Resolving Schematic Discrepancy in the Integration of Entity-Relationship Schemas

Qi He Tok Wang Ling

School of Computing, National University of Singapore
{heqi,lingtw}@comp.nus.edu.sg

Abstract. In schema integration, schematic discrepancies occur when data in one database correspond to metadata in another. We define this kind of semantic heterogeneity in general using the paradigm of context that is the meta information relating to the source, classification, property etc of entities, relationships or attribute values in entity-relationship (ER) schemas. We present algorithms to resolve schematic discrepancies by transforming metadata into entities, keeping the information and constraints of original schemas. Although focusing on the resolution of schematic discrepancies, our technique works seamlessly with existing techniques resolving other semantic heterogeneities in schema integration.

1 Introduction

Schema integration involves merging several schemas into an integrated schema. More precisely, [4] defines schema integration as "the activity of integrating the schemas of existing or proposed databases into a global, unified schema". It is regarded as an important work to build a *heterogeneous database system* [6, 22] (also called *multidatabase system* or *federated database system*), to integrate data in a data warehouse, or to integrate user views in database design. In schema integration, people have identified different kinds of semantic heterogeneities among component schemas: naming conflict (homonyms and synonyms), key conflict, structural conflict [3, 15], and constraint conflict [14, 21].

A less touched problem is schematic discrepancy, i.e., the same information is modeled as data in one database, but metadata in another. This conflict arises frequently in practice [11, 19]. We adopt a semantic approach to solve this issue. One of the outstanding features of our proposal is that we preserve the cardinality constraints in the transformation/integration of ER schemas. Cardinality constraints, in particular, functional dependencies (FDs) and multivalued dependencies (MVDs), are useful in verifying lossless schema transformation [10], schema normalization and semantic query optimization [9, 21] in multidatabase systems. The following example illustrates schematic discrepancy in ER schemas. To focus our contribution and simplify the presentation, in the example below, schematic discrepancy is the only kind of conflicts among schemas.

Example 1. Suppose we want to integrate supply information of products from

several databases (Fig. 1). These databases record the same information, i.e., product numbers, product names, suppliers and supplying prices in each month, but have discrepant schemas. In DB1, suppliers and months are modeled as entity types. In DB2, months are modeled as meta-data of entity types, i.e., each entity type models the products supplied in one month, and suppliers are modeled as meta-data of attributes, e.g., the attribute S1_PRICE records the supplying prices by supplier *s1*.[1] In DB3, months are modeled as meta-data of relationship types, i.e., each relationship type models the supply relation in one month. We propose (in Section 4) to resolve the discrepancies by transforming the metadata into entities, i.e., transforming DB2 and DB3 into a form of DB1. The statements on the right side of Fig. 1 provide the semantics of the constructs of these schemas using ontology, which will be explained in Section 3.□

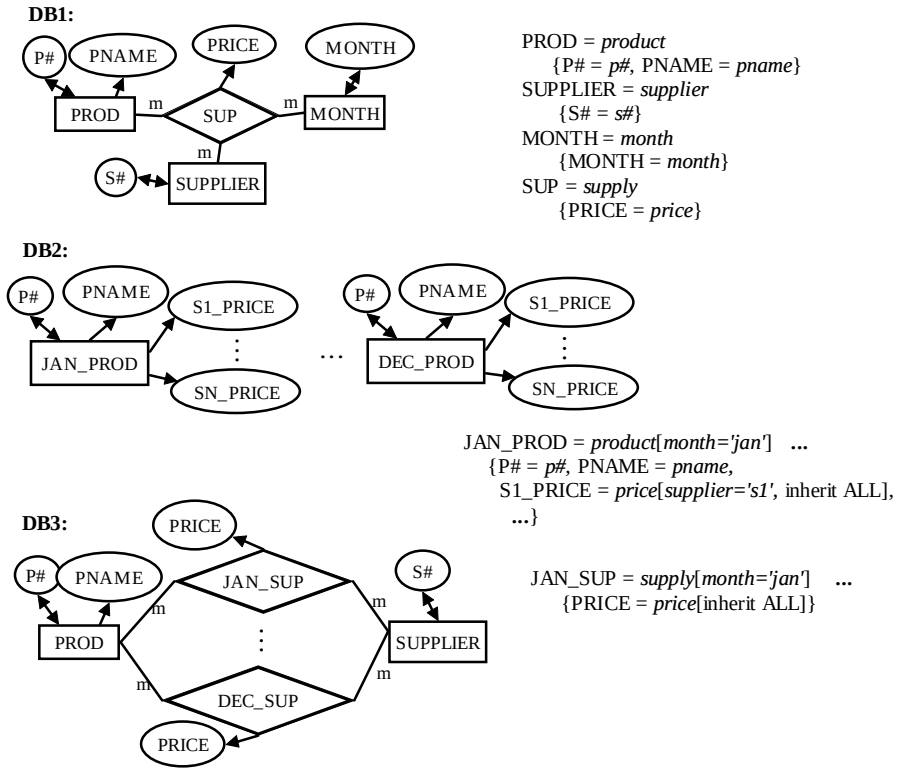


Fig. 1. Schematic discrepancy: months and suppliers modeled differently in DB1, DB2 and DB3

Paper organization. The rest of the paper is organized as follows. Section 2 is an introduction to the ER approach. Section 3 and 4 are the main contributions of this

[1] Without causing confusion, we blur the difference on entities and identifiers of entities. E.g., we use supplier number *s1* to refer to a supplier with identifier S# = *s1*, i.e., *s1* plays both the roles of an attribute value of S# and an entity of supplier.

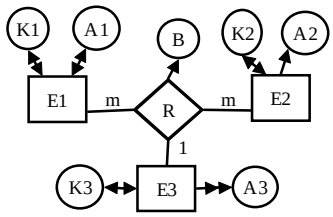
paper. In Section 3, we first introduce the concepts of ontology and context, and the mappings from schema constructs of ER schemas onto types of ontology. Then we define schematic discrepancy in general using the paradigm of context. In Section 4, we present algorithms to resolve schematic discrepancies in schema integration, without any loss of information and cardinality constraints. In Section 5, we compare our work with related work. Section 6 concludes the whole paper.

2 ER Approach

In the ER model, an entity is an object in the real world and can be distinctly identified. An *entity type* is a collection of similar entities that have the same set of predefined common *attributes*. Attributes can be *single-valued*, i.e., 1:1 (one-to-one) or m:1 (many-to-one), or *multivalued*, i.e., 1:m (one-to-many) or m:m (many-to-many). 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 relationship is an association among two or more entities. A *relationship type* is a collection of similar relationships that satisfy a set of predefined common attributes. A minimal set of attributes (including the identifiers of participating entity types) in a relationship type R that 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 type.

The cardinality constraints of ER schemas incorporate FDs and MVDs. For example, given an ER schema below, let K1, K2 and K3 be the identifiers of E1, E2 and E3, we have:

K1→A1 and A1→K1, as A1 is a 1:1 attribute of E1;
K2→A2, as A2 is a m:1 attribute of E2;
K3↠ A3, as A3 is a m:m attribute of E3;
{K1, K2}→K3, as the cardinality of E3 is 1 in R;
{K1, K2}→B, as B is a m:1 attribute of R.



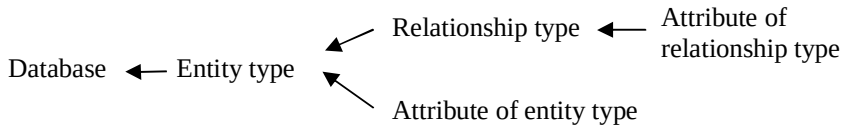
3 Ontology and Context

In this section, we first represent the constructs of ER schemas using ontology, then define schematic discrepancy in general based on the schemas represented using ontology. In this paper, we treat ontology as the specification of a representational vocabulary for a shared domain of discourse which includes the definitions of types (representing classes, relations, and properties) and their values. We present ontology at a conceptual level, which could be implemented by an ontology language, e.g., OWL [20].

For example, suppose ontology *SupOnto* describes the concepts in the universe of product supply. It includes the following types: *product, month, supplier, supply* (i.e., the supply relations among products, months and suppliers), *price* (i.e. the supplying prices of products), *p#, pname, s#,* etc. It also includes the values of these types, e.g. *jan, ..., dec* for *month*, and *s1, ..., sn* for *supplier*. Note we use lower case italic words

to represent types and values of ontology, in contrast to capitals for schema constructs of an ER schema. By use of OWL expression, *product, month, supplier* and *supply* would be declared as classes, *p#* and *pname* as properties of *product*, *s#* as a property of *supplier*, and *price* as a property of *supply*.

Conceptual modeling is always done within a particular context. In particular, the context of an entity type, relationship type or attribute is the meta-information relating to its source, classification, property etc. Contexts are usually at four levels: *database, object class, relationship type* and *attribute*. An entity type may "inherit" a context from a database (i.e., the context of a database applies to the entities), and so on. In general, the inheritance hierarchy of contexts at different levels is:



We'll give a formal representation of context below. Note as the context of a database would be handled in the object classes which inherit it, we will not care database level contexts any more in the rest of the paper.

Definition 1. Given an ontology, we represent an entity type (relationship type, or attribute) E as:

$$E = T\ [C_1=c_1, \ldots, C_m=c_m, \text{inherit } C_{m+1}, \ldots, C_n]$$

where T, C_1, …, C_n are types in the ontology, and each c_i is a value of C_i for $i \in \{1, \ldots, m\}$. C_{m+1}, …, C_n respectively have a value of c_{m+1}, …, c_n which are not explicitly given. This representation means that each instance of E is a value of T, and satisfies the conditions $C_i = c_i$ for each $i \in \{1, \ldots, n\}$. C_1, …, C_n with the values constitute the ***context*** within which E is defined; we call them ***meta-attributes***, and their values ***metadata*** of E. Furthermore, C_{m+1}, …, C_n with the values are from the context at a higher level (i.e. the context of a database if E is an entity type, the contexts of entity types if E is a relationship type, or the context of an entity type/relationship type if E is an attribute). We call E ***inherits*** the meta-attributes C_{m+1}, …, C_n with the values. If E inherits all the meta-attributes with values of the higher level context, we simply represent it as:

$$E = T\ [C_1=c_1, \ldots, C_m=c_m, \text{inherit ALL}].$$

For easy reference, we call the set $\{C_1=c_1, \ldots, C_m=c_m\}$ the ***self context***, and $\{C_{m+1}=c_{m+1}, \ldots, C_n=c_n\}$ the ***inherited context*** of E. □

In the above representation of E, either self or inherited context could be empty. Specifically, when the context of E is empty, we have E = T.

In the example below, we represent the entity types, relationship types and attributes in Fig. 1 using the ontology *SupOnto*.

Example 2. In Fig. 1, using the ontology *SupOnto*, the entity type JAN_PROD of DB2 is represented as:

$$\text{JAN\_PROD} = \textit{product}[\textit{month} = \textit{'jan'}].$$

That is, the context of JAN_PROD is *month='jan'*. This means that each entity of JAN_PROD is a product supplied in Jan.

Also in DB2, given an attribute S1_PRICE of the entity type JAN_PROD, we represent it as:

S1_PRICE = *price*[*supplier='s1'*, inherit ALL].

That is, the self context of S1_PRICE is *supplier='s1'*, and the inherited context (from the entity type) is *month='jan'*. This means that each value of S1_PRICE of the entity type JAN_PROD is a price of a product supplied by supplier *s1* in the month of Jan.

In DB3, given a relationship type JAN_SUP, we represent it as:

JAN_SUP = *supply*[*month = 'jan'*].

This means that each relationship of JAN_SUP is a supply relationship in the month of Jan.

Also in DB3, given an attribute PRICE of the relationship type JAN_SUP, we represent it as:

PRICE = *price*[inherit ALL].

PRICE inherits the context *month='jan'* from the relationship type. This means that each value of PRICE of the relationship type JAN_SUP is a supplying price in Jan.□

In contrast to original ER schemas, we call an ER schema whose schema constructs are represented using ontology symbols ***elevated schema***, as the ER schemas with the statements given in Fig. 1. The mapping from an ER schema onto an elevated schema should be specified by users. Our work is based on elevated schemas. Now we can define schematic discrepancy in general as follows.

Definition 2. Two elevated schemas are ***schematic discrepant***, if metadata in one database correspond to attribute values or entities in the other. We call meta-attributes whose values correspond to attribute values or entities in other databases ***discrepant meta-attributes***.□

For example, in Fig. 1, in DB2, *month* and *supplier* are discrepant meta-attributes as their values correspond to entities in DB1, so is the meta-attribute *month* in DB3.

Before ending this section, we define the *global identifier* of a set of entity types. In general, two entity types (or relationship types) E1 and E2 are ***similar***, if E1=T[Cnt1] and E2=T[Cnt2] with T an ontology type, and Cnt1 and Cnt2 two sets (possibly empty sets) of meta-attributes with values. Intuitively, a global identifier identifies the entities of similar entity types, independent of context.

Definition 3. Given a set of similar entity types $\mathcal{E}$, let K be an identifier of each entity type in $\mathcal{E}$. We call K a ***global identifier*** of the entity types of $\mathcal{E}$, provided that if two entities of the entity types of $\mathcal{E}$ refer to the same real world object, then the values of K of the two entities are the same, and vice versa. □

For example, in Fig. 1, the PROD entity types of *DB1* and *DB3*, and the entity types JAN_PROD, …, DEC_PROD of *DB2* are similar entity types, for they all correspond to the ontology type *product* without or with a context. Suppose P# is a global identifier of these entity types, i.e., P# uniquely identifies products from all the three databases. Similarly, we suppose S# is a global identifier of the SUPPLIER entity types of *DB1* and *DB3*.

In [13], Lee et al proposes an ER based federated database system where local schemas modeled in the relational, object-relational, network or hierarchical models are first translated into the corresponding ER export schemas before they are

integrated. Our approach is an extension to theirs by using ontology to provide semantics necessary for schema integration. In general, local schemas could be in different data models. We first translate them into ER or ORASS schemas (ORASS is an ER-like model for semi-structured data [25]). Then map the schema constructs of ER schemas onto the types of ontology and get elevated schemas with the help of semi-automatic tools. Finally, integrate the elevated schemas using the semantics of ontology; semantic heterogeneities among elevated schemas are resolved in this step. Integrity constraints on the integrated schema are derived from the constraints on the elevated schemas at the same time.

4 Resolving Schematic Discrepancies in the Integration of ER Schemas

In this section, we resolve schematic discrepancies in schema integration. In particular, we present four algorithms to resolve schematic discrepancies for entity types, relationship types, attributes of entity types and attributes of relationship types respectively. This is done by transforming discrepant meta-attributes into entity types. The transformations keep the cardinalities of attributes and entity types, and therefore preserve the FDs and MVDs. Note in the presence of context, the values of an attribute depend on not only the identifier of an entity type/relationship type, but also the metadata of the attribute. To simplify the presentation, we only consider the discrepant meta-attributes of entity types, relationship types and attributes, leaving the other meta-attributes out as they will not change in schema transformation.

In the rest of this section, we first present Algorithm TRANS_ENT and TRANS_REL, the resolutions of discrepancies for entity types and relationship types in Section 4.1, and then TRANS_ENT_ATTR and TRANS_REL_ATTR, the resolutions for attributes of entity types and attributes of relationship types in Section 4.2. Examples are provided to understand each algorithm.

4.1 Resolving Schematic Discrepancies for Entity types/Relationship types

In this sub-section, we first show how to resolve discrepancies for entity types using the schema of Fig. 1, then present Algorithm TRANS_ENT in general. Finally, we describe the resolution of discrepancies for relationship types by an example, omitting the general algorithm which is similar to TRANS_ENT.

As an example to remove discrepancies for entity types, we transform the schema of DB2 in Fig. 1 below.

Example 3 (Fig. 2). In Step 1, for each entity type of DB2, say JAN_PROD, we represent the meta-attribute *month* as an entity type MONTH consisting of the only entity *jan* that is the metadata of JAN_PROD. We change the entity type JAN_PROD into PROD after removing the context, and construct a relationship type R to associate the entities of PROD and the entity of MONTH. Then we handle the attributes of JAN_PROD. As PNAME has nothing to do with the context *month = 'jan'* of the entity type, it becomes an attribute of PROD. However, S1_PRICE, …,

SN_PRICE inherit the context of month; they become the attributes of the relationship type R. Then in Step 2, the corresponding entity types, relationship types and attributes are merged respectively. The merged entity type of MONTH consists of all the entities {*jan, …, dec*} of the original MONTH entity types, so do the entity type PROD, relationship type R and their attributes.□

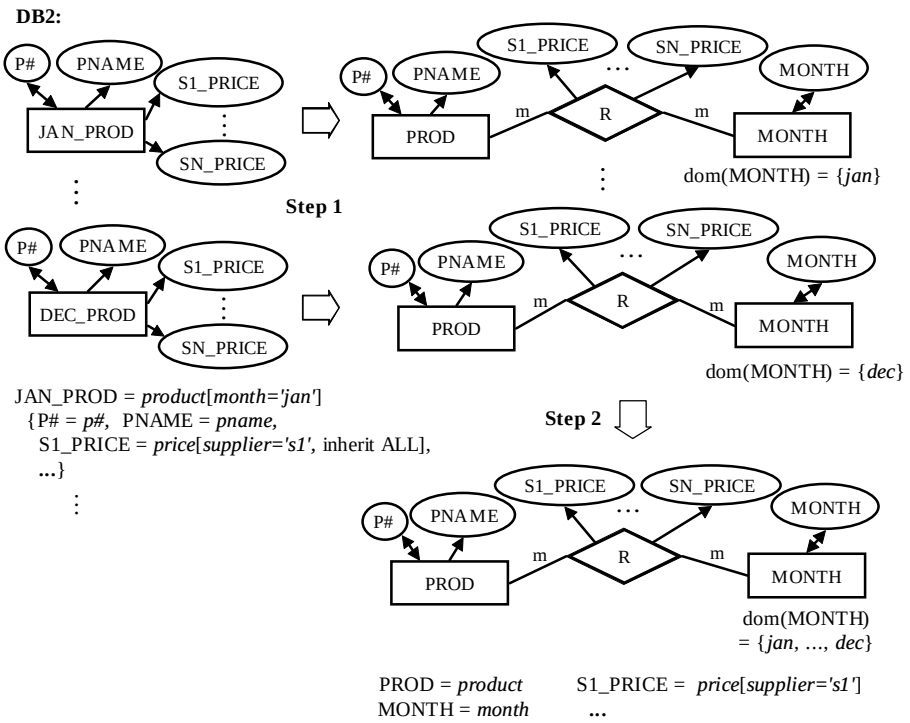


Fig. 2. Resolve schematic discrepancies for entity types

Then we give the general algorithm below.

Algorithm TRANS_ENT

Input: an elevated schema DB.

Output: a schema DB' transformed from DB such that all the discrepant meta-attributes of entity types are transformed into entity types.

Step 1: Resolve the discrepant meta-attributes of an entity type.

Let E = $E_{ont}[C_1=c_1, \ldots, C_m=c_m]$ be an entity type of DB, for E_{ont} a type in the ontology and discrepant meta-attributes $C_1, \ldots, C_m$ with the values $c_1, \ldots, c_m$. Let K be the global identifier of E.

Step 1.1: Transform discrepant meta-attributes $C_1, \ldots, C_m$ into entity types. Construct an entity type E' = E_{ont} with the global identifier K. E' consists of the entities of E without any context.

Construct entity types $E_{C_i} = C_i$ with identifier $K_{C_i} = C_i$ for each meta-attribute $C_i \in \{C_1, \ldots, C_m\}$. Each E_{C_i} contains one entity c_i.

//Construct a relationship type to represent the associations among the entities of E and the values of $C_1, \ldots, C_m$.

Construct a relationship type R connecting the entity types E' and $E_{C_1}, \ldots, E_{C_m}$.

Step 1.2: Handle the attributes of E.

Let A be an attribute (not part of the identifier) of E, and selfCnt, a set of meta-attributes with values, be the self context of A.

If A is a m:1 or m:m attribute, **then**

Case 1: attribute A has nothing to do with the context of E. Then A becomes an attribute of E'.

Case 2: attribute $A = A_{ont}$[selfCnt, inherit ALL] inherits all the context $\{C_1=c_1, \ldots, C_m=c_m\}$ from E. Then $A' = A_{ont}$[selfCnt] becomes an attribute of R.

Case 3: attribute $A = A_{ont}$[selfCnt, inherit S] inherits some discrepant meta-attributes $S \subset \{C_1, \ldots, C_m\}$ with the values from E, $S \neq \emptyset$. Then construct a relationship type R_A connecting E' and those E_{C_i} for each meta-attribute $C_i \in S$. Attribute $A' = A_{ont}$[selfCnt] becomes an attribute of R_A.

Else *// A is a 1:1 or 1:m attribute, i.e., the values of A determine the entities of E in the context. In this case, A should be modeled as an entity type to preserve the cardinality constraint. We keep the discrepant meta-attributes of A, and delay the resolution in Alg. TRANS_ENT_ATTR, the resolution for attributes of entity types.*

Construct an attribute $A' = A_{ont}$[Cnt] of E', with Cnt the (self and inherited) context of A as the (self) context of A'.

Step 1.3: Handle relationship types involving entity type E in DB.

Let R1 be a relationship type involving E in DB.

Case 1: R1 has nothing to do with the context of E. Then replace E with E' in R1.

Case 2: R1 inherits all the context $\{C_1=c_1, \ldots, C_m=c_m\}$ from E. Then replace E with R (i.e., treat R as a high level entity type) in R1.

Case 3: R1 inherits some discrepant meta-attributes $S \subset \{C_1, \ldots, C_m\}$ with the values from E, $S \neq \emptyset$. Then construct a relationship type R2 connecting E' and those E_{C_i} for each meta-attribute $C_i \in S$. Replace E with R2 in R1.

Step 2: Merge the entity types, relationship types and attributes respectively which correspond to the same ontology type with the same context, and union their domains. □

In the resolution of schematic discrepancies for relationship types, we should deal with a set of entity types (participating in a relationship type) instead of individual ones. The steps are similar to those of Algorithm TRANS_ENT, but without Step 1.3.

We omit the resolution algorithm TRANS_REL for lack of space, but explain it by any example below, i.e., transforming the schema of DB3 in Fig. 1.

Example 4 (Fig. 3). In Step 1, for each relationship type of DB3, say JAN_SUP, we represent the meta-attribute *month* as an entity type MONTH consisting of the only entity *jan* that is the metadata of JAN_SUP. We change JAN_SUP into the relationship type SUP after removing the context, and relate the entity type MONTH to SUP. Attribute PRICE of JAN_SUP inherits the context *month='jan'* from the relationship type, and therefore it becomes an attribute of SUP in the transformed schema. Then in Step 2, the MONTH entity types are merged into one consisting of all the entities {*jan, ..., dec*}; the SUP relationship types are also merged, and get the schema of DB1 in Fig. 1.□

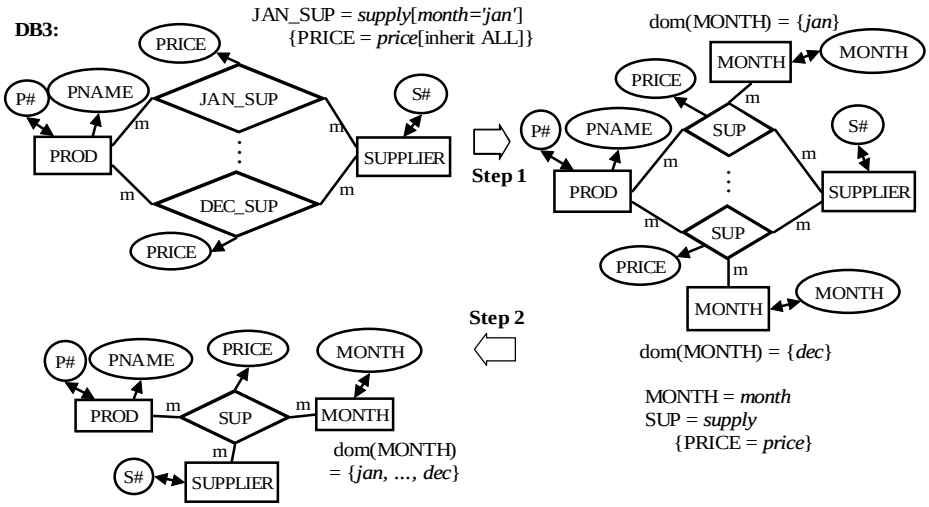


Fig. 3. Resolve schematic discrepancies for relationship types

4.2 Resolving Schematic Discrepancies for Attributes

In this sub-section, we first show how to resolve discrepancies for attributes of entity types using an example, then present Algorithm TRANS_ENT_ATTR in general. Finally, we describe the resolution of discrepancies for attributes of relationship types by an example, omitting the general algorithm which is similar to TRANS_ENT_ATTR.

The following example shows how to resolve discrepancies for attributes of entity types. Note the discrepancies of entity types should be resolved before this step.

Example 5 (Fig. 4). Suppose we have another database DB4 recording the supplying information, in which all the suppliers and months are modeled as contexts of the attributes in an entity type PROD. The transformation is given in Fig. 4. In Step 1, for each attribute with discrepant meta-attributes, say S1_JAN_PRICE, the meta-attributes *supplier* and *month* are represented as entity types SUPPLIER and MONTH consisting of one entity *s1* and *jan* respectively. A relationship type SUP is

constructed to connect PROD, MONTH and SUPPLIER. After removing the context, we change S1_JAN_PRICE into PRICE, an attribute of the relationship type SUP. Then in Step 2, we merge all the corresponding entity types, relationship types and attributes, and get the schema of DB1 in Fig. 1. □

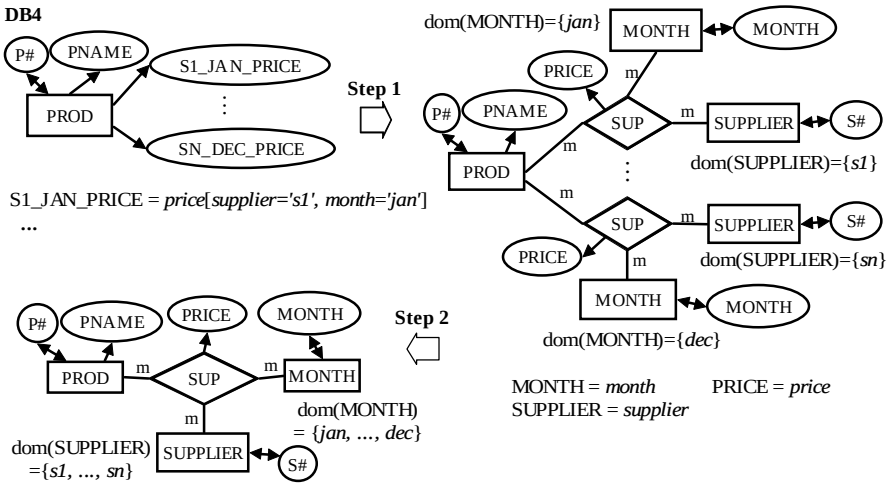


Fig. 4. Resolve schematic discrepancies for attributes of entity types

Then we give the general algorithm below.

Algorithm TRANS_ENT_ATTR

Input: an elevated schema DB.

Output: a schema DB' transformed from DB such that all the discrepant meta-attributes of attributes of entity types are transformed into entity types.

Step 1: Resolve the discrepant meta-attributes of an attribute in an entity type.

Given an entity type E of DB, let $A = A_{ont}[C_1=c_1, \ldots, C_m=c_m]$ be an attribute (not part of the identifier) of E, for A_{ont} a type in the ontology, and $C_1, \ldots, C_m$ the discrepant meta-attributes with the values $c_1, \ldots, c_m$. *// Note A has no inherited context which has been removed in Algorithm TRANS_ENT if any.*

// Represent the discrepant meta-attributes as entity types.

Construct entity types $E_{C_i} = C_i$ with identifiers $K_{C_i} = C_i$ for each meta-attribute $C_i \in \{C_1, \ldots, C_m\}$. Each E_{C_i} contains one entity c_i.

If A is a m:1 or m:m attribute, **then**

//Construct a relationship type to represent the associations among the entities of E and the values of $C_1, \ldots, C_m$.

Construct a relationship type R connecting the entity types E and $E_{C_1}, \ldots, E_{C_m}$.

Attribute A' = A_{ont} becomes an attribute of R.

Else *// A is a 1:1 or 1:m attribute, i.e., the values of A determines the entities of E in the context. A should be modeled as an entity type to preserve the cardinality constraint.*

Construct $E_{A'} = A_{ont}$ with the identifier A' = A_{ont}.

Construct a relationship type R connecting the entity types E, $E_{A'}$, and $E_{C_1}, \ldots, E_{C_m}$.

Represent the FD $\{A', C_1, \ldots, C_m\} \rightarrow K$ as the cardinality constraint on R.

If A is a 1:1 attribute, also represent the FD $\{K, C_1, \ldots, C_m\} \rightarrow A'$ on R.

Step 2: Merge the entity types, relationship types and attributes respectively which correspond to the same ontology type with the same context, and union their domains. □

The resolution of schematic discrepancies for the attributes of relationship types is similar to that for the attributes of entity types, as a relationship type could be treated as a high level entity type. We omit the resolution algorithm TRANS_REL_ATTR for lack of space, but explain it by an example below.

Example 6 (Fig. 5). Given the transformed schema of Fig. 2, we transform the attributes of the relationship type R as follows. In Step 1, for each attribute of R, say S1_PRICE, we represent the meta-attribute *supplier* as an entity type SUPPLIER with one entity *s1*, and construct a relationship type SUP to connect the relationship type R and entity type SUPPLIER. After removing the context, we change S1_PRICE into PRICE, an attribute of SUP. Then in Step 2, we merge the SUPPLIER entity types and SUP relationship

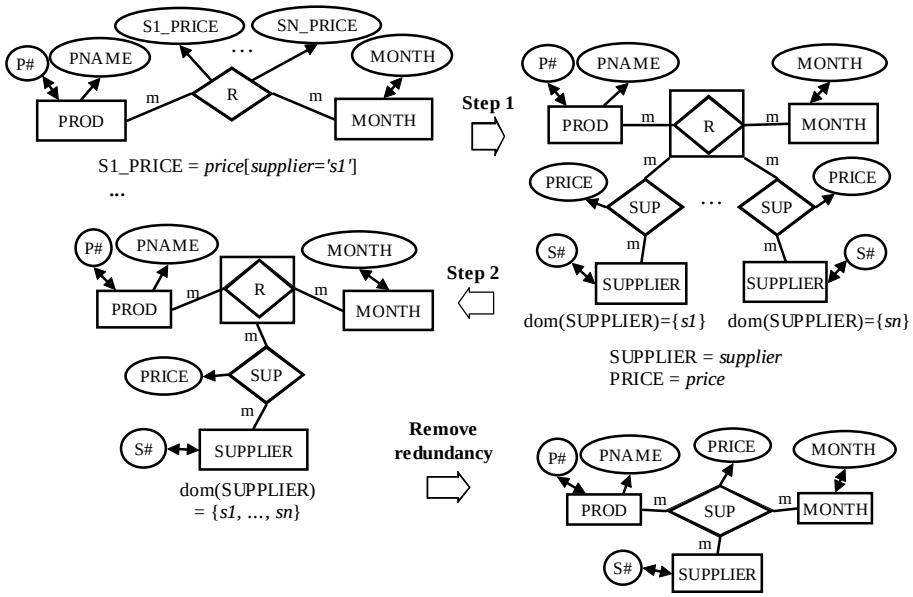


Fig. 5. Resolve schematic discrepancies for attributes of relationship types

types respectively. In the merged schema, the relationship type R is redundant as it is a projection of SUP and has no attributes. Consequently, we remove R and get the schema of DB1 in Fig. 1. □

The transformations of the algorithms (in Section 4.1 and 4.2) correctly preserve the FDs/MVDs in the presence of context, as shown in the following proposition.

Proposition 1. Let $\mathcal{E}$ be a set of similar entity types (or relationship types) with the same set of discrepant meta-attributes, and K be the global identifier of $\mathcal{E}$ (or a set of global identifiers of entity types if $\mathcal{E}$ is a set of relationship types). Suppose each entity type (or relationship type) of $\mathcal{E}$ has a set of attributes with the same cardinality:
$\mathcal{A} = \{A \mid A = A_{ont}[C_1=c_1, \ldots, C_m=c_m, \text{inherit } C_{m+1}, \ldots, C_n], c_i \in dom(C_i) \text{ for } 1 \leq i \leq m\}$.
Then in the transformed schema, $C_1, \ldots, C_n$ are modeled as entity types, and the following FDs/MVDs hold:

Case 1: $\mathcal{A}$ are m:1 attributes. Then A_{ont} is modeled as an attribute $A' = A_{ont}$, and a FD $\{K, C_1, \ldots, C_n\} \rightarrow A'$ holds.

Case 2: $\mathcal{A}$ are m:m attributes. Then A_{ont} is modeled as an attribute $A' = A_{ont}$, and a MVD $\{K, C_1, \ldots, C_n\} \twoheadrightarrow A'$ holds.

Case 3: $\mathcal{A}$ are 1:1 attributes. Then A_{ont} is modeled as an entity type with the identifier $A' = A_{ont}$, and FDs $\{K, C_1, \ldots, C_n\} \rightarrow A'$ and $\{A', C_1, \ldots, C_n\} \rightarrow K$ hold.

Case 4: $\mathcal{A}$ are 1:m attributes. Then A_{ont} is modeled as an entity type with the identifier $A' = A_{ont}$, and a FD $\{A', C_1, \ldots, C_n\} \rightarrow K$ holds. □

For lack of space, we only prove Case 1 when $\mathcal{E}$ is a set of entity types. In a transformed schema, given two relationships with values on A': $(k, c_1, \ldots, c_m, a)$ and $(k', c_1', \ldots, c_m', a')$ for k and k' values (or value sets) of K, $c_1, \ldots, c_m$ and $c_1', \ldots, c_m'$ values of $C_1, \ldots, C_m$, and a and a' values of A'. If $k=k'$, $c_1=c_1'$, $c_2=c_2', \ldots, c_m=c_m'$, then in the original schemas, the two relationships correspond to the same entity and same attribute, say $A1 \in \mathcal{A}$. As A is a m:1 attribute, we have a=a'. That is, the FD $\{K, C_1, \ldots, C_n\} \rightarrow A'$ holds in the transformed schema.

In schema integration, schematic discrepancies of different schema constructs should be resolved in order, i.e., first for entity types, then relationship types, finally attributes of entity types and attributes of relationship types. The resolutions for most of the other semantic heterogeneities (introduced in Section 1) follow the resolution of schematic discrepancies.

5 Related Work

Context is the key component in capturing the semantics related to the definition of an object or association. The definition of context as a set of meta-attributes with values is originally adopted in [7, 23], but is used to solve different kinds of semantic heterogeneities. Our work complements rather than competes with theirs. Their work is based on the context at the attribute level only. We consider the contexts at different levels, and the inheritance of context.

A special kind of schematic discrepancy has been studied in multidatabase interoperability, e.g. [11, 12, 16, 17, 19], and [2]. They dealt with the discrepancy when schema labels (e.g., relation names or attribute names) in one database correspond to attribute values in another. However, we use contexts to capture meta-information, and solve a more general problem in the sense that schema constructs could have multiple (instead of single) discrepant meta-attributes. Furthermore, their works are at the "structure level", i.e., they did not consider the constraint issue in the resolution of schematic discrepancies. However, the importance of constraints can never be overestimated in both individual and multidatabase systems. In particular, we preserve FDs and MVDs during schema transformation, which are expressed as cardinality constraints in ER schemas.

The purposes are also different. Previous works focused on the development of a multidatabase language by which users can query across schematic discrepant databases. However, we try to develop an integration system which can detect and resolve schematic discrepancies automatically given the meta-information on source schemas.

The issue of inferring view dependencies was introduced in [1, 8]. However, their works are based on the views defined using relational algebra. In other words, they did not solve the inference problem in the transformations between schematic discrepant schemas. In [14, 21, 24], people have begun to focus on the derivation of constraints for integrated schemas from constraints of component schemas in schema integration. However, these works failed to consider schematic discrepancy in schema integration. Our work complements theirs.

6 Conclusions and Future Works

Information integration provides a competitive advantage to businesses, and becomes a major area of investment by software companies today [18]. In this paper, we resolve a common problem in schema integration, schematic discrepancy in general, using the paradigm of context. We define context as a set of meta-attributes with values, which could be at the levels of databases, entity types, relationship types, and attributes. We design algorithms to resolve schematic discrepancies by transforming discrepant meta-attributes into entity types. The transformations preserve information and cardinality constraints which are useful in verifying lossless schema transformation, schema normalization and query processing in multidatabase systems.

We have implemented a schema integration tool to semi-automatically integrate schematic discrepant schemas from several relational databases. Next, we'll try to extend our system to integrate databases in different models and semi-structured data.

References

[1] S. Abiteboul, R. Hull, and V. Vianu: Foundations of databases. Addison-Wesley, 1995, pp 216-235

[2] R. Agrawal, A. Somani, Y. Xu: Storing and querying of e-commerce data. VLDB, 2001, pp 149-158
[3] C. Batini, M. Lenzerini: A methodology for data schema integration in the Entity-Relationship model. IEEE Trans. on Software Engineering, 10(6), 1984
[4] C. Batini, M. Lenzerini, S. B. Navathe: A comparative analysis of methodologies for database schema integration, ACM Computing Surveys, 18(4), 1986, pp 323-364
[5] P. P. Chen: The entity-relationship model: toward a unified view of data. TODS 1(1), 1976
[6] A. Elmagarmid, M. Rusinkiewicz, A. Sheth: Management of heterogeneous and autonomous database systems. Morgan Kaufmann, 1999
[7] C. H. Goh, S. Bressan, S. Madnick, and M. Siegel: Context interchange: new features and formalisms for the intelligent integration of information. ACM Transactions on Information Systems, 17(3), 1999, pp 270-293
[8] G. Gottlob: Computing covers for embedded functional dependencies. SIGMOD, 1987
[9] C. N. Hsu and C. A. Knoblock: Semantic query optimization for query plans of heterogeneous multidatabase systems. TKDE 12(6), 2000, pp 959-978
[10] Qi He, Tok Wang Ling: Extending and inferring functional dependencies in schema transformation. Technical report, TRA3/04. School of Computing, National University of Singapore, 2004
[11] R. Krishnamurthy, W. Litwin, W. Kent: Language features for interoperability of databases with schematic discrepancies. SIGMOD, 1991, pp 40-49
[12] V. Kashyap, A. Sheth: Semantic and schematic similarity between database objects: a context-based approach. The VLDB Journal 5, 1996, pp 276-304
[13] Tok Wang Ling, Mong Li Lee: Issues in an entity-relationship based federated database system, CODAS, 1996, pp 60-69
[14] Mong Li Lee, Tok Wang Ling: Resolving constraint conflicts in the integration of entity-relationship schemas. ER, 1997, pp 394-407
[15] Mong Li Lee, Tok Wang Ling: A methodology for structural conflicts resolution in the integration of entity-relationship schemas. Knowledge and Information Sys., 5, 2003, pp 225-247
[16] L. V. S. Lakshmanan, F. Sadri, S. N. Subramanian: On efficiently implementing schemaSQL on SQL database system. VLDB, 1999, pp 471-482
[17] L. V. S. Lakshmanan, F. Sadri, S. N. Subramanian: SchemaSQL—an extension to SQL for multidatabase interoperability. TODS, 2001, pp 476-519
[18] N. M. Mattos: Integrating information for on demand computing. VLDB, 2003, pp 8-14
[19] R. J. Miller: Using schematically heterogeneous structures. SIGMOD, 1998, pp 189-200
[20] Web ontology language, W3C recommendation. http://www.w3.org/TR/owl-guide/
[21] M. P. Reddy, B.E.Prasad, Amar Gupta: Formulating global integrity constraints during derivation of global schema. Data & Knowledge Engineering, 16, 1995, pp 241-268
[22] A. P. Sheth and S. K. Gala: Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Computing surveys, 22(3), 1990
[23] E. Sciore, M. Siegel, A. Rosenthal: Using semantic values to facilitate interoperability among heterogeneous information systems, TODS, 19(2), 1994, pp 254-290
[24] M. W. W. Vermeer and P. M. G. Apers: The role of integrity constraints in database interoperation. VLDB, 1996, pp 425-435
[25] Xiaoying Wu, Tok Wang Ling, Mong Li Lee, and Gillian Dobbie: Designing Semistructured Databases Using ORA-SS Model. WISE, 2001, pp 171-180