6-1996

Specifying Object-oriented Federated Database from Existing Databases

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

M. L. LIM

J. SRIVASTAVA

Citation

LIM, Ee Peng; LIM, M. L.; and SRIVASTAVA, J.. Specifying Object-oriented Federated Database from Existing Databases. (1996). *Intelligent information management systems: Proceedings of the IASTED/ISMM International Conference, June 5-7, 1996, Washington, DC.* Research Collection School Of Information Systems.
Available at: https://ink.library.smu.edu.sg/sis_research/908

Specifying Object-Oriented Federated Databases from Existing Databases

Ee-Peng Lim, Meng-Liang Lim, Jaideep Srivastava*
Sch. of Applied Science, Nanyang Technological Univ.
Nanyang Ave., Singapore 639798

1 Introduction

A major challenge in building an object-oriented (OO) federated database system (FDBS) is to be able to define the OO federated DBs from the existing DBs and to support queries on the federated DBs. In this paper, we present a mapping strategy that is based on a proposed set of DB integration operations. We first define an OO federated DB as a virtual view on multiple OO export DBs. Our DB mapping strategy systematically derives each of the class extents, deep class extents and relationships of the federated DB using an operator tree consisting of the integration operations. This mapping approach differs from the other existing approaches in that it is algebraic based, and is therefore very suitable for implementing federated query processing.

In the case of classic relational model, a core set of operations have been used to characterize the notion of **relational completeness**. The core operations are then used to design relational query languages (e.g. SQL), and to verify their correctness. In FDBS, we believe that a similar approach should be adopted. Hence, we begin with designing integration operations which are used in our proposed mapping strategy. With these integration operations, it will be easier to design declarative DB integration languages to construct a federated DB, as well as to compare the expressiveness of different integration approaches.

In our work, we adopt a 3-level schema architecture. The **local schemas** describe the local DBs in their respective data models. The **export schemas** describe the subsets of local DBs made available to the federated DB users. The export schemas, unlike the local schemas, are in the common OO data model. One or more export schemas may be defined upon a local schema allowing different aspects of the local schema to be tapped by different FDBSs. Each export schema can participate in the construction of none or more **federated schemas**[1]. At the federated schema level, we reconcile the discrepancies among export DBs.

*Dept. of Computer Science, University of Minnesota, Minneapolis.

[1]We allow some export schemas to be directly accessed by the global users.

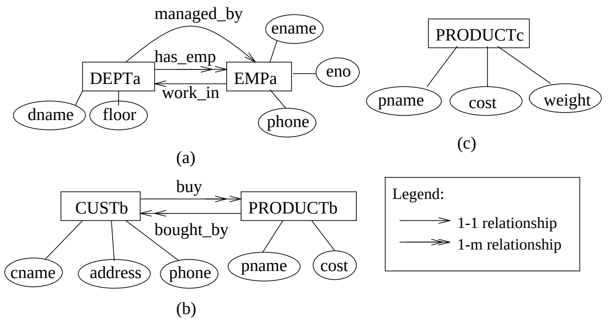

Figure 1: Export DBs: (a) DB_a (b) DB_b (c) DB_c

2 Example of OO Export schemas

Throughout this paper, three OO export DBs are used to demonstrate our DB mapping and query processing approaches. These export DBs are shown in Figure 1. DB_a, DB_b and DB_c are three export DBs modelling information about a company. Their classes and attributes are self-explanatory. In DB_a, each employee has a unique name and is assigned a unique employee number. DB_b maintains information about the customers and the products they purchased in the past. DB_c is a warehouse DB that keeps all the product information. The cost attribute in $PRODUCT_b$ is the amount the customer paid in the last transaction. The amount may be different from the cost provided by DB_c since small discounts on certain products may be given to customers in order to keep the company competitive.

3 Mapping from OO Export Schemas to OO Global Schema

OO-Myriad adopts an algebraic approach to express the DB mapping. In this approach, the federated schema can be freely specified as long as its objects can be computed from the participating export DBs. This achieves independence between the global schema structures and the local schema structures.

3.1 Deriving Global Classes

In deriving the global classes, the three main issues to be dealt with are: **entity identification** [2, 3], **attribute value conflict resolution** [4], and **global oid generation**. Entity identification is the process of identifying export objects that model the same real-world objects and hence to be merged. Attribute value conflict resolution determines how the attributes of global objects can be derived from the attributes of their export objects.

Entity Identification
We consider three ways in which a global class is derived from the export schemas: (i) A global class can be directly derived from an export class. In this case, a global class object corresponds to a single export DB object. (ii) A global class can be derived from multiple export classes where each global class object corresponds to a single export DB object. (iii) A global class can be derived from multiple export classes where each global class object corresponds to multiple export DB objects. Since each global object representing a real-world entity can be derived from a single or multiple export DB objects, it is essential that the export classes be merged together share some common attribute(s), which determines whether their objects correspond to the same real-world entities. We call these common attribute(s) the **entity key**.

Attribute Value Conflict Resolution
Once the export DB objects corresponding to the same real-world entities have been identified, it is often necessary to resolve any conflict in their attribute values before merging them into a global object. In this respect, OO-Myriad adheres quite closely to the Dayal's approach of using aggregate functions as attribute derivation functions[1]. To carry the idea further, OO-Myriad allows the functions to be user-defined.

Global Object Id
In OO-Myriad, each global object has a unique identifier. To ensure the uniqueness of oids across global classes, any oid generation method selected to yield global oids must be applied *throughout all global classes* in a federated DB.

- **Method 1:** The global oids can be computed by combining the entity key values and name of the global class. If the global class is involved in a class lattice, it is necessary to designate a representative global class for the lattice and use it (together with entity key values) to generate the global oids of objects which belong to the class lattice[2]. This is achieved by applying an oid generation function on the entity key and the representative global class name.

- **Method 2:** The global oids can be computed by combining the export oids and export DB name. If more than one export object models the same real-world entity, only an export oid - export DB name pair is needed to generate the global object id.

[2]To be formally correct, it should be class poset instead of class lattice.

While the merit of the first method is to accommodate pseudo-object oids, it forbids a global object to migrate from one global class to another. If a global object has to migrate, it will have to assume a different global oid. The second approach does not suffer from the above limitation but it requires the export DBs to support unique oids.

3.2 Deriving Global Relationship

A relationship attribute links one class to another. Let R_A be a relationship attribute that links a global class C_A to another global class C_B. There are several possible ways R_A can be derived. For example:

- C_A and C_B directly correspond to two export classes EC_A and EC_B respectively, both of which are in the same export DB. R_A corresponds to a relationship attribute ER_A that links EC_A to EC_B.

- C_A and C_B directly correspond to two export classes EC_A and EC_B respectively, both of which are in the same export DB. R_A corresponds to the reverse of a relationship attribute ER_B that links EC_B to EC_A.

- C_A and C_B directly correspond to two export classes EC_{A1} and EC_{B2} respectively, each in a different export DB. R_A corresponds to a simple attribute of EC_{A1} such that the attribute domain is the entity key of EC_{B2}.

- C_A is derived by merging export classes EC_{A1} and EC_{A2}. C_B is derived by merging export classes EC_{B1} and EC_{B2}. EC_{A1} and EC_{B1} are from export DB 1 and EC_{A2} and EC_{B2} are from export DB 2. R_A is a combination of relationship attributes ER_{A1} and ER_{B2}, where ER_{A1} and ER_{B2} link from EC_{A1} to EC_{B1} and from EC_{B2} to EC_{A2} respectively.

In OO-Myriad, each relationship attribute in a global class is assigned an algebraic expression that computes the object pairs linking the class to the destination class. Each object pair contains the oids or entity keys of the global objects involved in the relationship. Examples of deriving global relationships will be given in Section 3.5.

3.3 Deriving Deep Class Extent

In OO-Myriad, a global object belongs to only a global class. The set of objects that belong to a class is known as the **class extent**. The **deep extent** of a class refers to the union of the class extent and the extent of all its direct and indirect subclasses. Given a class C, we use C to denote its class extent and $C*$ to denote its deep extent. Despite the implicit mathematical relationship between the deep extent of a class and its extent, as well as the extent of all its subclasses, OO-Myriad allows the derivation of deep class extent to be based on integration semantics independent of this mathematical relationship. For each global class, we have to specify the algebraic expressions that compute its class extent and deep class extent.

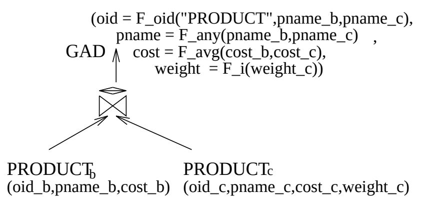
(oid = F_oid("PRODUCT",pname_b,pname_c),
pname = F_any(pname_b,pname_c) ,
GAD
cost = F_avg(cost_b,cost_c),
weight = F_i(weight_c))
PRODUCTb
(oid_b,pname_b,cost_b)
PRODUCTc
(oid_c,pname_c,cost_c,weight_c)

Figure 2: Example of using GAD operation

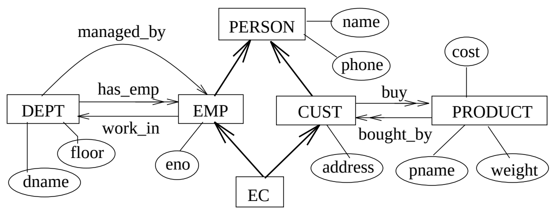

Figure 3: Example Global Schema

3.4 Integration Operations

In OO-Myriad, we focus on integration operations that are often used. In the domain of relational FDBS, the predecessor of OO-Myriad, i.e. Myriad, has adopted the conventional relational operations, e.g. $\bowtie$, σ, π, $-$ and $\cup$ as well as two additional integration operations, namely two-way outerjoin (denoted by $\overleftrightarrow{\bowtie}$) and *generalized attribute derivation* operation (denoted by GAD). The $\overleftrightarrow{\bowtie}$ operation assembles multiple sets of objects from different export DBs which model the same set of real-world entities. Predicates are associated with the $\overleftrightarrow{\bowtie}$ to determine the export objects that correspond to the same real-world entities and therefore can be merged together. GAD is an unary operation that derives attributes of global objects using any system- or user-defined resolution functions.

Example: Suppose we wish to integrate the objects from export classes $PRODUCT_b$ and $PRODUCT_c$ given in Section 2. We can first outerjoin the objects from $PRODUCT_b$ and $PRODUCT_c$ followed by a GAD operation that merges the matching objects together. Figure 2 depicts this[3].

In this example, F_oid is an **oid generation function**. It computes oid from a class name and an entity key value. In this case, we have adopted the oid generation method 1. F_any returns any non-null input values, F_avg returns average of input values and F_i is the identity function. In general, attribute resolution functions can be treated as black boxes. By permitting them to be user-defined, we have made the GAD operation flexible enough to resolve a wide variety of attribute value conflicts.

Other than the extended GAD operation, we have defined an outer-difference operation (denoted by $\ominus$) to distinguish the export objects representing real-world entities modeled by only one of the two export classes that model overlapping sets of real-world entities.

Definition: (Outer-difference) Let $O_1(A)$ and $O_2(B)$ be two sets of export objects and $p(X,Y)$ be a predicate on X and Y attributes of O_1 and O_2 respectively ($X \subseteq A$, and $Y \subseteq B$). p is the predicate that determines if two export objects represent the same real-world entity.
$O_1 \ominus_{p(X,Y)} O_2 = \{o_1 | o_1 \in O_1 \wedge \neg\exists o_2 \in O_2 \text{ s.t. } p(o_1.X, o_2.Y)\}$

[3]We have renamed the attributes of classes to avoid attribute name conflicts.

3.5 Example of Algebraic Approach to DB Mapping

In this section, we demonstrate the algebraic approach to DB mapping using a federated DB example. Suppose the federated schema the global users wish to construct upon our export DB example (refer to Section 2) is shown in Figure 3.

In this federated schema example, the real-world employee and customer entities have been grouped into EMP, $CUST$, $PERSON$ and EC classes. EC, being a common subclass of EMP and $CUST$, keeps information about people who are both employees and customers of the company.

In additional to the knowledge about federated schema, we assume that the following integration semantics are available:

- Each global object in $DEPT$ corresponds to an export object in $DEPT_a$ and the department name *dname* is the entity key for department entities.
- EMP_a and $CUST_b$ are to be combined in a way to form the global class lattice consisting EMP, $CUST$, $PERSON$ and EC. Suppose all entities modelled by these classes can be identified by their names, i.e. *name* is the entity key.
- $PRODUCT_b$ and $PRODUCT_c$ are to be combined into $PRODUCT$ global class. Some $PRODUCT$ objects correspond to either $PRODUCT_b$ or $PRODUCT_c$ export objects while other $PRODUCT$ objects can be merged from $PRODUCT_b$ and $PRODUCT_c$ export objects. Here, we assume that product entities have product name as the entity key.
- The oid of each global class can be derived by a function of the global class name and its respective entity key.

Apart from specifying the global and export schema information, the OO-Myriad federated DB administrators must define the algebraic mappings that derive (i) extent of global classes, (ii) deep extent of global classes, and (iii) object pairs of global relationships. Each algebraic mapping is represented as an operator tree involving the integration operations supported by OO-Myriad.

Figure 4 depicts the operator trees that define the class extent of EMP, EC, and $PRODUCT$. Since $PERSON$, EMP, $CUST$ and EC are involved in a class lattice, we have chosen $PERSON$ to be the

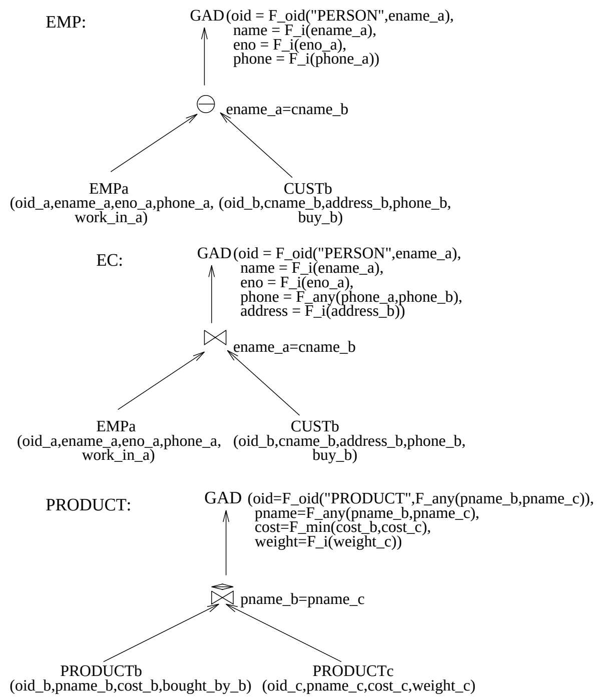
EMP:
GAD (oid = F_oid("PERSON",ename_a), name = F_i(ename_a), eno = F_i(eno_a), phone = F_i(phone_a))
ename_a=cname_b
EMPa (oid_a,ename_a,eno_a,phone_a, work_in_a)
CUSTb (oid_b,cname_b,address_b,phone_b, buy_b)
EC:
GAD (oid = F_oid("PERSON",ename_a), name = F_i(ename_a), eno = F_i(eno_a), phone = F_any(phone_a,phone_b), address = F_i(address_b))
ename_a=cname_b
EMPa (oid_a,ename_a,eno_a,phone_a, work_in_a)
CUSTb (oid_b,cname_b,address_b,phone_b, buy_b)
PRODUCT:
GAD (oid=F_oid("PRODUCT",F_any(pname_b,pname_c)), pname=F_any(pname_b,pname_c), cost=F_min(cost_b,cost_c), weight=F_i(weight_c))
pname_b=pname_c
PRODUCTb (oid_b,pname_b,cost_b,bought_by_b)
PRODUCTc (oid_c,pname_c,cost_c,weight_c)

Figure 4: Algebraic Mappings for Global Class Extent

representative class of this collection of global classes and use it in F_{oid} to generate oids for objects in these global classes. Figure 5 depicts the operator trees that define the deep extent of $PERSON$ and EMP.

Figure 6 depicts the operator trees that define the object pairs of the relationships *managed_by* and *has_emp*. In this example, each object pair contains the entity keys of the global entities involved in the relationship. To de-reference a relationship attribute of an export class such as *managed_by* in $DEPT_a$, we have introduced a new operation **DREF** which replaces one or more relationship attributes by some attributes of the destination classes.

Definition: (De-reference - DREF) Let O be a set of objects. Let $R_1, \cdots, R_m$ be relationship attributes of O and X be the remaining attributes. For each i, let $R_i.A_i$ be the attributes accessible through R_i. $DREF(O, R_1 \rightarrow R_1.A_1, \cdots, R_m \rightarrow R_m.A_m) = \{< o.X, o.R_1.A_1, \cdots, o.R_1.A_m > | o \in O\}$

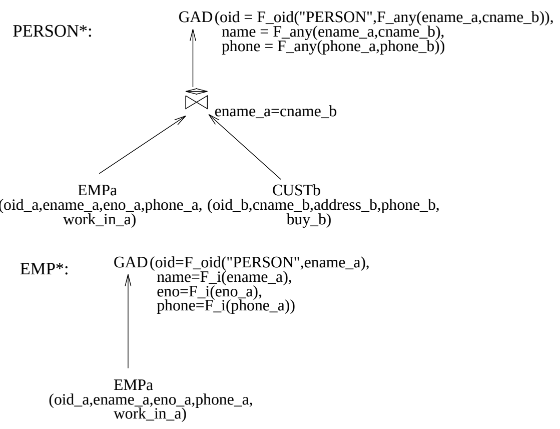
PERSON*:
GAD (oid = F_oid("PERSON",F_any(ename_a,cname_b)), name = F_any(ename_a,cname_b), phone = F_any(phone_a,phone_b))
ename_a=cname_b
EMPa (oid_a,ename_a,eno_a,phone_a, work_in_a)
CUSTb (oid_b,cname_b,address_b,phone_b, buy_b)
EMP*:
GAD (oid=F_oid("PERSON",ename_a), name=F_i(ename_a), eno=F_i(eno_a), phone=F_i(phone_a))
EMPa (oid_a,ename_a,eno_a,phone_a, work_in_a)

Figure 5: Algebraic Mappings for the Deep Extent of Global Classes

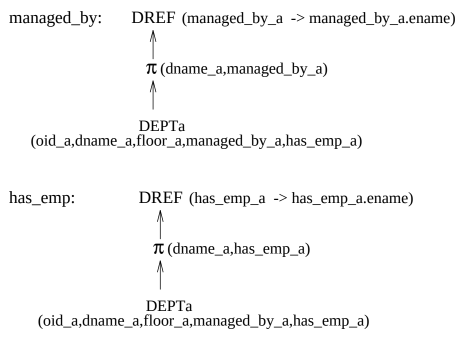
managed_by:
DREF (managed_by_a -> managed_by_a.ename)
π (dname_a,managed_by_a)
DEPTa (oid_a,dname_a,floor_a,managed_by_a,has_emp_a)
has_emp:
DREF (has_emp_a -> has_emp_a.ename)
π (dname_a,has_emp_a)
DEPTa (oid_a,dname_a,floor_a,managed_by_a,has_emp_a)

Figure 6: Algebraic Mappings for Global Relationships

4 Conclusions

In this paper, we begin with a careful examination of different object-oriented schema constructs in a federated DB and their derivation from the participating export DBs. The schema constructs that can be derived include the global object ids, the extent and deep extent of global classes, and global relationships. To derive a federated DB, we propose a set of integration operations to be used in an algebraic mapping approach that is designed to merge export DBs together. The proposed integration operations include two-way outerjoin, outer-difference($\ominus$), generalized attribute derivation(GAD), de-reference($DREF$), and other usual relational operations such as join, project and selection. This mapping approach is known to be flexible and extensible. We also illustrate the usefulness of the proposed DB mapping using an example federated DB and its component export DBs.

References

[1] U. Dayal and H-Y. Hwang. View definition and generalization for database integration in multibase: A system for heterogeneous distributed databases. *IEEE Trans. Software Eng.*, SE-10(6), November 1984.

[2] W. Kent. The entity join. In *Proc. of the 5th VLDB Conf.*, 1979.

[3] E-P. Lim, J. Srivastava, S. Prabhakar, and J. Richardson. Entity identification problem in database integration. *9th International Conference on Data Engineering*, 1993.

[4] E-P. Lim, J. Srivastava, and S. Shekhar. Resolving attribute incompatibility in database integration: An evidential reasoning approach. *10th International Conference on Data Engineering*, 1994.