Abstract—Information integration service has changed from the traditional query-oriented to the future user-oriented service. The current research on information integration service focuses on providing personalized service based on user's preference, interest and habit. In order to solve the problem of personality in information integration, a novel method based on Object Deputy Model is proposed and the architecture of Personalized Information Integration System by Object Deputy Model (PIISODM) is established. PIISODM has fully utilized the flexible object views, multiple roles and dynamic classification of Object Deputy Model. It can generalize personalized integration views very easily and efficiently. A use case of PIISODM applied in personalized electronic map system is shown to illustrate the validity of the method.

Key words-personalization, information integration, object deputy model, ontology

I. INTRODUCTION

With the pervasive application of network and the progress of information technologies, the information sources increase explosively, which is often heterogeneous, autonomous and distributed. The technology of bringing together heterogeneous and distributed information sources and establishing efficient information sharing is known as information integration. According to different demands, users need integrate information distributed in multiple locations by different approaches. However, most information integration systems are designed one-size-fits-all: all users see the exact same results regardless of interests, habits and previous interactions. But one size often does not fit all. For different users, the result of such integration often has only "scope" but no "depth" and exists plenty of rubbishes. Instead of presenting the same content, the integration mechanism should be flexible and personalized, adapting to user's preference, interest and habit. The personalized information integration of heterogeneous data has become the bottleneck problem of data sharing.

Information integration service has changed from the traditional query-oriented to the future user-oriented service. The current research on information integration service focuses on providing personalized service based on user's preference, interest and habit. There are 2 challenges to build personalized information integration system. The first challenge is how to accurately obtain user's personalized information. Personalization technology is the foundation of personalized information integration, which has become to be a focus research in the domain of information service. The crucial technologies related to personalization are introduced in [1], which include the representation and modification of user profile, the representation of resource, the recommendation technology, and the architecture of personalization. The second challenge is use what kind of mechanism to flexibly and efficiently integrate information based on user's personalized demand. The personalized integration mechanism according to user's preference, interest and habit, which should permit user chooses data sources by need and can transform data format on demand. Because users’ demands and habits are diverse, the mechanism should make all kinds of integration fashions coexist and not conflict each other. We will focus to resolve the second challenge in this paper.

The object-oriented data model is semantically rich and can provide a variety of abstraction mechanisms for information integration. Furthermore, it can define methods that enable arbitrary combinations of information stored in local databases and make it possible to integrate nontraditional databases through behavioral mapping. However, the object-oriented model is not enough for information integration because it has two serious problems. Firstly, it can only provide subclass constructor and support inheritance from superclass to subclass. Information integration needs not only specialization but also aggregation and generalization. Aggregation can be used to integrate component objects distributed in different databases. 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. Secondly, the view mechanism [2-6] 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, especially in personalized information integration.

In order to solve the problems, we extend the conventional object-oriented model with the concepts of deputy objects and deputy classes. A deputy object is used to extend and customize its source object(s). Its schema is defined by a deputy class which can be derived by an object deputy algebra. The object deputy model is more flexible than the traditional object-oriented model. It can provide inheritance for specialization, generalization and aggregation. In addition, as we discussed in the paper [7], the view
mechanism is easy to be implemented by the object deputy model. The object deputy model can flexibly choose data sources by object deputy algebra and transform data format by switching operations. The use of the object deputy model can facilitate the creation of the personalized information integration system.

This paper proposes a framework for personalized information integration system by object deputy model. Our system enables users to create their own integration views according to their needs or interests. This system has fully utilized the flexible object views, multiple roles and dynamic classification of Object Deputy Model. It can generalize personalized integration views very easily and efficiently.

The remainder of this paper is organized as follows. Section 2 introduces our object deputy model. Section 3 provides our solution on how to flexibly and efficiently integrate information based on user’s personalized demand. We present our object deputy mechanism and outline the personalized information integration architecture. Section 4 gives a use case and section 5 concludes.

II. OBJECT DEPUTY MODEL

The concept of deputy objects was at first introduced by the authors for the unified realization of object views, roles, and migration [7]. In order to illustrate that it is also useful for personalized information 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 defined as

\[ C = \{o\}, \{T_o:a\}, \{m: \{T_p:p\}\} \]

- \(o\) is the extent of C, where o is one of the instances of C.
- \(\{T_o:a\}\) is the set of attribute definitions of C, where a and Ta 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 \(T_o:a\), 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:

\[ \text{read}(o, a) \Rightarrow o.a, \quad \text{write}(o, a, v) \Rightarrow o.a := v \]

Here, \(\Rightarrow\), \(\Rightarrow\) 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, p and \(T_p\) represent parameter name and type, respectively. Applying method m to object o with parameters \(\{p\}\) is expressed as follows:

\[ \text{apply}(o, m, \{p\}) \]

Deputy objects are defined as extension and customization of objects. The schemata 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 as well as constraints. Note that the switching operation is peculiar to deputy objects. It is used to switch an operation request from a deputy object to its source object. Thus, the source object can execute the operation for the deputy object. From the user’s view, it seems as if the deputy object executed the operation. During the switching process, the operation request can be changed into the form suitable for the source object so that the capability of an object can be customized for different application situations. In general, deputy objects and deputy classes can be defined 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

\[ C' = \{o\}, \{T_o:a\}, \{m: \{T_p:p\}\} \]

be a source class.

Its deputy class \(C_{d}^d\) is defined as

\[ C_{d} = \{o\}, \{T_o:a\}, \{m: \{T_p:p\}\} \]

where

- \(\{m: \{T_p:p\}\}\) is the set of method definitions of \(C_{d}^d\), where m and \(\{T_p:p\}\) are method name and a set of parameters, p and \(T_p\) represent parameter name and type, respectively. Applying method m to object o with parameters \(\{p\}\) is expressed as follows:

\[ \text{apply}(o, m, \{p\}) \]

Deputy objects are defined as extension and customization of objects. The schemata 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 as well as constraints. Note that the switching operation is peculiar to deputy objects. It is used to switch an operation request from a deputy object to its source object. Thus, the source object can execute the operation for the deputy object. From the user’s view, it seems as if the deputy object executed the operation. During the switching process, the operation request can be changed into the form suitable for the source object so that the capability of an object can be customized for different application situation. In general, deputy objects and deputy classes can be defined 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

\[ C' = \{o\}, \{T_o:a\}, \{m: \{T_p:p\}\} \]

be a source class.

Its deputy class \(C_{d}^d\) is defined as

\[ C_{d} = \{o\}, \{T_o:a\}, \{m: \{T_p:p\}\} \]

where

- \(\{m: \{T_p:p\}\}\) is the set of method definitions of \(C_{d}^d\), where m and \(\{T_p:p\}\) are method name and a set of parameters, p and \(T_p\) represent parameter name and type, respectively. Applying method m to object o with parameters \(\{p\}\) is expressed as follows:

\[ \text{apply}(o, m, \{p\}) \]
\{T_d : a^d \} is the set of the attributes inherited from \{T_{as} : a^s \} of \ C^s \ of which switching operations are defined as:
\[
\begin{align*}
\text{read} \ (o^s, a^d) & \Rightarrow f_{T_{as} \rightarrow T_d} \ (\text{read} \ (o^s, a^s)) , \\
\text{write} \ (o^s, a^d, v^d) & \Rightarrow \text{write} \ (o^s, a^d, f_{T_{as} \rightarrow T_d} (v^s)) .
\end{align*}
\]

Here, function \( f_{T_{as} \rightarrow T_d} \) converts the value of one type \( T \) to the value of another type \( T' \).

\{T_p : a^p \} is the set of the additional attributes of \( C^d \), of which basic methods are defined as:
\[
\begin{align*}
\text{read} \ (o^s, a^d) & = \uparrow o^d \cdot a^d , \\
\text{write} \ (o^s, a^d, v^d) & = o^d \cdot a^d := v^d .
\end{align*}
\]

\{m^d : \{T_{ps} : p^s \}\} is the set of method definitions of \( C^d \), where
\[
\{m^d : \{T_{ps} : p^s \}\} \text{ is the set of the methods inherited from } \{m^s : \{T_{ps} : p^s \}\} \text{ of } \ C^s , \text{ which are applied through switching operations as:}
\[
\begin{align*}
\text{apply} \ (o^s, m^d, \{p^s \}) & \Rightarrow \text{apply} \ (o^s, m^s, f_{T_{ps} \rightarrow T_d} \{p^s \}) .
\end{align*}
\]

\{m_{as}^d : \{T_{a^s d} : p^s \}\} \text{ is the set of the additional methods of } \ C^d , \text{ which are applied as:}
\[
\begin{align*}
\text{apply} \ (o^s, m_{as}^d, \{p^s \}) .
\end{align*}
\]

The object deputy model provides an object deputy algebra for deputy class derivation, which contains the following six operations.

1) The Select operation is used to derive a deputy class of which instances are the deputy objects of the instances of a source class that are selected according to a selection predicate.

2) The Project operation is used to derive a deputy class which only inherits part of attributes and methods of a source class.

3) 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.

4) The Union operation is used to derive a deputy class of which extent consists of deputy objects of instances of more than one source class.

5) 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.

6) 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.

There are semantic constraints that are defined as predicates of deputy classes. They are expressed as Boolean expression, such as book.price > 30 being used as selection predicate. The selection predicate determines existence of a deputy object according to the state of its single source object. Only when the source object satisfies some special condition, its deputy object can exist. The combination and grouping predicates define the existence conditions between a deputy object and several source objects. In order to enforce these semantic constraints, data update propagations between deputy objects and their source objects need to be supported.

We have designed three procedures for data update propagation, which are caused by addition of an object, deletion of an object and modification of one attribute value of an object, respectively. They are realized based on bilateral links between objects/classes and their deputy objects/classes as well as various semantic constraints. That is, when the above basic update operations occur on some classes, their deputy classes will be examined so that the updates on deputy classes can be caused in order to maintain semantic constraints defined by deputy classes. The update procedures may be invoked recursively, since deputy classes may have their own deputy classes.

Based on the object deputy model, we have implemented a database system called TOTEM and designed an object deputy database language[8] which can create various kinds of deputy classes, including SelectionDeputyClass, JoinDeputyClass, UnionDeputyClass, and GroupDeputyClass.

III. PERSONALIZED INFORMATION INTEGRATION BY OBJECT DEPUTY MODEL

In this section we propose PIISODM (short for Personalized Information Integration System by Object Deputy Model). We first present PIISODM architecture, and then address several key technical issues. For each of them, we briefly propose our solution.

A. System Architecture

Figure 1 depicts the architecture of PIISODM. The key is to provide a logical view of one’s personalized information, based on meaningful deputy objects/classes. This logical view, described by domain ontology, is provided by constructing an object-oriented database, as TOTEM mentioned above, which complements current data storage. We now describe the three layers of PIISODM.

Figure 1. PIISODM architecture
Storage layer: The storage layer is composed with domain ontology database, object-oriented resource database (TOTEM) and user information database (UIDB), which is data part of the system. Local information sources such as relational databases (RDBs), object-oriented databases (OODBs) and XML files are wrapped respectively into object-oriented form in object deputy database---TOTEM. Users’ personalized information such as user’s background information, interests, habits and visit behavior, are stored in UIDB.

Personalized presentation layer: it is composed with 3 modules. The user information collection module (UICM) and user interesting model (UIM) are corresponding to the implementation of personalized service. The UICM updates UIDB and maintains UIM by alternating records between user and system. Then system can recommend personalized resources to user from UIM. Using conceptions in domain ontology, the UIM considers the relationship between conceptions and describes user’s personalization from multidimensional. The object deputy integration mechanism (ODIM) implements flexible and efficient personalized integration based on UIM.

Integration layer: This layer displays personalized integration views for different users. From UIM, we can obtain user’s personalized information. The ODIM according to user’s preference, interest and habit, which can permit user chooses data sources by needs and transform data format on demand. Because users’ demands and habits are diverse, the mechanism can make all kinds of integration fashions coexist and not conflict each other.

It is crucial to provide personalized views to different users in personalized information integration system. Our object deputy mechanism can achieve it easily and efficiently. Next, we’ll introduce it.

B. Object Deputy Integration Mechanism

In order to define personalized integration interfaces for different users, the concept of views was introduced. Because views in RDB can be defined conveniently using relational algebra, RDB achieve a great success. However, in order to realize high performance applications, the object-oriented data model encapsulates data and methods in terms of objects. It is difficult to divide and combine objects, thus flexible view mechanism of conventional OODB is rather limited. To allow objects to be divided and combined freely, we extend the conventional object-oriented data model with the concepts of deputy objects and deputy classes. A deputy object is used to extend and customize its source object(s). The concept of deputy objects can be used to realize object views, because they inherit attributes and methods of source objects by switching operations, which can customize attributes and methods of source objects.

The schema of deputy objects is defined by a deputy class that can be derived by an object deputy class like relational algebra defines its schema. An object can have many deputy objects that may represent its much faceted nature. A deputy object can have its own deputy object as well. Many objects can be combined through a single deputy object. Thus, objects can be divided and combined indirectly through their deputy objects so that the flexibility of OODBs can be easily achieved.

![Figure 2. Personalized Views by Object Deputy Mechanism](image)

In addition, data update propagation between deputy objects and objects can be supported by three procedures, which are caused by addition of an object, deletion of an object and modification of one attribute value of an object, respectively. This can be used to realize dynamic classification.

As shown in Figure 2, we discuss about modeling for personalized information integration system using ObjectDeputy Mechanism. The major advantages are summarized as follows.

1) User can create a deputy class that includes deputy objects inherited from objects or other deputy objects. This deputy class can be regarded as a layer and can be shown as a personalized view. So to speak, user can freely choose data sources by need. In addition, the mechanism makes all kinds of personalized integration views coexist and not conflict each other because the different definitions of deputy classes.

2) A deputy object has its own persistent identifier, and may have additional attributes and methods that are not derived from its source object(s). So, the user can append additional attributes and operations to the deputy class according to his need and interest.

3) The attributes and methods of an object can be inherited by its deputy objects. The inheritance is realized through switching operations that can change the names and types of the inherited attributes and methods. Thus, to the same integration result, different users can show it in different fashions by their own preferences.

4) There is a bilateral link between an object and one of its deputy objects, which allows not only inheritance but also update propagation between them. Thereby, it guarantees the result of personalized integration is updated.

From above, we found our object deputy model is favorable for the personalized information integration. It is more flexible and efficient than traditional ones.

IV. A USE CASE

Our personalized information integration system is very useful to build personalized digital libraries, personalized
After creating the personalized map, the user can modify the value of the attributes at any moment. In our system, we set a maximum number of customized maps for every user. Organizing data like this is convenient for the users to observe the geographical information from different point of views. And the most important advantage is that there is not great data redundancy. Our approach can provide less storage space consumption and shorter query response time.

V. RELATED WORK AND CONCLUSION

A number of PIM (Personal Information Integration) projects studied the method to organize and search information effectively. They all discard the traditional hierarchical directory model. Haystack [9] and MyLifeBits [10] resort to annotations in building a graph model of information; Haystack puts more emphasis on personalization. Placeless Documents [11] annotates documents with property/value pair, and group documents into overlapping collections according to the property value. Stuff I’ve Seen (SIS) [12] indexes all types of information and provides a unique full-text search interface. Finally, LifeStreams [13] organizes documents based on a chronological order. All of the above projects manage information at the document level. Our approach distinguishes from them by taking objects as the search and organization unit and facilitating the search with associations between objects. The system uses an ontology to guide information management, allowing manipulation and personalization of the ontology.

This paper serves to bring personal information management closer to the mainstream of data management research, and as a platform for the next generation of information integration systems. We described the current implementation of PIISODM that performs personalized information management and integration.

Personalized information integration is a rich area for further research. Further down the road, we plan to use the PIISODM to discover useful patterns in one's data set, such as clusters of people who are related in ways that are not explicit in one's data. Finally, we will use PIISODM to coordinate multiple PIM devices and provide a flexible tool for merging multiple data sets of a user.

REFERENCES


