View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Institutional Knowledge at Singapore Management University

Singapore Management University

Institutional Knowledge at Singapore Management University

Research Collection School Of Information Systems

School of Information Systems

11-1998

# A global object model for accommodating instance heterogeneities

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

Roger Hsiang-Li CHIANG
*University of Cincinnati*

**DOI:** https://doi.org/10.1007/978-3-540-49524-6_34

Follow this and additional works at: https://ink.library.smu.edu.sg/sis_research

Part of the Databases and Information Systems Commons, and the Numerical Analysis and Scientific Computing Commons

Citation

LIM, Ee Peng and CHIANG, Roger Hsiang-Li. A global object model for accommodating instance heterogeneities. (1998). *Conceptual Modelling ER '98: Proceedings of the 17th International Conference on Conceptual Modeling, Singapore, November 16-19*. 1507, 435-448. Research Collection School Of Information Systems.
**Available at:** https://ink.library.smu.edu.sg/sis_research/972

This Conference Proceeding Article is brought to you for free and open access by the School of Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email libIR@smu.edu.sg.

# A Global Object Model for Accommodating Instance Heterogeneities

Ee-Peng Lim[1] and Roger H.L. Chiang[2]

[1] Centre for Advanced Information Systems
School of Applied Science, Nanyang Technological University
Singapore 639798, SINGAPORE

[2] Information Management Research Centre
Nanyang Business School, Nanyang Technological University
Singapore 639798, SINGAPORE

**Abstract.** To completely address database integration problems in the context of multidatabase[10] and data warehousing systems, one has to examine various integration and query requirements. Due to various reasons such as poor data quality in local databases, ongoing local database updates, and instance heterogeneities, some instance differences have to be accommodated by the integrated databases. We have therefore proposed a new object-oriented global data model, called $OO_{RA}$, that can accommodate attribute and relationship instance heterogeneities in the integrated database. In addition, 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.

## 1 Introduction

To fully address the schema and instance integration issues in both multidatabase and data warehousing systems, one has to examine the database integration process at the macro level. Throughout the entire database integration process, inter-database heterogeneities should be handled appropriately. 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**.

– **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. 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 the global schema, as well as a mapping from local instances to global instances are specified.
– **Evolution:** Due to the autonomy of local database systems, updates to the local databases may violate the mapping from local instances to global

© Springer-Verlag Berlin Heidelberg 1998

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 multidatabase or a data warehouse.

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[1,3,4,11,14]. 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 accommodates instance heterogeneities become necessary.

## Literature Review

Most previous database integration research focused on resolving schema conflicts. Lately, as researchers begin to address instance integration problems, several solutions of instance conflict resolution have been proposed [7,8,13]. We review some data modeling research in handling instance conflicts as follows.

– *Polygen model [13]* 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 include 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 [5]* was proposed to accommodate entity and attribute conflicts in a relational integrated database. A special source attribute is assigned to every relations. 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 [9]* considered 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 (see Sect. 2).

## Research Objectives

Our research addresses 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:

– Resolving all instance differences may not be desirable because some global applications may want to retain and view these differences. 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 integrated database for different global application requirement.
– 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.
– 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.

In this paper, we 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 to support queries on integrated databases containing resolved and unresolved instances.

## 2 Instance Heterogeneities

Instance heterogeneities can be classified into **entity conflicts**, **attribute conflicts**, and **relationship conflicts** [6,8]. 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.

As pointed out by a number of researchers [2,5], instance integration may be difficult due to imperfect data quality in the legacy databases. The methods used for resolving discrepancies may also be different for different attributes. 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.

Although instances from different databases can be 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 data instances could be added to a database 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

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

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. 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 global 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. 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 differs in expected patterns, there is an 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 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.

Attribute conflicts which fail the specified thresholds are defined to be **intolerable**. Database integrators should be alerted for intolerable attribute conflicts by having the intolerable attribute conflicts recorded in a log file.

### 2.3 Relationship Conflicts

Relationship conflicts, first discussed in [6], arise when the relationship between two real-world entities may not be represented consistently in different databases. In [6], 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.

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 A Database Integration Example

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 methods predefined by database integrators. For new instance conflicts that cannot be resolved automatically, database integrators have to be called upon to handle them. Nevertheless, before any actions can be taken by the database integrators, 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.

We employ an integration scenario to demonstrate the attribute and relationship conflicts. Figure 1 depicts the object-oriented schemas of the local databases ($DB_A$ and $DB_B$) containing company information. Here, we assume that schema integration has been performed and the schemas of existing databases have been made compatible to facilitate instance comparisons.

The instances of $DB_A$ and $DB_B$ are shown in Figs. 2 and 3 respectively. Suppose all entity conflicts are resolved by matching *ename* and *dname* of employee and department instances respectively. We notice that John in $DB_A$ works in the Marketing department but John in $DB_B$ works in the Research department. This is a relationship conflict. On the other hand, the difference in salary for Chen in $DB_A$ and $DB_B$ is an attribute conflict. Figure 4 depicts the integrated schema derived from $DB_A$ and $DB_B$.

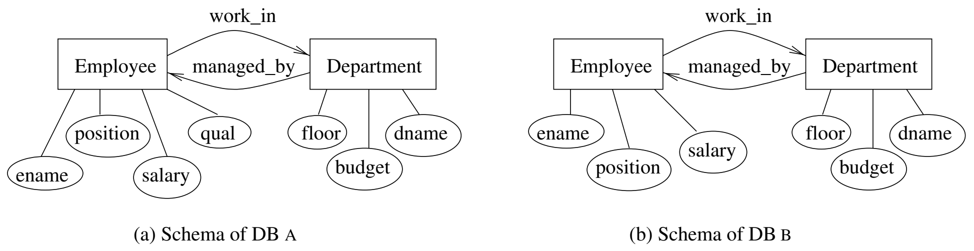

**Fig. 1.** Schemas of Local Databases

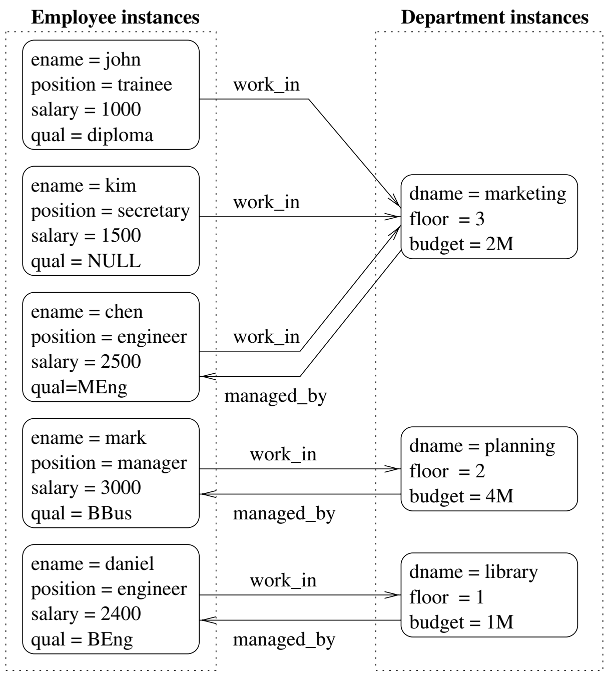

**Fig. 2.** Instances of $DB_A$

## 4 The $OO_{RA}$ Object-Oriented Data Model

In this section, we propose the $OO_{RA}$[2], 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 integra-

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

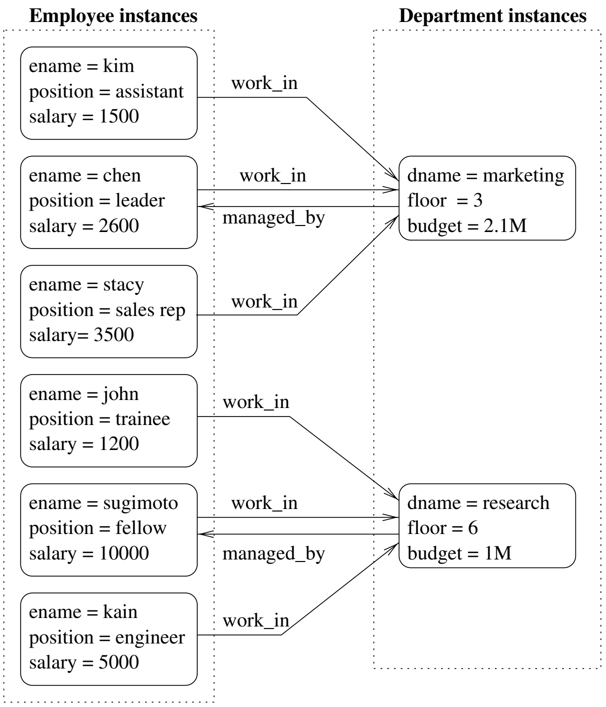

**Fig. 3.** Instances of $DB_B$

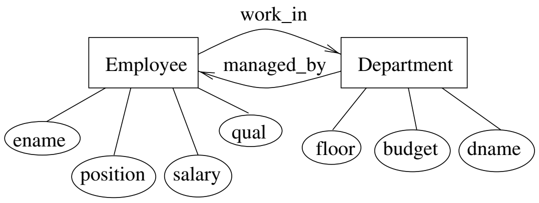

**Fig. 4.** Integrated Schema

tion 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 `ename` to match Employee objects, and `dname` to match Department objects from $DB_A$ and $DB_B$. The following two data definition statements have been used to identify matching local objects:

```
DERIVE EMP from Employee@DB_A, Employee@DB_B
   USING Employee@DB_A(ename), Employee@DB_B(ename);

DERIVE DEPT from Department@DB_A, Department@DB_B
   USING Department@DB_A(dname), Department@DB_B(dname);
```

### 4.2 Threshold Predicates and Resolution Functions

A **threshold predicate** and a **resolution function** can be defined for each attribute in the global schema. Given an attribute in a class of global objects, the threshold predicate determines for each global object if a difference between local values of the attribute is tolerable. The resolution function is then specified to resolve tolerable attribute conflicts automatically. Depending on the characteristics of attributes, different threshold predicates and resolution functions should be defined and be implemented using system-defined functions/operators or general programs.

Given an attribute in the global schema, three combinations of threshold predicates and resolution functions can be constructed[3]:

– *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

[3] 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 shouldn't be resolved by a resolution function automatically.

`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.

The `A.ovalue` of a global object is defined to be a set of $(value, database\_id)$ pairs where *value* denotes the attribute value contributed by the corresponding object from the existing database identified by *database_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 `A.threshold()`. If the conflict is tolerable, `A.rvalue` is obtained by applying `A.resolution()` on the local attribute values. In the event where the conflict is intolerable or `A.resolution()` is undefined, NULL is assigned to `A.rvalue`.

Depending on the original attribute values and the threshold predicate defined for the attribute, different conflict types can be derived and be captured in `A.conflictType`. `A.conflictType` is assigned 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 a intolerable conflict.

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

```
DEFINE salary.threshold@EMP(s1,s2) =    (abs(s1-s2) ≤ 100)
DEFINE salary.resolution@EMP(s1,s2) =   max(s1,s2)
DEFINE position.threshold@EMP(p1,p2) =  (p1=p2) or
                                        (p1=secretary and p2=assistant)
```

With the above definition, the salary values of 2500 and 2600 for the employee Chen constitute a resolvable attribute conflict. The global object for the employee Chen will have a resolved salary value of 2600 computed by the resolution function. On the other hand, the salary values of 1000 and 1200 for the employee John constitute an intolerable attribute conflict. In this situation, database integrators should be alerted and the conflict should be resolved manually. Since only threshold predicate is defined for the position attribute, the position values of secretary and assistant constitute an acceptable conflict.

## 4.4 Relationship Conflicts

In the $OO_{RA}$ 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 the attribute conflict, 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.

For illustration, let say the research department is in fact part of the marketing department. The following threshold predicate and resolution function can be defined.

```
DEFINE work_in.threshold@EMP(o1,o2) = (o1 = o2) ∨ (∀ o, o.oid ∈ {o1,o2},
o.dname ∈ {research,marketing})
DEFINE work_in.resolution@EMP(o1,o2) = o1 if (o1=o2), marketing otherwise
```

In the above statements, `o1` and `o2` denotes global object ids of DEPT objects.

### 4.5 Integrated Database Instances

The $OO_{RA}$ objects of the integrated database are shown in Tables 1 and 2. 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 and 2, 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. For example, the following threshold predicate and resolution function are defined for budget attribute.

DEFINE budget.threshold@DEPT(b1,b2) = ($\frac{abs(b1-b2)}{max(b1,b2)} \leq$ 5%)
DEFINE budget.resolution@DEPT(b1,b2) = min(b1,b2)

**Table 1.** EMP's Global Objects

| **Attrib-element** | **oid** | **ename** | **position** | **salary** | **qual** | **work_in** |
|---|---|---|---|---|---|---|
| ovalue | e1 | (john,A)(john,B) | (trainee,A) (trainee,B) | (1000,A) (1200,B) | (diploma,A) | (d1,A)(d4,B) |
| rvalue | | john | trainee | NULL | diploma | d1 |
| conflictType | | NULL | NULL | Intolerable | NULL | Resolvable |
| ovalue | e2 | (kim,A)(kim,B) | (secretary,A) (assistant,B) | (1500,A) (1500,B) | (NULL,A) | (d1,A)(d1,B) |
| rvalue | | kim | NULL | 1500 | NULL | d1 |
| conflictType | | NULL | Acceptable | NULL | NULL | NULL |
| ovalue | e3 | (chen,A)(chen,B) | (engineer,A) (leader,B) | (2500,A) (2600,B) | (MEng,A) | (d1,A)(d1,B) |
| rvalue | | chen | NULL | 2600 | MEng | d1 |
| conflictType | | NULL | Intolerable | Resolvable | NULL | NULL |
| ovalue | e4 | (mark,A) | (manager,A) | (3000,A) | (BBus,A) | (d2,A) |
| rvalue | | mark | manager | 3000 | BBus | d2 |
| conflictType | | NULL | NULL | NULL | NULL | NULL |
| ovalue | e5 | (daniel,A) | (engineer,A) | (2400,A) | (BEng,A) | (d3,A) |
| rvalue | | daniel | engineer | 2400 | BEng | d3 |
| conflictType | | NULL | NULL | NULL | NULL | NULL |
| ovalue | e6 | (stacy,B) | (sales rep,B) | (3500,B) | (NULL,B) | (d1,B) |
| rvalue | | stacy | sales rep | 3500 | NULL | d1 |
| conflictType | | NULL | NULL | NULL | NULL | NULL |
| ovalue | e7 | (sugimoto,B) | (fellow,B) | (10000,B) | (NULL,B) | (d4,B) |
| rvalue | | sugimoto | fellow | 10000 | NULL | d4 |
| conflictType | | NULL | NULL | NULL | NULL | NULL |
| ovalue | e8 | (kain,B) | (engineer,B) | (5000,B) | (NULL,B) | (d4,B) |
| rvalue | | kain | engineer | 5000 | NULL | d4 |
| conflictType | | NULL | NULL | NULL | NULL | NULL |

## 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 $OOQL_{RA}$. $OOQL_{RA}$ adapts the existing SQL syntax for object-oriented queries. In addition, it is specially designed to support the query requirement for an integrated database containing attribute and relationship conflicts in the derivation and evolution phases of database integration. A $OOQL_{RA}$ SELECT query statement can be expressed as:

**Table 2.** DEPT's Global Objects

| **Attribute-element** | **oid** | **dname** | **floor** | **budget** | **managed_by** |
|---|---|---|---|---|---|
| ovalue | d1 | (marketing,A)(marketing,B) | (3,A)(3,B) | (2M,A)(2.1M,B) | (e3,A)(e3,B) |
| rvalue | | marketing | 3 | 2M | e3 |
| conflictType | | NULL | NULL | Resolvable | NULL |
| ovalue | d2 | (planning,A) | (2,A) | (4M,A) | (e4,A) |
| rvalue | | planning | 2 | 4M | e4 |
| conflictType | | NULL | NULL | NULL | NULL |
| ovalue | d3 | (library,A) | (1,A) | (1M,A) | (e5,A) |
| rvalue | | library | 1 | 1M | e5 |
| conflictType | | NULL | NULL | NULL | NULL |
| ovalue | d4 | (research,B) | (6,B) | (1M,B) | (e7,B) |
| rvalue | | research | 6 | 1M | e7 |
| conflictType | | NULL | NULL | NULL | NULL |

```
SELECT <target attribute 1>, ..., <target attribute m>
FROM <table 1>, ..., <table n>
WHERE <predicate expression>;
```

Unlike the usual SQL statements, every non-oid attribute (say `A`) found in a $OOQL_{RA}$ 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.

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

## 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 (Q1) could be used to identify unresolved intolerable attribute conflict in the `EMP` class.

**Example (Q1):**

```
SELECT E.oid,E.ename.ovalue,E.position.ovalue,
E.salary.ovalue,E.qual.ovalue
FROM EMP E
WHERE E.ename.conflictType=Intolerable OR
E.position.conflictType=Intolerable OR
E.salary.conflictType=Intolerable OR
E.qual.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 $OOQL_{RA}$ statement. An example of such queries is illustrated in Q2.

**Example (Q2):**

```
SELECT E.oid,E.ename.ovalue(A),E.position.ovalue(A),
E.salary.ovalue(A),E.qual.ovalue(A),E.work_in(A).dname.ovalue(A)
FROM EMP E;
```

**Table 3.** Query Result of Q1

| oid | ename.ovalue | position.ovalue | salary.ovalue | qual.ovalue |
|---|---|---|---|---|
| e1 | (john,A)(john,B) | (trainee,A)(trainee,B) | (1000,A)(1200,B) | (diploma,A) |
| e3 | (chen,A)(chen,B) | (engineer,A)(leader,B) | (2500,A)(2600,B) | (MEng,A) |

**Table 4.** Query Result of Q2

| oid | ename.ovalue(A) | position.ovalue(A) | salary.ovalue(A) | qual.ovalue(A) | work_in.ovalue(A).dname.ovalue(A) |
|---|---|---|---|---|---|
| e1 | john | trainee | 1000 | diploma | marketing |
| e2 | kim | secretary | 1500 | NULL | marketing |
| e3 | chen | engineer | 2500 | MEng | marketing |
| e4 | mark | manager | 3000 | BBus | planning |
| e5 | daniel | engineer | 2400 | BEng | library |

## 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.

**Example (Q3):**

```
SELECT E.ename.rvalue,E.work_in.rvalue.dname.rvalue,
E.work_in.rvalue.budget.rvalue
FROM EMP E;
```

**Table 5.** Query Result of Q3

| ename.rvalue | E.work_in.rvalue.dname.rvalue | E.work_in.rvalue.budget.rvalue |
|---|---|---|
| john | marketing | 2M |
| kim | marketing | 2M |
| chen | marketing | 2M |
| mark | planning | 4M |
| ⋮ | ⋮ | ⋮ |

## 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 Q1 can be used to identify unresolved attribute and relationship conflicts.

**Example (Q4):**

```
SELECT E.oid,E.ename,E.position,E.salary,E.qual,E.work_in
FROM EMP E
WHERE E.ename.conflictType=Intolerable OR
E.position.conflictType=Intolerable OR
```

```
E.salary.conflictType=Intolerable OR
E.qual.conflictType=Intolerable OR
E.work_in.conflictType=Intolerable;
```

**Table 6.** Query Result of Q4

| oid | ename | position | salary | qual | work_in |
|---|---|---|---|---|---|
| e1 | (john,A)(john,B)<br>john<br>NULL | (trainee,A)(trainee,B)<br>trainee<br>NULL | (1000,A)(1200,B)<br>NULL<br>Intolerable | (diploma,A)<br>diploma<br>NULL | (d1,A)(d4,B)<br>d1<br>Resolvable |
| e3 | (chen,A)(chen,B)<br>chen<br>NULL | (engineer,A)(leader,B)<br>NULL<br>Intolerable | (2500,A)(2600,B)<br>2600<br>Resolvable | (MEng,A)<br>MEng<br>NULL | (d1,A)(d1,B)<br>d1<br>NULL |

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 database, 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.

## 6 Conclusions

This research introduces a novel 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.

## References

1. C. Batini, M. Lenzerini and S.B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Survey*, 18(4), December 1986.
2. M. Garcia-Solaco, F. Saltor and M. Castellanos. *Semantic Heterogeneity in Multidatabase Systems*, chapter 5, pages 129–202. Prentice Hall, 1996.
3. W. Kim and J. Seo. Classifying Schematic and Data Heterogeneity in Multidatabase Systems. *IEEE Computer*, December, 1991.
4. J.A. Larson, S.B. Navathe and R. Elmasri. A Theory of Attribute Equivalence in Databases with Application to Schema Integration. *IEEE Transactions in Software Engineering*, 15(4), April 1989.

5. E.-P. Lim, R.H.L. Chiang and Y.Y. Cao. Tuple Source Relational Model: A Source-Aware Data Model for Multidatabases. in *Data & Knowledge Engineering*, Forthcoming, 1999.
6. E.-P. Lim and R.H.L. Chiang. Resolving instance-level relationship conflicts in database integration. In *Workshop on Information Technologies and Systems (WITS'97)*, Atlanta, Georgia, December 1997.
7. E.-P. Lim, J. Srivastava, S. Prabhakar, and J. Richardson. Entity Identification Problem in Database Integration. In *Proceedings of IEEE Data Engineering Conference*, Vienna, Austria, 1993.
8. E.-P. Lim, J. Srivastava, and S. Shekhar. Resolving Attribute Incompatibility in Database Integration: An Evidential Reasoning Approach. In *IEEE International Conference on Data Engineering*, Houston, TX, February 1994.
9. P. Scheuermann, and E.I. Chong. Role-based Query Processing in Multidatabase Systems. In *International Conference on Extending Database Technology*, Cambrige, U.K., March 1994.
10. A.P. Sheth and J.A. Larson. Federated database systems for managing distributed heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3), September 1990.
11. S. Spaccapietra, C. Parent and Y. Dupont. Model Independent Assertions for Integration of Heterogeneous Schemas. *Very Large Database Journal*, 1(1), 1992.
12. Y.R. Wang and S.E. Madnick. The Inter-database Instance Identification Problem in Integrating Autonomous Systems. In *IEEE International Conference on Data Engineering*, 1989.
13. R. Wang and S. Madnick. A Polygen Model for Heterogeneous Database Systems: The Source Tagging Perspective. In *International Conference on Very Large Data Bases*, pages 519–538, Brisbane,Australia, 1990.
14. M.W.W. Vermeer and P.M.G. Apers. On the applicability of schema integration techniques to database interoperation. In *Entity-Relationship Conference*, Cottbus,Germany, 1996.