution of structural conflicts have led us to conclude that these conflicts are actually avoidable once we have resolved any structural conflicts between an entity type in one schema and an attribute in another schema.

Example 3 Consider the two schemas shown in Figure 5. Using the approach in [1] to integrate these two schemas S3 and S4, they would reconcile the relationship set OWNS in Figure 5a and the entity type OWNERSHIP in Figure 5b. [1] would transform the relationship set OWNS in Figure 5a into an entity type called OWNS-E. This new entity type OWNS-E is connected to PERSON and VEHICLE in Figure 5a by two new relationship sets R_1 and R_2. Figure 5c shows the schema obtained. There are several problems with this approach. First, we do not know the semantics of R_1 and R_2 or the cardinalities of the entity types participating in these two relationship sets. We should not merge the schemas of Figure 5b and 5c just because structurally they are identical. This is because we can have more than one relationship sets which have distinctly different meanings between the same entity types. Second, the new entity type OWNS-E in Figure 5c has no identifier. Even if we give an identifier for OWNS-E, say O#, we do not know whether CERT# in Figure 5b and O# are equivalent. Third, it is not known how we can populate R_1 and R_2 from the original relationship set OWNS. Finally, there may be loss of information if we arbitrary split a relationship set into two or more relationship sets.

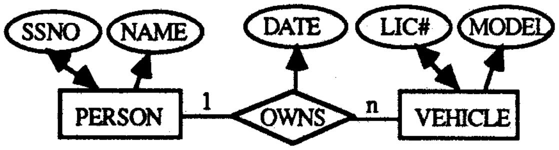


Figure 5a: Schema S3

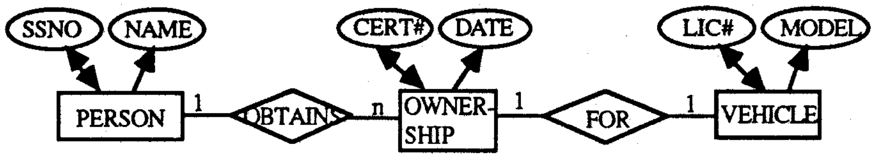


Figure 5b: Schema S4

In our approach, we first resolve any structural conflicts between an entity type in one schema and an attribute in another schema; in this example there is none. Then, we proceed to merge the two schemas S3 and S4 (Figure 5d). We observe that there is a cycle. We check with the DBA if the relationship set OWNS is actually derived from a join of the two relationship sets OBTAINS and FOR over the common entity type OWNERSHIP and the attribute CERT# has been projected out. If it is, this implies that OWNS is redundant. Therefore we drop OWNS from the integrated schema. The integrated schema is S4. Otherwise, if OWNS is not a derived relationship set and therefore it is not redundant, then the integrated schema is shown in Figure 5d. Note that any conflict between the attributes DATE in S3 and S4 can be easily resolved (see discussion in Example 2).

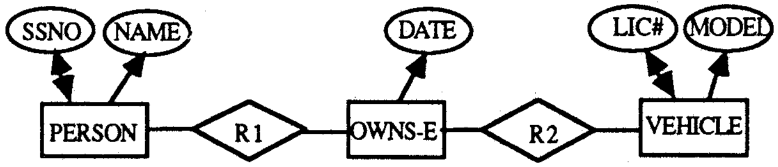


Figure 5c: Relationship set OWNS in S3 transformed to entity type OWNS-E by [Bati84].

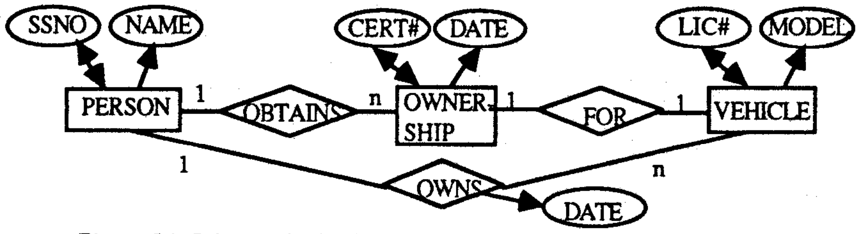
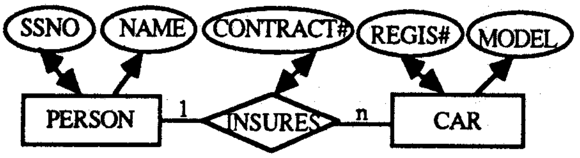


Figure 5d: Schema obtained by merging S3 and S4 by our approach.

Example 4 Consider the following schemas S5 and S6 which is given in [19].



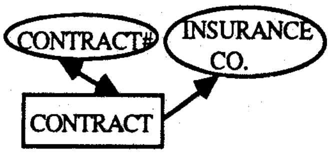


Figure 6a: Schema S5 *Figure 6b: Schema S6*

[19] asserts that the relationship set INSURES in Figure 6a is equivalent to the entity type CONTRACT in Figure 6b and transforms INSURES into an entity type CONTRACT whose identifier is CONTRACT# and two binary relationship sets SIGNS and OWNS which connects CONTRACT to PERSON and CAR respectively (Figure 6c). The transformed schema S7 is then integrated with S6. This solution is unsatisfactory because it results in loss of semantics. In schema S5, INSURES associates a person who insures a car with a particular contract. However, in the integrated schema, this association is split into two relationships: SIGNS associates a person who signs a contract, and INSURANCE associates an insurance contract with a car. The semantics in the original schema S5 is not maintained in the integrated schema S7. This is because in schema S7, it is possible to have an insurance contract for a car without requiring that the contract by signed by a person. On the other hand, in schema S5, we require a person to be associated (or sign a contract) to insure a car.

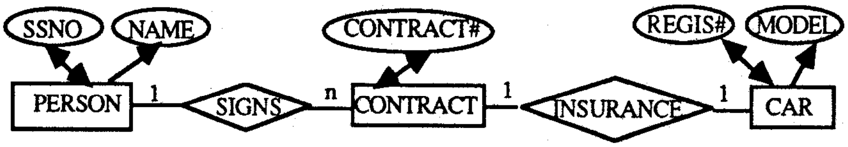


Figure 6c: Schema S7 transformed from S5 by [Spac92].

In our approach, we observe that the attribute CONTRACT# in schema S5 has been modeled as an entity type CONTRACT in schema S6. We transform the attribute into an entity type. The new entity type CONTRACT simply becomes another participating entity type in INSURES (Figure 6d). Therefore INSURES becomes a ternary relationship set. The entity type CONTRACT in Figure 6b is equivalent to the entity type CONTRACT in Figure 6d and can therefore be merged. Compare the semantics captured in Figure 6d with that in Figure 6c. There is no loss of semantics in our final integrated schema because the ternary relationship set INSURES still maintains the association of a person who insures a car with a particular contract.

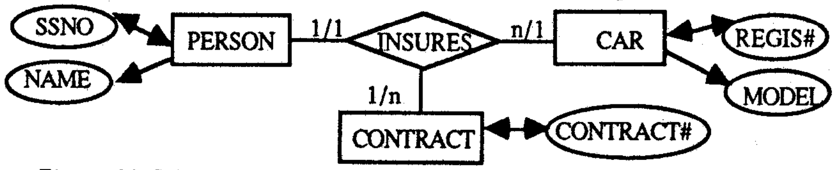


Figure 6d: Schema S8 obtained by transforming the attribute CONTRACT# in S5 into an entity type CONTRACT by our approach.

6. Conclusion

In this paper, we have presented a different methodology to integrate ER schemas. We have focus on the resolution of structural conflicts and have given a detailed algorithm to transform an attribute into an entity type without any loss of semantics. This algorithm takes into consideration whether the attribute belongs to an entity type or a relationship set, whether the attribute is part of an identifier, a key or a composite attribute, and the cardinality of the attribute. Unlike previous works, we have shown that our approach needs to resolve only one type of structural conflict, which is the structural conflict between an entity type in one schema and an attribute in another schema, and the other types of structural conflict are automatically resolved.

References

[1] Batini, C. and Lenzerini, M., A Methodology for Data Schema Integration in the Entity-Relationship Model, IEEE Trans. Software Engineering, vol 10, 6, 1984.

[2] Batini, C., Lenzerini, M. and Navathe, S.B., A Comparative Analysis of Methodologies for Database Schema Integration, ACM Computing Surveys, Vol 18, No 4, December 1986, pp 323-364.

[3] Bouzenhoub and Commyn-Wattiau: View Integration by Semantic Unification and Transformation of Data Structures, Proc. of the Int. ER Approach, 1990.

[4] Chen, P.P., The Entity-Relationship Model: Toward a Unified View of Data, ACM Transactions on Database Systems vol 1, no 1, 1976, pp 166-192.

[5] D'Atri, A., Sacca, D., Equivalence and Mapping of Database Schemas, in Proc. 10th VLDB Conf., 1984.

[6] Dayal, U. and Hwang, H., View Definition and Generalization for Database Integration in Multibase: A System for Heterogenous Distributed Databases, IEEE Transaction on Software Engineering Vol 10, 6, 1984, pp 628-644.

[7] Effelsberg, W. and Mannino, M.V., Attribute equivalence in global schema design for heterogenous distributed databases, Information Systems 9(3), 1984.

[8] Elmasri, R. and Navathe, S., Object integration in logical database design, IEEE First Int. Conf. on Data Engineering, 1984, pp 426-433.

[9] Hainaut, J.-L., Entity-Generating Schema Transformations for Entity-Relationship Models, 10th Int. Conf. on Entity-Relationship Approach, 1991.

[10] Kambayashi, Y., Rusinkiewicz, M. and Sheth, A., First Int. Workshop on Interoperability in Multidatabase Systems, Kyoto, 1991.

[11] Kaul, M. et. al, ViewSystem: Integrating Heterogenous Information Bases by Object-Oriented Views, IEEE 6th Int. Conf. on Data Engineering, LA, 1990.

[12] Kobayashi, I., Losslessness and Semantic Correctness of Database Schema Transformation: another look at Schema Equivalence, Information Systems Vol 11 No 1, 1986.

[13] Larson, J., Navathe, S. and Elmasri, R., A Theory of Attribute Equivalence in Database with Application to Schema Integration, IEEE Trans. on Software Engineering, 15:449-463, 1989.

[14] Ling, T.W., "A Normal Form for Entity-Relationship Diagrams", Proc. 4th International Conference on Entity-Relationship Approach, 1985.

[15] Lee, M.L. and Ling, T.W., A Methodology for Structural Conflict Resolution in the Integration of Entity-Relationship Schemas, Technical Report, August 1995.

[16] Motro, A., Superviews: Virtual integration of multiple databases, IEEE Trans. on Software Engineering, 13(7), 1987, pp 785-798.

[17] Navathe, S.B. and Gadgil, S., A Methodology for View Integration in Logical Database Design, 8th International Conference on VLDB, 1982, pp 142-152.

[18] Spaccapietra, S., Parent, C, and Dupont, Y., View Integration: A step forward in solving structural conflicts, Technical Report, 1990.

[19] Spaccapietra, S., Parent, C., and Dupont, Y., Model independent assertions for integration of heterogenous schemas, VLDB Journal, (1), 1992, pp 81-126.