Using Object Deputy Model to Prepare Data for Data Warehousing

Zhiyong Peng, *Member*, *IEEE*, Qing Li, *Senior Member*, *IEEE*, Ling Feng, *Member*, *IEEE*, Xuhui Li, and Junqiang Liu

Abstract—Providing integrated access to multiple, distributed, heterogeneous databases and other information sources has become one of the leading issues in database research and the industry. One of the most effective approaches is to extract and integrate information of interest from each source in advance and store them in a centralized repository (known as a *data warehouse*). When a query is posed, it is evaluated directly at the warehouse without accessing the original information sources. One of the techniques that this approach uses to improve the efficiency of query processing is *materialized view(s)*. Essentially, materialized views are used for data warehouses, and various methods for relational databases have been developed. In this paper, we will first discuss an object deputy approach to realize materialized object views for data warehouses which can also incorporate object-oriented databases. A framework has been developed using Smalltalk to prepare data for data warehousing, in which an object deputy model and database connecting tools have been implemented. The object deputy model can provide an easy-to-use way to resolve inconsistency and conflicts while preparing data for data warehousing, as evidenced by our empirical study.

Index Terms—Data preparation, data warehousing, data fusion/integration, object deputy model, conflict resolution, duplicate handling.

1 Introduction

The ability to act quickly and decisively in today's increasingly competitive marketplace is critical to the success of many organizations. The volume of information that is available to corporations is rapidly increasing and frequently overwhelming. Those organizations that effectively and efficiently manage this vast amount of data and use this information to make business decisions realize a significant competitive advantage in the marketplace. Data analysis, reporting, and query tools help business workers sift through tomes of data to extract valuable information from it. Software and hardware systems that support such "Business Intelligence" (BI) applications are often known as Decision Support Systems (DSS).

Research surveys predict an explosion in implementation and sales for the business intelligence/data warehousing (BI/DW) industry. According to IDC's study [16], the market for packaged data marts/warehouses was $112.6 million in 1999, while it is expected to grow at a compound annual growth rate (CAGR) of 43.1 percent to reach $674.5 million by the end of 2004. In a white paper by Knightsbridge [20], it quotes META Group as saying that "all enterprises are anticipating exponential data warehouse growth, with average raw data exceeding one terabyte in 2003/04 and three terabytes in 2005/06." Another report by Survey.com [28], based on surveying hundreds of organizations, concludes that the overall worldwide investment in data warehousing and business intelligence will exceed $150 billion by 2005. Over the years, BI/DW has evolved in industries like retail (customer profiling, inventory management, trend forecasting), financial services (risk analysis, fraud detection), and manufacturing (order shipment, customer support) to new areas in clinical analysis, governmental surveys, and others. Decision Support Systems in these domains clearly exhaust traditional manual methods of data analysis such as spreadsheets and ad hoc queries. This motivates the need for a new breed of DSS—Data Warehousing and Online Analytic Processing, recent initiatives which have resulted in many successful, high-return applications of information technology.

1.1 Problem Overview

The issues to be dealt with while designing any data warehouse solution can, in a way, be thought of as an extension of database design issues (distributed database or multidatabase systems). While the output specifications of a DW are similar, as far as integrating schema and maintaining semantic heterogeneity are concerned and, in some cases, as far as developing views go, there are certain key issues unique to each environment.

The work presented in this paper highlights some issues related to data preparation for data integration in a distributed data warehouse environment. The framework of our research, as illustrated in Fig. 1, is called *FOODMAW* (standing for "Framework of Object Oriented Data Mining And Warehousing") which is a modified version of [17]. Here, the *wrappers* convert data from each source into a common model and also provide a common query language [7], [10]. The knowledge base is used to store information such as correspondence between local database schemas and conflict resolution for any mismatched

- *Z. Peng, X. Li, and J. Liu are with the State Key Laboratory of Software Engineering, Computer School, Wuhan University, LuoJia Mountain, Wuhan, Hubei, China, 430072. E-mail: {peng, lixuhui}@whu.edu.cn, liujunqiang@msn.com.*
- *Q. Li is with the Department of Computer Science, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon Hong Kong, China. E-mail: itqli@cityu.edu.hk.*
- *L. Feng is with the Department of Computer Science, University of Twente, PO Box 217, 7500 AE, Enschede, Netherlands. E-mail: ling@ewi.utwente.nl.*

Manuscript received 24 Nov. 2004; revised 29 Mar. 2005; accepted 7 Apr. 2005; published online 19 July 2005.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDESI-0488-1104.

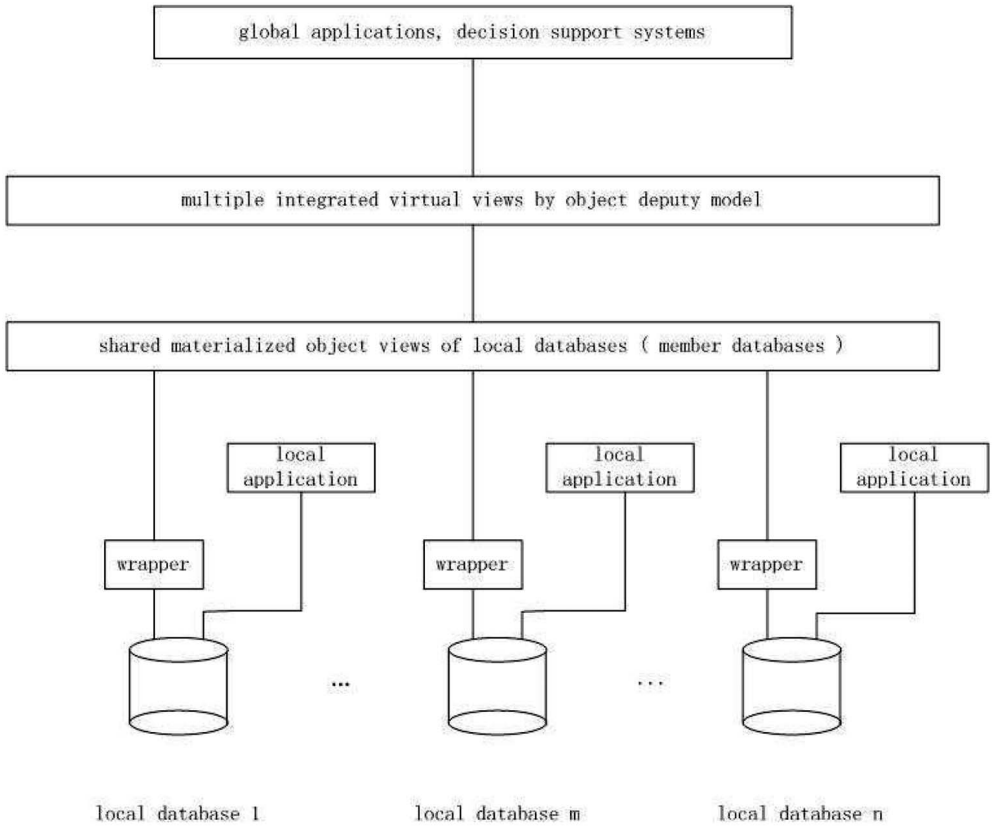

Fig. 1. FOODMAW—Framework of object-oriented data mining and warehousing.

structures of local database schemas. These components are common in many integration projects [22], [7], [3], [21], [19]. However, the focus of our project is on the following two components:

- *Shared materialized view set (member databases):* These are derived databases through what we call "mirroring process" [17], the purpose of which is to convert the local heterogeneous databases into a set of homogeneous databases which can be efficiently managed by a single robust database management system.
- *Multiple integrated (virtual) views:* Based on the shared materialized view set, integrated (virtual) views are defined for efficient support of different global and decision support applications on a data warehouse.

Essentially, materialized views are used for data warehouses and various methods for relational databases have been developed. In this paper, we will first discuss a method to realize materialized views for object-oriented databases (OODB) using deputy objects [24]. Since materialized views require the generation of duplicate data, we will address the issue of duplication handling by utilizing inheritance property of object-oriented databases.

While it is arguable whether the object-oriented (OO) approach is more practical (at least not proven by the current market yet) in handling traditional data warehouse applications which involve tera or petabytes of data, we do see emerging applications whose more complex data and (new) query types can benefit from adopting the OO paradigm [1], [29]. Particularly, in addition to being an interesting academic exercise, a data warehouse adopting the OO model can be more expressive and efficient than the relational or multidimensional models in accommodating "multifact" and "ad hoc" cube queries and such *semantic* cubes' derivation and mining [12], [13]. In this paper, we describe our work of developing such an OO data warehousing system by focusing on the data preparation phase and investigate some of the basic issues including object identification and fusion (for data objects coming from relational as well as nonrelational databases such as OODB or object-relational ones), as well as inconsistency and conflict resolution; other issues involved in maintaining and handling such an OO data warehousing system as well as efficient (cube) query processing, etc., are outside the scope of this special issue, hence, they are not included in this paper.

1.2 Paper Contributions and Organization

The main contributions of our paper include the following:

1. We provide an object framework for data warehousing including complex information not only from relational databases but also from object-oriented or object relational databases.
2. We give a methodology to resolve inconsistency and conflicts based on an object deputy model which is flexible enough for data fusion/integration compared with traditional relational or object-oriented data models.
3. We develop the framework by using Smalltalk as the data integration environment in which the object deputy model has been implemented and various database connecting tools are provided.

4. An application case is provided to illustrate the effectiveness of our framework for data integration and, furthermore, data analysis.

The remainder of the paper is organized as follows: Section 2 reviews the background of our research, including works on object-based data preparation in distributed databases, object-oriented data warehousing, and object deputy model. In Section 3 and Section 4, we study the issues of data fusion/integration in FOODMAW, particularly the issues of data extraction, conflict resolution, and duplication handling. Section 5 describes an experimental prototype of FOODMAW, in terms of its implementation in Smalltalk and associated facilities such as the database connecting tools and data analysis functionalities. Finally, we conclude the presentation of our work by summarizing the results and suggesting some ideas for future work in Section 6.

2 Background of Research

In this section, we review some works that are closely relevant to this study.

2.1 Object-Based Data Preparation

A lot of research efforts have been made to integrate heterogenous information sources in a distributed environment. The object-oriented approach is thought of as a good solution to the problem of the heterogeneous system interoperability because the object-oriented model is semantically rich and can define complex mapping even for information sources without database schema [5], [6], [9], [18], [19], where methods can be used to resolve various syntactic and semantic conflicts. However, the traditional object-oriented data model has two serious problems. First, it can only provide subclass constructor and support inheritance from superclass to subclass. Data integration needs not only specialization but also aggregation and generalization. Aggregation can be used to integrate component objects distributed in different databases [11]. The attributes and methods of the component objects are inherited by the complex object in the global schema. The generalization can be used to integrate specific objects into general ones. Thus, the object-oriented model should be extended with superclass constructor and can define inheritance from a subclass to a superclass and from the component objects to the complex object. Second, the view mechanism is very difficult to be implemented. In order to integrate databases, their schemas should be dynamically defined and modified. The view mechanism plays an important role in restructuring the schema resulting from the merging of component schemas. Although many view mechanisms [2], [4], [14], [18], [26] were published, to our knowledge, almost no commercial object-oriented databases provide true view supports. We know that the flexibility of relational databases is due to their data independence that enables data to be divided and combined very easily. Similarly, a flexible object-oriented database should also allow objects to be restructured. Without this feature, view mechanisms are difficult to incorporate into object-oriented databases.

2.2 Object-Oriented Data Warehousing

Data warehouse systems equip users with effective decision support tools by integrating enterprise-wide corporate data into a single repository from which business end-users can run reports and perform ad hoc data analysis [8]. In recent years, there has been considerable interest in tapping object systems for handling multidimensionality in OnLine Analytical Processing (OLAP) data. While relational OLAP (ROLAP) and multidimensional OLAP (MOLAP) systems are used successfully in data warehouses and can be tailored for specific applications, each system has its inherent pros and cons [30]. Current forage into object systems has partly been motivated by modeling issues (ROLAP) and scalability, sparse data (MOLAP) [31], and partly by the natural modeling ability of object systems to handle complex structures and relationships present in real-world applications. In [13], object cubes are proposed by utilizing semantic links between concept hierarchies in object cubes, so as to study generalization-based data mining in object-oriented databases. Abello et al. [1] also motivate object-oriented treatment of complex dimensions due to issues in aggregation levels, reference fan-outs, and interdimensional relationships. Trujillo et al. [29] utilize relationship semantics and constraints to conceptually model data warehouses in object and object-relational systems. Extensions to UML are used to model structural and dynamic (user queries, OLAP operations) levels. A logical federated model is advocated by [23], letting object systems handle complex relationships, while the OLAP system handles aggregate queries. They also provide an integrated OLAP query mechanism to access and return combinations of object and OLAP data, keeping the physical implementation separate. In the context of an object-relational data warehouse (ORDW) environment, [12] proposes an iterative logical and physical ORDW design mechanism. They also optimize the view/cube selection process by using horizontal partitioning techniques based on query and structural semantics.

As in the traditional data warehouses, the mechanism of materialized views is also applicable to object-oriented data warehouses. In [15], [32], the authors point out that the hybrid integrated views, i.e., the combination of fully materialized and virtual views, are beneficial, and [15] presents a framework for data integration using the materialized and virtual view approaches, while [32] provides an algorithm for selecting (sub)views to be materialized. In this paper, we will advocate an object deputy approach, which can actually facilitate such a hybrid approach. The basic idea is to 1) select the objects of interest from information sources, 2) create their materialized deputy objects to avoid communication delays, and 3) form an integrated and application-specific view by properly selecting a combination of computed and materialized deputy objects, considering the trade-off problems between view maintenance cost and computation overhead.

2.3 Object Deputy Mechanism

The concept of deputy objects was at first introduced by the authors for the unified realization of object views, roles, and migration [24]. In order to illustrate that it is also useful for database integration, we will review its definition.

The object-oriented data model represents real-world entities in term of `objects`. Objects are identified by system-defined identifiers which are independent of objects' states. An object has attributes which represent properties of a corresponding real-world entity. The state of an object is represented by its attribute values, which are read and written by basic methods. In addition, there are

general methods that represent the behavior of objects. Objects having the same attributes and methods are clustered into `classes`, which make it possible to avoid specification and storage of redundant information. A formal definition of objects and classes is given as follows:

Definition 1. *Each object has an identifier, some attributes, and methods. The schema of objects with the same attributes and methods is defined by a class which consists of a name, an extent, and a type. The extent of a class is a set of objects belonging to it, called its instances. The type of a class is definitions of its attributes and methods. A class named as C is represented as*

$$C = \langle \{o\}, \{T_a : a\}, \{m : \{T_p : p\}\}\rangle.$$

- *$\{o\}$ is the extent of C, where o is one of the instances of C.*
- *$\{T_a : a\}$ is the set of attribute definitions of C, where a and T_a represent the name and type of an attribute, respectively. The value of attribute a of object o is expressed by $o.a$. For each attribute $T_a : a$, there are two basic methods: $read(o, a)$ for reading $o.a$ and $write(o, a, v)$ for writing $o.a$ with the new value v, expressed as follows:*

  $$read(o, a) \Rightarrow \uparrow o.a, \quad write(o, a, v) \Rightarrow o.a := v.$$

  Here, $\Rightarrow$, $\uparrow$, and $:=$ stand for operation invoking, result returning, and assignment, respectively.
- *$\{m : \{T_p : p\}\}$ is the set of method definitions of C, where m and $\{T_p : p\}$ are method name and a set of parameters and p and T_p represent parameter name and type, respectively. Applying method m to object o with parameters $\{p\}$ is expressed as follows:*

  $$apply(o, m, \{p\}).$$

Deputy objects are defined as an extension and customization of objects. An object can have many deputy objects that are used to customize the object for different applications or represent its many faceted nature. The schemas of deputy objects are defined by deputy classes that are derived by creating deputy objects as their instances, generating switching operations for inheritance of attributes and methods and adding definitions for their additional attributes and methods. A formal definition of deputy objects and deputy classes is given as follows:

Definition 2. *A deputy object is generated from object(s) or other deputy object(s). The latter is called source object(s) of the former. A deputy object can inherit some attributes/methods from its source object(s). The schema of deputy objects with the same properties is defined by a deputy class, which includes a name, extent, and type. Deputy classes are derived from classes of source objects, called source classes. In general, let $C^s = \langle \{o^s\}, \{T_{a^s} : a^s\}, \{m^s : \{T_{p^s} : p^s\}\}\rangle$ be a source class. Its deputy class C^d is defined as*

$$\begin{aligned} C^d = &\langle \{o^d \mid (o^d \rightarrow o^s) \vee (o^d \rightarrow \ldots \times o^s \times \ldots) \vee (o^d \rightarrow \{o^s\}),\\ &sp(o^s) \ \vee \ cp(\ldots \times o^s \times \ldots) \ \vee \ gp(\{o^s\}) == true\},\\ &\{T_{a^d} : a^d\} \cup \{T_{a^d_+} : a^d_+\}, \{m^d : \{T_{p^d} : p^d\}\} \cup\\ &\{m^d_+ : \{T_{p^d_+} : p^d_+\}\}\rangle, \end{aligned}$$

where

1. $$\{o^d \mid (o^d \rightarrow o^s) \vee (o^d \rightarrow \ldots \times o^s \times \ldots) \vee (o^d \rightarrow \{o^s\}),\ sp(o^s) \ \vee \ cp(\ldots \times o^s \times \ldots) \ \vee gp(\{o^s\}) == true\}$$

   is the extent of C^d, where $(o^d \rightarrow o^s) \vee (o^d \rightarrow \ldots \times o^s \times \ldots) \vee (o^d \rightarrow \{o^s\})$ denotes that o^d is the deputy object of a certain source object o^s, or some combined source objects represented using $\ldots \times o^s \times \ldots$, or a set of source objects represented using $\{o^s\}$; and, sp, cp, and gp represent selection, combination, and grouping predicate, respectively.
2. *$\{T_{a^d} : a^d\} \cup \{T_{a^d_+} : a^d_+\}$ is the set of attribute definitions of C^d, where*
   - *$\{T_{a^d} : a^d\}$ is the set of the attributes inherited from $\{T_{a^s} : a^s\}$ of C^s, of which switching operations are defined as*

     $$read(o^d, a^d) \Rightarrow\uparrow f_{T_{a^s} \mapsto T_{a^d}}(read(o^s, a^s)),$$
     $$write(o^d, a^d, v^d) \Rightarrow write(o^s, a^s, f_{T_{a^d} \mapsto T_{a^s}}(v^d)).$$

     Here, function $f_{T \mapsto T'}$ converts the value of one type T to the value of another type T'.
   - *$\{T_{a^d_+} : a^d_+\}$ is the set of the additional attributes of C^d, of which basic methods are defined as*

     $$read(o^d, a^d_+) \Rightarrow\uparrow o^d.a^d_+,$$
     $$write(o^d, a^d_+, v^d_+) \Rightarrow o^d.a^d_+ := v^d_+.$$

3. *$\{m^d : \{T_{p^d} : p^d\}\} \cup \{m^d_+ : \{T_{p^d_+} : p^d_+\}\}$ is the set of method definitions of C^d, where*
   - *$\{m^d : \{T_{p^d} : p^d\}\}$ is the set of the methods inherited from $\{m^s : \{T_{p^s} : p^s\}\}$ of C^s, which are applied through switching operations as*

     $$apply(o^d, m^d, \{p^d\}) \Rightarrow\uparrow$$
     $$apply(o^s, m^s, \{f_{T_{p^d} \mapsto T_{p^s}}(p^d)\}).$$

   - *$\{m^d_+ : \{T_{p^d_+} : p^d_+\}\}$ is the set of the additional methods of C^d, which are applied as*

     $$apply(o^d, m^d_+, \{p^d_+\}).$$

According to the above definition, deputy objects have persistent identifiers but their attribute values inherited from source objects are still computed through switching operations that need to communicate with the underlying information sources. In order to improve performance, queries are required to be evaluated locally since information sources may be remote or unavailable for some time. For this reason, we extend deputy mechanisms to allow deputy objects to materialize their inherited attribute values. The definitions of basic methods for the inherited attribute of which value is materialized are changed as follows:

$$read(o^d, a^d) \Rightarrow\uparrow o^d.a^d,$$
$$write(o^d, a^d, v^d) \Rightarrow o^d.a^d := v^d \wedge$$
$$write(o^s, a^s, f_{T_{a^d} \mapsto T_{a^s}}(v^d)),$$
$$update(o^d, a^d) \Rightarrow o^d.a^d := f_{T_{a^s} \mapsto T_{a^d}}(read(o^s, a^s)).$$

That is, the inherited attribute values are precomputed and can be directly read from the deputy object. Thus, queries on the integrated views need not interfere with objects at remote sources. The update of the inherited attribute value of a deputy object needs to be reflected in its source object(s). Therefore, the writing method is first to update the precomputed value and then propagate the change into the original one through the switching operation. On the other hand, the update of the original attribute value requires recomputing the inherited attribute value of the deputy object. This operation is realized by introducing another basic method for each inherited attribute. The basic update method updates the inherited attribute value according to its dependence relationship defined by the switching operation. It is triggered when the original attribute value is updated.

3 Data Extraction in FOODMAW

In FOODMAW, we first build object-oriented interfaces (viz., wrappers) on top of nonobject-oriented information sources (see Fig. 1). Such an interface consists of a set of classes, each of which defines the properties and messages for a set of objects. Each definitional property has a name followed by its domain. The domain of a definitional property can be a basic type (for example, integer, Boolean, string, etc.) or a class. The basic operations for messages are implemented in terms of primitives provided by the local information sources. That means each local information source is converted into an object-oriented view. It mainly solves conflicts due to syntactical heterogeneity.

In object-oriented databases, objects are classified into classes, which are organized into a hierarchy. The upper classes are called superclasses of the lower ones that are conversely called subclasses of the upper ones. The super class includes instances of its subclasses and a subclass inherits attributes and methods of its superclass. Suppose there are two classes, Rectangle and Right_Rectangle, where the former is a superclass of the latter. As shown in Fig. 2a, if both of them are extracted into data warehouses without considering their inheritance relationship, right rectangles will be duplicated since right rectangles also indirectly belong to the class Rectangle.

In order to avoid the duplication of instances of the subclass, we extract the superclass by first creating its deputy class that only includes deputy objects of its direct instances. The deputy class is then merged with the deputy class of the subclass by the union operation to derive a deputy class that can be used at data warehouses as if it would be the superclass. Since the deputy class derived by the union operation is not materialized, right rectangles will not be duplicated as shown in Fig. 2b. Let $S = \{C_1, \ldots, C_m\}$ be the classes selected from an object-oriented database. We assume that T is a set of the classes that have been extracted, D_i is a deputy class of C_i and D'_i is a deputy class of C_i with the extent only including deputy objects of direct instances of C_i. In general, we can give the following algorithm that can derive deputy classes without unnecessary instance duplication from the selected classes according to their inheritance relationships:

While ($S \neq \emptyset$) do
{
  For each $C_i \in S$ that has not any subclass in S,
    if C_i has subclasses $\{C_{i_1}, \ldots, C_{i_n}\}$ in T,
  then
    {
      if (C_i has direct instances) then
          1) To create a materialized deputy class D'_i,
          2) To create a computed deputy class D_i
          as the union of $D_{i_1}, \ldots, D_{i_n}$ and D'_i

      else
          To create a computed deputy class D_i
          as the union of $D_{i_1}, \ldots, D_{i_n}$
    }
  else
      To create a materialized deputy class D_i
  To delete C_i from S and add C_i into T
}

In order to illustrate the above algorithm, we give an example as shown in Fig. 3. According to the algorithm, the class hierarchy of an OODB system is extracted into the data warehouse in the following way:

1. $S = \{C_1, C_2, C_3, C_4\}$ and $T = \emptyset$;
2. C_2 and C_4 are selected since they have no subclass in S;
3. The materialized deputy classes D_2 and D_4 are created since C_2 and C_4 have no subclass in T;
4. $S := S - \{C_2, C_4\}$ and $T := T + \{C_2, C_4\}$;
5. $S = \{C_1, C_3\}$ and $T = \{C_2, C_4\}$;
6. C_3 is selected since it has no subclass in S;
7. A materialized deputy class D'_3 is created since C_3 has a subclass C_4 in T and has a direct instance;
8. A computed deputy class D_3 is derived as the union of D_4 and D'_3;
9. $S := S - \{C_3\}$ and $T := T + \{C_3\}$;
10. $S = \{C_1\}$ and $T = \{C_2, C_3, C_4\}$;
11. C_1 is selected since it has no subclass in S;
12. A computed deputy class D_1 is created as the union of D_2 and D_3 since C_1 has subclasses C_2, C_3 in T and has no direct instance;
13. $S := S - \{C_1\}$ and $T := T + \{C_1\}$;
14. $S = \emptyset$ and $T = \{C_1, C_2, C_3, C_4\}$;
15. Stop since $S = \emptyset$.

4 Data Integration in FOODMAW

After the objects of interest are extracted from multiple heterogeneous information sources by creating their materialized deputy objects, their integration will be realized by an object deputy algebra [25], which consists of six algebraic operations: Select, Project, Extend, Union, Join, and Grouping.

In order to offer an integrated view, the schema mismatch of objects must be overcome at first. Since the inheritance by switching operations allows the names and types of attributes between objects and their deputy objects to be different, the synonym, homonym, and type mismatch problems can be solved by defining appropriate switching operations. After the schema mismatch of objects has been solved, object integration can be realized by applying the object deputy algebra, of which the selection operation is mainly used to

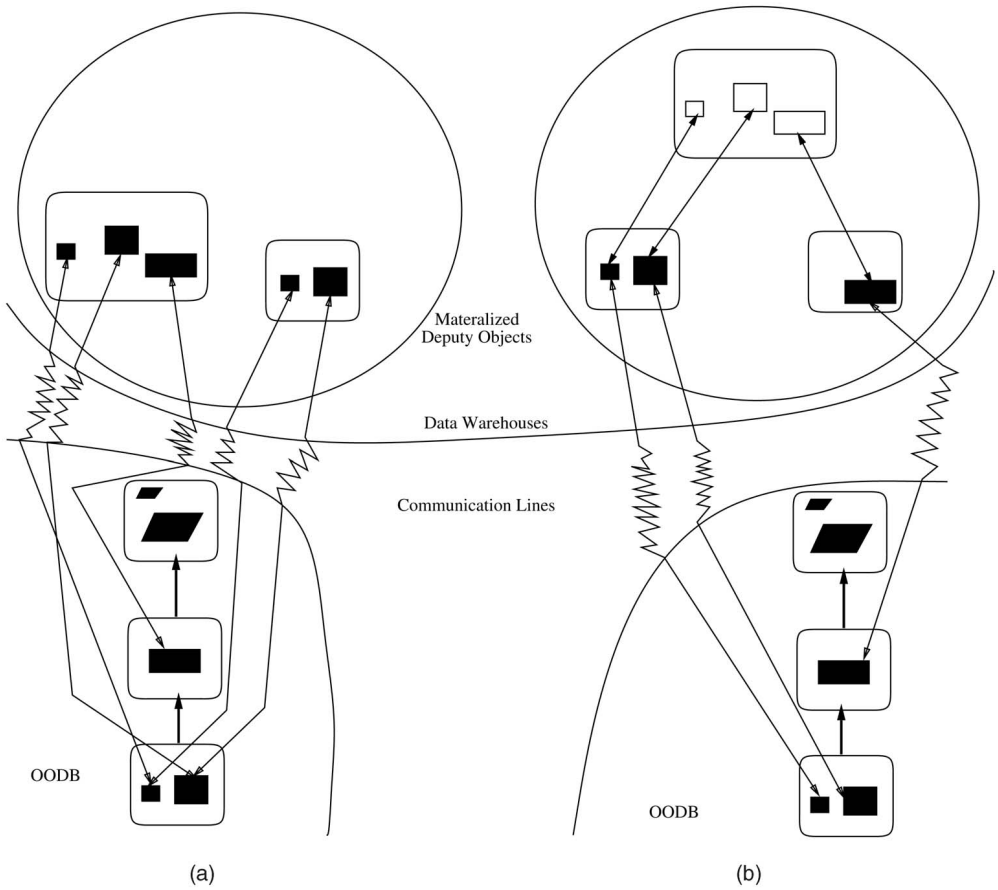

Fig. 2. Comparison of two object extraction approaches. (a) An approach not considering inheritance. (b) An approach considering inheritance.

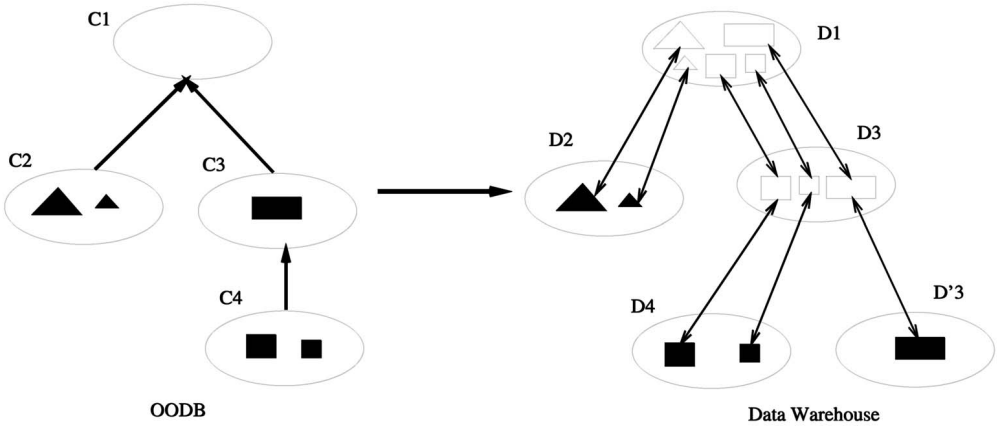

Fig. 3. An example for illustrating class extraction.

select objects of interest for integration. The same kind of real-world entities may be defined in multiple sources with attributes and methods more or less than what need be included in the integrated view. The project operation can be used to hide unnecessary ones, and the extend operation to add necessary ones (with default values). The objects with the same attributes and methods from different sources can be integrated into a single class by the union operation. If attributes and methods of a real-world entity are distributed in different sources, the conceptually related components can be combined by the join operation. In addition, the same replicated objects defined in different sources can be grouped by the grouping operation.

To illustrate data integration by object deputy model, let us consider the following application scenario. XYZ is a global electronic corporate with divisions located in different parts of the world. Each of the divisions makes certain kinds of products, like home appliances, medical systems, lighting, etc. Assume each division has its own human resource management department, R and D department, sales department, etc., and maintains information of its employees, products, etc., in its local database. The local database in the headquarters maintains the performance appraisal results for all its employees within the corporation. Fig. 4 shows a sample set of local database examples, described in the tabular format for simplicity.

name	*age*	*dept*	*salary* (S$)	*CPF* (S$)
Jack Louis	32	Personnel	2K	100
Martin Smith	36	R&D	4K	200
Alice Leong	40	Sales	5K	250
..	..	..	..	..

(a)

name	*age*	*dept*	*payment* (US $)
Kavin Ho	27	R&D	1K
Steven Lous	28	R&D	2K
Jone Klein	30	Personnel	1K
..	..	..	..

(b)

name	*category*	*model*
TV	home-appliance	TH42PX25
TV	home-appliance	TH42PHD7
DVD	home-appliance	DVD9002
..	..	..

(c)

name	*category*	*model*
ultrasound	medical-systems	SSD-210DX
X-ray	medical-systems	3630
medi-Scan	medical-systems	8400
..	..	..

(d)

name	*region*	*evaluation*
Martin Smith	region-A	A
Kavin Ho	region-B	A
Stin Louwes	region-B	B+
Alice Leong	region-A	B-
..	..	..

(e)

Fig. 4. A sample set of local database examples. (a) Class $Employee_A$ at region A. (b) Class $Employee_B$ at region B. (c) Class $Product_A$ at region A. (d) Class $Product_B$ at region B. (e) Class $EmpEvaluation$ at the headquarters.

4.1 Resolving Conflicts by Object Deputy Model

Referring to Fig. 1, an important step is to integrate semantically heterogeneous schemas in an integrated view (viz., a member of the shared materialized view set of Fig. 1). There may be various conflicts during integration, which can be resolved through homogenizing, specialization, generalization, and aggregation.

4.1.1 Homogenizing Objects

When integrating several heterogeneous information sources, there are problems of having different names for equivalent entities or attributes, or having the same name for different entities or attributes. In addition, different expressions, units, or levels of precision may be used to denote similar information. These conflicts can be resolved by defining deputy objects with switching operations which can rename attributes/methods and change their expressions, units, or levels of precision. For example, as shown in Fig. 4, an employee's $salary$ in one database, $Employee_A$, is expressed as $payment$ in another database, $Employee_B$. Suppose that they have different units, Singapore dollars (S$) for the $salary$ and US dollar (US) for the $payment$. If they need be homogenized into $payment$ with US dollar, as the unit, we can define the deputy object with the following switching operations:

$$Read(d, payment) \Rightarrow Read(o, salary) * su\_rate,$$
$$Write(d, payment, v) \Rightarrow Write(o, salary, v/su\_rate),$$

where su_rate represents the exchange rate from Singapore dollars to US dollars.

Conflicts involving "missing attributes" arise when the numbers of attributes are different in semantically equivalent entities across several information sources. For instance, the attribute CPF (Central Provident Fund) is only applicable to the employees positioned in Singapore, but not in the USA. There are two ways to resolve this type of conflict: One way is to hide the extra attributes from the entities which have more attributes than other entities. This can be realized by deriving a deputy class using the algebraic operation: Project. The **Project** operation is used to derive a deputy class which only inherits part of attributes and methods of a source class. Its formal definition is as follows:

Definition 3. *Let $C^s = \langle \{o^s\}, \{T_{a^s} : a^s\}, \{m^s : \{T_{p^s} : p^s\}\}\rangle$ be a source class, $\{T_{a^s_-} : a^s_-\}$ and $\{m^s_- : \{T_{p^s_-} : p^s_-\}\}$ be subsets of attributes and methods of C^s which are allowed to be inherited. A deputy class derived by the **Project** operation is represented as $C^d = \mathbf{Project}(C^s, \{T_{a^s_-} : a^s_-\}, \{m^s_- : \{T_{p^s_-} : p^s_-\}\})$.*

1. *The extent of C^d is the set of deputy objects of instances of C^s, which is expressed as*

$$\{o^d | o^d \rightarrow o^s\}.$$

2. *The set of attributes of C^d is defined as $\{T_{a^d_-} : a^d_-\}$, which are inherited from the attributes $\{T_{a^s_-} : a^s_-\}$ of C^s. The switching operations for inheriting $T_{a^s_-} : a^s_-$ in the form of $T_{a^d_-} : a^d_-$ are realized in the following way:*

$$read(o^d, a^d_-) \Rightarrow\uparrow f_{T_{a^s_-} \mapsto T_{a^d_-}}(read(o^s, a^s_-)),$$
$$write(o^d, a^d_-, v^d_-) \Rightarrow write(o^s, a^s_-, f_{T_{a^d_-} \mapsto T_{a^s_-}}(v^d_-)).$$

3. *The set of methods of C^d is defined as*

$$\{m^d_- : \{T_{p^d_-} : p^d_-\}\},$$

which are inherited from the methods $\{m^s_- : \{T_{p^s_-} : p^s_-\}\}$ of C^s. The switching operation for inheriting $m^s_- : \{T_{p^s_-} : p^s_-\}$ in the form of $m^d_- : \{T_{p^d_-} : p^d_-\}$ is realized in the following way:

$$apply(o^d, m^d_-, \{p^d_-\}) \Rightarrow\uparrow$$
$$apply(o^s, m^s_-, \{f_{T_{p^d_-} \mapsto T_{p^s_-}}(p^d_-)\}).$$

The other way is to add the extra attributes to the entities which have less attributes than other entities. This can be realized by deriving a deputy class using the algebraic operation: Extend. The **Extend** operation is used to derive a deputy class of which instances are extended with additional attributes and methods that cannot be derived from a source class. Its formal definition is as follows:

Definition 4. *Let* $C^s = \langle\{o^s\}, \{T_{a^s} : a^s\}, \{m^s : \{T_{p^s} : p^s\}\}\rangle$ *be a source class,* $\{T_{a^d_+} : a^d_+\}$ *and* $\{m^d_+ : \{T_{p^d_+} : p^d_+\}\}$ *be sets of additional attributes and methods. A deputy class derived by the* ***Extend*** *operation is represented as* $C^d = \mathbf{Extend}\ (C^s, \{T_{a^d_+} : a^d_+\}, \{m^d_+ : \{T_{p^d_+} : p^d_+\}\})$.

1. *The extent of* C^d *is the set of deputy objects of instances of* C^s*, which is expressed as*

$$\{o^d | o^d \rightarrow o^s\}.$$

2. *The set of attributes of* C^d *is defined as the union of attributes* $\{T_{a^d} : a^d\}$ *inherited from the attributes* $\{T_{a^s} : a^s\}$ *of* C^s *and its additional attributes* $\{T_{a^d_+} : a^d_+\}$*, expressed as* $\{T_{a^d} : a^d\} \cup \{T_{a^d_+} : a^d_+\}$.

   a. *The switching operations for inheriting* $T_{a^s} : a^s$ *in the form of* $T_{a^d} : a^d$ *are realized in the following way:*

$$read(o^d, a^d) \Rightarrow\uparrow f_{T_{a^s} \mapsto T_{a^d}}(read(o^s, a^s)),$$
$$write(o^d, a^d, v^d) \Rightarrow write(o^s, a^s, f_{T_{a^d} \mapsto T_{a^s}}(v^d)).$$

   b. *For each additional attribute* $T_{a^d_+} : a^d_+$*, the following two basic methods are realized, which are operated independently of the source object:*

$$read(o^d, a^d_+) \Rightarrow\uparrow o^d.a^d_+,$$
$$write(o^d, a^d_+, v^d_+) \Rightarrow o^d.a^d_+ := v^d_+.$$

3. *The set of methods of* C^d *is defined as the union of methods* $\{m^d : \{T_{p^d} : p^d\}\}$ *inherited from the methods* $\{m^s : \{T_{p^s} : p^s\}\}$ *of* C^s *and its additional methods* $\{m^d_+ : \{T_{p^d_+} : p^d_+\}\}$*, expressed as*

$$\{m^d : \{T_{p^d} : p^d\}\} \cup \{m^d_+ : \{T_{p^d_+} : p^d_+\}\}.$$

   a. *The switching operation for inheriting* $m^s : \{T_{p^s} : p^s\}$ *in the form of* $m^d : \{T_{p^d} : p^d\}$ *is realized in the following way:*

$$apply(o^d, m^d, \{p^d\}) \Rightarrow\uparrow$$
$$apply(o^s, m^s, f_{T_{p^d} \mapsto T_{p^s}}\{p^d\}).$$

   b. *For each additional method* $m^d_+ : \{T_{p^d_+} : p^d_+\}$*, the following switching operation is realized, which is applied independently of the source object:*

$$apply(o^d, m^d_+, \{p^d_+\}).$$

4.1.2 Specialization

Each application may have its own integrated views which select their needed objects from local information sources. The specialization abstract mechanism can be used to achieve such an objective. For example, suppose an application wants to have an overview of all the employees working in the R & D department. To bring this information together, we can define a deputy class, which is a specialization of the local ones, so that the deputy class only contains deputy objects of employees in the R & D department.

The **Select** operation is used to derive a deputy class of which instances are the deputy objects of the instances of a source class selected according to a selection predicate. Its formal definition is as follows:

Definition 5. *Let* $C^s = \langle\{o^s\}, \{T_{a^s} : a^s\}, \{m^s : \{T_{p^s} : p^s\}\}\rangle$ *be a source class. A deputy class derived by the* ***Select*** *operation is represented as* $C^d = \mathbf{Select}(C^s, sp)$*, where* sp *is a selection predicate:*

1. *The extent of* C^d *is the set of deputy objects of instances of* C^s *which satisfy the selection predicate* sp*, expressed as*

$$\{o^d | o^d \rightarrow o^s, sp(o^s) == true\}.$$

2. *The set of attributes of* C^d *is defined as* $\{T_{a^d} : a^d\}$*, which are inherited from the attributes* $\{T_{a^s} : a^s\}$ *of* C^s*. The switching operations for inheriting* $T_{a^s} : a^s$ *in the form of* $T_{a^d} : a^d$ *are realized in the following way:*

$$read(o^d, a^d) \Rightarrow\uparrow f_{T_{a^s} \mapsto T_{a^d}}(read(o^s, a^s)),$$
$$write(o^d, a^d, v^d) \Rightarrow write(o^s, a^s, f_{T_{a^d} \mapsto T_{a^s}}(v^d)).$$

3. *The set of methods of* C^d *is defined as*

$$\{m^d : \{T_{p^d} : p^d\}\},$$

   which are inherited from the methods $\{m^s : \{T_{p^s} : p^s\}\}$ *of* C^s*. The switching operation for inheriting* $m^s : \{T_{p^s} : p^s\}$ *in the form of* $m^d : \{T_{p^d} : p^d\}$ *is realized in the following way:*

$$apply(o^d, m^d, \{p^d\}) \Rightarrow\uparrow$$
$$apply(o^s, m^s, \{f_{T_{p^d} \mapsto T_{p^s}}(p^d)\}).$$

4.1.3 Generalization

Data with the same attributes and methods may be distributed in different databases. For instance, based on the places where products are made, product information is maintained within the respective private local database. Suppose the headquarter wants to have a regular check application over all the company products made by different divisions, the scattered data items then need to be included in a single class so as to offer an integrated view. We can define their deputy objects by a single deputy class which can be automatically derived by using the **Union** operation on the existing data classes. The deputy class can be treated as the union of the existing data classes.

The **Union** operation is used to derive a deputy class of which the extent consists of deputy objects of instances of more than one source class. Its formal definition is as follows:

Definition 6. *Let*

$$C_1^s = \langle \{o_1^s\}, \{T_{a_1^s} : a_1^s\}, \{m_1^s : \{T_{p_1^s} : p_1^s\}\}\rangle, \ldots,$$
$$C_m^s = \langle \{o_m^s\}, \{T_{a_m^s} : a_m^s\}, \{m_m^s : \{T_{p_m^s} : p_m^s\}\}\rangle$$

be source classes $\{T_{a^s} : a^s\} = \{T_{a_1^s} : a_1^s\} \cap \ldots \cap \{T_{a_m^s} : a_m^s\}$ *and*

$$\{m^s : \{T_{p^s} : p^s\}\} = \{m_1^s : \{T_{p_1^s} : p_1^s\}\} \cap \ldots \cap \{m_m^s : \{T_{p_m^s} : p_m^s\}\}$$

be common sets of attributes and methods of $C_1^s, \ldots, C_m^s$. *A deputy class derived by the* ***Union*** *operation is represented as* $C^d = \mathbf{Union}(C_1^s, \ldots, C_m^s)$:

1. *The extent of* C^d *is the union of sets of deputy objects of instances of* $C_1^s, \ldots, C_m^s$, *which is expressed as*

$$\{o_1^d | o_1^d \rightarrow o_1^s\} \cup \ldots \cup \{o_m^d | o_m^d \rightarrow o_m^s\}.$$

2. *The set of attributes of* C^d *is defined as* $\{T_{a^d} : a^d\}$, *which are inherited from the common attributes* $\{T_{a^s} : a^s\}$ *of* $C_1^s, \ldots, C_m^s$. *The switching operations for inheriting* $T_{a^s} : a^s$ *in the form of* $T_{a^d} : a^d$ *are realized in the following way:*

$$read(o_1^d, a^d) \Rightarrow\uparrow f_{T_{a^s} \mapsto T_{a^d}}(read(o_1^s, a^s)),$$
$$write(o_1^d, a^d, v^d) \Rightarrow write(o_1^s, a^s, f_{T_{a^d} \mapsto T_{a^s}}(v^d)),$$
$$\ldots$$
$$read(o_m^d, a^d) \Rightarrow\uparrow f_{T_{a^s} \mapsto T_{a^d}}(read(o_m^s, a^s)),$$
$$write(o_m^d, a^d, v^d) \Rightarrow write(o_m^s, a^s, f_{T_{a^d} \mapsto T_{a^s}}(v^d)).$$

3. *The set of methods of* C^d *is defined as*

$$\{m^d : \{T_{p^d} : p^d\}\},$$

which are inherited from the common methods $\{m^s : \{T_{p^s} : p^s\}\}$ *of* $C_1^s, \ldots, C_m^s$. *The switching operations for inheriting* $m^s : \{T_{p^s} : p^s\}$ *in the form of* $m^d : \{T_{p^d} : p^d\}$ *are realized in the following way:*

$$apply(o_1^d, m^d, \{p^d\}) \Rightarrow\uparrow$$
$$apply(o_1^s, m^s, \{f_{T_{p^d} \mapsto T_{p^s}}(p^d)\}),$$
$$\ldots$$
$$apply(o_m^d, m^d, \{p^d\}) \Rightarrow\uparrow$$
$$apply(o_m^s, m^s, \{f_{T_{p^d} \mapsto T_{p^s}}(p^d)\}).$$

4.1.4 Aggregation

Attributes of a complex entity of real world may be distributed in different databases. For example, a correlation analysis between the performance evaluation result regarding one employee and his/her other personal information will call for the data scattered in two separate classes, i.e., *employee* and *empEvaluation*. These two conceptually related components need be combined. A complex deputy object, namely, a deputy object having several source component objects, is useful for such a purpose. The algebraic operation **Join** provided by a object deputy model can be used to derive automatically a complex deputy class.

The **Join** operation is used to derive a deputy class of which instances are deputy objects for aggregating instances of source classes according to a combination predicate. Its formal definition is as follows:

Definition 7. *Let*

$$C_1^s = \langle \{o_1^s\}, \{T_{a_1^s} : a_1^s\}, \{m_1^s : \{T_{p_1^s} : p_1^s\}\}\rangle, \ldots,$$
$$C_n^s = \langle \{o_n^s\}, \{T_{a_n^s} : a_n^s\}, \{m_n^s : \{T_{p_n^s} : p_n^s\}\}\rangle$$

be source classes. A deputy class derived by the ***Join*** *operation is represented as* $C^d = \mathbf{Join}(C_1^s, \ldots, C_n^s, cp)$, *where cp is a combination predicate:*

1. *The extent of* C^d *is the set of deputy objects of aggregations of instances of* $C_1^s, \ldots, C_n^s$, *satisfying the combination predicate cp, which is expressed as*

$$\{o^d | o^d \rightarrow o_1^s \times \ldots \times o_n^s, cp(o_1^s \times \ldots \times o_n^s) == true\}.$$

2. *The set of attributes of* C^d *is defined as the union of attribute sets* $\{T_{a_1^d} : a_1^d\}, \ldots, \{T_{a_n^d} : a_n^d\}$, *respectively, inherited from* $C_1^s, \ldots, C_n^s$, *expressed as* $\{T_{a_1^d} : a_1^d\} \cup \ldots \cup \{T_{a_n^d} : a_n^d\}$. *The switching operations for attributes* $\{T_{a_1^d} : a_1^d\}, \ldots, \{T_{a_n^d} : a_n^d\}$, *respectively, inherited from the attributes* $\{T_{a_1^s} : a_1^s\}$ *of* $C_1^s, \ldots, \{T_{a_n^s} : a_n^s\}$ *of* C_n^s *are realized in the following way:*

$$read(o^d, a_1^d) \Rightarrow\uparrow f_{T_{a_1^s} \mapsto T_{a_1^d}}(read(o_1^s, a_1^s)),$$
$$write(o^d, a_1^d, v_1^d) \Rightarrow write(o_1^s, a_1^s, f_{T_{a_1^d} \mapsto T_{a_1^s}}(v_1^d)),$$
$$\ldots$$
$$read(o^d, a_n^d) \Rightarrow\uparrow f_{T_{a_n^s} \mapsto T_{a_n^d}}(read(o_n^s, a_n^s)),$$
$$write(o^d, a_n^d, v_n^d) \Rightarrow write(o_n^s, a_n^s, f_{T_{a_n^d} \mapsto T_{a_n^s}}(v_n^d)).$$

3. *The set of methods of* C^d *is defined as the union of method sets* $\{m_1^d : \{T_{p_1^d} : p_1^d\}\}, \ldots, \{m_n^d : \{T_{p_n^d} : p_n^d\}\}$, *respectively, inherited from* $C_1^s, ..., C_n^s$, *expressed as*

$$\{m_1^d : \{T_{p_1^d} : p_1^d\}\} \cup \ldots \cup \{m_n^d : \{T_{p_n^d} : p_n^d\}\}.$$

The switching operations for methods

$$\{m_1^d : \{T_{p_1^d} : p_1^d\}\}, \ldots, \{m_n^d : \{T_{p_n^d} : p_n^d\}\},$$

respectively, inherited from the methods $\{m_1^s : \{T_{p_1^s} : p_1^s\}\}$ *of* $C_1^s, \ldots, \{m_n^s : \{T_{p_n^s} : p_n^s\}\}$ *of* C_n^s *are realized in the following way:*

$$apply(o^d, m_1^d, \{p_1^d\}) \Rightarrow\uparrow$$
$$apply(o_1^s, m_1^s, \{f_{T_{p_1^d} \mapsto T_{p_1^s}}(p_1^d)\}),$$
$$\ldots$$
$$apply(o^d, m_n^d, \{p_n^d\}) \Rightarrow\uparrow$$
$$apply(o_n^s, m_n^s, \{f_{T_{p_n^d} \mapsto T_{p_n^s}}(p_n^d)\}).$$

4.2 Handling Duplication via Object Deputy Model

When integrating several databases, there may exist data duplication. Duplication means that the same entities or attributes may appear at different databases. In the integrated views, the duplications need to be eliminated. The integrated system is usually used both to access the data and to update the stored information. When an object is modified in the integrated view, all of the duplications at the local information sources should be updated at the same time. In addition, any modification on one of the duplications should be propagated into the others. Therefore, we need a mechanism to handle duplications in the integrated views.

The attribute duplication may arise when several objects are combined into a complex deputy object. In this case, all of the duplicated attributes are inherited as a single attribute by the complex deputy object. For example, an employee may be recorded in two different databases. One record has attributes including $name$ and age in Region A's database while the other has attributes including $name$ and $evaluation$ in the headquarters' database. When the two records are integrated, the $name$ attribute will be duplicated. The attributes of the employee are inherited by the deputy object in the following way:

$$
\begin{aligned}
&d \rightarrow r_1 \times r_2,\\
&read(d, name) \Rightarrow read(r_1, name) \vee read(r_2, name),\\
&write(d, name, v) \Rightarrow write(r_1, name, v) \wedge\\
&\qquad\qquad write(r_2, name, v),\\
&read(d, age) \Rightarrow read(r_1, age),\\
&write(d, age, v) \Rightarrow write(r_1, age, v),\\
&read(d, evaluation) \Rightarrow read(r_2, evaluation),\\
&write(d, evaluation, v) \Rightarrow write(r_2, evaluation, v).
\end{aligned}
$$

Here, r_1 and r_2 represent two records of the employee and d is a deputy object used to integrate r_1 and r_2. That means any one of the duplicated attributes can be read but all of them should be written with the new value at the same time.

The entity duplication may arise when a deputy class is defined as the union of several classes. For example, when some products are manufactured by more than one division, the entity duplication problem will exist across different local databases. The duplication can be handled by deriving a deputy class through the algebraic operation Grouping.

The **Grouping** operation is used to derive a deputy class of which instances are deputy objects for grouping instances of a source class according to a grouping predicate. That is, the duplicated objects are grouped by a single deputy object. Its formal definition is as follows:

Definition 8. *Let $C^s = \langle \{o^s\}, \{T_{a^s} : a^s\}, \{m^s : \{T_{p^s} : p^s\}\}\rangle$ be a source class. A deputy class derived by the **Grouping** operation is represented as $C^d = \mathbf{Grouping}\ (C^s, gp)$, where gp is a grouping predicate:*

1. *The extent of C^d is the set of deputy objects for grouping instances of C^s according to the grouping predicate gp, which is expressed as*

$$\{o^d | o^d \rightarrow \{o^s\}, gp(\{o^s\}) == true\}.$$

2. *The set of attributes of C^d is defined as $\{T_{a^d} : a^d\}$ which are inherited from the attributes $\{T_{a^s} : a^s\}$ of C^s. The switching operations for inheriting $T_{a^s} : a^s$ in the form of $T_{a^d} : a^d$ are realized in the following way:*

$$
\begin{aligned}
&read(o^d, a^d) \Rightarrow \uparrow f_{\{T_{a^s} \mapsto T_{a^d}\}}(\{read(o^s, a^s)\}),\\
&write(o^d, a^d, v^d) \Rightarrow \{write(o^s, a^s, f_{T_{a^d} \mapsto T_{a^s}}(v^d))\}.
\end{aligned}
$$

3. *The set of methods of C^d is defined as $\{m^d : \{T_{p^d} : p^d\}\}$ which are inherited from the methods $\{m^s : \{T_{p^s} : p^s\}\}$ of C^s. The switching operation for inheriting $m^s : \{T_{p^s} : p^s\}$ in the form of $m^d : \{T_{p^d} : p^d\}$ is realized in the following way:*

$$
\begin{aligned}
&apply(o^d, m^d, \{p^d\}) \Rightarrow\\
&\{\uparrow apply(o^s, m^s, \{f_{T_{p^d} \mapsto T_{p^s}}(p^d)\})\}.
\end{aligned}
$$

Suppose o_1, o_2, and o_3 with attributes a_1 and a_2 are the duplicated objects which are grouped by the deputy object d. d inherits attributes a_1 and a_2 of o_1, o_2, and o_3 in the following way:

$$
\begin{aligned}
&d \rightarrow \{o_1, o_2, o_3\},\\
&read(d, a_1) \Rightarrow read(o_1, a_1) \vee read(o_2, a_1) \vee read(o_3, a_1),\\
&write(d, a_1, v_1) \Rightarrow write(o_1, a_1, v_1) \wedge write(o_2, a_1, v_1) \wedge\\
&\qquad\qquad write(o_3, a_1, v_1),\\
&read(d, a_2) \Rightarrow read(o_1, a_2) \vee read(o_2, a_2) \vee read(o_3, a_2),\\
&write(d, a_2, v_2) \Rightarrow write(o_1, a_2, v_2) \wedge write(o_2, a_2, v_2) \wedge\\
&\qquad\qquad write(o_3, a_2, v_2).
\end{aligned}
$$

The above way can guarantee that any modification on the integrated view can be reflected in all of the duplicated attributes and entities. Because the integrated view can be used as the bridge among the autonomous information sources, any update on an object in an information source can be propagated onto another information source.

For example, if object o_1 is modified with a new name, the deputy object d will be notified with the update because there exists a pointer from o_1 to d. d can read the new name and then write the new name to o_2. Thus, o_2 is also updated with the new name.

In the similar way, to illustrate how to deal with updates occurring in the local information sources, consider that attribute a_1 of the object o_2 is modified with a new value v_1. The deputy object d will be notified with the update. Because the update is from the object o_2, the new value can be read from o_2. Using the writing operation, the new value can be written into all of the duplicated objects o_1, o_2, and o_3. Because the attribute a_1 of the object o_2 has been modified before the update broadcasting, it does not need to be updated once again. It can be detected by comparing the new value with the current attribute value of the object o_2. This method can avoid a lot of unnecessary modification.

5 Implementation of FOODMAW

In order to demonstrate the feasibilty of FOODMAW, we have implemented a prototype using Smalltalk which can integrate data from heterogeneous databases including Oracle, Sybase, PostgreSQL, and so on. To do this, first, a class hierarchy is defined to facilitate the implementation of the object deputy model in Smalltalk. Next, a database connecting tool is developed for PostgreSQL (an open

source object-relational database system), which is similar to the database connecting tools for Oracle and Sybase in Smalltalk. These tools can help connect various databases and map data into objects that are defined by classes in Smalltalk. In order to allow users to derive deputy classes based on object deputy algebra for data integration, we further design an object deputy language and implement its compiler in Smalltalk. Finally, an object deputy browser is developed for users to utilize various data modelling functionalities provided by the object deputy model. We have implemented an application case to show the effectiveness of our approach.

5.1 Class Hierarchy of Object Deputy Model

Smalltalk is an object-oriented programming environment which can support software reuse through "is-a" hierarchy. That is, common attributes and methods are defined by the superclass, which are inherited by all of its subclasses. In the object deputy model, classes and deputy classes have some common attributes and methods which will be defined by the class "*Deputy*." Classes and various deputy classes have some different properties which will be defined by the classes: "*BaseClass*," "*SelectionDeputyClass*," "*UnionDeputyClass*," "*JoinDeputyClass*," and "*GroupingDeputyClass*." These classes have some common attributes and methods inherited from the class "*Deputy*" and therefore are defined as its subclasses. Since they can be predefined, we call them primitive classes in Smalltalk. Any classes or deputy classes created by users are defined as subclasses of these primitive classes. So, they are called nonprimitive classes. In summary, all classes and deputy classes of the object deputy model in Smalltalk are organized as shown in Fig. 5.

5.2 Database Connecting Tools

A large amount of data is stored in commercial databases like Oracle, Sybase, or open-source databases like PostgreSQL and Mysql. To integrate data from these heterogeneous databases, our framework needs database connecting tools. Smalltalk can provide tools to connect Oracle database and Sybase database, but cannot support connection with open source databases like PostgreSQL. In order to connect PostgreSQL, we develop a database connecting tool in Smalltalk which has the same connecting functionalities as tools for Oracle and Sybase.

The database connecting tool allows users to choose tables in PostgreSQL database. When a table is chosen, a class will be created as a subclass of the primitive class "*BaseClass*" in Smalltalk. All tuples of that table are mapped to instances of that class, constituting the materialized object views of the relational table. The tool also helps to maintain the consistency of relational data and objects. When data in databases is updated, the change will be propagated onto Smalltalk, where the update will be propagated onto all of the related deputy objects and deputy classes.

5.3 Object Deputy Language Compiler

In order to allow users to define various deputy classes for data integration, we have designed an object deputy language in the style of Smalltalk language. The language compiler has been implemented for an automatic generation of deputy classes from statements defined by the language.

For instance, a deputy class for specialization can be defined by the object deputy language as follows:

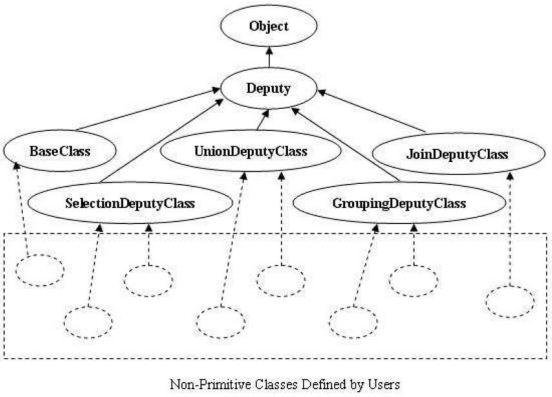

Fig. 5. Class hierarchy of object deputy model in Smalltalk.

$SelectionDeputyClass$ **define** $: \#A$
select $: '*' \,|\#([' \; A.a := f_{T_b \mapsto T_a}(B.b) \; ']^*$
$[' \; B.b := f_{T_a \mapsto T_b}(A.a) \; ']^* [\; 'A.a :=: B.b \; ']^* [\; 'B.b \; ']^*)$
from : B
where $: sp(\ldots, B.b, \ldots)$
extend $:' \; [T : attName]^* \; '$

The statement means that A will be created as the selection deputy class of the source class B with additional attributes $'[attName]^* '$. The selection predicate is $sp(\ldots, B.b, \ldots)$. The switching operations for attributes inherited from the source class B are defined by $'*'|\#([' A.a := f_{T_b \mapsto T_a}(B.b)']^* [' B.b := f_{T_a \mapsto T_b}(A.a)']^* [' A.a :=: B.b']^* \; [' B.b']^*)$. The option $*$ means that all of attributes of B are inherited by A without changing their types and names. For each attribute b of B, the options $A.a := f_{T_b \mapsto T_a}(B.b)$ and $B.b := f_{T_a \mapsto T_b}(A.a)$ mean that the attribute $T_b : b$ of B is inherited by A in the form of $T_a : a$. The option $A.a :=: B.b$ means that the attribute b of B is inherited by A with the different name a while the type remains the same; The option $B.b$ means that the attribute b of B is inherited by A in its original form.

The language compiler can generate a deputy class from the statement. The deputy class is created as a subclass of the primitive class $SelectionDeputyClass$. The generation method is implemented as a class method of $SelectionDeputyClass$.

5.4 Object Deputy Browser

Given an object deputy language, we need a tool to browse classes as well as deputy classes in Smalltalk, create new deputy classes from them, and manipulate objects as well as deputy objects. The tool is called object deputy browser. It is implemented by the class $DeputyBrowser$ that is defined as a subclass of the system class $Browser$. It can provide four panes for displaying and manipulating classes, instances, attributes, and texts, respectively. Since methods can be displayed and manipulated by the system browser of Smalltalk, the deputy browser does not provide a pane for methods in order to simplify our implementation.

5.4.1 Pane for Classes

The pane for classes is used to list all of base/deputy classes. Once a base/deputy class is selected, its definition, instances, and attributes will be displayed in the text, instance, and attribute panes, respectively. When the operating menu button is pressed, a menu of commands is to be displayed. It allows users to select the following command items:

- createClass: create a deputy class,
- deleteClass: delete a deputy class, and
- classDefinition: show definition of a base/deputy class.

Since database connecting tools are responsible for the creation and deletion of base classes which are materialized object views of databases, the object deputy browser does not allow users to create or delete a base class. When the command item createClass is selected, a submenu of commands is to be displayed. It allows users to create different types of deputy classes as follows:

- selection: create a deputy class for specialization,
- union: create a deputy class for generalization,
- Join: create a deputy class for aggregation, and
- grouping: create a deputy class for grouping.

When one of the above command items is selected, a template of the statement for the corresponding deputy class definition is displayed in the text pane, users can edit the template for defining their needed deputy class. Once the definition is accepted, the object deputy language compiler will be invoked to generate the corresponding deputy class from the definition.

The current selected deputy class can be deleted by executing the menu command $deleteClass$. When a deputy class is deleted, all of its deputy classes may be deleted automatically. If users want to know the definition of an existing base/deputy class, he/she can first select the base/deputy class and then execute the menu command $classDefinition$, the command picks up the definition from the selected base/deputy class and shows it in the pane for text.

5.4.2 Pane for Instances

The pane for instances is used to list all of instances of the current selected class. Instances of deputy classes are not allowed to be directly added/deleted by users. They are added/deleted indirectly through data update propagations.

The addition and deletion of instances of base classes are done by database connecting tools. An instance is added to the base class by invoking the procedure $add(C, o)$ which may cause an addition of deputy objects of the new instance. When some data is deleted from the database, its corresponding object o will be deleted from the base class C by invoking the procedure $delete(C, o)$. The deputy objects of o may be deleted by the procedure $delete(C, o)$ since it can cause data update propagation.

5.4.3 Pane for Attributes

The pane for attributes is used to list all of the attributes of the current selected class. When both an attribute and an instance are selected, the attribute value of the instance will be displayed in the text pane, where the attribute value can be edited so that it can be updated.

Displaying the value of an attribute of an instance is done by invoking the reading method defined for that attribute. The value is first translated into the text format and then displayed in the text pane.

Since data is usually not allowed to be updated in data warehouses, the text pane only allows users to edit the additional attribute value of deputy objects. The updated attribute value is first translated from the text format into the acceptable one and then stored into the attribute of the current selected instance by invoking the writing method defined for that attribute.

Updating the attribute value is requested to adjust the deputy classes of the selected class and the deputy objects of the selected instance by invoking the subroutines $adjustDeputyClasses(o)$ and $adjustDeputyObjects(o)$, which may cause addition and deletion of deputy objects of the selected instance.

5.4.4 Pane for Texts

The pane for texts is used to display and edit deputy class definitions and attribute values as described above. The associated menu provides commands for editing and accepting the displayed text. The editing commands are implemented by the system classes of Smalltak. The accepting command is implemented by the class $DeputyBrowser$ for creating deputy classes and updating additional attribute values.

5.5 An Application Case

The FOODMAW implementation system has been successfully used to integrate some simple biological information like cells and microbes, which are distributed in several heterogeneous databases and an XML document repository. The information resources consist of several remote databases managed by Oracle and Sybase, respectively, a local database managed by PostgreSQL, and an XML document repository. The databases use tables to store information about classification and characteristics of cells and microbes and the XML document repository uses XML files to contain some simply preprocessed data of microbes. They are integrated in the Smalltalk environment for biological feature analysis.

First, the data objects are extracted from the information sources with FOODMAW's data extraction routines and wrappers. For the data in the tables and the XML documents with certain data schemas defined by DTD and XML Schema, FOODMAW just deploys the data extraction routines to fetch the data schema from the data sources and invoke the routine, *createBaseClass*, to create a class for each table or class as the subclass of *BaseClass*. Then, the corresponding base classes have instances which are created by the wrappers using the records received from those data sources. However, many original XML documents may have no formal schemas but exhibit similar structures. A flexible semiautomatic approach is adopted to process these documents. At first, the samples of the elements are chosen in each document. The information of the nested tags in the samples are extracted and then processed by a data mining tool to find the common structures of each subset of samples. After the manual confirmation of the structures, all of the elements in XML documents are transformed into roughly defined deputy objects, which consist of a set of attributes with the types representing the confirmed structures and a set of attributes with the type indicating a common XML content. For example, an XML element

```
<Repository>
<Company> Beijing Medicine Development Company
</Company>
<CompanyNo>20331344</CompanyNo>
<Address>Unknown</Address>
<Condition>under normal temperature, dry environment
</Condition>
</Repository>
```

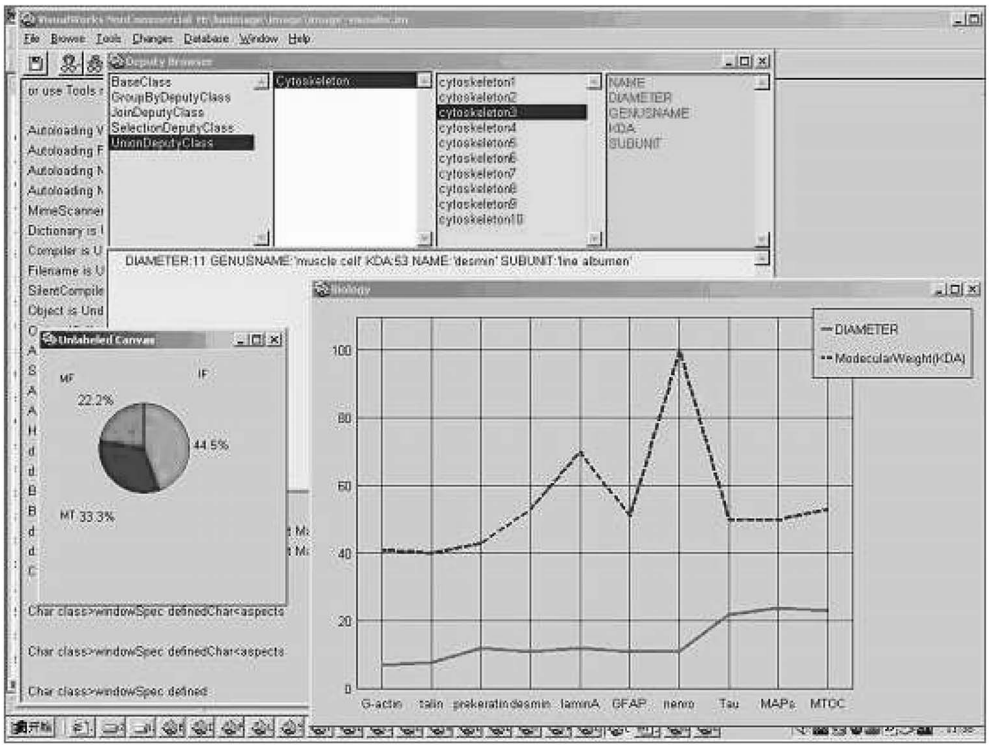

Fig. 6. An application case.

can be extracted into a class *Repository* whose attributes including *Company*, *CompanyNo*, *Address*, and *Condition* if the corresponding tags are confirmed. However, the *Repository* class may also only hold the attributes *Company*, *CompanyNo*, and *Condition* if the *Address* tag is not confirmed. In that case, the *Address* tag with its content is stored in an attribute *others* with a predefined type *XMLTags*, waiting for further processing. After data extraction, the objects would be integrated according to certain standards. Since the data sources are established by different colleges and companies, the naming convention and the structure of the data might be various for different data sources. FOODMAW utilizes the mechanism given in Section 4 to resolve the conflicts and build the data warehouse for data analysis. For example, two deputy classes, *MFIF* standing for *microtubulins* (*MF*) and middle *fibrins* (*IF*) and *MT* standing for *microfilament* albumen, are derived from data sources and defined in the Smalltalk environment. The definitions of the two classes are shown as follows:

MFIF{name: String, diameter: Int, subunit: String, weight: Int, price: Int},
MT{name:String, diameter: Int, subunit:String, Molecularweight:Int, price: Int}.

For the convenience of data analysis, a new class called *Cytoskeleton* is created to merge all of the instances of the two classes and the data from *MFIF* are required to be divided into two classes *Microtubulins* and *Middle Fibrins* before they are merged. Meanwhile, according to the standard naming conventions, the weight attribute should be renamed as *Molecularweight* in the new classes, and the prices should be represented in RMB uniformly although they are represented in original classes using other currencies (e.g., US dollars).

The diameter of middle fibrins is less than 15. We use the condition to differentiate between microtubulins and middle fibrins. Therefore, two new classes *IF* and *MF* are defined according to the range of diameters as follows:

SelectionDeputyClass define: #IF
select: #('MFIF.name', 'MFIF.diameter', 'MFIF.subunit',
'IF.Molecularweight:=MFIF.weight',
'IF.price:=MFIF.price/rate")
from: MFIF
where: 'MFIF.diameter $<$ 15'

SelectionDeputyClass define: #MF
select: #('MFIF.name', 'MFIF.diameter', 'MFIF.subunit',
'IF.Molecularweight:=MFIF.weight',
'IF.price:=MFIF.price/rate')
from: MFIF
where: 'MFIF.diameter $\geq$ 15'

Thus, the class *Cytoskeleton* can be defined as the union of *MF*, *IF*, and *MT* in the following way:

UnionDeputyClass define: #Cytoskeleton
select: #('Cytoskeleton.name:={MF.name, IF.name,
MT.name}'
...
'Cytoskeleton.price:={MF.price, IF.price, MT.price/rate}')
union:#(MF, IF, MT)

Further data analysis can be performed upon the integrated data. To illustrate, we show here how to analyze and discover sizes and weights of different molecular.

For example, we can employ the Nearest Neighbor clustering method to calculate the sizes of different molecular, where the adopted distance for clustering is no

more than 2nm. Three clusters of data (i.e., MF, IF, and MT) are obtained, together with the sum of data within each cluster. From them, we can find out that the size of MF is approximately 7nm, the size of IF is approximately 12nm, and the size of MT is approximately 22nm. According to the sum of every kind of data, we can also easily visualize the percentage of different kinds of data.

To obtain the weights of different molecular, we further apply the decision tree ID3 algorithm to the three obtained clusters and generalize that if molecular weight is less than 45kDa, these molecular are called small albumen; if molecular weight is between 45 kDa and 50kDa, these molecular are called normal albumen and the remainder are called big albumen. Since the weight of MF is about 43kDa, MF thus belongs to small albumen. Since the weight of MT is about 50kDa, MT belongs to normal albumen. Among the three kinds of data in IF, prekeratin has a small weight, approximately 40kDa, thus belonging to small albumen; GFAP has weight 50kDa, thus belonging to normal albumen, and the rest has a heavier weight, thus belonging to big albumen.

Fig. 6 shows the above analysis results: the average diameters are 7 (*MT*), 12 (*IF*), and 22 (*MF*), respectively. The average molecular weights of *MT* and *MF* are 43 and 50, respectively, and the molecular weight of IF varies apparently according to the types of the concrete albumens.

As the above application case illustrates, a deputy object in FOODMAW usually stores the references to the attributes of its source object. Such a deputy object can be regarded as an index to a materialized view over related data objects. Therefore, the deputy objects built from the data integration stage are effectively "skimmed" materialized views over the original deputy objects in the data extraction stage. As most of the attributes' values of such "skimmed" views are references to the source object attributes, their sizes are much smaller in comparison with conventional data warehouses' materialized views whose construction would involve the expensive processes of copying the values from the data sources to the views.

6 Conclusion

An important yet often undermined step toward building a data warehouse is data preparation, the result of which is critical for any online analytical processing and decision support applications. In this paper, we have presented an object-oriented framework (viz., FOODMAW) that we have developed as an elegant support of data warehousing (and mining) through an object deputy approach. In particular, the traditional object-oriented data model extended with deputy objects facilitates better data fusion/integration of the heterogeneous underlying data sources, by enabling various object views to be easily realized in a flexible manner. This is significant for the task of data integration (as part of the data preparation) particularly, since the object deputy approach is superior in resolving data inconsistency and reducing data duplication, thereby increasing the data quality. (In addition, this approach also facilitates efficient object view maintenance incrementally.) Our experimental prototype system of FOODMAW, implemented in Smalltalk, is able to effectively integrate data from different underlying data sources including both commercial databases and open-source ones.

As part of our ongoing research, we are applying several data mining algorithms upon the experimental prototype, along with an empirical study of incremental object view maintenance based on the deputy approach; the effectiveness and efficiency resulted from the FOODMAW framework in these aspects are validated and evaluated. Due to the space and the scope of this special issue, however, our results from those aspects are not included in this paper, which will be reported in our subsequent papers. Moreover, we plan to investigate in our future research such further issues as scalability and its impact on maintaining and handling *semantic* cubes and multifact cube computation based on the object deputy model, as well as online mining of association rules across transactions upon the FOODMAW data warehousing environment.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (60273072,60473076), the Hubei Natural Science Foundation for Distinguished Youth (2002AC003), and the Program for New Century Excellent Talents at the University of China (NCET-04-0675) and State Key Lab of Software Engineering (Wuhan University, China) under grant: SKLSE03-01.

References

[1] A. Abello, J. Samos, and F. Saltor, "Understanding Analysis Dimensions in a Multidimensional Object-Oriented Model," *Proc. Third Int'l Workshop Design and Management of Data Warehouses (DMDW '01),* June 2001.

[2] S. Abiteboul and A. Bonner, "Objects and Views," *Proc. Int'l Conf. Management of Data,* pp. 238-247, 1991.

[3] R. Ahmed et al., "The Pegasus Heterogeneous Multidatabase System," *Computer,* vol. 24, no. 12, pp. 19-27, Dec. 1991.

[4] E. Bertino, "A View Mechanism for Object-Oriented Databases," *Proc. Third Int'l Conf. Extending Database Technology,* pp. 136-151, 1992.

[5] E. Bertino, "Application of Object-Oriented Technology to the Integration of Heterogeneous Database Systems," *J. Distributed and Parallel Databases,* vol. 2, no. 4, pp. 343-370, 1994.

[6] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati, "Information Integration: Conceptual Modeling and Reasoning Support," *Proc. Sixth Int'l Conf. Cooperative Information Systems,* pp. 280-291, 1998.

[7] M.J. Carey et al., "Towards Heterogeneous Multimedia Information Systems: The Garlic Approach," *Proc. Fifth Int'l Workshop Research Issues in Data Eng.: Distributed Object Management,* pp. 124-131, 1995.

[8] S. Chaudhuri and U. Dayal, "An Overview of Data Warehousing and OLAP Technology," *ACM SIGMOD Record,* vol. 26 no. 1, pp. 65-74, 1997.

[9] B. Czejdo and M.C. Taylor, "Integration of Database Systems Using an Object-Oriented Approach," *Proc. IEEE First Workshop Research Issues in Data Eng.—Interoperability among Multidatabase Systems,* pp. 30-37, 1991.

[10] J.C. Franchitti and R. King, "Amalgame: A Tool for Creating Interoperating Persistent, Heterogeneous Components," *Advanced Database Systems,* pp. 313-36, 1993.

[11] G. Yu, K. Kaneko, G. Bai, and A. Makinouchi, "Transaction Management for a Distributed Object Storage System WAKASHI—Design, Implementation and Performance," *Proc. IEEE 12th Int'l Conf. Data Eng.,* pp. 460-468, 1996.

[12] V. Gopalkrishnan, Q. Li, and K. Karlapalem, "Semantic Query Optimization Based on Class Partitioning Techniques in an Object Relational Data Warehousing Environment," *Int'l J. Information Technology,* vol. 7, no. 2, 2001.

[13] J. Han, S. Nishio, H. Kawano, and W. Wang, "Generalization-Based Data Mining in Object-Oriented Databases Using an Object Cube Model," *Data and Knowledge Eng.,* vol. 25, no. 1-2, pp. 55-97, 1998.

[14] S. Heiler and S. Zdonick, "Object Views: Extending the Vision," *Proc. IEEE Sixth Int'l Conf. Data Eng.,* pp. 86-93, 1990.

[15] R. Hull and G. Zhou, "A Framework for Supporting Data Integration Using the Materialized and Virtual Approaches," *SIGMOD Record,* vol. 25, no. 2, pp. 481-92, 1996.

[16] IDC, "Data Warehousing Tools: Market Forecast and Analysis: 2000-2004," IDC report no. 23712, 2004.

[17] K. Karlapalem, Q. Li, and C. Shum, "An Architectural Framework for Homogenizing Heterogeneous Legacy Databases," *SIGMOD Record,* vol. 24, no. 1, pp. 15-20, 1995.
[18] M. Kaul, K. Drosten, and E.J. Neuhold, "ViewSystem: Integrating Heterogeneous Information Bases by Object-Oriented Views," *Proc. IEEE Sixth Int'l Conf. Data Eng.,* pp. 2-10, 1990.
[19] W. Kim, I. Choi, S. Gala, and M. Scheevel, "On Resolving Schematic Heterogeneity in Multidatabase Systems," *Distributed and Parallel Databases,* vol. 1, no. 3, pp. 251-279, 1993.
[20] Knightsbridge Solutions LLC, "Top 10 Trends in Data Warehousing (White Paper)," http://www.knightsbridge.com, 2005.
[21] W. Litwin, L. Mark, and N. Roussopoulos, "Interoperability of Multiple Autonomous Databases," *ACM Computing Surveys,* vol. 22, no. 3, pp. 267-293, 1990.
[22] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom, "Object Exchange across Heterogeneous Information Sources," *Proc. Int'l Conf. Data Eng.,* pp. 251-60, 1995.
[23] T.B. Pedersen, A. Shoshani, J. Gu, and C.S. Jensen, "Extending OLAP Querying to External Object Databases," *Proc. Conf. Information and Knowledge Management,* pp. 405-413, 2000.
[24] Z. Peng and Y. Kambayashi, "Deputy Mechanisms for Object-Oriented Databases," *Proc. IEEE 11th Int'l Conf. Data Eng.,* pp. 333-340, 1995.
[25] Z. Peng and Y. Kambayashi, "Handling Conflicts and Replication During Integration of Multiple Databases by Object Deputy Model," *Proc. 20th Int'l Conf. Conceptual Modeling (ER '01),* pp. 285- 298, 2001.
[26] E.A. Rundensteiner, "MultiView: A Methodology for Supporting Multiple Views in Object-Oriented Databases," *Proc. 18th Very Large Data Bases Conf.,* pp. 187-198, 1992.
[27] C.S. dos Santos, "Design and Implementation of Object-Oriented Views," *Proc. Sixth Int'l Conf. Database and Expert Systems Applications,* pp. 91-102, 1995.
[28] N. Stewart, "Data Warehousing and Business Intelligence Market Forecast 2001-2005," Survey.com market forecast report, 2001.
[29] J. Trujillo, M. Palomar, J. Gomez, and I. Song, "Designing Data Warehouses with OO Conceptual Models," *Computer,* vol. 34, no. 12, pp. 66-75, Dec. 2001.
[30] P. Vassiliadis and T.K. Sellis, "A Survey of Logical Models for OLAP Databases," *ACM SIGMOD Record,* vol. 28, no. 4, pp. 64-69, 1999.
[31] J. Widom, "Research Problems in Data Warehousing," *Proc. Conf. Information and Knowledge Mnagement,* pp. 25-30, 1995.
[32] J. Yang, K. Karlapalem, and Q. Li, "Algorithms for Materialized View Design in Data Warehousing Environment," *Proc. Int'l Conf. Very Large Data Bases (VLDB '97),* pp. 137-146, 1997.

Zhiyong Peng received the BSc degree from Wuhan University in 1985, the MEng degree from Changsha Institute of Technology of China in 1988, respectively, and the PhD degree from Kyoto University of Japan in 1995, all in computer science. He is a professor and vice director of the State Key Laboratory of Software Engineering, Wuhan University of China. He worked as a researcher at the Advanced Software Technology and Mechatronics Research Institute of Kyoto from 1995 to 1997 and was a member of the technical staff at Hewlett-Packard Laboratories, Japan, from 1997 to 2000. His research interests include object-oriented programming, advanced database systems, information integration, and Web services. He is a member of the IEEE and the IEEE Computer Society, the ACM SIGMOD, and the Database Society of Chinese Computer Federation, and is vice director of the VLDB School, China. He has served on many program committees of international conferences and workshops including WISE2001, APWEB2003, WAIM2004, ER2004, DASFAA2005, etc.

Qing Li received the BEng degree from Hunan University, China, and the MSc and PhD degrees from the University of Southern California (USA), all in computer science. Before joining the City University of Hong Kong in 1998, where he is an associate professor in the Department of Computer Science, he taught at the Hong Kong Polytechnic University, the Hong Kong University of Science and Technology, and the Australian National University (Canberra, Australia). His main research interests are in applied object-oriented systems, multimedia/video databases, flexible workflow systems, data warehousing, and multimodality search engines. He has authored or coauthored more than 180 publications in refereed journals and conference proceedings in these areas. Dr. Li has been actively involved in the research community and served as a guest/associate editor for numeral technical journals including the *IEEE Transactions on Multimedia*, *Information Systems*, *WWW* journal, the *Journal of Web Engineering*, *Knowledge and Information Systems*, etc., and as an organizer/coorganizer of many international conferences including WAIM'2004, HSI'2003, VLDB'2002, IFIP DS-9, PAKDD'2001, and WISE'2000. In addition, he has been a programme committee member for more than 40 international conferences (including VLDB, ER, ICDCS, CIKM, CAiSE, DASFAA, CoopIS, DaWaK, and FODO). Locally, he has been an executive committee (EXCO) member of the IEEE Hong Kong Computer Chapter, ACM Hong Kong Chapter, and is currently the chairman of the Hong Kong Web Society (http://www.hkws.org/); in addition, he is a steering committee member of the WISE Society (http://www.i-wise.org/). Dr. Li is a senior member of the IEEE and a member of the IEEE Computer Society.

Ling Feng received the BSc and PhD degrees in computer science from Huazhong University of Science and Technology. She is currently an associate professor at the University of Twente in the Netherlands and was an assistant professor at Tilburg University in the Netherlands (1999-2002) and a lecturer at the Department of Computing in Hong Kong Polytechnic University in China (1997-1999). Her research interests are distributed object-oriented database management systems, knowledge-based information systems, data mining and its applications, data warehousing, data/knowledge management issues in the Internet era, including the integration of database and Web-based information technologies, XML databases, and knowledge-based digital libraries. She is a member of the IEEE and the IEEE Computer Society.

Xuhui Li received the BS degree in information science and the MS and the PhD degrees in computer science from Wuhan University in 1996, 1999, and 2003, respectively. He is an assistant professor in the State Key Laboratory of Software Engineering, Wuhan University of China. His research interests include data integration, formal semantics, mobile computing and grid computing, and computer simulation.

Junqiang Liu received the BS degree from Henan University, China, in 2002 and the MS degree from Wuhan University, China, in 2004. He is a PhD candidate in the State Key Laboratory of Software Engineering, Wuhan University. His research interests include information integration from heterogeneous data sources, data mining, and bioinformatics.