2000

The Integration of Relationship Instances from Heterogeneous Databases

Ee Peng LIM
Singapore Management University, eplim@smu.edu.sg

Roger Hsiang-Li CHIANG

DOI: https://doi.org/10.1016/S0167-9236(00)00070-1

ELSEVIER

Decision Support Systems 29 (2000) 153–167

Decision Support Systems

www.elsevier.com/locate/dsw

The integration of relationship instances from heterogeneous databases

Ee-Peng Lim [a,*], Roger H.L. Chiang [b]

[a] *Centre for Advanced Information Systems, School of Applied Science, Nanyang Technological University, Nanyang Avenue, N4-2A-12, Singapore 639798, Singapore*

[b] *College of Business Administration, University of Cincinnati, Cincinnati, OH 45221-0211, USA*

Accepted 1 April 2000

Abstract

In the process of integrating legacy databases, one has to resolve inter-database conflicts at both the schema and instance levels. In this paper, we discuss relationship conflicts as a special type of conflicts to be resolved during the database integration. Relationships are properties that relate real world objects. So far, most inter-database relationship conflicts are addressed at the schema-level by various schema integration techniques. However, instance-level relationship conflicts are largely neglected. This paper therefore investigates the causes of instance-level relationship conflicts and proposes a taxonomy for classifying instance-level relationship conflicts. In addition, we develop a systematic process to resolve instance-level relationship conflicts and incorporate the resolution steps into the overall database integration process. Instance-level relationship conflict detection algorithms have also been developed to aid the resolution process. This research should facilitate database integration work for both multidatabase and data warehousing approaches. Most importantly, it should improve the data quality of the integrated databases.

Keywords: Instance-level relationship conflicts; Database integration; Multidatabases; Data warehouses

1. Introduction

In the past, much database integration research has focused on solving the schema integration problem which involves combining the conceptual schemas of heterogeneous databases [1,5,7,14]. Typically, schema integration assumes the availability of conceptual schemas for the databases involved, and the schemas are usually represented in the Entity-Relationship (ER) model [3]. Schema integration tasks include matching schema constructs (i.e. entity, relationship and attribute types) from different but related conceptual schemas, resolving schema construct conflicts, and deriving an integrated conceptual schema [1]. While schema integration is a well-established database research area and several fully and semi-automatic schema integration tools and techniques have been developed, the integration of database instances, in contrast, has not been addressed very much in database integration.

The integration of database instances is more difficult than schema integration. Firstly, integration of instances is carried out after schema integration

* Corresponding author. Tel.: +65-799-4802; fax: +65-792-6559.

E-mail addresses: aseplim@ntu.edu.sg (E.-P. Lim), roger.chiang@uc.edu (R.H.L. Chiang).

0167-9236/00/$ - see front matter
PII: S0167-9236(00)00070-1

and hence very much relies on the correctness of schema integration. Schema integration research and methods normally tend to assume that required semantics for database integration can be captured by their schemas and different kinds of *semantic-related assertions* [5,14,15]. As it is not easy to obtain required domain semantics and further exhaustively validate these semantics, schema integration process may generate an integrated schema that contains flaws. These flaws will later surface during the integration of data instances. Flaws resulted from schema integration therefore have to be detected and resolved during the instance integration process. Secondly, the volume of instances in databases for integration is much larger than their schema elements. The large volume of data prohibits a database integrator from performing the integration task by meticulously matching and reconciling the database instances without making use of domain knowledge and meta-data (e.g. the integrated schema) to guide the integration process.

For database integration, the schema integration is better performed at the conceptual level. Likewise, in order to perform correct instance integration, we advocate that instances should be integrated at the conceptual level making it necessary to tap the conceptual-level knowledge about local schemas and the integrated schema to perform correct instance integration. In our research, instances from the databases to be integrated are viewed as entity and relationship instances, and integrated entity and relationship instances are derived by combining these local entity and relationship instances.

The integration of entity instances has been studied in the *entity identification* and *attribute value conflict* [9,10]. There are a few solution techniques proposed for these two instance integration issues. However, the integration of relationship instances has not been properly addressed. There is also a lack of database integration methodology that incorporates the integration of not only entity, but also relationship instances. Without such a methodology, relationship instances captured as tuples in the underlying local databases may have to be treated as entity instances and be wrongly integrated without considering the semantics of relationship instances.

For example, assume that there are two databases containing overlapping but not identical sets of employee and project instances. Let EMPPROJ_A -(eno,pno) and EMPPROJ_B(eno,pno) be the relational tables storing information about the many-to-many relationship instances between employees and projects in the two databases. Let *e* and *p* be an employee instance and a project instance, respectively, and they can be found in both databases. They are found together in a EMPPROJ_A tuple but not in EMPPROJ_B. If tuples from the two tables are integrated as entity instances, it would not be possible to discover that the relationship between *e* and *p* is missing in EMPPROJ_B. If it is known that EMPPROJ_B contains more up-to-date information, one should ensure that the relationship instance between *e* and *p* be excluded from the integrated database. Unfortunately, this instance-level relationship conflict is usually not considered in the existing database integration approaches. By blindly treating relationship instances as entity instances and overlooking the semantics of relationships, one could result in incorrectly integrated databases.

In this paper, the integration of relationship instances from heterogeneous databases is studied as an essential part of instance integration. Our main objective is to develop an overall database integration methodology to address both the schema and instance integration issues by incorporating the relationship semantics. In summary, there are three research objectives.

- We present a novel database integration methodology that encompasses schema and instance integration tasks as well as other essential integration tasks such as reverse engineering of data instances. The resolution of instance-level relationships conflicts has been incorporated into the methodology. We also elaborate on the inter-dependence of schema and instance integration tasks. To our best knowledge, such a complete treatment of database integration problem has not been mentioned in the research literature.
- We provide a classification of instance-level relationship conflicts. The causes of different types of conflicts have also been identified.
- We outline a systematic process to detect and resolve the instance-level relationship conflicts. It is a step-by-step instance integration process that can be carried out by a database integrator manually or with some help from a semi-automatic tool.

Our research studies and probes the semantic issues of instance integration. It is different from existing data cleaning techniques [13] and tools [2] available in the market for data warehousing and data mining. Data cleaning methods focuses on automatic identification and correction of inconsistent data format and value coding together with incomplete and incorrect data in a target data warehouse. It focuses mainly on the syntactic inconsistency resolution in data consolidation and integration. However, without resolving semantic differences between instances, incorrectly integrated databases are most likely produced during data integration. Therefore, instance integration should consist of not only data cleaning but also semantic conflict identification and reconciliation.

In the following discussions, we assume that databases for integration are relational. We adopt ER model to represent the export conceptual schemas and instances, and the integrated databases. The reminder of this paper is organized as follows. In Section 2, we present our database integration methodology. This methodology divides the integration task into a set of processes, which can be adapted for use in either multidatabases or data warehouses. Section 3 discusses schema-level relationship conflicts. The discussion and classification of these conflicts will facilitate the understanding of relationship conflicts at the instance level. Section 4 examines the instance-level relationship conflicts and their causes. Sections 5 and 6 discuss the instance-level relationship conflicts detection and resolution, respectively. Section 7 provides the conclusions and discusses future research directions.

2. Overview of database integration

2.1. Physical vs. virtual database integration

Database integration is performed whenever two or more databases have to be combined together either physically or virtually. Physical database integration requires the original databases to be discarded after the integrated database has been constructed and all existing application software to be migrated to the database systems operating the integrated database. Virtual database integration, on the other, deploys a multidatabase or data warehousing system to support queries on an integrated *view* constructed upon the original databases. It retains both the original databases and its application software.

Regardless the mode of database integration, the basic integration tasks remain essentially the same. We view the entire database integration as a set of processes which derives the integrated schema and instances. For physical database integration, the schemas and instances of heterogeneous databases are combined at the time physical integrated databases are created. For virtual database integration, only the local schemas are integrated at the time integrated views are defined upon the heterogeneous databases. The integrated view definition specifies how the integrated instances can be obtained.

The physical instance integration is performed when the global applications/users issue queries for multidatabase systems. For data warehouse systems, local data instances are extracted and integrated prior to global query processing [16]. Therefore, the instance integration is performed according to the changes in the local databases rather than according to the ad hoc global queries. The global queries are evaluated directly against the data warehouse. To construct a data warehouse, *monitoring systems* must be implemented on the local databases to actively detect changes to them, and to propagate the changes to the data warehouse. These changes are transformed into updates to the data warehouse by a *data integrator subsystem.*

2.2. Database integration methodology

Our proposed database integration methodology is shown in Fig. 1. The methodology is unique in the way it considers not only the schema integration but also the instance integration. It highlights the requirement of integrating instance at the conceptual level making it possible to explore relationship semantics. It also presents database integration as a set of inter-related processes so that it can be tackled by solutions to inter-related sub-problems. There are two inter-related dichotomies of database integration issues. The first dichotomy is based on the database components (i.e. schemas and instances) for integration. The second one is based on the representation

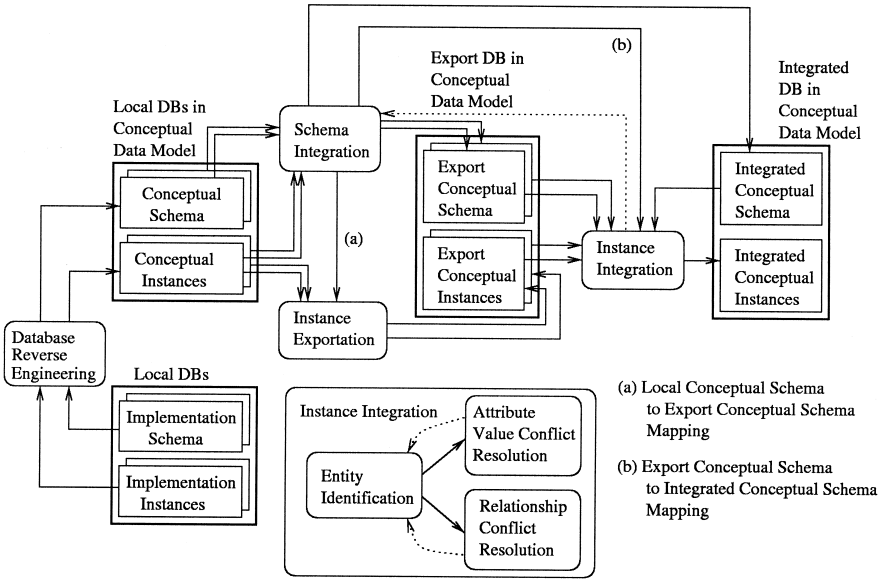

Fig. 1. Database integration methodology.

abstraction level of database components to be integrated (i.e. implementation data models and conceptual data models).

As most local databases are implemented on relational database systems, they are stored as implementation schemas and instances, i.e. relational tables and tuples. By performing *database reverse engineering* [4,12], the conceptual schemas and instances of these local databases can be derived. In some cases, the conceptual schemas of local databases are well documented and maintained by database designers and the reverse engineering process could focus only on deriving the conceptual entity and relationship instances from the local database tuples. Here, we have extended the traditional database reverse engineering process to include the task of reverse engineering instances. Traditionally, database reverse engineering research deals with only database schemas. There is no research done on the reverse engineering of database instances. However, it is better to reverse engineer and represent instances of heterogeneous databases for integration at the conceptual level to facilitate the integration work.

Schema integration combines the conceptual schemas of local databases into an integrated conceptual schema. Unlike traditional schema integration, the conceptual schemas of the local databases and some mapping information[1] are determined during the schema integration process. In schema integration, an export conceptual schema is defined for each local database according to the portion of its instances to be integrated (i.e. export database). The export schemas also serve to ensure that export databases look compatible before their instances are to be further integrated.

Based on the conceptual schemas of local databases, *instance exportation* can be performed on the instances in order to derive the conceptual instances for integration, i.e. entity and relationship instances. Instance exportation requires the instances from different local databases to be formatted according to the export conceptual schemas. Each entity instance consists of a unique entity identifier and the values for attributes prescribed to the corresponding entity type. Each relationship instance consists of the identifiers of participating entity instances and the values for attributes prescribed to the corresponding relationship type. The mappings from local conceptual schema to export conceptual schema, as part

[1] These are mappings from local conceptual schema to export conceptual schema, and from export conceptual schemas to integrated conceptual schema.

of the output of schema integration, facilitates the derivation of conceptual instances. Since export conceptual schemas are compatible with one another, instance integration can then be performed on the conceptual instances.

In this proposed database integration methodology, we examine *instance integration at conceptual level*. It involves three major tasks: *entity identification*, *attribute value conflict resolution*, and *relationship conflict resolution*. Entity identification involves matching local database instances that correspond to the same real world objects. Attribute value conflict resolution handles the differences between attribute values of matching local database instances. In some ways, entity identification and attribute value conflict resolution for instance integration are similar to matching entity and attribute types, respectively, at the schema level. In previous research, entity identification and attribute value conflict resolution have been investigated within the implementation data model, e.g. relational. Relationship conflict resolution reconciles the different instance values that represent the same real-world relationship. Relationship conflicts can occur due to undetectable flaws during schema integration, entity identification, or inconsistent content of local databases.

A close relationship exists between entity identification and relationship conflict resolution. Each relationship instance is defined by a set of inter-related entity instances. In the case of an integrated database, a relationship instance is derived from one or more local relationship instances from the local databases. When the local entity instances involved in these relationship instances are identified wrongly in the integrated database, the integrated relationship instance will inevitably be incorrect. Nevertheless, erroneous entity identification is not the only cause of relationship conflicts. Further details about the causes of instance-level relationship conflicts will be given in Section 4.1.

To carry out instance integration, the mapping from export conceptual schemas to integrated conceptual schema has to be provided by schema integration. As shown in Fig. 1, entity identification is performed first in order to obtain the matching instances for attribute value and relationship conflict resolutions. The dotted edge from relationship conflict resolution to entity identification indicates that erroneous entity identifications may be detected during relationship conflict resolution. The detected erroneous entity identification has to be fedback for resolution. Similarly, the dotted edge indicates that flaws in the integrated schema may be detected during instance integration. The detected flaws should be fedback to schema integration for correction. Finally, the integrated schema and instances are transformed into an implementation data model (e.g. relational) that are later utilized by the new applications developed on the integrated database. The transformation is performed as part of a *database mapping* process.

As illustrated by the example given in Section 1, relationship conflicts at the instance level have been largely neglected by the database integrators as local instances are often integrated at the implementation level without considering their conceptual level semantics. Therefore, in this paper, we focus on the resolution of instance-level relationship conflicts. Our proposed integration methodology attempts to overcome the deficiency of previously proposed integration approaches in dealing with this problem. To do so, we first examine the schema integration of relationship types. We point out that this integration work does not necessarily resolve instance-level relationship conflicts. However, this is a common misbelief in database integration. Then, we discuss the classification, detection and resolution of instance-level relationship conflicts, respectively, in Sections 4 to 6.

3. Relationship conflicts at schema level

At the schema level, relationship conflicts occur when associations in real world objects are modeled differently in the conceptual schemas of local databases. Typically, schema-level relationship conflicts are handled for the relationship types among those local entity types to be included into the integrated schema. Hence, the matching entity types from different local conceptual schemas have to be determined before one can proceed to resolve schema-level relationship conflicts.

For example, Fig. 2 depicts the conceptual schemas of two local databases A and B that are to

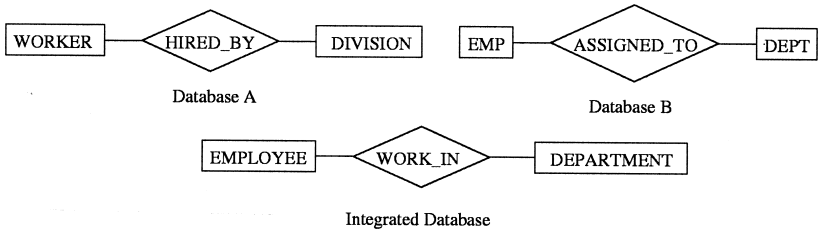

Fig. 2. Schema-level relationship conflicts.

be integrated. Having determined that the entity types WORKER and DIVISION from database A matches with entity types EMP and DEPT from database B, respectively, one proceeds to create the corresponding EMPLOYEE and DEPARTMENT entity types in the integrated schema. After the matching entity types have been identified, one has to determine if the relationship type HIRED_BY matches with ASSIGNED_TO from database B. As shown in the figure, the two relationship types have been matched and integrated into the WORK_IN relationship type in the integrated schema.

Note that the above example scenario captures only one type of schema-level relationship conflicts. There are other types of schema-level relationship conflicts that require attention during schema integration. Interestingly, in the traditional schema integration research [8,11,14], schema-level relationship conflicts are rarely mentioned. Some classification of schema level conflicts has been proposed in Refs. [6,11]. Schema-level conflicts have been classified into *entity-vs.-entity*, *attribute-vs.-attribute*, *entity-vs.-attributes*, and *different representation for equivalent data* in Ref. [6]. On the other hand, schema-level conflicts are classified into *naming*, *structural*, *key*, *cardinality* and *domain conflicts* in Ref. [11]. We adopt the classification scheme proposed in Ref. [11] to classify schema-level relationship conflicts as follows.

- *Naming conflicts*: Naming conflicts occur when different names are used for relationship types that are semantically the same, or when same names are used for semantically different relationship types. For example, a WORK_IN relationship type in a conceptual schema may correspond to WORK_FOR relationship type in another conceptual schema. This is an example of synonym problems. Note that homonym problem can also occur when the local relationship types with the same name actually carry different semantics. For example, there are two IS ASSIGNED_TO relationships between EMPLOYEE and DEPARTMENT. One of the relationships is the employee work-for relationship. Another is the manager-assignment relationship between employees and departments.
- *Structural conflicts*: Relationship types in the integrated database may be modeled by different schema constructs in the local databases. For example, at the conceptual level, a real world association may be represented as a relationship type in one local database but as an entity type in another local database. Often, structural conflicts at the logical schema level arise from the differences in representing the one-to-many $(1{:}N)$ binary relationship types. A $1{:}N$ binary relationship type can be represented as a foreign key into an entity relation with the relationship attributes as the non-key attributes of the entity relation. Or, it can be represented by an individual relationship relation.
- *Identifier conflict*: Different identifiers are used for uniquely determining relationships. The identifier of a relationship type is usually defined by the concatenation of key attributes of the participating entity types, or the relationship type's own attribute(s). If different identifiers are used in local databases for the same relationship type, they have to be resolved.
- *Cardinality conflict*: Cardinality conflicts arise when the local databases have different cardinalities for the same relationship type in the integrated schema. For example, a local database may have one-to-many $(1{:}N)$ WORK_IN relationship type while another local database may have many-to-many $(N{:}M)$ WORK_IN relationship type.
- *Domain conflict*: Domain conflicts arise when the attributes of local databases corresponding to the

same relationship type in the integrated schema have different domain values, or data types. For example, a WORK_IN relationship type in a local database may have start date represented as a text attribute while WORK_IN relationship type in another local database may have start date represented as three integer attributes, i.e. DAY, MONTH and YEAR.

4. Relationship conflicts at instance level

Instance-level relationship conflicts occur when export instances are not related to one another in a consistent manner, or when export instances are not consistent with the relationship types defined in the integrated schema. Like other instance-level conflicts, these conflicts cannot be addressed by examining the schemas alone. In this section, we identify different types of instance-level relationship conflicts and their causes.

4.1. Causes of instance-level relationship conflicts

Instance-level relationship conflicts are resulted from three most common categories of problems.

4.1.1. Incorrect or stale schema integration

Schema integration is itself a complicated process that may require users (or DBA) supply domain knowledge, schema as well as instance-level semantics. However, it is possible that users may supply incorrect knowledge. In some cases, users may give semantics that are not validated against local databases for performance reasons. It is also possible that semantics extracted from local databases for schema integration may not be valid for the same databases some time later for instance integration. For example, starting from 1999, employees can work for more than one department at the same time. Therefore, the relationship cardinality derived previously during schema integration is not valid anymore for current instance integration. Any such errors will contribute to either incorrect or stale schema integration as well as instance-level conflicts including relationship conflicts.

4.1.2. Incorrect instance-level entity identification

Instance-level entity identification essentially matches entity instances from different local databases representing same real world objects. Entity identification also requires knowledge given by users. For example, to integrate an Employee database containing employee id as key with another database containing social security number as key, one may have to rely on the name and age attributes to match the employee entity instances. If the two attributes fail to distinguish some pairs of entity instances, one may have to refer to additional attributes such as address, phone, etc. A rule-based approach to entity identification has been given in Ref. [9]. Similar to schema integration, errors in the given knowledge will affect the entity identification outcome as well as the relationship conflict resolution. In Ref. [9], several entity identification approaches have been mentioned. These approaches differ in their *identity rules*, which are criteria used for matching entity instances, as well as other knowledge about the databases.

4.1.3. Instance inconsistencies existed in local databases

Inter-database conflicts can be manifested at the instance level in many ways. Even when export databases consist of compatible schemas, it is still possible to find some export instances in one database but not in another, and corresponding instances from different export databases having different values. For example, one database may indicate that an employee is not affiliated to any department. However, another local database indicates that the same employee works for some department. We classify the common causes of instance inconsistency in database integration as follows.

- Incomplete data: Databases for integration may contains missing instances, or null values which may be either resulted from no verification and editing during data entry, or various update anomalies.
- Incorrect data: It is similar to the previous case that databases for integration may contains inconsistent data which may be either entered without verification and editing, resulted from invalid updates, or inconsistent calculation for the derived attributes. For example, the sales data for a firm or an order can be calculated by different functions, or even with the same function at different time using different currency exchange rates.

• Representation inconsistency: When databases for integration are developed autonomously, there may easily exist inconsistent representations (e.g. format, data type, measurement unit, precision level, coding, etc.) to capture same data. We call them instance-level synonyms. Another type of representation inconsistency is instance-level homonyms, the same data from different databases represent different real world objects. For example, two warehouses use the same item code for different products. The representation inconsistencies can be detected at the schema level with domain or schema information, and reconciled at the instance level through mapping functions derived from schema integration. However, some of them can only detected by examining data instances. For example, two databases have the same attribute for sex, which has the same data type, size and data domain. But, one database uses 0 and 1 to represent male and female, respectively, and the other uses these two codes vice verse. Another common representation inconsistency example is the date format. For instance, the date, August 9, 1998, can be represented as 8/9/98 in one database and 9/8/98 in another database. However, information regarding the coding of sex and the format of date is not available during the schema integration.

• Time-variancy (temporal conflicts): Temporal conflicts arise when local databases for integration capture instance values of real world objects at different moments of time. This is due to that databases for integration may have different requirement of timeliness of data. This difference will result in update inconsistency. The data instances from local databases may capture values with different time snapshots. For example, there should be only one salary level for each employee. However, if an organization has multiple databases capture employees' information, due to the update inconsistency, there may exist more than one salary level for an employee. The temporal conflicts may be caused by the evolution of real world objects across time. For example, the marketing department was split into three departments, sales, customer service and adverting on January 1, 1998, and the sales and accounting departments were merged into one department called finance on September 1, 1998. All employees in these departments should be reassigned into the new departments accordingly. During instance integration, there may exist WORK_IN relationship instances which have not been updated. Department reassignment table for involved employees should be established to facilitate the detection and resolution of this type of temporal conflicts.

• Instance heterogeneities: Some instance inconsistencies are real heterogeneities which should be accommodated and retained for processing global queries in the integrated database. These heterogeneities are resulted from different observation, judgement, business rules and requirements, or application need. For example, different departments may have different definitions of the volume of 'inventory' and 'sales'. To support business transactions' needs, a firm may assign multiple classification codes for the same product. Furthermore, different departments may assign different sets of classification codes for the same supplier. From the business point of view, these heterogeneities are necessary to reflect in the integrated database. Then the challenge in instance integration is to accommodate these heterogeneities rather than to adopt only one sales definition (calculation), one code for each item, or the same set of classification codes for each supplier in the integrated database. In order to accommodate instance heterogeneities, we need to acquire and adopt these heterogeneous definitions, business rules and requirements in the integration process.

Without knowing the domain (context) knowledge of each database for integration along with the global requirements of the integrated database, sometimes, it is difficult to determine whether an instance conflict is an inconsistency to reconcile or a heterogeneity to accommodate. In addition, most of the required knowledge for instance integration is not captured by the data dictionaries of local databases. Therefore, it is better to acquire and accumulate instance integration knowledge during the integration process.

4.2. Taxonomy of instance-level relationship conflicts

Since an instance-level relationship conflict may be resulted from different causes, we must not only detect the conflicts, but also find out the exact cause of each conflict in order to resolve or accommodate the each conflicts accordingly. Therefore, we classify the instance-level relationship conflicts according to

the way that they can be detected into the following three categories:

- Relationship Cardinality Conflicts
- Missing Relationship Instances
- Inconsistent Relationship Attribute Values.

We use an integration example to discuss and illustrate each type of instance-level relationship conflicts . Let A and B be two local databases to be integrated. Suppose the schemas of these local databases have undergone schema integration and the integrated schema is derived. Using the integrated schema, we also determine the schema and instances to be exported from A and B. The integrated schema and the export schemas are therefore compatible. To simplify our explanation, we assume that they are identical and the schema is shown in Fig. 3.

As shown in Fig. 3, each export schema consists of the Product and Factory entity types. Each factory can only produce one product. Some of the factories have the capacity to store the manufactured products. To keep the stock-taking process simple, the stock of each product can only be stored at one factory. The quantity of each product stored in a factory is captured in the schema.

4.2.1. Relationship cardinality conflicts

While export instances always conform to the relationship cardinalities in the export schemas, exports instances after integration may not always conform to the relationship cardinalities in the integrated schema.

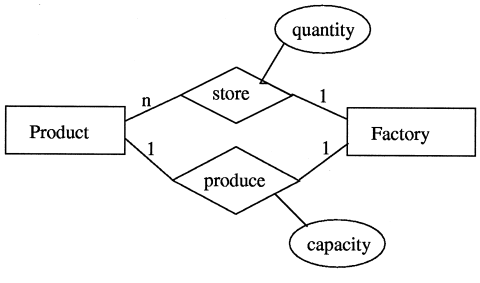

Fig. 3. The integrated schema and export schemas for databases A and B.

Example. The 1:1 PRODUCE relationships in export schema A and export schema B, are integrated into a 1:1 PRODUCE relationship in the integrated schema. At the instance level, product P1 is produced by F1 in export database A but is produced by F2 in export database B where F1 and F2 do not represent the same real world factory. Conflicts with the relationship cardinalities in the integrated schema can be attributed to the following.

Incorrect integrated schema: In the example, the conflict may be caused by errors made during schema integration. Instead of 1:1 PRODUCE relationship type, one may want to choose 1:*M* PRODUCE relationship type for the integrated schema. Clearly, when a relationship cardinality in the integrated schema can be wrongly determined, it may not be possible to detect the error unless some export instances are found to violate the constraint.

Incorrect entity identification: In the above example, the conflict may be caused by two export FACTORY instances wrongly identified to represent the same factory entity. This can occur due to two reasons. First, there exist instance inconsistencies between the export databases. For example, the factory information of the real world object represented by F1 in export database B may not be updated. Second, there are some errors in the knowledge used to perform entity identification. For example, we may have wrongly used factory name to determine if export factory instances correspond to the same real world factory object (i.e. homonym and synonym problems at the instance level).

Erroneous local data: When local relationship instances are created with errors, they may not be detectable in the local databases. However, when erroneous relationship instances from multiple databases are to be integrated together, the errors may be revealed as relationship cardinality conflicts. For example, the relationship instance associating P1 and F2 may be incorrect. In reality, P1 should not be produced by F2.

Temporal conflicts: The conflict may be caused by the time variancy of local databases' data. Temporal conflicts arise when local databases for integration capture snapshots of real world associ-

ations at different moments of time. For example, product P1 can be produced by both factories F1 and F2 as shown in local databases A and B, respectively. Actually, each product can be produced by only one factory at a time, and P1 is produced by F1 and F2 at different points in time. To resolve this type of conflicts, integrated schema should be modified (e.g. including the time stamp attribute) to incorporate time semantics of relationships.

4.2.2. Missing relationship instances

This type of conflicts arises when a relationship instance from one export database cannot be found in another export database.

Example. In export database A, product P1 is stored in factory F1 but in export database B, such relationship instance does not exist between P1 and F1. When we say that relationship instance does not exist in export database B, it can be due to: (1) P1 or F1 or both are not represented in export database B, or (2) P1 and F1 are represented in export database B but there is not relationship instance between them. This type of conflict can be caused by the following factors.

Incorrect entity identification: The P1 and F1 have not been properly identified in export database A or B, or both.

Erroneous local data: It is possible that P1 should actually be stored in a factory different from F1 in database A. It is also possible that the information about the factory storing P1 is erroneous in database B. Both kinds of errors in local relationship instances will therefore lead to missing relationship instances.

Temporal conflict: In this case, an export database just does not capture the relationship. This is possible when one or both of the export databases is not up-to-date.

4.2.3. Inconsistent relationship attribute values

This kind of conflicts arises when export relationship instances identified to represent the same real world association do not share the same attribute value(s). Their corresponding attribute values may be inconsistent, or some may be missing. It can be detected by examining the export relationship instances.

Example. The 1:1 PRODUCE relationships in export schema A and export schema B are integrated into a 1:1 relationship in the integrated schema. Suppose product instance P1 is produced by the factory F1, product P2 is produced by factory F2, P1 and P2 have been determined to represent the same product entity, and F1 and F2 have been determined to represent the same factory entity. However, the two export relationships have conflicting relationship attributes' values. For example, P1 is produced by F1 with a capacity different from that between P2 and F2. Or, one of the capacities is missing. The conflicts can be caused by the following.

Incorrect integrated schema: When the two capacities carry different semantics (e.g. capacity at peak or average capacity), it is clearly incorrect to consider the relationship attributes with the same name attribute to be the same attribute (homonym problem).

Incorrect entity identification: This can occur when: (1) there are instance inconsistencies among the export databases, or (2) there are some errors in the knowledge used to perform entity identification, for example, either F1 and F2, or P1 and P2 are homonyms.

Instance inconsistencies of relationship attributes: The inconsistency may be caused by the error in one or both of the capacities.

Temporal conflict: Inconsistent relationship attribute values can also be caused by time differences in updating the databases involved.

5. Instance-level relationship conflict detection

In this section, we outline algorithms for detecting instance-level relationship conflicts. As pointed out in Section 4, the instance-level relationship conflicts can be detected in three ways, i.e. relationship cardinality conflicts, missing relationship instances, and inconsistent relationship attribute values. Hence, we have developed the appropriate algorithms for detect-

ing the three types of anomalies. To simplify our explanation, we have restricted the discussion to binary relationship types. The algorithms can be further extended to handle relationship types involving more than two entity types.

Since instance-level relationship conflicts are only meaningful after entity identification is performed on the export entity instances, we assume that one-to-one mappings between export entity instances have been established. Each matching pair of entity instances is represented by the pair consisting of their key values. In other words, as part of entity identification, a mapping table *MapEntity* is constructed for a pair of entity types (say $E1$ and $E2$) representing the same global entity type (say E) as follows:

$$\begin{aligned}\text{MapEntity}_{(E1,E2)} &= \{(e_1 \cdot \text{key}, e_2 \cdot \text{key}) \mid e_1 \in E1, e_2 \in E2, \quad \text{and} \\ &\quad \text{they correspond to the same real-world entity}\} \\ &\cup \{(e_1 \cdot \text{key}, \text{NULL}) \mid e_1 \in E1, \\ &\quad e_1 \text{ does not have matching entity in } E2\} \\ &\cup \{(\text{NULL}, e_2 \cdot \text{key}) \mid e_2 \in E2, \\ &\quad e_2 \text{ does not have matching entity in } E1\}\end{aligned}$$

5.1. Relationship cardinality conflict detection

Let e_1 be an export entity instance of entity type $E1$, and e_2 be an export entity instance of entity type $E2$ where $E1$ and $E2$ are integrated into a global entity type E. Let $E1$ and $F1$ be related by a relationship type $R1$ and $E2$ and $F2$ be related by a relationship type $R2$ such that $F1$ ($R1$) and $F2$ ($R2$) are matching entity types (relationship types). In other words, these entity types and relationship types should be integrated together, as shown in Fig. 4. Given the export entity instance e_1 (e_2), we define the $F1$ ($F2$) instances that are related to e_1 (e_2) by the following functions:

$$\text{Inst}(e_1, R1) = \{f_i \cdot \text{key} \mid e_1 \text{ is related to } f_i \text{ via } R1\}$$

$$\text{Inst}(e_2, R2) = \{f_j \cdot \text{key} \mid e_2 \text{ is related to } f_j \text{ via } R2\}.$$

Note that Inst(e_1; $R1$) (Inst(e_2; $R2$)) returns an empty set if e_1(e_2) is not related to any $F1$($F2$) instances via $R1$($R2$).

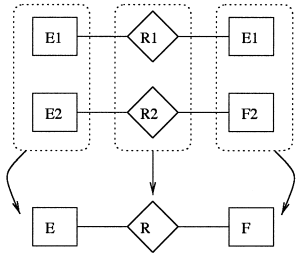

Fig. 4. Integrating matching entity and relationship types.

To check for relationship cardinality conflicts, the following algorithm is used.

- Step 1: For each pair of matching relationship types, say $R1$ and $R2$, do Step 2.
- Step 2: For each entity pair $(ek_1, ek_2) \in \text{MapEntity}_{(E1, E2)}$, $ek_1 \neq \text{NULL}$, $ek_2 \neq \text{NULL}$, do Steps 3 and 4.
- Step 3: Evaluate:

$$FK1 = \text{Inst}(e_1, R1) \text{ where } e_1 \cdot \text{key} = ek_1$$

$$FK2 = \text{Inst}(e_2, R2) \text{ where } e_2 \cdot \text{key} = ek_2$$

$$T1 = \{(fk_1, fk_2) \mid fk_1 \in FK1, fk_2 \in FK2 \text{ and } (fk_1, fk_2) \in \text{MapEntity}_{(F1, F2)}\}.$$

- Step 4: Check if $|T1|$ is consistent with the relationship cardinality assigned to F of R. For example, if the relationship cardinality assigned to F is 1, $|T1|$ should be ≤ 1. If the above condition is not met, the relationship cardinality conflict is reported.

Note that the above detection is not required for those global relationship types with many-to-many cardinalities.

5.2. Missing relationship instance detection

To detect missing relationship instances, the following algorithm is used:

- Step 1: For each pair of matching relationship types, say $R1$ and $R2$, do Step 2.
- Step 2: For each $(e_1, \cdot \text{key}, e_2 \cdot \text{key}) \in \text{MapEntity}_{(E1, E2)}$ such that $e_1 \cdot \text{key} \neq \text{NULL}$ and $e_2 \cdot \text{key} \neq \text{NULL}$, for each $fk_1 \in \text{Inst}(e_1, R1)$, do Steps 3 and 4.

- Step 3: Find fk_2 such that $(fk_1, fk_2) \in \text{MapEntity}_{(F1, F2)}$, $fk_2 \neq \text{NULL}$.
- Step 4: Check if $fk_2 \in \text{Inst}(e_2, R2)$. If the above condition is not met, a missing relationship instance is found.

5.3. Inconsistent relationship attribute value detection

Inconsistent relationship attribute values can only occur for those relationships that have relationship attributes. For such relationship types, the following conflict detection algorithm can be used.

- Step 1: For each pair of matching relationship types, say $R1$ and $R2$, do Step 2.
- Step 2: For each entity pair $(e_1 \cdot \text{key}, e_2 \cdot \text{key}) \in \text{MapEntity}_{(E1, E2)}$ such that $e_1 \cdot \text{key} \neq \text{NULL}$ and $e_2 \cdot \text{key} \neq \text{NULL}$, for each entity pair $(f_1 \cdot \text{key}, f_2 \cdot \text{key}) \in \text{MapEntity}_{(F1, F2)}$ such that $f_1 \cdot \text{key} \in \text{Inst}(e_1, R1)$, $f_2 \cdot \text{key} \in \text{Inst}(e_2, R2)$, $f_1 \cdot \text{key} \neq \text{NULL}$ and $f_2 \cdot \text{key} \neq \text{NULL}$, do Step 3.
- Step 3: If the relationship attributes of $(e_1, R1, f_1)$ are different from those of $(e_2, R2, f_2)$, a relationship attribute value conflict is detected.

6. Instance-level relationship conflicts resolution

The resolution of instance-level relationship conflicts is a complicated matter. First, it is not easy to detect the existence of conflicts. Second, when a relationship conflict is detected, it is difficult to identify its cause in order to take the necessary actions to resolve the conflict. According to the nature of relationship conflicts and the present state-of-the-art database integration technology, we believe that human involvement is required in the resolution process although some semi-automatic integration tools can be useful.

We propose a systematic process to detect different types of relationship conflicts and to resolve them. This process is depicted in Fig. 5. There are three conflict resolution steps and they are illustrated using the example presented in Section 4.2.

6.1. Instance matching

The process begins with checking if an export relationship instance exists in another export database. For example, to resolve the relationship instances of STORE between export databases A and

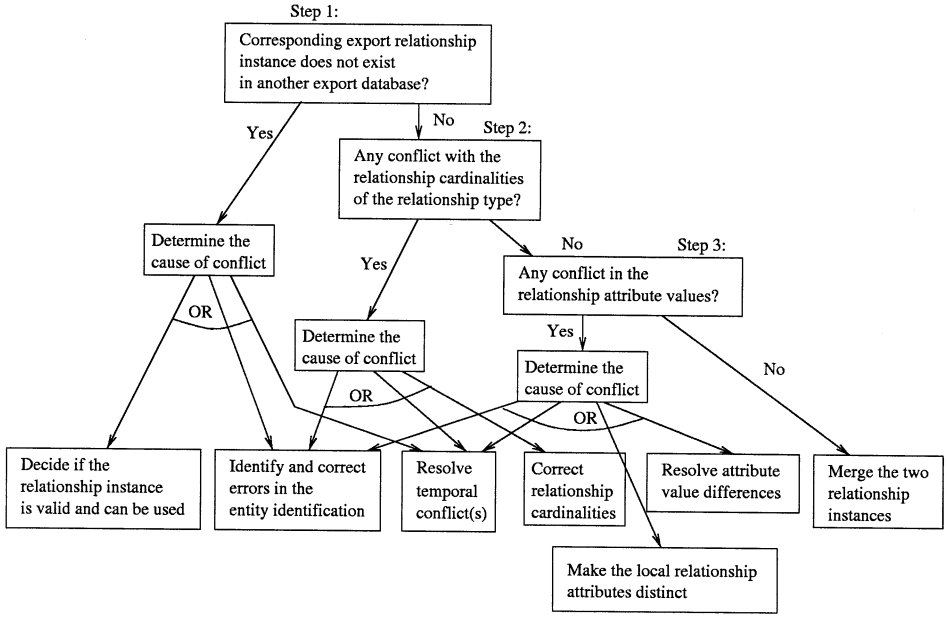

Fig. 5. Instance-level relationship conflict resolution.

B, we match every STORE instance in A with STORE instances in B. If the corresponding STORE instance in B cannot found for a STORE instance in A, there are three possibilities: (a) the relationship instance is valid but not captured by database B or the relationship instance is not valid in database A; (b) there is an error in the entity identification process; or (c) one of the databases is not up-to-date. The appropriate rectification can be performed by consulting the database integrator and/or local database administrators.

If a valid relationship instance is captured by one database but not the other, the integrator or administrator can decide whether to capture this relationship instance in the integrated database, while informing the administrators of the affected local databases. If the integrator or administrator determines a relationship instance that is not valid, it should be excluded from the integrated database.

To further assist the database integration process, a semi-automatic tool can be developed to identify all entities affected if the missing relationship instance is caused by incorrect entity identification. For example, let database A contain the information that product P1 is stored in factory F1 but the relationship instance does not exist in database B. It would be useful for an intelligent tool to suggest the factory records in database B that may be matched with F1 in database A, and the product records in database B that may be matched with P1 in database A. In this case, the database integrator can save some effort in re-examining the affected entities and correcting the entity identification errors.

In Ref. [9], *Instance-level Functional Dependency* (ILFD), as a special form of knowledge about the entity instances, was proposed to aid the resolution of entity identification conflicts. It can also be applied here to further disambiguate different possible matching of entity instances.

When a missing relationship instance is caused by temporal conflicts between two local databases, here are two ways to resolve the conflict. The database integrator can add a temporal attribute to the relationship making it a part of the relationship's key. With the additional temporal attribute, the validity duration of a relationship instance can be accommodated by the integrated database. If it is the intention of the database integrator to keep only the latest information in the integrated database, the out-dated relationship information should be discarded.

6.2. Cardinality verification

If the corresponding relationship instance is found in another export database, one may check if the integration of the two relationship instances will violate the relationship cardinalities. Violation of relationship cardinalities can be due to entity identification errors or incorrect relationship cardinalities.

Given a relationship type, it may be useful to count the number of relationship instances that violate the relationship cardinalities. It will give the database integrator an idea about the possibility of incorrect relationship cardinalities assigned to the relationship type. In resolving the relationship conflicts due to entity identification, the techniques mentioned in instance matching (Section 6.1) can also be used.

If the conflict with relationship cardinalities is caused by incorrect relationship cardinalities in the integrated schema, the database integrator should correct the schema and determine if the correction leads to other changes in the integrated schema.

6.3. Attribute value verification

Even when there is no conflict with the relationship cardinalities, it is necessary to check if the two corresponding relationship instances have conflicts in their attribute values. The conflicts in attribute values may be caused by incorrect entity identification or inconsistent database states. The approaches to deal with conflicts due to the two causes have been described as part of instance matching.

Sometimes, the conflict in relationship attribute values may be caused by different semantics associated with the relationship attributes to be merged. In this case, the local database administrators have to consider making the two attributes distinct in the integrated schema.

If the conflict in relationship attribute values is caused by some inconsistency between the local databases, some additional semantics about the database domain may be required to resolve the inconsistency in relationship attribute values between the local databases. One possible approach here is to

examine the reliability of databases and determine the database(s) that gives the most credible values for the relationship attribute concerned. In this case, the less credible relationship attribute value will be discarded.

7. Conclusions

Relationship conflicts have been studied only at the schema level. In this paper, we address the relationship conflicts at the instance level by investigating the different types of conflicts and their causes. We first describe the overall database integration methodology and present instance-level relationship conflict resolution as a subtask to be handled during instance integration. As the conflict resolution process is inherently complicated, it is not practical to propose automatic solutions. Therefore, we propose a systematic approach to classify instance-level relationship conflicts as well as their causes. This approach allows us to detect the different types of instance-level relationship conflicts and to suggest solutions corresponding to sources of conflicts. This research should facilitate and enhance the database integration work and result in the improvement of the data quality of integrated databases. More importantly, this research advocates a novel database research methodology to consider and examine a broad of integration issues and their inter-relationships.

As part of our future work, we will develop a semi-automatic intelligent integration tool to assist users to detect and resolve different types of instance-level relationship conflicts. Furthermore, by examining data instances of export databases, the tool should help to accelerate the instance integration process, reduce the inter-database inconsistencies, and accommodate necessary instance heterogeneities into integrated databases.

Acknowledgements

The authors wish to thank the Editor-in-chief and the anonymous reviewer for their very supportive comments during the preparation of this manuscript.

References

[1] C. Batini, M. Lenzerini, S.B. Navathe, A comparative analysis of methodologies for database schema integration, ACM Comput. Surv. 18 (4) (1986) December.
[2] A. Berson, S.J. Smith, Data extraction, cleanup, and transformation tools,Data Warehousing, Data Mining, and OLAP, McGraw-Hill, New York, 1997, Chap. 10.
[3] P. Chen, The Entity-Relationship model — toward a unified view of data, ACM Trans. Database Syst. 1 (1) (1976) 9–36.
[4] R.H.L. Chiang, T.M. Barron, V.C. Storey, Reverse engineering of relational databases: extraction of an eer model from a relational database,, March Data Knowl. Eng. 12 (2) (1994) .
[5] M. Garcia-Solaco, F. Saltor, M. Castellanos, Semantic heterogeneity in multidatabase systems, in: O.A. Bukhres, A.K. Elmagarmid (Eds.), Object-Oriented Multidatabase Systems: A Solution for Advanced Applications, Prentice-Hall, Englewood Cliffs, NJ, 1996, pp. 129–202, Chap. 5.
[6] W. Kim, J. Seo, Classifying schematic and data heterogeneity in multidatabase systems, IEEE Comput. (1991) December.
[7] R. Krishnamurthy, W. Litwin, W. Kent, Interoperability of heterogeneous databases with schematic discrepancies, in: International Workshop on Interoperability in Multidatabase Systems,1991.
[8] J.A. Larson, S.B. Navathe, R. Elmasri, A theory of attribute equivalence in databases with application to schema integration, IEEE Trans. Software Eng. 15 (4) (1989) April.
[9] E.-P. Lim, J. Srivastava, S. Prabhakar, J. Richardson, Entity identification problem in database integration, in: Proceedings of IEEE Data Engineering Conference, Vienna, Austria, 1993.
[10] E.-P. Lim, J. Srivastava, S. Shekhar, Resolving attribute incompatibility in database integration: an evidential reasoning approach, in: IEEE International Conference on Data Engineering, Houston, February, 1994.
[11] T.W. Ling, M.L. Lee, Issues in an Entity-Relationship federated database system, in: International Symposium on Cooperative Database Systems for Advanced Applications, Kyoto, Japan, December, 1996.
[12] W.J. Premerlani, M.R. Blaha, An approach for reverse engineering of relational databases, in: IEEE Working Conference on Reverse Engineering, Baltimore, 1993.
[13] E. Simoudis, B. Livezey, R. Kerber, Using recon for data cleaning, in: Proceedings of the First Int. Conference on Knowledge Discovery and Data Mining, Montreal, Ouebec, Canada, 1995.
[14] S. Spaccapietra, C. Parent, Y. Dupont, Model independent assertions for integration of heterogeneous schemas, Very Large Database J. 1 (1) (1992) 81–126.
[15] M.W.W. Vermeer, P.M.G. Apers, On the applicability of schema integration techniques to database interoperation, in: Entity-Relationship Conference, Cottbus, Germany, 1996, pp. 282–287.
[16] J. Widom, Integrating heterogeneous databases: lazy or eager?, ACM Comput. Surv. 28 (4es) (1996) 91.

Ee-Peng Lim received his BS (Honours) degree in Information Systems and Computer Science from the National University of Singapore, in 1989, and his PhD degree in Computer Science from the University of Minnesota, Minneapolis, in 1994. Since 1994, he has been on the faculty of the School of Applied Science at the Nanyang Technological University, Singapore, where he founded the Centre for Advanced Information Systems (CAIS). His current research interests include database integration, web warehousing, and digital libraries.

Roger H.L. Chiang is an Associate Professor of Information Systems at College of Business Administration, University of Cincinnati. His research interests are in data and knowledge management and intelligent systems, particularly in database reverse engineering, database integration, data mining, and common sense reasoning and learning. His research has been published in a number of international journals including *ACM Transactions on Database Systems*, *Data and Knowledge Engineering*, *Decision Support Systems*, and the *Journal of Database Administration*.