11-2004

Accommodating Instance Heterogeneities in Database Integration

Ee Peng LIM

Singapore Management University, eplim@smu.edu.sg

Roger Hsiang-Li CHIANG

DOI: https://doi.org/10.1016/S0167-9236(03)00103-9

Available online at www.sciencedirect.com

SCIENCE DIRECT®

Decision Support Systems 38 (2004) 213–231

Decision Support Systems

www.elsevier.com/locate/dsw

Accommodating instance heterogeneities in database integration

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

[a] Center for Advanced Information Systems, School of Computer Engineering, Nanyang Technological University, N4-2a-32, Nanyang Avenue, Singapore 639798, Singapore

[b] Information Systems Department, College of Business, University of Cincinnati, Cincinnati, OH 45221, USA

Received 30 March 2002; accepted 30 June 2003
Available online 6 August 2003

Abstract

A complete data integration solution can be viewed as an iterative process that consists of three phases, namely *analysis*, *derivation* and *evolution*. The entire process is similar to a software development process with the target application being the derivation rules for the integrated databases. In many cases, data integration requires several iterations of refining the local-to-global database mapping rules before a stable set of rules can be obtained. In particular, the mapping rules, as well as the data model and query model for the integrated databases have to cope with poor data quality in local databases, ongoing local database updates and instance heterogeneities. In this paper, we therefore propose a new object-oriented global data model, known as OO_{RA}, that can accommodate attribute and relationship instance heterogeneities in the integrated databases. The OO_{RA} model has been designed to allow database integrators and end users to query both the local and resolved instance values using the same query language throughout the derivation and evolution phases of database integration. Coupled with the OO_{RA} model, we also define a set of local-to-global database mapping rules that can detect new heterogeneities among databases and resolve instance heterogeneities if situations permit.

Keywords: Federated databases; Database integration; Schema integration; Instance integration; OO_{RA}; $OOQL_{RA}$

1. Introduction

To integrate two or more pre-existing or local databases without violating their local autonomy, either a *multidatabase* approach or a *warehousing* approach can be adopted. In both approaches, the database integration tasks are similar. Before pre-existing databases can be integrated together, the differences among them must first be identified and further resolved if possible. All inter-database heterogeneities can be generally classified into *schema* and *instance* ones. Schema heterogeneities refer to differences among schema elements from different local databases. Instance heterogeneities refer to conflicts that arise when data from different local databases have to be integrated into multidatabases or data warehouses. When these multidatabases (or data warehouses) and local databases are represented in relational model, the instance level conflicts are local database tuples corresponding to the same real world entities but carrying different attribute values. A classification of instance heteroge-

* Corresponding author. Tel.: +65-6790-4802; fax: +65-6792-6559.

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

neities will be given in Section 2. The resolutions of schema and instance heterogeneities are also known as *schema integration* and *instance integration*, respectively. In general, schema integration must be done before instance integration. In the traditional database integration research, focuses have been given to schema integration [2,7,9,10,20,23]. Instance heterogeneities, in contrast, have not been fully addressed.

To fully address the schema and instance integration issues, one has to examine the database integration process at the macro level. We believe that inter-database heterogeneities should be handled throughout the entire database integration process. While there has not been a well-accepted database integration methodology, we proposed to divide the entire integration process into three phases, namely *analysis*, *derivation* and *evolution* as shown in Fig. 1.

- *Analysis*: Analysis is essentially a knowledge acquisition phase. In this phase, database integrators are expected to understand pre-existing databases at both the conceptual and implementation levels. These local database semantics can be acquired from local database owners. Database integrators are also required to find out from the integrated database users their global application requirements in order to derive the global schema and instances.
- *Derivation*: The actual derivation of global schema and integrated instances is done in this phase. Once the derivation is done, queries on the integrated database can be evaluated. It is in this phase a complete mapping from local schemas to global schema, as well as a mapping from local instances to global instances is specified.
- *Evolution*: Due to the autonomy of local database systems, updates to the local databases may violate the mapping from local instances to global instances. Evolution therefore refers to the ongoing refinement of integrated databases as the local database schemas and instances evolve. It becomes the most important phase to maintain a multi-database or data warehousing system.

Among the above three phases, evolution has been largely ignored in the database integration research primarily due to two reasons. Firstly, most researchers focus on schema integration issues. While a lot of schema integration issues have to be investigated for different databases during the derivation phase, it is uncommon to investigate schema integration issues during the evolution phase due to rare modification to pre-existing local schemas. Secondly, research on the integration of instances has been pre-occupied by query processing issues instead of local database updates during the evolution phase. In this paper, we argue that instance integration may not be complete in the derivation phase. During the evolution phase, one also has to consider local database updates, which lead to new instance conflicts that cannot be handled by pre-defined integration methods. Hence, new global data models that can accommodate instance heterogeneities become necessary.

1.1. Related work

Most previous database integration research focused on resolving schema conflicts. A taxonomy of schema conflicts can be found in [6,8]. Depending on the tightness of integration between component databases, different schema conflict resolution methods can be adopted [19]. Loosely integrated component databases often involve the creation of a wrapper to

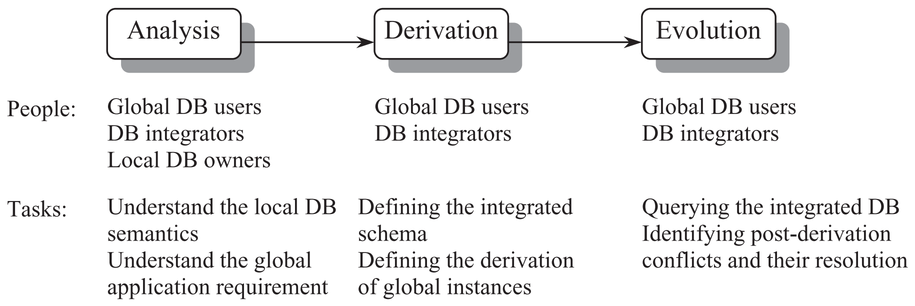
Analysis
Derivation
Evolution
People:
Global DB users
DB integrators
Local DB owners
Global DB users
DB integrators
Global DB users
DB integrators
Tasks:
Understand the local DB semantics
Understand the global application requirement
Defining the integrated schema
Defining the derivation of global instances
Querying the integrated DB
Identifying post-derivation conflicts and their resolution

Fig. 1. Database integration phases.

allow a local database system to have a unified access to the schemas and instances of one or more remote databases. In this case, the schema conflicts to be resolved arise from mainly the incompatibility between data models. The schema conflict resolution methods for loosely integrated component databases include [16,25]. For tightly integrated component databases, an integrated database that merges the schemas and instances of component databases will be derived. In other words, a global schema that merges the local schemas will be required and it should hide the heterogeneity of local schemas from the integrated database users [1,7,9,20]. Kaul, Drosten and Neuhold proposed *schema transformation mapping* and *schema integration mapping* as the two major steps to derive a global schema from a few local schemas [7]. They also proposed an incremental approach to derive global schemas as the knowledge required by schema integration may only be available over time. The stepwise incremental approach to schema integration was further developed by Aslan and McLeod [1].

Lately, as researchers begin to address instance integration problems, several solutions of instance conflict resolution have been proposed [5,14,15,22,25]. Most of these solutions resolve instance conflicts in some pre-determined approaches [5,14,15,22], e.g., using probabilistic reasoning, uncertainty theories, etc. In the work by Lu et al. [17], a statistical approach to discover conflict resolution rules from different relations has been developed. Nevertheless, the data models adopted by these solutions neither facilitate different ways to resolve conflicts, nor accommodate much information about instance heterogeneities. The impact of instance conflicts on query processing and optimization has also been studied in [26].

In the following, we describe some previous work in extending data model to accommodate instance conflicts.

- Polygen model [25] was proposed to capture source information of attribute values that come from different local relations. A source value is associated with every attribute value of the tuples of polygen relations. The source information captured includes the sites the attributes originated from and the intermediate sites at which they are processed. The model, however, does not provide the mechanism to accommodate or resolve instance heterogeneities.
- TS Relational model [11] was proposed to accommodate entity and attribute conflicts in a relational integrated database. A special source attribute is assigned to every relation. An extended relational algebra has been proposed to manipulate the TS relations. Like the Polygen model, TS Relational model is not designed to represent resolved instance values.
- Role-based multidatabase model [18] extended Litwin's multidatabase model [16] by considering the different roles (or relations) assumed by real-world objects. Queries on a role-based multidatabase are decomposed into queries on different combinations of roles. Apart from not handling resolved instance values, the role-based multidatabase model does not classify between tolerable and intolerable relationship and attribute value conflicts. The notions of tolerable and intolerable conflicts will be elaborated in Section 4.

1.2. Objectives and scope

In the paper, we address the problem of accommodating instance heterogeneities (conflicts) in the global data model adopted for integrated databases. There are a number of reasons for accommodating instance heterogeneities in integrated databases:

- Resolving all instance differences may not be desirable because some global applications may want to retain and view these differences in the integrated database. For example, the different prices for the same product sold in different stores may be required to be retained and queried in the integrated database.
- Preserving the instance heterogeneities allows database integrators to apply different resolution techniques on the same instance-level conflicts for different global application requirement. For example, when there are differences between delivery times for the same products from different databases, the shortest ones might be used for web advertisement purposes while the longer ones might be recommended to the users upon purchases.
- Resolving all instance conflicts may not be possible because the information and knowledge

required for the complete conflict resolution is not available during the moment of instance integration. It is also impossible to anticipate and resolve all possible instance conflict as the local database evolves with time.

- Resolving all instance conflicts may not be feasible because the amount of processing time to resolve conflicts for large number of instances may be so much that integrated information may not be available on time. For example, the complete instance conflict resolution may exceed the available window time for loading data in the data warehouse environment.

We therefore present an object-oriented global data model that can accommodate instance heterogeneities for attributes and relationships in the integrated database. The new global data model supports different integration and query requirements from the database integrators and database users during the derivation and evolution phases of database integration. To our best knowledge, this is the first attempt in developing a global data model with the purpose of supporting queries on integrated databases containing both resolved and unresolved values.

1.3. Contributions

On the whole, our research contributes to database integration in a number of ways. Firstly, it presents the different types of instance heterogeneities one may encounter during database integration. Secondly, it clearly points out the different integration and query requirements from the database integrators and database users during the derivation and evolution phases of database integration. Thirdly, we introduce the concept of threshold predicates and resolution functions to detect and reconcile instance heterogeneities. Fourthly, an extended object-oriented data model has been introduced to accommodate resolved as well as unresolved instance conflicts in the integrated databases. The proposed model is also equipped with the necessary query and integration primitives required.

This paper is organized as follows. In Section 2, we describe the various types of instance conflicts and the overall approaches to handle them in database integration. Section 3 describes requirements for accommodating instance heterogeneities during the derivation and evolution phases. This study motivates the development of an extended object-oriented data model called OO_{RA} to be defined in Section 4. Section 5 presents the query language for the OO_{RA} data model and some query examples are given. The conclusions and future research directions are given in Section 6.

2. Instance heterogeneities

Instance heterogeneities can be classified into *entity conflicts*, *attribute conflicts* and *relationship conflicts* [12,15]. Entity conflicts arise when it is not known which entity instances from matching entity types[1] correspond to the same real-world entities. Relationship conflicts occur when it is not known which relationship instances from matching relationship types correspond to the same real-world relationships. Attribute conflicts arise when the matching entity (or relationship) instances (determined by resolving entity or relationship conflicts) do not have the same attribute values.

Consider the following integration scenario. Let DB_A and DB_B be two databases containing employee information. The former is owned by the headquarter, while the latter is maintained by the regional office.

DB_A: Staff(ename, position, salary)
DB_B: Emp(ename, title, salary, qual)

Assume that during the derivation phase, the database integrator defines an integrated relation Employee directly from the above two relations as shown below.

Employee(ename, position, salary, qual)

The integration of the two local relations Staff and Emp into Employee at the schema level can be performed prior to integrating their instances. To integrate employee tuples from the two local relations at the instance level, one has to address the issues of matching tuples that represent the same real world

[1] Matching entity types are determined by schema integration.

entities, and resolving their attribute value and relationship conflicts.

As pointed out by a number of researchers [6,12], instance integration constitutes an important step in database integration. Instance level heterogeneities are caused by imperfect data quality in the legacy databases. For example, a typo-error in an employee's ename in Staff may lead to the failure of matching the Staff data instance with the corresponding instance in Emp. When customers do not wish to reveal their ages, they may provide erroneous age values to the databases causing problems in the determination of correct age values. There are many different methods for resolving instance level conflicts. Nevertheless, the method used for resolving discrepancies in employees' positions may also be different from that used for salary. In this paper, we will further point out that the tolerance of instance conflicts varies among different attributes. In fact, it is common that not all instances from legacy databases can be properly integrated during the derivation phase.

Assuming that all instances of Staff and Emp have been properly integrated during the derivation phase, one still has to handle integration issues arising from the updates to local database(s) during the evolution phase. For example, new employee data instances could be added to Emp making it necessary to perform instance integration on the new instances. Similarly, instance integration is required for changes to attributes of some pre-existing data instances. There are essentially *two approaches* to handle instance integration problems during the derivation and evolution phases. The first approach requires the database integrator to anticipate all possible integration scenarios during the derivation phase and define the instance integration methods accordingly, hoping that all integration problems in the evolution phase can be predicted in advance. When the integration scenarios cannot be predicted in advance (which is often the case), one has to resort to accommodating instance conflicts in the integrated database before these conflicts can be finally resolved sometime in the future or may not be resolved at all.

2.1. Entity conflicts

To resolve entity conflicts, the knowledge for identifying instances representing the same real-world entities is required. For simple cases, common keys among entity instances could be used to match instances. For example, the employee name attribute can be used to match data from DB_A and DB_B. Complicated entity conflicts arise when there is no common attribute that can be used to match instances from different databases. Special techniques designed for resolving entity conflicts have been proposed [4,13,14,24]. Although it may not be possible to resolve all entity conflicts, instances that could not be determined to represent the same real-world entities can still be retained as separated instances in the integrated database.

2.2. Attribute conflicts

Given two instances that represent the same real-world entity, the differences in their equivalent attributes are known as attribute conflicts.[2] We distinguish two main types of attribute conflicts, namely tolerable and intolerable attribute conflicts that should be handled in database integration. Tolerable attribute conflicts are those expected by a database integrator at the time an integrated database is derived. Intolerable attribute conflicts, on the other hand, refer to attribute conflicts that should not be resolved automatically by any predefined resolution methods.

To distinguish between the above two types of attribute conflicts, we introduce the concept of *threshold predicate*. When the difference between two or more conflicting attribute values is smaller than a threshold value or when the conflicting attribute values differ in expected patterns, there is a straightforward pre-defined approach to handle the conflicts. The exact conflict handling approach can be readily specified during the derivation phase of database integration. The primary purpose of a threshold predicate is therefore to explicitly capture the criteria to be satisfied by tolerable attribute conflicts. In other words, we define tolerable attribute conflicts to be those satisfying the threshold predicates defined for the attributes involved.

It is necessary to resolve tolerable attribute conflicts derived from different local databases. To do

[2] If the attribute values are identical, the attribute conflict does not exist.

so, resolution functions should be defined to reconcile the corresponding tolerable attribute values. However, it is not always possible to apply resolution functions to resolve all possible tolerable attribute conflicts. Sometime, one may not know the correct resolution function to be specified or used. In other occasions, the attribute conflicts are considered to be valid and acceptable by the integrated database users. Thus, no resolution function is required.

When the conflicting local attribute values cannot satisfy the threshold predicate defined for the corresponding attribute, database integrators should be alerted. Attribute conflicts that fail the specified thresholds are defined to be *intolerable*. Database integrators can be alerted for intolerable attribute conflicts by having the intolerable attribute conflicts recorded in a log file.

2.2.1. Attribute conflict example

When the price of a product has different values in different databases, it is an attribute conflict. However, a very small difference in price may not be significant enough to warrant the database integrators' attention For example, the company may practise price discrimination strategy. In this case, the price difference of a product is considered tolerable if the difference is less than 5% of the lowest price. Otherwise, it is intolerable. For tolerable price differences, the multidatabase system or data warehousing system can apply a pre-defined resolution function (e.g., choosing the large value, averaging the two values) to derive an integrated product price.

2.3. Relationship conflicts

Relationship conflicts, first discussed in [12], arise when the relationship between two real-world entities may not be represented consistently in different databases. In [12], different types of relationship conflicts have been derived, and they can be caused by incorrect schema integration, incorrect entity conflict resolution and inaccurate database content.

2.3.1. Relationship conflict example

For example, employee e1 is known to work for department d1 in database A but d2 in database B where d1 and d2 are determined to represent different departments in the real-world after entity conflict resolution. This relationship conflict could be caused by incorrect relationship cardinalities in the integrated schema. It could also be caused by d1 and d2 being wrongly determined to represent the same department. Or, it could be the case that the employee data in either database A or B is not accurate.

Like other instance-level conflicts, a complete resolution of relationship conflicts may not always be possible. When relationship conflicts cannot be resolved by the multidatabase system or data warehousing system, they should be retained and accommodated.

3. Requirements for accommodating instance heterogeneity

If inter-database conflicts at both the schema and instance levels are resolved completely, the integrated databases can be represented by using a standard data model. However, when instance heterogeneities are part of integrated databases, we have to extend the standard data models and their query languages to accommodate and manipulate instance heterogeneities. Before we propose a global object model for this purpose, we first investigate the potential query and integration requirements imposed by two types of users of the integrated databases, the *database integrators (system users)* and *end users*.

3.1. Database integrators

A database integrator's involvement in database integration encompasses both the derivation and evolution phases as shown in Fig. 1. During the derivation phase, the database integrator first performs schema integration on the local databases. He or she later defines transformations on the instances from each local database so that they conform to the integrated schema before instances from different local databases are integrated together. The database integrator's tasks during the evolution phase are similar except that schema integration is usually not required and the instances to be dealt with are fewer. A detailed discussion about the transforma-

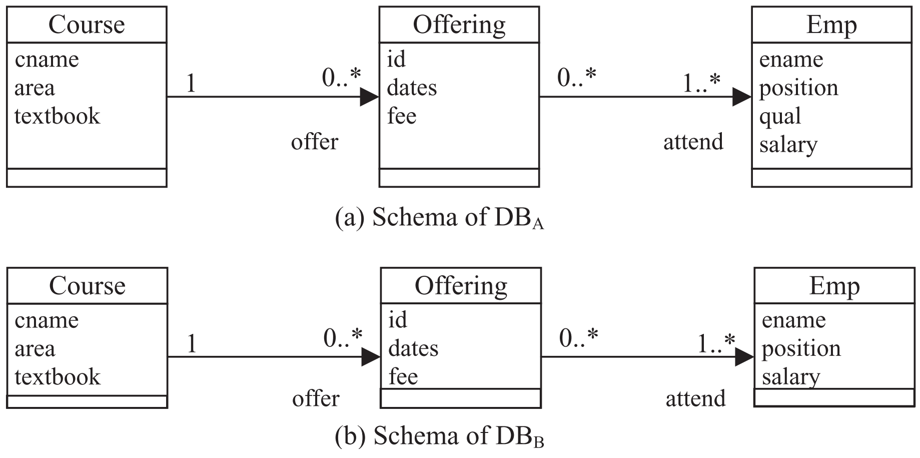

Fig. 2. Schemas of local databases.

tion of local instances is beyond the scope of this paper. We will instead focus on the database integrator's activities in resolving the conflicts between transformed local instances. Henceforth, without loss of generality, we assume that the database integrator is given local databases that conform to the integrated schema.

To efficiently integrate local instances, database integrators require some mechanisms to easily compare local instances and discover conflicts between them. Subsequently, appropriate resolution methods can be applied to the tolerable conflicts. Throughout this integration process, a global data model that accommodates both instance conflicts and resolved instances is required. With such a global data model, the database integrator can query the integrated database containing instance conflicts and view these conflicts in the query results. The global data model should also allow database integrators to define different resolution functions to resolve conflicts or enforce consistency of data stored in local databases.

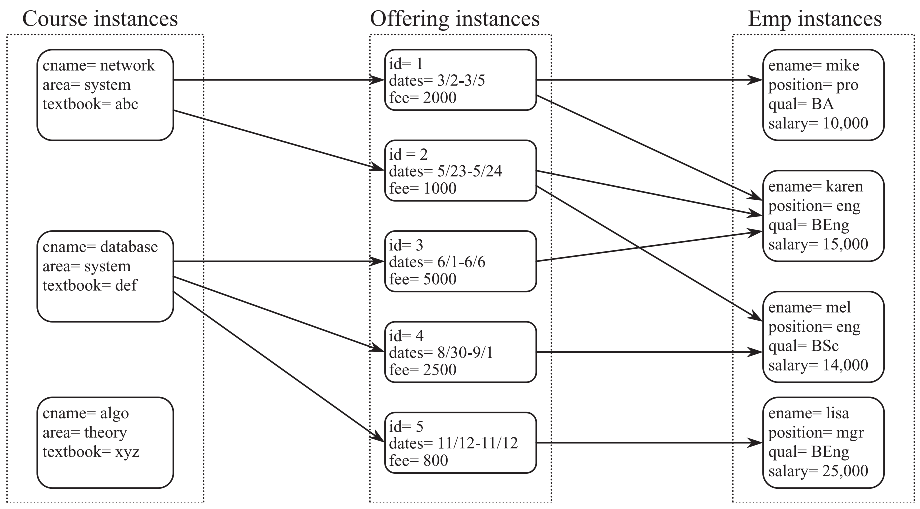

Fig. 3. Instances of DB$_A$.

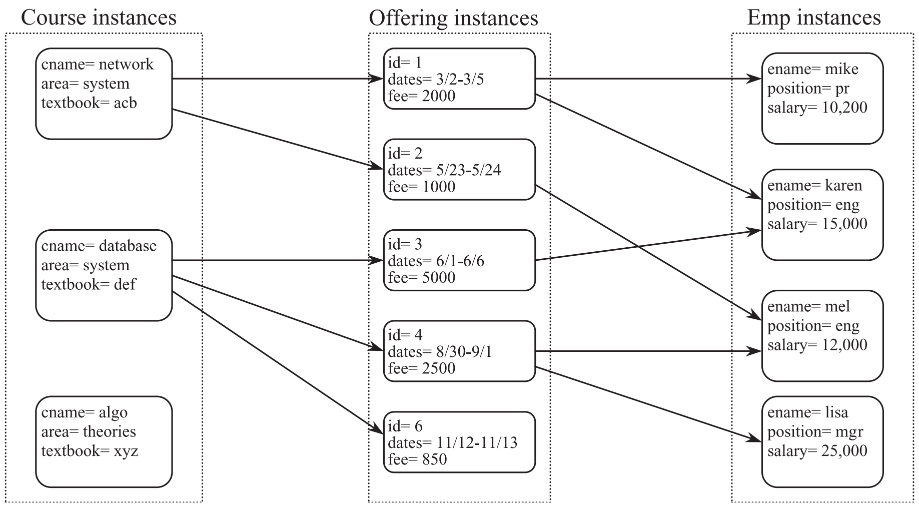

Fig. 4. Instances of DB_B.

3.2. End users

End users utilize the integrated database for report generation and decision making. Depending on their needs, they may choose to query the resolved or original instance values. For example, one may want to query the resolved salary values of employees who hold manager positions in a particular local database. Hence, a global data model should support flexible queries on both original and resolved instance values.

As both multidatabase and data warehousing systems preserve the autonomy of local database systems, updates to local databases can often introduce new instance conflicts to integrated databases. Some of these new instance conflicts could be handled automatically by the resolution functions predefined by database integrators and hence no further actions would be required by the end users. For other new instance conflicts that cannot be resolved automatically, database integrators have to be called upon to handle them. Nevertheless, before the database integrators take any actions, these new conflicts have to be accommodated by the global data model and the end users should be allowed to continue using the integrated database.

3.3. An example integration scenario

We employ an integration scenario to demonstrate the attribute and relationship conflicts. Fig. 2 depicts the object-oriented schemas (in the form of UML class diagrams) of the local databases DB_A and DB_B containing employee training information. Both the schemas show that a course can be offered at different dates with different fees. Each course-offering can be

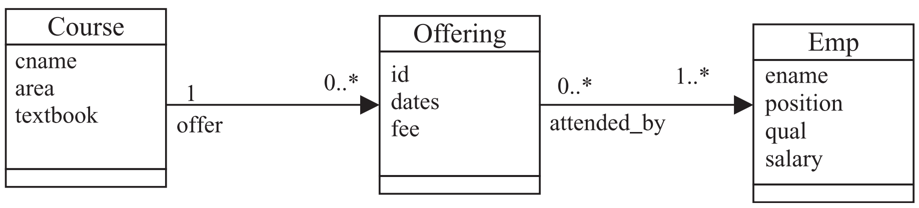

Fig. 5. Integrated schema.

attended by one or more employees. Here, we assume that schema integration has been performed and the schemas of existing databases have been made compatible to facilitate instance level comparisons. The schema and data transformation approaches taken to homogenize the existing databases have been reported in several papers [20,21].

The instances of DB_A and DB_B are shown in Figs. 3 and 4, respectively. To keep the figures concise, we have intentionally used simplified attribute values for the instances (e.g., course offering ids, course name, etc.). Suppose all entity conflicts are resolved by matching *cname*, *id* and *ename* of course, course offering (i.e., Offering), and employee (i.e., Emp) instances, respectively. We notice that database course in DB_A has been offered in the Offering instances with ids 3, 4 and 5, but the same course in DB_B has been offered in the Offering instances with ids 3, 4 and 6. This is a relationship conflict. On the other hand, the difference in area (i.e., "theory" vs. "theories") for the algorithm course in DB_A and DB_B is an attribute conflict. Fig. 5 depicts the integrated schema derived from DB_A and DB_B.

4. The OO_{RA} object-oriented data model

We propose the OO_{RA},[3] the extended object-oriented data model, to accommodate instance heterogeneities in the integrated databases. Specifically, the OO_{RA} model is able to accommodate attribute and relationship conflicts. The OO_{RA} data model is also designed to support queries on the integrated databases. Furthermore, the OO_{RA} data model ensures that the source of instance heterogeneities can be identified in order to support subsequent integration work on the partially integrated database. OO_{RA} differs from the traditional OO data model in a number of ways:

- Identification of matching criteria for deriving global objects;
- Specification of threshold predicates and resolution functions;
- Representation of original and resolved attribute values; and
- Uniform treatment of attribute and relationship conflicts.

In the following, we describe the unique features of OO_{RA} model in detail.

4.1. Global objects

A global object in the integrated database is derived from one or more local objects that represent the same real-world entity. Like in the traditional OO data model, each global object is assigned a unique *global object id (oid)*. In OO_{RA}, we assume that local objects corresponding to the same global objects (or real-world entities) can be matched by examining some common attribute values. These common attribute(s) can be specified as matching criteria by the database integrator. For example, a database integrator may use *cname* to match Course objects, *id* to match Offering objects and *ename* to match Emp objects from DB_A and DB_B. The following three data definition statements have been used to identify matching local objects:

```
DERIVE COURSE FROM Course@DB_A, Course@DB_B
USING Course@DB_A(cname), Course@DB_B(cname);
DERIVE OFFERING FROM Offering@DB_A, Offering@DB_B
USING Offering@DB_A(id), Offering@DB_B(id);
DERIVE EMP FROM Emp@DB_A, Emp@DB_B
USING Emp@DB_A(ename), Emp@DB_B(ename);
```

In general, a data definition statement is defined based on the following grammar rules:

⟨*Derive Global Object Class*⟩	::-DERIVE⟨*Global Class*⟩ FROM⟨*Local Class List*⟩ USING⟨*Local Attribute List*⟩
⟨*Local Class List*⟩	::-⟨*Local Class*⟩, {⟨*Local Class*⟩}*
⟨*Local Class*⟩	::-⟨*Class Name*⟩@ ⟨*Local Database Name*⟩
⟨*Local Attribute List*⟩	::-⟨*Local Class*⟩ (⟨*Local Attribute Name*⟩), {⟨*Local Class*⟩ (⟨*Local Attribute Name*⟩)}*

[3] In OO_{RA}, R represents the relationship conflicts. A represents the attribute conflicts. The name OO_{RA} indicates that both relationship and attribute conflicts can be accommodated.

Once the above data definition statements are specified, we effectively construct a set of global classes.

Definition 1 (Global class). Let C_1, C_2, ..., C_n be n local classes that are schema compatible, a global class G derived from them can be defined as:

$$G = \mathrm{OJ}(K, \{C_1, \ldots, C_n\})$$

where OJ is an n-way outerjoin that merges the objects from all C_i's sharing the same identifying attribute (i.e., K) values.. Note that the resultant global class G contains the union of all attributes from C_1, C_2, ..., C_n denoted by A_1, A_2, ..., A_n, respectively. Suppose a global object g in G is derived from local objects l_{j1}, ..., l_{jk} in C_{j1}, ..., C_{jk}, respectively, $1 \leq j1 \leq jk \leq n$. We use *g.a.ovalue*[4] to denote the attribute value of g where a appears in A_{j1}, ..., A_{jk} and is defined as follows:

$$\text{g.a.ovalue} = \{(l_{ji}.a, \mathrm{DB}_{ji}) \mid 1 \leq i \leq k\}$$

4.2. Threshold predicates and resolution functions

One or more pairs of *threshold predicates* and *resolution functions* can be defined for each attribute in the global schema. Given an attribute in a class of global objects, the threshold predicates determine for each global object if a difference between local values of the attribute is tolerable. As long as one of the predicates holds, the difference between the local attribute values is considered tolerable. The corresponding resolution function (if defined) is then adopted to resolve tolerable attribute conflicts automatically. Depending on the characteristics of attributes, different threshold predicates and resolution functions should be defined by the database integrators and be implemented using system-defined functions/operators or general programs.

Examples of thresholds and resolution functions for tolerable instance conflicts are:

- A price difference of less than 5 cents for an apparel product in different databases of a department store may be tolerable. The highest price can be assigned to the product in the integrated database.
- A floor area difference of less than 1 ft^2 for the same apartment in different property databases may be considered insignificant. Such difference could be resolved by choosing the smallest floor area as the resolved value.

[4] The notation *ovalue* represents the original attribute values.

We define a pair of threshold predicate and resolution function for an attribute a using the following statement:

⟨*Define Threshold Predicate*⟩	::-DEFINE⟨*global attribute*⟩.threshold⟨*id*⟩@⟨*global class*⟩(⟨*parameter list*⟩) = ⟨*conjunction list*⟩
⟨*parameter list*⟩	::-⟨parameter⟩{,⟨parameter list⟩}*
⟨*conjunction list*⟩	::-⟨*conjunct*⟩{AND⟨*conjunction list*⟩}*
⟨*Define Resolution Function*⟩	::-DEFINE⟨*global attribute*⟩.resolution⟨*id*⟩@⟨*global class*⟩(⟨*parameter list*⟩)=[*value*|⟨*function*⟩(⟨*parameter list*⟩)]

Given a global attribute, multiple pairs of threshold predicates and resolution functions can be defined. When the resolution function is not defined for a specific threshold predicate, we say that the resolution function is a NULL function that returns NULL value for any given value. Each pair of threshold predicate and resolution function is assigned a unique id for easy identification.

Each conjunct in the threshold predicate is a Boolean condition on the *ovalue*(s) of the global attribute. In the simplest case, a threshold predicate can be a conjunction of Boolean comparison of attribute values. In other more complex cases, a threshold predicate involves *distance functions* on the attribute values to measure the extent of difference among attribute values from different sources. For example, to detect for a given product whether the difference between the largest and smallest price values from different sources is less than 10 cents, we can define following threshold predicate.

$$\text{DEFINE price.threshold1@PRODUCT}(a1, a2, a3)$$
$$= (\text{max} - \text{dist}(a1, a2, a3) < 0.10)$$

In the above example, the max-dist() function is an implemented function that can be invoked with the input values and it returns the difference between the largest and smallest input values.

Similarly, the resolution function may involve a constant value or a *merge function* that combines the attribute values from difference sources. For example, to integrate the price values of a given product, we may define the resolution function to take the average value as follows:

$$\text{DEFINE price.resolution1@PRODUCT}(a1, a2, a3) = \text{average}(a1, a2, a3)$$

The average() function is an implemented function, similar to the *distance function*, that can be invoked with input values and it returns a single combined value.

There are many different ways to define the distance and merge functions. Broadly, we classify them into three main categories, namely *distributive*, *algebraic* and *holistic*. A distance or merge function f is distributive if it can be computed in a distributed manner, e.g., $f(x1,\ldots,xk)=f(\ldots f(f(x1,x2),x3)\ldots)$. In other words, we can apply the same function in any order on different subsets of the input values. Examples of distributive function include sum(), max() and min(). A distance or merge function f is algebraic if it can be computed by an algebraic function with a fixed number p of arguments, and each argument can be computed by applying a distributive function. An example of algebraic function is the average() function as it can be computed by two arguments sum() and count(), which can be derived by distributive functions. When a distance or merge function is neither distributive nor algebraic, it is known to be holistic.

When a distance or merge function is distributive and algebraic, it is possible to compute the function value in different ordering of the source values. Such distributive and algebraic distance and merge functions allow local databases to be integrated in different orderings. They can also easily accommodate new local databases to be integrated. When holistic distance and merge functions are used, the ordering of local databases will be restricted, and it is more difficult to accommodate any new local databases.

Given an attribute in the global schema, three combinations of threshold predicates and resolution functions can be constructed[5]:

- *Both the threshold predicate and resolution function are undefined*: This implies that any difference between the corresponding attribute values is considered an intolerable attribute conflict. Unless all corresponding attribute values given are identical, the resolved attribute value is always NULL.
- *The threshold predicate is defined, but not the resolution function*: This implies that tolerable attribute conflicts can exist among distinct instances. These conflicts are also acceptable. However, unless the acceptable attribute conflict involves identical values, the resolved attribute value is always NULL.
- *Both the threshold predicate and resolution function are defined*: This implies that tolerable attribute conflict can exist among distinct instance and the resolution function will return the resolved attribute values.

4.3. Elements of attribute values

In the OO_{RA} model, every non-oid attribute has a domain consisting of three elements, namely the original values (denoted by ovalue), resolved values (denoted by rvalue) and conflict type (denoted by conflictType). The resolved value, original value, and conflict type of an attribute A are represented by A.rvalue, A.ovalue and A.conflictType respectively. Formally,

Definition 2 (Conflict type and resolved value of a global object attribute). Let g be a global object in the global class G derived from local objects c_1, c_2,

[5] Note that, when the threshold predicate is not defined for an attribute, it is meaningless to define the resolution function for the attribute since any difference between corresponding attribute values is considered intolerable, and such conflict should not be resolved by a resolution function automatically.

..., c_n from local classes C_1, C_2, ..., C_n, respectively. Let a be a global object attribute defined with threshold and resolution functions g.a.threshold's and g.a.resolution's. The conflict type and resolved value of a global object attribute is defined as follows:

$$\text{g.a.conflictType} = \begin{cases} \text{Resolvable} & \text{if } \exists k, \text{g.a.threshold}_k(c_1.a,\ldots,c_n.a) = \text{TRUE} \wedge \text{g.a.resolution}_k(c_1.a,\ldots,c_n.a) \text{ is defined} \\ \text{Acceptable} & \text{if } \exists k, \text{g.a.threshold}_k(c_1.a,\ldots,c_n.a) = \text{TRUE} \wedge \text{g.a.resolution}_k(c_1.a,\ldots,c_n.a) \text{ is undefined} \\ \text{Intolerable} & \text{if } \forall k, \text{g.a.threshold}_k(c_1.a,\ldots,c_n.a) = \text{FALSE} \\ \text{NULL} & \text{otherwise} \end{cases}$$

$$\text{g.a.rvalue} = \begin{cases} \text{g.a.resolution}_k(c_1.a,\ldots,c_n.a) & \text{if } \exists k, \text{g.a.threshold}_k(c_1.a,\ldots,c_n.a) = \text{TRUE} \wedge \text{g.a.resolution}_k(c_1.a,\ldots,c_n.a) \text{ is defined} \\ x & \text{if } \forall a \in \text{g.a.ovalue}, a = x \\ \text{NULL} & \text{otherwise} \end{cases}$$

The A.ovalue of a global object is defined to be a set of (value,database$_{\text{id}}$) pairs where value denotes the attribute value contributed by the corresponding object from the existing database identified by database$_{\text{id}}$. The A.rvalue of a global object is defined to be any A value contributed by local objects if there is no attribute conflict. If a difference is found among the local A values, the tolerance of the conflict is first determined using the threshold predicate(s) defined for A, i.e., A.threshold$_i$()'s. If any of the threshold predicates holds, the conflict is tolerable and A.rvalue is obtained by applying the corresponding A.resolution$_i$() on the local attribute values. In the event where the conflict is intolerable or the resolution function is undefined, NULL is assigned to A.rvalue.

Depending on the original attribute values and the threshold predicate(s) defined for the attribute, different conflict types can be derived and is stored in A. conflictType. A.conflictType is NULL if there is no conflict, Resolvable if there is a tolerable conflict that can be resolved by the pre-defined resolution function, Acceptable if there is a tolerable conflict and there is no pre-defined resolution function for resolving the conflict, and Intolerable if there is an intolerable conflict.

In our integrated database example, we can define the threshold predicates and resolution function for the area, textbook and salary attributes as follows:

DEFINE area.threshold1@COURSE($a1$,$a2$) = (($a1$ EQUALS "theory") AND ($a2$ EQUALS "theories"))
DEFINE area.resolution1@COURSE($a1$,$a2$) = "theory"
DEFINE area.threshold2@COURSE($a1$,$a2$) = (($a1$ EQUALS "system") AND ($a2$ EQUALS "systems"))
DEFINE area.resolution2@COURSE($a1$,$a2$) = "system"
DEFINE textbook.threshold1@COURSE($t1$,$t2$) = (($t1$ EQUALS "abc") AND ($t2$ EQUALS "abcd"))
DEFINE textbook.resolution1@COURSE($t1$,$t2$) = "abc"
DEFINE position.threshold1@EMP($p1$,$p2$) = (($p1$ EQUALS "pro") AND ($p2$ EQUALS "pr"))
DEFINE salary.threshold1@EMP($s1$,$s2$) = (max-dist($s1$,$s2$) $\leq$ 200)
DEFINE salary.resolution1@EMP($s1$,$s2$) = max($s1$,$s2$)

With the above definition, the area values of "theory" and "theories" for the Algorithm course constitute a resolvable attribute conflict according to area.threshold$_1$(). The global object for the Algorithm course will therefore have a resolved area value of "theory" computed by the resolution function, area.

resolution$_1$(). On the other hand, the textbook values of "abc" and "acb" for the Network course constitute an intolerable attribute conflict. In this situation, database integrators should be alerted and the conflict should be resolved manually. Since only the threshold predicate is defined for the position attribute of the EMP class, the position values of "pro" and "pr" constitute an acceptable conflict. This may be because "pro" (public relation officer) and "pr" (public relation) are synonymous. However, there is no corresponding resolution function for this conflict.

For example, the attribute elements of the area and textbook attribute of the Network and Algorithm course objects, and those of the position attribute of the employee object mike are shown below:

```
network: area.ovalue = {(system, A), (system, B)}
         area.rvalue = system
         area.conflictType = NULL
         textbook.ovalue = {(abc, A), (acb, B)}
         textbook.rvalue = NULL
         textbook.conflictType = Intolerable
algo:    area.ovalue = {(theory, A), (theories, B)}
         area.rvalue = theory
         area.conflictType = Resolvable
         textbook.ovalue = {(xyz, A), (xyz, B)}
         textbook.rvalue = xyz
         textbook.conflictType = NULL
mike:    position.ovalue = {(pro, A), (pr, B)}
         position.rvalue = NULL
         position.conflictType = Acceptable
```

To keep the discussion simple, we have so far only mentioned attributes with simple domain. OO$_{\mathrm{RA}}$ can handle multivalued attributes and attributes with complex data types in a similar manner.

4.4. Relationship conflicts

In the object-oriented data model, relationships can be treated as attributes that provide references to objects in other classes. These attributes are known as *reference attributes*. The domain of reference attributes consists of object ids. A one-to-one or many-to-one relationship from class C1 to class C2 can be represented as an single-valued reference attribute in C1, while a one-to-many or many-to-many relationship can be represented as an multi-valued reference attribute in C1. For multi-valued reference attributes, the reference attribute values are represented as sets of object ids.

In the OO$_{\mathrm{RA}}$ model, global relationships are derived from relationships between objects of existing databases. The global relationships, represented as reference attributes in the global schemas, relate global objects from different classes in the integrated database. Similar to attribute conflicts, we represent the original and resolved values of a reference attribute R in the global schema by R.ovalue and R.rvalue, respectively. Threshold predicates and resolution functions can also be defined on reference attributes.

Definition 3 (Conflict type and resolved value of a global object reference attribute). Let g be a global object in the global class G derived from local objects $c_1, c_2, \ldots, c_n$ from local classes $C_1, C_2, \ldots, C_n$, respectively. Let r be a global object reference attribute defined with threshold and resolution functions g.r.threshold's. and g.r.resolution's. The conflict type and resolved value of a global object reference attribute is defined as follows:

$$\text{g.r.conflictType} = \begin{cases} \text{Resolvable} & \text{if } \exists k, \text{g.r.threshold}_k(c_1.r,\ldots,c_n.r) = \text{TRUE} \wedge \text{g.r.resolution}(c_1.r,\ldots,c_n.r) \text{ is defined} \\ \text{Acceptable} & \text{if } \exists k, \text{g.r.threshold}_k(c_1.r,\ldots,c_n.r) = \text{TRUE} \wedge \text{g.r.resolution}(c_1.r,\ldots,c_n.r) \text{ is undefined} \\ \text{Intolerable} & \text{if } \forall k, \text{g.r.threshold}_k(c_1.r,\ldots,c_n.r) = \text{FALSE} \\ \text{NULL} & \text{otherwise} \end{cases}$$

$$\text{g.r.rvalue} = \begin{cases} \text{g.r.resolution}_k(c_1.r,\ldots,c_n.r) & \text{if } \exists k, \text{g.r.threshold}_k(c_1.r,\ldots,c_n.r) = \text{TRUE} \wedge \text{g.r.resolution}_k(c_1.r,\ldots,c_n.r) \text{ is defined} \\ x & \text{if } \forall r \in \text{g.r.ovalue}, r = x \\ \text{NULL} & \text{otherwise} \end{cases}$$

For illustration, let say the course offering with id 5 is actually part of that with id 6. The following threshold predicate and resolution function can be defined.[6]

DEFINE offer.threshold$_1$@COURSE(a,b) = (a − {5} + {6} = = b)
DEFINE offer.resolution$_1$@COURSE(a,b) = a − {5} + {6}

In the above statements, a and b denotes global object ids of OFFERING objects. Using the threshold predicate and resolution function for offer, the elements of offer relationships for Network and Database are shown below. Note that co1 to co6 are global object ids for the course offerings with ids 1 to 6, respectively.

Network:	offer.ovalue = {({co1,co2},A), ({co1,co2},B)}
	offer.rvalue = {co1,co2}
	offer.conflictType = NULL
Database:	offer.ovalue = {({co3,co4,co5},A), ({co3,co4,co6},B)}
	offer.rvalue = {co3,co4,co6}
	offer.conflictType = Resolvable

4.5. Integrated database instances

The OO$_{RA}$ objects of the integrated database are shown in Tables 1–3. Note that the Attribute-element columns in the above tables are included simply to illustrate the three elements of attribute values. As shown in Tables 1–3, the OO$_{RA}$ data model retains both attribute and relationship conflicts while holding the matching objects from different databases together by assigning global object ids to them. Respective resolution functions are defined to perform various resolutions of instance conflicts when they are tolerable.

As shown in the above tables, the oid attributes are unlike the other attributes. They do not have any conflicts and resolved values. However, to allow them to be queried in a way consistent with the other attributes, the oid attributes are assumed to have identical ovalues and rvalues, and NULL values for conflictType.

[6] In practice, the threshold predicate and resolution function can be implemented as general programs.

5. OO$_{RA}$ query language and examples

To query the global objects represented in the OO$_{RA}$ data model, one has to formulate queries in a language we refer to as OOQL$_{RA}$. OOQL$_{RA}$ uses the OQL syntax since the latter has been included by the Object Data Management Group (ODMG) as the standard object-oriented query language [3]. OOQL$_{RA}$ has further extended OQL to support the query requirement for an integrated database containing attribute and relationship conflicts in the derivation and evolution phases of database integration. An OOQL$_{RA}$ SELECT query statement can be expressed as:

SELECT⟨path expression 1⟩, . . ., ⟨path expression m⟩
FROM⟨object set expression 1⟩, . . .,⟨object set expression n⟩
WHERE⟨predicate expression⟩

The FROM clause consists of one or more expressions each representing a set of objects that belong to some class. Unlike the usual OQL statements, every non-oid attribute, say A, found in an OOQL$_{RA}$ query statement must be in one of the forms, A, A.ovalue, A.ovalue(D), A.rvalue and A.conflictType where D is some local database id. Only attributes of the forms A.ovalue, A.ovalue(D), A.rvalue and A.conflictType can be used in the WHERE clause. In other words, the attribute in the form of attribute name can only appear in the SELECT clause. For example, in the following query Q1, we retrieve the id, name, area and textbook information of courses (Table 4).

Example (Q1)

SELECT C.oid, C.cname, C.area, C.textbook
FROM COURSE C

In the following subsections, we will use several query examples to illustrate other essential features of OOQL$_{RA}$.

Table 1
COURSE's global objects

Attr-element	oid	cname	area	Textbook	offer
ovalue	c1	(network,A)(network,B)	(system,A)(system,B)	(abc,A)(acb,B)	({co1,co2},A)({co1,co2},B)
rvalue		network	system	NULL	{co1,co2}
conflictType		NULL	NULL	intolerable	NULL
ovalue	c2	(database,A)(database,B)	(system,A)(system,B)	(def,A)(def,B)	({co3,co4,co5},A)({co3,co4,co6},B)
rvalue		database	system	def	{co3,co4,co6}
conflictType		NULL	NULL	NULL	resolvable
ovalue	c3	(algo,A)(algo,B)	(theory,A)(theories,B)	(xyz,A)(xyz,B)	NULL
rvalue		algo	theory	xyz	NULL
conflictType		NULL	resolvable	NULL	NULL

Table 2
OFFERING's global objects

Attr-element	oid	id	Dates	fee	attended_by
ovalue	co1	(1,A)(1,B)	(3/2–3/5,A)(3/2–3/5,B)	(2000,A)(2000,B)	({e1,e2},A)({e1,e2},B)
rvalue		1	3/2–3/5	2000	{e1,e2}
conflictType		NULL	NULL	NULL	NULL
ovalue	co2	(2,A)(2,B)	(5/23–5/24,A)(5/23–5/24,B)	(1000,A)(1000,B)	({e2,e3},A)({e3},B)
rvalue		2	5/23–5/24	1000	{e3}
conflictType		NULL	NULL	NULL	intolerable
ovalue	co3	(3,A)(3,B)	(6/1–6/6,A)(6/1–6/6,B)	(5000,A)(5000,B)	({e2},A)({e2},B)
rvalue		3	6/1–6/6	5000	{e2}
conflictType		NULL	NULL	NULL	NULL
ovalue	co4	(4,A)(4,B)	(8/30–9/1,A)(8/30–9/1,B)	(2500,A)(2500,B)	({e3},A)({e3,e4},B)
rvalue		4	8/30–9/1	2500	{e3}
conflictType		NULL	NULL	NULL	intolerable
ovalue	co5	(5,A)(5,B)	(11/12–11/12,A)	(800,A)	({e4},A)
rvalue		5	11/12–11/12	800	{e4}
conflictType		NULL	NULL	NULL	NULL
ovalue	co6	(6,A)(6,B)	(11/12–11/13,B)	(850,B)	NULL
rvalue		6	11/12–11/13	850	NULL
conflictType		NULL	NULL	NULL	NULL

Table 3
EMP's global objects

Attr-element	oid	Ename	Position	Qual	Salary
ovalue	e1	(mike,A)(mike,B)	(pro,A)(pr,B)	(BA,A)	(10000,A)(10200,B)
rvalue		mike	NULL	BA	10200
conflictType		NULL	acceptable	null	resolvable
ovalue	e2	(karen,A)(karen,B)	(eng,A)(eng,B)	(BEng,A)	(15000,A)(15000,B)
rvalue		karen	eng	BEng	15000
conflictType		NULL	NULL	NULL	NULL
ovalue	e3	(mel,A)(mel,B)	(eng,A)(eng,B)	(BSc,A)	(14000,A)(12000,B)
rvalue		mel	eng	BSc	NULL
conflictType		NULL	NULL	NULL	intolerable
ovalue	e4	(lisa,A)(lisa,B)	(mgr,A)(mgr,B)	(BEng,A)	(25000,A)(25000,B)
rvalue		lisa	mgr	BEng	25000
conflictType		NULL	NULL	NULL	NULL

Table 4
Query result of Q1

oid	cname	area	Textbook
c1	(network,A)(network,B)	(system,A)(system,B)	(abc,A)(acb,B)
	network	system	NULL
	NULL	NULL	intolerable
c2	(database,A)(database,B)	(system,A)(system,B)	(def,A)(def,B)
	database	system	def
	NULL	NULL	NULL
c3	(algo,A)(algo,B)	(theory,A)(theories,B)	(xyz,A)(xyz,B)
	algo	theory	xyz
	NULL	resolvable	NULL

5.1. Queries on original attribute/relationship values

The original attribute and relationship values in the existing databases have to be examined by the database integrators during the process of deriving objects in the integrated databases in both the derivation and evolution phase. For example, the following OOQL$_{RA}$ statement (Q2) could be used to identify unresolved intolerable attribute conflict in the COURSE class (Table 5).

Example (Q2)

```
SELECT C.cname.ovalue, C.area.ovalue, C.textbook.ovalue
FROM COURSE C
WHERE C.textbook.conflictType = Intolerable
```

Since the OO$_{RA}$ model accommodates all the original attribute and relationship values in the integrated database, users can query local databases via the global schema using OOQL$_{RA}$. An example of such queries is illustrated in Q3 (Table 6).

Example (Q3)

```
SELECT C.oid, C.cname.ovalue(A), C.area.ovalue(A), C.textbook.
   ovalue(A), F.date.ovalue(A)
FROM COURSE C, C.offer.ovalue(A) F
```

In the Q3 statement, F.date.ovalue(A) represents a path expression that involves the relationship values provided by DB$_A$.

5.2. Queries on resolved attribute/relationship values

Once an integrated database is derived, OOQL$_{RA}$ allows end users to query only the resolved attribute and relationship values in the integrated database while hiding the conflicts from the users (Table 7).

Example (Q4)

```
SELECT C.cname.rvalue, F.dates.rvalue, E.ename.rvalue
FROM COURSE C, C.offer.rvalue F, F.attended_by.rvalue, E
WHERE E.position.rvalue = "eng"
```

5.3. Evolution of local databases

When an integrated database is first derived during the derivation phase, all conflicts between local instances may be fully resolved. As the local database evolves, new records are added to the databases, some old ones are removed, and other old ones get updated. These local changes may lead to un-anticipated conflict(s) in the integrated database. In this case, queries similar to Q2 can be used to identify unresolved attribute and relationship conflicts (Table 8).

Example (Q5)

```
SELECT C.oid, C.cname, C.textbook
FROM COURSE C
WHERE C.textbook.conflictType = Intolerable
```

To resolve the identified conflicts, one has to examine the cause of conflicts. If the conflicts are due to flaws in the derivation of integration data-

Table 5
Query result of Q2

cname.ovalue	area.ovalue	textbook.ovalue
(network,A)(network,B)	(system,A)(system,B)	(abc,A)(acb,B)

Table 6
Query result of Q3

C.oid	C.cname.ovalue(A)	C.area.ovalue(A)	C.textbook.ovalue(A)	F.date.ovalue(A)
c1	network	system	abc	3/2–3/5
c1	network	system	abc	5/23–5/24
c2	database	system	def	6/1–6/6
c2	database	system	def	8/30–9/1
c2	database	system	def	11/12–11/12
c3	algo	theory	xyz	NULL

base, we can define attribute threshold predicates and resolution functions using the DEFINE statements. Otherwise, the conflicts may be caused by erroneous information introduced to some local database, e.g., typographical errors made during data entry. In this case, the appropriate local database administrator can be advised to correct the error. We can further define triggers using $\mathrm{OOQL_{RA}}$ to detect intolerable attribute or relationship conflicts in the integrated objects.

5.4. Aggregate queries

Like other query languages, $\mathrm{OOQL_{RA}}$ supports aggregate queries that are often used in decision making. For example, to calculate the number of courses that are offered with a fee higher than $1000, the query below (Q6) is required.

Example (Q6)

```
COUNT(SELECT C FROM COURSE C
WHERE EXISTS F IN C.offer.rvalue: F.fee.rvalue>1000)
```

As shown in Table 9, Q6 first identifies the COURSE objects satisfying the predicate F.fee. rvalue>1000. The qualified objects are summarized by the COUNT function. Note that COUNT() is performed by counting the number of unique global objects.

Table 7
Query result of Q4

C.cname.rvalue	F.dates.rvalue	E.ename.rvalue
Network	3/2–3/5	karen
Network	5/23–5–24	mel
Database	6/1–6/6	karen
Database	8/30–9/1	mel

5.5. Characteristics of $OOQL_{RA}$

From the above query examples, we show that $\mathrm{OOQL_{RA}}$ allows users to query either conflicting or resolved information in an integrated database. It also supports direct queries on some selected local database. This flexibility is not available in the traditional object-oriented data models. The source information assigned to the attribute values not only help to distinguish the origin of the attribute values, but also provide the essential meta-information required for further conflict resolution. For example, if the course fee information from database A is more reliable than that from database B, by examining the source information in the attribute values, one derive the appropriate course fee information for the global objects.

If necessary, special mapping functions that relate the source information to other meta-information of the existing databases can be developed. These meta-information further capture the additional semantics about the existing databases that may be useful during database integration or when the attribute values are interpreted.

Table 8
Query result of Q5

oid	cname	textbook
c1	(network,A)(network,B)	(abc,A)(acb,B)
	network	NULL
	NULL	intolerable

Table 9
Query result of Q6

COUNT
2

6. Conclusions and future research

This research introduces a fresh and comprehensive approach to examine the database integration process. To support the query and integration activities in all phases of database integration, we believe that some amount of instance-level conflicts have to be accommodated by the integrated databases. Furthermore, not all instance conflicts can always be resolved during database integration. This paper examines the impact of instance conflicts on global data model. The concept of threshold predicate and resolution function have been adopted to handle both attribute and relationship conflicts. An extended object-oriented data model called OO_{RA} has been proposed to accommodate attribute and relationship conflicts. Its query language $OOQL_{RA}$ and some query examples were given.

This research can be seen as an initial effort to systematically devise different solutions to resolve as well as to accommodate instance heterogeneity in the integrated databases. This is in contrast to past database integration research, which often emphasized on conflict resolution only.

The following are some future research directions:

- *$OOQL_{RA}$ query algebra*: The query algebra of $OOQL_{RA}$ will be developed so that the formal theory of the $OOQL_{RA}$ language can be defined. The query algebra will also be useful for designing the query evaluation strategies for $OOQL_{RA}$ queries.
- *Design and implementation of a database engine based on the OO_{RA} model*: The ultimate goal of our research is to provide a thorough solution to the construction and maintenance of multidatabase or data warehousing systems. As part of our effort, a database engine based on OO_{RA} model will be developed to support both the integration and query requirements of multidatabase and data warehouse users.
- *Comprehensive classification of schema and instance conflicts*: Based on the classification of instance conflicts given in this paper and further research on schema conflicts, a comprehensive classification of schema and instance conflicts can be derived. The classification will provide better understanding of inter-database conflicts and their solutions.
- *Multidatabase views*: In Ref. [19], a five-level schema architecture similar to the CODASYL schema architecture has been proposed for multidatabase systems. With the different query and integration requirement imposed by the global application, we believe that multidatabase users should be given a flexibility to decide how the conflicts can be viewed and resolved. In this case, a flexible multidatabase view definition mechanism based on OO_{RA} model can be extremely useful. For example, users can choose different threshold predicates and resolution functions for different multidatabase views defined over the same set of local databases.

Acknowledgements

The authors wish to thank the Editor-in-Chief and anonymous reviewers for their supportive comments on earlier version of this manuscript.

References

[1] G. Aslan, D. McLeod, Semantic heterogeneity resolution in federated databases by metadata implantation and stepwise evolution, VLDB Journal 8 (2) (1999) 120–132.

[2] C. Batini, M. Lenzerini, S.B. Navathe, A comparative analysis of methodologies for database schema integration, ACM Computing Surveys 18 (4) (1986) 323–364 (December).

[3] R.G.G. Cattell (Ed.), The Object Database Standard: ODMG 2.0, Morgan Kaufmann Publishers, 1997.

[4] A. Chatterjee, A. Segev, Resolving data heterogeneity in scientific and statistical databases, International Working Conference on Scientific and Statistical Database Management, Switzerland (June), 1992.

[5] L.G. DeMichiel, Resolving database incompatibility: an approach to performing relational operations over mismatched domains, IEEE Transactions on Knowledge and Data Engineering 1 (4) (1989) 485–493 (December).

[6] M. Garcia-Solaco, F. Saltor, M. Castellanos, Semantic heterogeneity in multidatabase systems, chapter 5, in: O.A. Bukhres, A.K. Elmagarmid (Eds.), Object Oriented Multidatabase Systems: A Solution For Advanced Applications, Prentice Hall, 1996, pp. 129–202.

[7] M. Kaul, K. Drosten, E.J. Neuhold, ViewSystem: integrating heterogeneous information bases by object-oriented views, International Conference on Data Engineering, 1990, pp. 2–10, February, Los Angeles.

[8] W. Kim, J. Seo, Classifying schematic and data heterogeneity in multidatabase systems, IEEE Computer 24 (12) (1991) 12–18 (December).

[9] J.A. Larson, S.B. Navathe, R. Elmasri, A theory of attribute equivalence in databases with application to schema integration, IEEE Transactions in Software Engineering 15 (4) (1989) 449–463 (April).

[10] W.-S. Li, C. Clifton, Semantic integration in heterogeneous databases using neural networks, Proceedings of International Conference on Very Large Data Bases, 1994, pp. 1–12, Chile.

[11] E.-P. Lim, R.H.L. Chiang, Tuple source relational model: a source-aware data model for multidatabases, Data and Knowledge Journal 29 (1) (1999) 83–114.

[12] E.-P. Lim, R.H.L. Chiang, The integration of relationship instances from heterogeneous databases, Decision Support Systems 29 (2) (2000) 153–167.

[13] E.-P. Lim, J. Srivastava, Entity identification in database integration, Proceedings of International Symposium on Next Generation Database Applications, Fukuoka, Japan, 1993.

[14] E.-P. Lim, J. Srivastava, S. Prabhakar, J. Richardson, Entity identification problem in database integration, Proceedings of IEEE Data Engineering Conference, 1993, pp. 294–301, Vienna, Austria.

[15] E.-P. Lim, J. Srivastava, S. Shekhar, Resolving attribute incompatibility in database integration: an evidential reasoning approach, Proceedings of IEEE International Conference on Data Engineering, 1994, pp. 154–163, February, Houston.

[16] W. Litwin, A. Adbellatif, Multidatabase interoperability, IEEE Computer 19 (12) (1986) 10–18.

[17] H. Lu, W. Fan, C.H. Goh, S.E. Madnick, D.W. Cheung, Discovering and reconciling semantic conflicts: a data mining perspective, The Proceedings of the 7th IFIP 2.6 Working Conference on Data Semantics (DS-7), Leysin, Switzerland, 1997.

[18] P. Scheuermann, E.I. Chong, Role-based query processing in multidatabase systems, Proceedings of International Conference on Extending Database Technology, 1994, pp. 95–108, March, Cambridge.

[19] A.P. Sheth, J.A. Larson, Federated database systems for managing distributed heterogeneous, and autonomous databases, ACM Computing Surveys 22 (3) (1990) 183–236 (September).

[20] S. Spaccapietra, C. Parent, Y. Dupont, Model independent assertions for integration of heterogeneous schemas, Very Large Database Journal 1 (1) (1992) 81–126.

[21] C. Thieme, A. Siebes, An approach to schema integration based on transformations and behaviour, Proceedings of the 6th Intern'l Conf. on Advanced Information Systems Engineering (CAiSE'94), 1994.

[22] F.S.C. Tseng, A.L.P. Chen, W.-P. Yang, Answering heterogeneous database queries with degrees of uncertainty, Distributed and Parallel Databases 1 (3) (1993) 281–302.

[23] M.W.W. Vermeer, P.M.G. Apers, On the applicability of schema integration techniques to database interoperation, Proceedings of International Conference on Conceptual Modeling, 1996 pp. 179–194, Cottbus.

[24] Y.R. Wang, S.E. Madnick, The inter-database instance identification problem in integrating autonomous systems, Proceedings of IEEE International Conference on Data Engineering, 1989, pp. 46–55.

[25] R. Wang, S. Madnick, A polygen model for heterogeneous database systems: the source tagging perspective, Proceedings of International Conference on Very Large Data Bases, 1990, pp. 519–538, Brisbane.

[26] L.L. Yan, T. Ozsu, Conflict tolerant queries in AURORA, Fourth IFCIS Conference on Cooperative Information Systems (CoopIS'99), Edinburgh, Scotland (September), 1999.

Ee-Peng Lim is an Associate Professor with the School of Computer Engineering, Nanyang Technological University, Singapore. He obtained his PhD from the University of Minnesota, Minneapolis in 1994. Upon graduation, he started his academic career at the Nanyang Technological University (NTU). In 1997, he established the Centre for Advanced Information Systems and was appointed the Centre Director. He was later appointed a visiting professor at the Chinese University of Hong Kong from December 2001 to June 2003. Upon his return to NTU, he started heading the Division of Information Systems within the School of Computer Engineering. He has published more than 120 refereed journal and conference articles in the area of web warehousing, digital libraries and database integration. He is currently an Associate Editor of the ACM Transactions on Information Systems (TOIS). He is also a member of the Editorial Review Board of the Journal of Database Management (JDM). At present, he is the Program Co-Chair of the 2004 Joint Conference on Digital Libraries (JCDL2004) and also the Program Co-Chair of the Sixth International Conference on Asian Digital Libraries (ICADL 2003).

Roger Chiang is Associate Professor of Information Systems at College of Business, University of Cincinnati. He received his BS degree in Management Science from National Chiao Tung University, Taiwan, MS degrees in Computer Science from Michigan State University and in Business Administration from University of Rochester, and PhD degree in Computers and Information Systems from University of Rochester. His research interests are in data and knowledge management and intelligent systems, particularly in database reverse engineering, database integration, data mining, common sense reasoning and learning, and semantic information retrieval of Web data. He is currently on the editorial board of *Journal of AIS*, *Journal of Database Management* and *International Journal of Intelligent Systems in Accounting, Finance and Management.* He is the Program Co-Chair of 22nd International Conference on Information Systems, Research in Progress Track, 2001, and ACM International Workshop on Web Information and Data Management in 2001, 2002 and 2003. His research has been published in a number of international journals including *ACM Transactions on Database Systems, Data Base, Data and Knowledge Engineering, Decision Support Systems*, *Journal of Database Administration* and *Very Large Data Base Journal*.