AN APPROACH TO RESOLVING SEMANTIC HETEROGENEITY IN A FEDERATION OF AUTONOMOUS, HETEROGENEOUS DATABASE SYSTEMS

JOACHIM HAMMER and DENNIS McLEOD
Computer Science Department
University of Southern California
Los Angeles, CA 90089-0781, USA
(213)740-4504
E-mail: {joachim, mcleod}@cs.usc.edu

ABSTRACT

An approach to accommodating semantic heterogeneity in a federation of interoperable, autonomous, heterogeneous databases is presented. A mechanism is described for identifying and resolving semantic heterogeneity while at the same time honoring the autonomy of the database components that participate in the federation. A minimal, common data model is introduced as the basis for describing sharable information, and a three-pronged facility for determining the relationships between information units (objects) is developed. Our approach serves as a basis for the sharing of related concepts through (partial) schema unification without the need for a global view of the data that is stored in the different components. The mechanism presented here can be seen in contrast with more traditional approaches such as "integrated databases" or "distributed databases". An experimental prototype implementation has been constructed within the framework of the Remote-Exchange experimental system.

Keywords: Autonomy, federation, heterogeneous databases, interoperability, resolution, semantic heterogeneity.

1 Introduction

With the information age and the accompanying rapid advances in information technology has come an abundance of data that is overwhelming and cannot be processed efficiently by humans alone. For example, in the scientific community, in order to keep pace with this wealth of information, researchers continue to specialize in smaller and smaller areas. Therefore, the quality and progress of scientific endeavors depends on the researcher's ability to efficiently store large quantities of heterogeneous data and, even more importantly, to share and exchange this knowledge with his/her colleagues. Such environments consisting of a collection of data/knowledge bases and their supporting systems, and in which it is desired to accommodate the controlled sharing and exchange of information among the collection are extremely common in but not exclusively confined to the scientific community. Cooperative work, computer-based manufacturing, scientific databases, and traditional data processing are only a few of the environments where collaboration among autonomous, heterogeneous components is desired [37, 42]. It is obvious that the next generation of database systems must keep pace with this current trend in order to ensure support for large inter-disciplinary projects such as the Human Genome Project and Informatics[a] (HGI) [15]. Considering an example from a non-scientific application domain, we can observe that many of the major airlines use their own online reservation systems to keep track of flight schedules and customer reservations. In order to find the lowest possible airfare

[a]This project is a major scientific venture under the auspices of both the National Institute of Health (NIH) and the Department of Energy (DOE).

available or put together a trip involving more than one airline, travel agencies must have the ability to access and share data from many different reservation systems. These systems may have been created using different specifications and different data models and it is imperative to find ways to overcome these differences before any sharing can take place.

In light of these observations, participants in the 1990 Workshop on Future Database Systems Research sponsored by the National Science Foundation (NSF) have identified the following goal as one of the key areas in future database research [50]: The creation of an environment that allows for the controlled sharing and exchange of information among autonomous, heterogeneous databases. This is a very ambitious goal and its complete realization depends on the realization of several important sub-goals. In order to support this kind of information sharing, a mechanism is needed to make selected data objects from one database "usable" by other databases of similar or different constitution. For example, the respective databases may differ in their database management system (DBMS), their data model (DM), or their conceptual schema; in this research we are not concerned with differences in hardware or operating systems. Such a collection of cooperating but heterogeneous, autonomous component database systems (DBSs) may be termed a *federated database system* (FDBS) or *federation* for short [17, 19]. A key characteristic of a federation is the cooperation among independent systems, which is reflected by controlled and sometimes limited integration of its autonomous components. This kind of cooperation is often referred to as *interoperability*. To this extent, an FDBS provides an explicit interface to its database components that facilitates sharing while at the same time allowing each component to function independently.

A key aspect of making data be "usable" across heterogeneous databases involves *(partial) schema unification*, and it is imperative to develop methods that can perform this task with as little human intervention as possible. In this paper, we use the term "unification" to denote the process of combining different (meta-)data objects together. While the term "(schema) integration" prevails as the predominant expression for activities of this kind, we stress here partially uniting meta-data specifications. By meta-data we mean objects that represent the structural part of a database (i.e., the conceptual schema) as opposed to "factual" data which represents the contents of the database. Rather than unifying complete schemas with each other, a process that is both costly and difficult, partial schema unification is concerned with uniting non-local objects[b] that are selected on demand by components in the federation. One of the fundamental problems that must be solved before one can successfully unify schemas (or parts thereof) is that of *semantic heterogeneity* or *semantic diversity*. In the database context, this heterogeneity refers to differences in the meaning and use of data that make it difficult to identify the various relationships that exist between similar or related objects in different components.

The goal of our work is to demonstrate a new, modular facility for resolving semantic heterogeneity in a federation of autonomous, heterogeneous, object-based databases in order to achieve interoperability. By resolving semantic heterogeneity we mean two things: (1) determine the relationships between objects that model similar information, and (2) detect possible conflicts in their representations that pose problems during the unification of shared data. Our approach to resolving semantic heterogeneity can be utilized by a variety of intelligent and cooperative information systems (ICISs), such as *Remote-Exchange* [11], for example.

The remainder of this paper is organized as follows: In Section 2, we review related work. In Section 3, we introduce a typical sharing scenario in a federated database environment where individual components can share and exchange objects. Section 3.1 gives a detailed definition of semantic heterogeneity and other important terms used in this research. A spectrum for heterogeneity is provided and the most common causes for schema diversities are discussed using examples. Section 4 describes the different steps that are needed to achieve interoperability between individual components. We consider the use of an intelligent *sharing advisor* that will locate relevant information sources in other components. In Section 5, we provide

[b]In this context and throughout the remainder of this paper, the term "object" may refer to type objects, function objects (also termed methods or behavioral objects), and instance objects.

the object context in which we have couched our research. In order for any sharing to take place among heterogeneous components, the components must agree on some common model for describing shared data. The model used in this work is called Minimal Object Data Model (MODM), which is a simplified version of the Kernel Object Data Model (KODM) from *Remote-Exchange*. We also present an overview of the different ways in which (meta-)data objects (i.e., type objects) can be related to each other depending upon the relationships of the real-world concepts they represent. In Section 6, we show in detail how our approach to resolving semantic heterogeneity operates. Specifically, we present a strategy for determining the relationship between two objects which is based on three separate mechanisms (parts) for gathering structural as well as semantic information about the objects in question. In Section 7, we investigate three scenarios for unifying foreign objects with local objects once ambiguities between the objects have been resolved. Finally, Section 8 contains concluding observations with a critical evaluation of our results and their potential impact.

2 Related Research

Several projects and prototype development efforts to support heterogeneous, distributed databases started in the late 1970's and early 1980's, mostly focusing on providing methodologies for relational database design. This work addressed methodologies and mechanisms to integrate individual, user-oriented schemata into a single global conceptual schema (view integration). In their 1986 survey paper, Batini et al. [4] investigate twelve of these early methodologies and compare them on the basis of five commonly accepted integration activities (preintegration, comparison of schemas, conforming of schemas, merging, restructuring). However, most of the approaches examined in this survey do not directly address the diversity problem described above. The problem of achieving semantic interoperability has been studied extensively in view integration [3, 18, 39, 41, 48] using the relational and semantic data models, but (partial) unification of multiple heterogeneous object-based databases is still in its infancy.

Research in the area of heterogeneous database systems (HDBSs) began only a decade ago [13, 51]. The term "heterogeneous databases" was originally used to distinguish work which included database model and conceptual schema heterogeneity from work in "distributed databases[c]" which addressed issues solely related to distribution [7]. Recently, there has been a resurgence in the area of heterogeneous database systems. This work may be characterized by the different levels of integration of the component DBSs and by different levels of global (federation) services. In Mermaid [52], for example, which is considered a tightly coupled HDBS, component database schemas are integrated into one centralized global schema with the option of defining different user views on the unified schema. While this approach supports pre-existing component databases, it falls short in terms of flexible sharing patterns. Furthermore, the integration process is very expensive and tends to be difficult to change.

The federated architecture proposed in [19], which is similar to the multidatabase architecture of [34], involves a loosely coupled collection of database systems, stressing autonomy and flexible sharing patterns through inter-component negotiation. Rather than using a single, static global schema, the loosely coupled architecture allows multiple import schemas, enabling data retrieval directly from the exporter and not indirectly through some central node as in the tightly coupled architecture. Examples of loosely coupled FDBSs are MRSDM [33], Omnibase [44], and Calida [23].

One reason for the relatively slow progress in the area of integrating multiple heterogeneous databases is that unlike views, heterogeneous databases provide complimentary information, e.g., their object groupings may not have many properties in common. This is not the case with views, which are typically designed for rather narrow domains and thus exhibit a great deal of redundancy in their vocabulary. View integration mainly performs structural transformations from one conceptual schema to another and the process of disambiguation (viz., solving semantic diversity) is much more straightforward than in any other context.

[c]The term distributed database is used here as it has been mainly used in the literature, denoting a relatively tightly coupled, homogeneous system of logically centralized, but physically distributed component databases.

There are several different approaches to view integration that can be distinguished by the set of operators used to perform the transformation [3, 19, 40]. Schema integration of heterogeneous databases, on the other hand, poses substantially new problems. Since names of objects are used much more inconsistently in multiple heterogeneous databases, previous approaches taken from view integration can only be partially applied to the comparison and integration of heterogeneous databases. Another problem that must be addressed when integrating schemas is that of inter-database object correspondences. Work in this area has focused on inter-database object reference by description [46] and object identification via language constructs [27]. Based on these earlier results, there are now several efforts focusing on database schema integration, with varied approaches to handling the problem of semantic diversity.

One common approach to schema integration is to reason about the meaning and resemblance of heterogeneous objects in terms of their structural representation. Larson et al. [29], for example, base their approach to schema integration on the so-called basic principle of integrated attributes: "... any pair of objects whose identifying attributes can be integrated can themselves be integrated." In this case, the meaning of an attribute is approximated in terms of its value type (set of possible values), cardinality constraints, integrity constraints, and allowable operations. However, one can argue that any such set of characteristics does not sufficiently describe the real-world meaning of an object, and thus their comparison can lead to unintended correspondences or fail to detect important ones. Other promising methodologies that were developed include heuristics to determine the similarity of objects based on the percentage of occurrences of common attributes [18, 41, 48]. More accurate techniques use classification for choosing a possible relationship between classes [45]. Whereas most of these methods primarily utilize schema knowledge, techniques utilizing semantic knowledge (based on real-world experience) have also been investigated. Fankhauser et al. [12] present an approach to integrating heterogeneous database schemas utilizing fuzzy and possibly incomplete real world knowledge. In their methodology, class definitions, or more generally schemas, are disambiguated by matching unknown terms with concepts in an interconnected knowledge base.

A different approach to schema integration uses behavior to solve domain and schema mismatch problems [26]. Domain and schema mismatch are two important semantic integration problems for interoperating heterogeneous databases. The domain mismatch problem generally arises when some commonly understood concept, for example money, is represented differently in different databases (i.e., U.S. dollars vs. English pounds). Schema mismatch arises when similar concepts are expressed differently in the schema (i.e., a relationship that is being modeled as one-to-one in one schema and one-to-many in another). Kent [26] proposes to use an object-oriented database programming language to express mappings between these common concepts that allow a user to view them in some integrated way. It remains to be seen if a language that is sophisticated enough to meet all of the requirements given by Kent in his solution can be developed in the near future.

A very recent approach to interoperability by Mehta et. al. [38] uses so-called path-methods to access distant information in a federation of database components. Instead of integrating foreign objects into the intended target schema (as we propose in this research), explicit inter-component and inter-object mappings are created between the source and the target classes in order to retrieve and update related data objects. The obvious drawback of this approach is the large overhead in calculating and maintaining the mappings which may be impractical for large federations with extensive sharing patterns. No mention is made as to how a relationship between objects belonging to different components is determined in the first place.

3 The Federated Database System Context

As observed above, the trend towards decentralization of computing that has occurred over the past decade has accelerated the need for effective principles, techniques and mechanisms to support information sharing and exchange among distributed heterogeneous databases. Consider the following scenario involving the travel business. Several travel agencies located in different cities of the country decide that it would be mutually beneficial to join forces and form a nation-wide **F**ederation **O**f **T**ravel **A**gencies (FOTA). The

Figure 1: A loosely interconnected federation of travel agencies

goal of FOTA is to share and exchange travel related information between the individual agencies in order to stay competitive and keep up to date with the fast pace in the world of business and pleasure travel. However, each travel agency wants to retain autonomy over its own database with respect to organization and administration. Therefore, when a component agrees to joining the federation it will keep its own local DBMS together with its original conceptual schema. The main advantage of this is that the costly and inefficient process of restructuring the component's existing data is avoided. Furthermore, there is no need for travel agencies to retrain their employees on a new system. We may assume that since the agencies are located in different parts of the country, the contents of their databases reflect the different travel habits of their customers. Thus FOTA depicted in Figure 1 is an example of a loosely coupled collection of heterogeneous database systems, or FDBS, as described in Section 2.

Components of FOTA keep track of the following kinds of information: rental cars, airline information, train information, pleasure cruise information, sight-seeing information such as museums and historic buildings, local entertainment such as shows and events, and hotel information. We also assume that each travel agency will agree to use an object-based database model at the federation interface. Components may update their own local schema at any time, and it is the responsibility of an importing component to obtain access to new information. Figure 2 shows a snapshot of the information stored in the federation at a given instance during its lifetime. As mentioned before, the content and organization of each database differs from travel agency to travel agency. We can see, for example, that travel agency B in Miami is the only component with information on pleasure cruises, due to the heavy demand and the large number of cruises departing from and arriving in that area. All components contain data on hotels for the specific area to which the travel agency is primarily catering. With the exception of some large chains such as Holiday Inn, or Hilton, for example, hotel information is relatively localized and not as readily available in the remaining parts of the country. For example, travel agency E which is located in Washington, has data on hotels in the Northeastern US, whereas agency A in Los Angeles represents "places-to-stay" in California. All components contain national airline and train information.

It is important to note that an additional difficulty in finding a solution to the problem of achieving

Figure 2: A conceptual overview of the federation and its components

interoperability in FOTA and other similar federations stems from the conflicting nature of sharing and autonomy. On one hand, a travel agency would like to share information with other components of FOTA. On the other hand, the same component would also like to exercise some degree of control over the sharing process, e.g., control over the information it is willing to "export" to the other components. Since the focus of our work is on a solution to resolving semantic heterogeneity, several other important issues such as security, i.e., access control, and automatic update of shared data are not dealt with in their entirety here. In consequence, we assume that all the information stored in a specially "marked" section of a travel agent's database, called the *export schema*, is available to every other travel agency in the federation.

3.1 Semantic Heterogeneity in Federated Databases

As noted above, a central problem of interoperability that must be addressed in order to support the sharing of information among a collection of autonomous, heterogeneous databases is *semantic heterogeneity*. By this we mean variations in the manner in which data is specified and structured in different components. Semantic heterogeneity is a natural consequence of the independent creation and evolution of autonomous databases which are tailored to the requirements of the application system they serve. For the remainder of this paper, we refer to the problem of overcoming semantic heterogeneity in order to enable information sharing among autonomous, heterogeneous databases the *heterogeneity problem* in federated databases. Before we can present a solution to the heterogeneity problem, it is useful to examine the different kinds of semantic heterogeneity that may occur.

3.1.1 The Spectrum of Heterogeneity

Within the context of a federation of loosely coupled components, we can identify a spectrum of heterogeneity based on the following levels of abstraction:

(i) Meta-data language (conceptual database model):
The components may use different collections of and techniques for combining the structures, con-

straints, and operations used to describe data. With respect to the federation of travel agencies, we can observe that travel agency A, for example, uses an OSQL-like data definition language (DDL), while travel agency E uses Postquel and its associated DDL.

(ii) Meta-data specification (conceptual schema):
While the components share a common meta-data language (conceptual database model), they may have independent specifications of their data (varied conceptual schemas). For example, this refers to the different schemas used by the members of FOTA.

(iii) Object comparability:
The components may agree upon a conceptual schema, or more generally agree upon common subparts of their schemas; however, there may be differences in the manner in which information facts are represented [25]. This variety of heterogeneity also relates how information objects are identified, and to the interpretation of atomic data values as denotations of information modeled in a database (e.g., naming). Looking ahead to travel agency D's schema in Figure 3, we can see that the type **Accommodations** represents information that is comparable to the information represented in E's **Places in Northeastern Region**.

(iv) Low-level data format:
While the components agree at the model, schema, and object comparability levels, they may utilize different low-level representation techniques for atomic data values (e.g., units of measure). In terms of FOTA this refers to the problem that arises when travel agency A, for example, decides to represent all its fare prices in English pounds rather than in U.S. dollars.

(v) Tool (database management system):
The components may utilize different tools to manage and provide an interface to their data. This kind of heterogeneity may exist with or without the varieties described immediately above. In the context of FOTA, this kind of heterogeneity is due to the fact that components may use a different DBMS.

We assume that the components utilize a common language (see Section 5.1), thus ruling out the occurrence of heterogeneities of type (i), conceptual database model. Furthermore, tool heterogeneity, which is type (v) in the above spectrum, is treated as somewhat orthogonal to our concern in this paper. As a result, for the purpose of this work we term types (ii) through (iv) in the heterogeneity spectrum *semantic heterogeneity*.

3.1.2 Causes of Semantic Heterogeneity

According to Batini et al. [4] there are three major causes for semantic heterogeneity:

(i) Different perspectives:
This is a modeling problem that finds its roots mostly during the design phase of a database schema. Different user groups or designers adopt their own viewpoints when modeling the same information. For instance, in example 1 in Figure 3, different names were attached to the same concept (**Accommodations** versus **Places in the Northeastern Region**) in the two schemas of travel agencies D and E, respectively.

(ii) Equivalent constructs:
The rich set of constructs in data models allows for a large number of modeling possibilities, which results in variations in the conceptual database structure [25]. Typically, in conceptual models, several combinations of constructs can model the same real-world domain equivalently. In example 2 in Figure 4, the association between flights and fare information was modeled as a function *has_price* in schema A as opposed to a separate type **Flight-Fare-Combinations** connecting **Flights** and **Fare-Types** in schema B.

Figure 3: Example 1. Semantic heterogeneity in FOTA

(iii) Incompatible design specifications:
Different design specifications result in different schemas. In example 3 in Figure 5, the relationship between **Travelers** and **Bookings** in schema B indicates that a customer can only have one booking at a time, since the cardinality constraint $1 : n$ has been specified. The more realistic situation (that a customer may have several reservations at once) appears in schema C.

Thus far, we have discussed the nature of the interoperability problem and identified the causes and implications of semantic heterogeneity. In the following sections, we present the details of our mechanism for accomplishing interoperability between two components. In order to illustrate our solution we use the framework of FOTA.

4 The Interoperability Context for Semantic Heterogeneity Resolution

Database interoperability refers to the ability to allow partial and controlled sharing of data among autonomous, heterogeneous database components [47]. Due to the complexity and difficulty in achieving interoperability in such systems, we have divided up this task into three subtasks which can be performed (iteratively) during different operational phases in the lifetime of a federation. The approach that we take in implementing each phase is interactive to the extent that the user (e.g., the user of a database component that is engaged in a sharing procedure) or the administrator of the federation has the ability to intervene at any time during each phase. In addition, human input is essential for resolving semantic heterogeneity.

4.1 Operational Phases

When a federation is initially formed or when a new component joins an existing federation, an intelligent *sharing advisor* must first locate all sharable information in each component and store the (partially) identified concepts in a separate repository called the *semantic dictionary*. The semantic dictionary is just another component that can be accessed by every member of the federation. In our approach, sharable information corresponds to all those data objects that a component is willing to *export* to the rest of the federation. Each component has a so-called *export schema* [19] that contains only those objects that are non-private and

Figure 4: Example 2. Semantic heterogeneity in FOTA

thus sharable by other components. All other objects are kept in the component's *private schema*. This is a relatively simple way for each component to exercise control over its database but it suffices for our purpose since the focus here is not on security and access control mechanisms. The sharing advisor tool also responds to requests from database components to locate information that is in structure and content "relevant" to certain information in its own local schema. This first phase of locating and identifying non-local, relevant information is termed *resource discovery and identification* and currently under investigation in a related research project at USC (see [11]). During the second phase, the exact relationship between the requested non-local information and the local information in the schema of the integrating component is determined. This phase is termed *resolution of semantic heterogeneity* and is the focus of our work. After the exact relationship between a foreign object and one or more local objects has been established in phase two, the third phase, called *sharing and transmission* supports efficient access to shared object(s). A picture of the operational phases is given in Figure 6. In the next three sections, we will describe in more detail each of these three phases using the familiar example of FOTA.

4.2 Resource Discovery and Identification

A travel agency, for example, travel agency A located in Los Angeles, needs information on bed & breakfast places in the New England area. A only has bed & breakfast information for California and wants to incorporate this additional information into its own schema so that it can be accessed transparently like any other local information in A's database. Assuming that A has no prior knowledge of where to find bed & breakfast places in New England within the federation, it consults with the intelligent sharing advisor that assists components in discovering relevant, sharable information throughout the federation. The sharing advisor uses the information stored in the semantic dictionary and a set of heuristic rules (see Figure 6); it returns to the component initiating the inquiry those concepts that it considers relevant to the requested information. In our case it locates two possible sources of bed & breakfast places in the New England area: the types **Lodgings**[d], **Inexpensive**, and **Expensive** of travel agency E in Washington, and **Private**

[d]Throughout the rest of this paper, the **boldface** type will be used for type objects, the *italicized* type will indicate function objects.

Figure 5: Example 3. Semantic heterogeneity in FOTA

Accommodations of travel agency D in New York. Figure 7 displays the three (partial) schemas of the travel agencies involved in this identification process.

The goal of identifying relevant information is to retrieve information in other components that is *identical*, *similar*, or *related* to the requested information. Depending upon the amount and kinds of information in which a user is interested, s/he may wish for a large intersection or a small one. Large intersections correspond to concepts that are identical or similar to the concepts in his/her database (e.g., all components of FOTA are in the travel business, hence their databases contain many similar concepts such as **Accommodations** and **Hotels**, **Flights** and **Airtravel**, etc.). Small intersections correspond to related types of information (e.g., a travel agency and an automobile club share a few commonalities like **Travel** and **Trips**, hence their databases have a smaller overlap as far as data objects are concerned), or *completely disjoint* information (e.g., a travel agency and a telephone company will probably have no common concepts in their database schemas).

4.3 Resolution of Semantic Heterogeneity

Having identified two possible sources of information, component A has to find the exact relationship(s) between its own type **Bed & Breakfast**, the types **Lodgings**, **Inexpensive**, and **Expensive** of component E, and **Private Accommodations** of component D in order to determine if and how they can be folded into its conceptual framework. This is the problem of resolving semantic heterogeneity as depicted in Figure 6; it must be addressed before any sharing can take place. Using so-called *meta-functions*, a *local dictionary* or *lexicon* as well as the semantic dictionary mentioned earlier, the meanings of concepts unknown to another component are "derived". Meta-functions return structural information about an object (supertype, subtype(s), properties, etc.). The local lexicon, which is created and updated by each component separately, contains a semantic description of every sharable type object in the database. In order to make the local lexica usable throughout the entire federation, a common knowledge representation is used. Terms in question can be located and compared with each other. The semantic dictionary describes the relationships between terms in the local lexica.

Figure 6: Interoperability in a federation of databases

4.4 Sharing and Transmission

In general, there is a spectrum of the kinds of inter-component sharing that may be desired. At one extreme, a *copy* of the foreign object can be created in the importing database. At the other, a surrogate (i.e., handle or place-holder) for the foreign object can exist in the local database. In the later case, objects such as instances, types, and functions are added to the importing schema at specially created places using local surrogates. Surrogates are essentially references that are used to refer to the original object in the exporting database whenever it is used by the importing database. Thus, local surrogates enable individual components to exchange and use each others' objects in a transparent way without making physical copies of the shared objects. The goal is to place the shared object into the best possible location in the existing schema of the importing component. In addition to sharing type objects with or without instances, it is also possible to share methods and individual instances by themselves. In order to share methods and instances, the importing component must already have the underlying meta-data (i.e. the type(s) that the method or instances belong to) in its schema.

Based upon this sharing mechanism the component that wishes to import the foreign concept will be able to add the (meta-)data representing the concept to its own local schema. Adding (meta-)data to an already existing schema is a two step process. First, conflicts (e.g., naming, modeling, scaling) between the objects of the importing database and the external objects from the other components must be resolved (conflict resolution). Second, these objects must be imported into the new schema as gracefully as possible and unified with already existing local types (unification). Figure 8 shows the final schema of travel agency A after conflict resolution and unification are completed.

5 The Object Database Model

In order for any collaboration to take place among the heterogeneous components of a federation, a common model for describing the sharable data must be established. This model must be semantically expressive enough to capture the intended meanings of conceptual schemas which may reflect several of the kinds of heterogeneity enumerated above. Further, this model must be simple enough so that it can be readily

Figure 7: Three partial conceptual schemas for travel agencies A, E, and D

understood and implemented. The advantage of a simple model is that it can be implemented using a variety of already existing object-oriented database management systems saving both time and effort. To this end, we have chosen to use a **M**inimal **O**bject **D**ata **M**odel (MODM) as the common data model for describing the structure, constraints, and operations for sharable data.

5.1 MODM

MODM is a generic functional object data model, which supports the usual object-based constructs. In particular, it draws upon the essentials of functional database models, such as those proposed in in Daplex [49], Iris [14], and Omega [16]. MODM contains the basic features common to most semantic [1, 21] and object-oriented models [2], such as GemStone [36], *O*2 [30], and Orion [28]. The model supports complex objects (aggregation), type membership (classification), subtype to supertype relationships (generalization), inheritance of stored functions (attributes) from supertype to subtypes, and user-definable functions (methods). Not supported at this point are run-time binding of functions (method override), overloading of operations, constraints (semantic integrity rules) on types and functions, and remote transparency. We expect that MODM will eventually incorporate some of the concepts from [31, 35] for additional support in the unification process.

Some of the advantages of using an object-based common data model include the ability to encapsulate the functionality of shared objects [5], its extensible nature [32], and object uniformity [8]. This last item is especially important for the unification phase where one can ask about the equivalence of actual data-values, types, and operations.

5.2 Relationships Among Objects

Before we present the details of our approach to resolving semantic heterogeneity, we first present an overview of the various relationships that can exist among objects that model the same or similar real-world concepts in different components of a federation [4].

Figure 8: Travel agency A's schema after the importation of "Bed & Breakfast"-like information

5.2.1 Common Concepts

As a direct result of the different causes for schema diversity described above, it may happen that the same concept of an application domain is modeled by different representations R_1 and R_2 in different schemas. Returning to the example in Figure 7 of Section 4, we can see that the concept of "bed & breakfast" as it can be found in travel agency A's schema is also represented by the type **Lodgings** in travel agency E and by the type **Private Accommodations** in travel agency D. In addition to the obvious naming differences, both abstract objects mirror closely related real-world information but use different modeling constructs in their representations. Several types of *semantic relationships* can exist between two representations R_1 and R_2: they may be identical, equivalent, agreeable, or incompatible:

(i) Identical:
R_1 and R_2 are exactly the same. This happens when the same modeling constructs are used, the same perceptions are applied, and no extraneous information enters into the specification. For example, **Reservations** in schema B and **Bookings** in schema D are equal representations for the same real world concept, namely records of services rendered by the travel agency to its customers.

(ii) Equivalent:
R_1 and R_2 are not exactly the same because different modeling constructs have been applied. For example, in Figure 3, the types **Accommodations** in travel agency D, and **Places in Northeastern Region** in travel agency E can be used to model the same information, namely hotels in the Northeastern region of the U.S., but the representation is different. In D's schema the location is explicitly modeled through a separate function called *is_located_in*. In E's schema, the location information is implicit since it is part of the type name.

(iii) Compatible:
R_1 and R_2 are neither identical nor equivalent. However, their representation is not contradictory. For example, **Private Accommodations** and **Resorts** both model the same basic hotel information

(i.e., rooms to rent) but differ in the type of services offered.

(iv) Incompatible:
R_1 and R_2 are contradictory because of inconsistent design specifications or fundamental differences in the underlying information. For example, in Figure 5, the two partial schemas displaying **Travelers** and **Reservations** and **Customers** and **Bookings** are incompatible because of the cardinalities assigned to their respective relationships.

5.2.2 Related Concepts

In addition to common concepts, related concepts arise frequently; we can enumerate the following most commonly used types of interschema (binary) relationships of this kind:

(i) Generalization/Specialization:
Generalization is the result of taking the union of two or more types to produce a higher-level type. In terms of the travel agency example, **Places-to-Stay** of schema A is the generalization of **Resorts & Spas**, **Hotels**, and **Bed & Breakfast**. Specialization is the opposite of generalization.

(ii) Positive Association:
It is impossible to accurately classify all kinds of relationships that can exist between objects. This category includes concepts that are "synonyms" in some context (e.g., **Bed & Breakfast** and **Private Accommodations**, and those that are typically used in the same context (e.g., **Hotel** and **Reservations**).

This list is by no means exhaustive but rather indicative of useful inter-relationships vis-a-vis semantic heterogeneity resolution.

6 A Mechanism for Semantic Heterogeneity Resolution

The fundamental goal of this work is to provide a mechanism to support the resolution of semantic heterogeneity when sharing information among components in a federation. In general, sharing is possible at many different levels of abstraction and granularity, ranging from specific information units (data objects), to meta-data[e], to behavior. For this paper, we limit our investigation to the sharing of type objects, a process we term *type-level* sharing. Sharing of individual instances (*instance-level* sharing) and sharing of behavior or functions (*function-level* sharing) has been addressed in [10] as we mentioned earlier. The findings from this previous research are particularly useful for our research since when sharing a type object, all its instances as well as its stored functions will be shared at the same time. The sharing of computed functions requires more work in the sense that there may be problems with side-effects[f]. As observed in Section 2, most of the previous work on resolving semantic heterogeneity has concentrated in the area of determining *structural equivalence* or *behavioral equivalence* at the schema level. In what follows, we first present a brief overview of these two approaches (Section 6.1). In Section 6.2 we describe our mechanism; it is based on three parts, each addressing a different aspect of the problem of establishing the relationship between type objects in different components. Here, we present a solution to this problem using two federation components only; for this simplified case, we work out a solution that later serves as a basis for the more general case (described in Section 8).

[e] This includes structural schema specifications and semantic integrity constraints.

[f] By side-effects we really mean: (1) any kind of implicit input other than the input argument, and (2) any modifications to the state of the database where the function executes.

6.1 Basic Structural and Behavioral Equivalence

A widely used approach for disambiguating two objects[g] is to determine if they are structurally equivalent (at some level of abstraction). As an example, consider a (local) object, say `:joachim`, of type **Traveler** in travel agency B's database and a foreign object, say `:dennis`, of type **Customer** in travel agency C's database, that are being compared. In this case, where both objects are atomic, the comparison is relatively easy. One can simply apply some sort of "eq" or "equal" semantics. Otherwise, further comparisons are needed. When comparing type objects this includes comparing the type names, the instances of that type, the subtypes of the type, and so forth. In addition, function information such as value types, function names, missing functions, and mapping constraints, for example, can prove useful on structural characteristics. Consider the two abstract objects **Accommodations** of travel agency D and **Places in Northeastern Region** of travel agency E as shown in Figure 3. In this case, the functions *has_name* with value type **Name** and *is_managed_by* with value type **Person** in D's schema and *has_name* with value type **Business-Name** and *owned_by* with value type **Owner-Name** in E's schema suggest a certain similarity between the two concepts. In order to determine a more exact connection, one has to make further comparisons. The more commonalities there are with respect to the above criteria the higher the correlation between **Accommodations** and **Places in Northeastern Region**.

Another approach to determining object equivalence is to look at the operations (i.e., methods) that are defined on the objects in order to establish a behavioral equivalence between them. The general idea behind this approach is to apply all operations that are associated with a given local object, say **Cities**, and compare the results with those obtained when running the same methods on a foreign object, say **Destinations**. Examples of such methods are *sights()*, which takes a city instance as input and returns its sights, or *location()*, which returns the country in which a particular city is located. Behavioral equivalence requires the ability to compare methods and their results when executed in different environments. Thus, its success largely depends on how well a particular federation environment supports the remote execution of procedures. Another problem that remains in this approach is that of deciding when the results of the procedure applications are equal.

6.2 A Three-Pronged Approach to Relative Object Equivalence

In order to determine the relationship between objects within a broader context, we realize that not one single method (such as the structural approach) but a combination of several different approaches taken together is highly promising. While it is nearly impossible to completely automate such a procedure, the following mechanism provides substantially useful functionality. We now describe our three-pronged approach for resolving semantic heterogeneity, viz., determining relative object equivalence.

6.2.1 Meta-Functions

The first aspect of our mechanism is based on the existence of so-called *meta-functions*, which return meta-data information about objects in remote database components[h]. Examples of such functions are *ShowAll-Types()* which returns a list of all types that are part of the export schema, *HasFunctions()* which returns a list of all functions defined on a given type, *HasSubtypes()*, *HasInstances()*, and so forth. The following is a

[g] In this section the term object can refer to a type object as well as an instance object.
[h] This requires the ability to compare methods and their results when executed in different environments.

list of meta-functions:

Meta-Function	Description
ShowAllTypes()	Returns a list of all types that can be shared
HasStoredFunctions(t:Type)	Returns a list of all stored functions defined on type t
HasComputedFunctions(t:Type)	Returns a list of all computed functions defined on type t
HasInstances (t:Type)	Returns a list of all instances defined on user defined type t
HasValue(i:Instance, f:Function)	Returns the value of the stored function f on instance i
HasValueType(f:Function)	Returns the value type for a stored function f
HasDirectSubtypes(t:Type)	Returns a list of all direct subtypes of type t
HasDirectSuperType(t:Type)	Returns the direct supertype of type t

Note that we cannot depend on examining supertypes of an object, since the supertype might not be part of a component's export schema whereas we can always assume that subtypes are. Components that wish to share and exchange information (such as the travel agencies in FOTA) must agree on a common interface (MODM) that can provide the functionality described above.

6.2.2 Local Lexicon

The second aspect provides semantic information about the sharable objects in each component. For this purpose, each component must maintain a local *lexicon* where every type object that is part of its export schema is defined. The common vocabulary in which shared knowledge is represented in a lexicon draws some ideas from declarative knowledge representation forms such as the *Knowledge Representation Language* (KRL) [6], semantic networks [43], and the *Cyc* knowledge base [20]. In our approach, knowledge is represented as a static collection of facts of the simple form:

`<term>` relationship descriptor `<term>`

A term on the left side of a relationship descriptor represents the unknown concept which is described by the term on the right side of a relationship descriptor. The set of descriptors is extensible and specifies the relationships that exist between the two terms. The following is a preliminary list of conceptual relationship descriptors utilized in our mechanism:

R-Descriptors	Meaning
IDENTICAL	Two types are the same
EQUAL	Two types are equivalent
COMPATIBLE	Two type are transformable
KINDOF	Specialization of a type
ASSOC	Positive association between two types
COLLECTIONOF	Collection of related types
INSTANCEOF	Instance of a type
COMMON	Common characteristic of a collection
FEATURE	Descriptive feature of a type
HAS	Property belonging to all instances of a type

For example in FOTA, we may have the following in a local lexicon:

Bed & Breakfast KINDOF **Hotel** A bed & breakfast place is a specialization of hotel

Figure 9: A partial view of three local lexica

”	FEATURE	**economical**	and one of its features is its low price.
Sightseeing	KINDOF	**Entertainment**	Sightseeing is a special form of entertainment
”	COLLECTIONOF	**Trips**	that is a collection of trips
”	HAS	**Place-to-visit**	to places
”	FEATURE	**Pleasure**	and is meant to be enjoyable.
MexicoFlights	COLLECTIONOF	**Flight**	MexicoFlights is a collection of flights
”	COMMON	**Destination**	with the common destination of Mexico.

The terms that are used to describe the unknown concept are taken from a dynamic list that characterizes commonalities in a federation. Since interoperability only makes sense among components that model similar or related information, it is reasonable to expect a common understanding of a minimal set of concepts taken from the application domain. In case of FOTA a subset of commonly understood concepts could be:

$$\mathcal{C}_{sub} = \{Airlines,\ Arrival,\ Booking,\ Cruises,\ Customer,\ Destination,\ Departure,\ Entertainment,\ Fare,\ Flight,\ Hotel,\ Public\_Transportation,\ Rental\_Car\_Companies,\ Trains,\ Trips\}$$

The underlying idea of the lexicon is to represent the real-world meaning of all shared terms in order to complement the results of the meta-functions. In some cases, it is not possible to "derive" the meaning of a term only by looking at its structure (for example, **Pleasure Cruises** and **Flights** both have functions containing departure and arrival information but the types themselves are not related). By the same token, the real-world meaning by itself might not be enough to correctly integrate a term into another type hierarchy (for example, two types **Fare** and **Airfare** that represent the cost of tickets but store their prices in different denominations cannot simply be merged into a new type). Thus, by using meta-functions together with local lexica, we are able to achieve a higher degree of confidence in the correctness of our mechanism when integrating objects. An example of the partial contents of three local lexica is given in Figure 9.

Figure 10: Sharing architecture and the various interactions among its components

6.2.3 Semantic Dictionary

The third module is the semantic dictionary, which is created and maintained by the sharing advisor. There is an important connection between local lexica and the semantic dictionary. Each local lexicon describes the meaning of all type objects that are part of the component's export schema to which the lexicon belongs. Local lexica contain only semantic information and no knowledge about any relationships among its entries. This kind information is provided by the semantic dictionary which contains partial knowledge about the relationships between all the terms in the local lexica in the federation. An overview of our architecture displaying the various interactions between the different parts is shown in Figure 10.

6.3 A Strategy for Resolving Object Relationships

The basic problem addressed by the semantic heterogeneity resolution mechanism may be expressed, without loss of generality, as: given two objects, a local and a foreign one, return the relationship that exists between the two. Specifically, our strategy is based on structural knowledge (meta-functions) and the (known) relationships that exist between keywords and the two objects in questions (local lexicon, semantic dictionary). One characteristic of our approach is that the majority of user input occurs before the resolution step is performed (i.e., when selecting the set of keywords and creating the local lexicon) rather than during. As a final note, we can make the following observation on the use of meta-functions. All information about the structure of a type object is provided through meta-functions: an approach that can be viewed as a variation to the usual paradigm of behavior encapsulation in object-oriented programming languages. Rather than encapsulating behavior, meta-functions "encapsulate" the structure of a type object. The advantage of this approach is that meta-functions that are essentially computed or foreign functions in our data model can be part of each component's schema without modifications to the underlying architecture.

7 The Unification of Remote and Local Information

Thus far we have discussed the nature of the schema integration problem and identified the various causes of and a solution to the heterogeneity problem. In this section we describe the individual activities that a component must go through when importing a type object. As mentioned before, there are two steps that must be performed when adding imported (meta-)data to an already existing schema. First, once conflicts (e.g., naming, modeling, scaling) between a source and a target schema are detected, these conflicts must be resolved so that the unification of the foreign type(s) with the corresponding local objects is possible (conflict resolution). Second, the foreign object(s) must be imported into the local schema as gracefully and naturally as possible (unification). In the context of this research, "schema" refers to a collection of type objects that are connected through the usual object-based constructs. As noted above, we consider here only the importation of type objects or collections thereof.

7.1 Conflict Resolution

The goal of this activity is to resolve inconsistencies between the imported type(s) and the target schema before the unification step. However, automatic conflict resolution is in general infeasible. Sometimes conflicts cannot be resolved because they arose as a result of some basic inconsistencies. In cases where automatic resolution is not possible, the conflicts are reported to the users who must guide the unification mechanism in the process. The following specific activities are performed during conflict resolution:

- Operations on atomic data values: This is an attempt to resolve type (iv) heterogeneities (low-level data format. For example, a conversion of fare prices that are represented in English pounds into U.S. dollars (this might also affect the respective value types of functions associated with fare prices).

- Renaming: This activity addresses the problem of *homonyms* (where the same name is used for two different concepts) and *synonyms* (in which the same concept is described by two or more names). As an example for homonyms, consider the following scenario. Two components A and B of FOTA want to share fare information. Both schemas include an object named **Fare-Prices**. However, **Fare-Prices** in schema A includes airport and sales taxes whereas in schema B it represents the "true" fare price without any taxes added. It is obvious that merging the two types would result in a problem. Therefore, one of the two types must be renamed before unification can take place in order to reflect the differences in their representations. As an example of synonyms, consider two schemas representing customer information, **Clients** and **Customers**, where both types contain the same data. In this case, keeping two distinct types in the integrated schema would result in modeling a single object by means of two different types.

Additional conflicts are resolved during the following unification phase.

7.2 Unification

At this point, the foreign object(s) can be unified with the corresponding local object(s). If necessary the target schema must be restructured to achieve a result that is (1) complete, (2) minimal, and (3) understandable. Complete since the new, integrated schema must contain all concepts that were present before the unification process took place. Minimal since concepts should only be represented once, and understandable since the integrated schema should be easy to understand for the end user.

Upon importing the (meta-)data, structural conflicts with existing types in the component's local type hierarchy may arise. Several possibilities exist, and we demonstrate our approach with the help of several sample scenarios from FOTA. These scenarios differ in the complexity of the schemas to be integrated, starting with the most simple one: the importation of a single foreign type object. The second scenario examines the cases when a component wishes to import one or more types that are inter-related. The types to be unified are usually part of a more complex type hierarchy, hence we also need to be concerned with inheritance issues. The last scenario is a special case of the previous one and focuses on the relationship(s) between the objects to be unified. In this scenario, it is possible to construct the situation in which a component wishes to import objects that are connected in a complex manner involving one or more types not present in the local schema. In assembling this framework, we want to point out the relevance of previous work in this area described in [4, 22, 24, 39].

Below, we assume the following naming convention. Let C_{imp} denote an importing component of FOTA, C_{exp} denote an exporting component of FOTA, L denote a local object belonging to C_{imp}, and F denote a foreign object belonging to C_{exp}.

(i) In the first scenario, we assume that a component C_{imp} wishes to import a single type object F from component C_{exp}. In increasing order of generality, this scenario can be decomposed into the following three cases.

  (a) *F does not exist in C_{imp}'s schema.* For example, travel agency D wishes to import the object type **Museums** from travel agency E. Since **Museums** does not exist in D's schema, it can be added without further modifications. The designated place for the new type is either as a subtype of "root" or "system" (i.e., the highest place in the type hierarchy), or as a subtype of some other user-specified local type. It is important to note that "adding" a type requires some additional work in the sense that the value types for the new functions do not always exist in the target schema and may have to be imported as well.

  (b) *F is (semantically) equivalent to some local object L in C_{imp}'s schema.* In this case we make F a subtype of L and add the necessary functions to both L as well as F. For example, travel agency A wishes to import the object type **Private Accommodations** from travel agency D (this scenario is depicted in Figure 7). An equivalent type, namely **Bed & Breakfast** already exists in A's schema. Thus, the imported type **Private Accommodations** is added as a subtype of **Bed & Breakfast** and all its functions are created for both the supertype as well as the subtype. Note that "subsetting" is a unification practice that is used by most methodologies [4]. In fact, it is considered to be the basis for accommodating multiple user perspectives on comparable types.

  The case where F is exactly identical to L (e.g., structurally as well as semantically), is merely a simplification and only requires the importation of the type instances in which C_{imp} is interested (this is essentially a type merge as mentioned earlier).

  (c) *F is related to an object L in C_{imp}'s schema.* In the case when L and F are similar in their semantic meaning but not equivalent, i.e., they are identical with respect to some functions and different with respect to others, a new supertype is created that contains only the identical functions of L and F. The functions in which L and F differ are associated with two new subtypes which inherit the functions common to both L and F. Together, the new supertype and its two subtypes contain the same information as L and F before the unification. This method was proposed in [9]. In terms of FOTA, the following example as shown in Figure 11 best illustrates this method. Assume that travel agency D is about to integrate **Hotels in Northeastern US** into its own schema. It has determined that **Hotels in the Northeastern US** is related to its own **Accommodations in New England**. Although there is considerable overlap between the two types (e.g., both contain hotel information, both cover a similar geographic area), there are still enough differences to prevent D from simply "merging" the two types. Using the above method, a new supertype **Accommodations** is created that has two subtype **Northeastern US** and **New-England**. Together, all three types contain the same information as the two original types.

(ii) In the second scenario C_{imp} wishes to import (meta-)data consisting of several, inter-related objects from C_{exp}. For simplicity, we use only two different foreign type objects, namely F_1 and F_2, but the

Figure 11: Integration of two related objects

following discussion can be adapted to situations where more than two foreign type objects are involved in the unification process. As before, we can distinguish the following cases, based on their degree of generality:

(a) *F_1 and F_2 do not exist in C_{imp}'s schema.* Here the type objects and all instances as required by C_{imp} are added. For example, travel agency B that has no train information in its schema so far wishes to import two related types called **Trains** and **Train-Fares** from travel agency C.

(b) *Either F_1 or F_2 is (semantically) equivalent to some object L in C_{imp}'s schema.* If we assume that foreign type F_1 is equivalent to a local type, say L_1, several things need to be done. First a new subtype of L_1 is created, in order to hold the imported instances from F_1. Then F_2 is added as L_2 to C_{imp}'s local schema, including all of its instances. Finally, new functions relating L_1's subtype and L_2 are created. For example, travel agency D is importing the type **Airtravel** and the value type **Price** of function *has_price* from travel agency C (see Figure 4). Since **Airtravel** is related to C's **Flights** it is added as a subtype of it. Then the type **Price** is added to the newly imported **Airtravel** just as in the original schema of travel agency C.

In the case where both types F_1 and F_2 are present in C_{imp}'s schema, all foreign instances of F_1 and F_2 can be imported into the already existing local types L_1 and L_2 (type merge).

(c) *Either F_1 or F_2 is related to some object L in C_{imp}'s schema.* Assume that F_1 is related to L. In this case, which is similar to case (c) of the first scenario, the same method applies. The functions that are common to both F_1 and L are associated with a new supertype, which has two subtypes containing the functions that distinguished F_1 and L. Since F_2 was related to the original type F_1, it will also be related to the new supertype to which it is added as a stored function. For example, travel agency D is importing the two types **Sightseeing** and **Cities** (containing the cities where the sights are) from travel agency C. C's type **Sightseeing** is related to D's local type **Entertainment**. Following the method outlined above, a new supertype is created by travel agency D, say **Things-to-do**, which contains all the information common to

Figure 12: Example 1. Integration of several inter-related objects

D's **Entertainment** and C's **Sightseeing**. The new supertype **Things-to-do** has two subtypes, namely **Entertainment** and **Sightseeing**, that reflect the differences between the original types. **Cities**, which is a stored function of both subtypes, is connected to **Things-to-do** via a stored function called *has_location* (see Figure 12).

(d) *Both F_1 and F_2 are related to objects L_1 and L_2 in C_{imp}'s schema.* For example, travel agency E with types **Places in Northeastern Region** and **Owner-Name** wants to import the types **Accommodations** and **Person** from travel agency D (see Figure 3). In this case, both types **Accommodations** and **Person** are imported separately. If the relationship between the two is important for the importing component, the appropriate function(s), in this case *is_managed_by* can be added afterwards.

(iii) In the third scenario, consider the case where component C_{imp} wishes to import types from C_{exp} whose relationships are modeled differently then the corresponding type relationships in C_{imp}'s local schema. For example, L_1 and L_2 are related through a ternary relationship that includes an additional type L_3 whereas the corresponding foreign types, F_1 and F_2, are related directly through functions and their inverses. Applying the method of "subsetting", two new subtypes for F_1 and F_2 are necessary. Then, for each instance pair that C_{imp} imports, a new instance for L_3 is created, which relates the imported objects. As an example in FOTA, consider the following situation depicted in Figure 13. Travel agency C models its "flight-fare" information through a ternary relationship consisting of the types **Flights**, **Flight-Fare-Combinations**, and **Fare-Types**. In order to obtain fare information on charter flights from Miami to the Florida Keys, it decided to import this information from travel agency B. B uses two types **Airtravel** and **Price** and a function *has_price* to model the relationship between flights and fares. Using the procedure outlined above, two new subtypes for **Flights** and **Fare-Types** are created, namely **Southern-Flights** and **Southern-Fare-Types** respectively (to represent the fact that they contain information about the southern parts of the U.S.). Further, for every related pair of objects in the new subtypes, a new object is created in C's **Flight-Fare-Combinations** relating each flight

Figure 13: Example 2. Integration of several inter-related objects

from **Southern-Flights** to its corresponding price object in **Southern-Fare-Types**.

These three scenarios are an attempt to examine the most important sharing situations that arise during the life-time of FOTA. We have only examined the importation and unification of single type objects or pairs of related type objects. However, the mechanisms presented in this section can be extended to accommodate the unification of three or more inter-related type objects. Some of these cases will be studied during subsequent research in this area. Although it is difficult to propose a useful measure of the "completeness" of this enumeration, we shall attempt to use the scenarios presented here as a foundation for our work, and argue that they are sufficient to prove the feasibility of this approach.

8 Concluding Remarks

We have presented an approach and mechanism for resolving semantic heterogeneity in the context of a federation of autonomous database system components. For this mechanism to operate effectively, each participating component must agree to meet two principal conditions: First, the MODM data model must be supported at the federation interface, including the sharing meta-functions. Second, a local lexicon must be provided, wherein a component describes the meaning of the (type) objects it is willing to share with other components in federation. These objects must be described using the conceptual relationship descriptors supported by our mechanism.

The approach to semantic heterogeneity presented in this paper is based on the following three "services" that are provided by the federation to its components: the sharing advisor, the disambiguation algorithm, and the unification tool. When a new component initially joins the federation it must first register, and invoke the sharing advisor. The sharing advisor enters the data that the component is willing to share into the semantic dictionary so that it can be used by other components in the federation. Assuming that the new component has met the two basic conditions above, it is then ready to participate in the exchange of information. Sharing takes place on a component-pairwise basis when the sharing advisor has located the sources of relevant information. The importing component selects those relevant foreign objects that it wants to integrate into its local framework. Given a foreign object, a related local object and the relationship between the two, the unification tool places place the foreign object (including its instances and stored functions) into the appropriate place in the local meta-data framework (type hierarchy). At this point, the unification is complete and the newly imported object can be used by the remote component.

We note that it is not at present possible to completely automate the tasks of the sharing advisor, particularly in supporting new components. In consequence, in practice one or more humans will likely be required to assist. In order to facilitate the establishment of new federations, we employ a notion termed a "federation seed", which is application environment dependent and contains an initial set of terms that can be used to describe unknown concepts in the local lexicon of each component. This initial set of terms describes general information from the application domain, and will evolve and grow with time to accommodate additional, more complex concepts within a given federation. Such seeds can be provided for particular domains, e.g., the travel industry or genetic information.

The result of this research may have both direct and practical impact on information sharing among heterogeneous databases, specifically in the following areas:

- *Framework:* We have presented a framework for accommodating semantic heterogeneity in interoperable object-based database systems. We specifically use an functional object-based data model (extended with meta-functions) for describing the sharable data as well as their relationships to the real-world concepts they represent.

- *Architecture:* We have introduced an architecture and experimental system for resolving semantic heterogeneity. Our system is based on the interaction between meta-functions that provide structural information, local lexica that provide semantic information, and the semantic dictionary that provides incomplete knowledge about the relationships between the concepts in the local lexica. Relationships are described using a set of fundamental descriptors, which is extensible.

- *Existing Components and Autonomy:* Throughout this paper we have paid careful attention to limiting required modifications to existing DBMS software and conceptual schemas. As a result, our approach requires no modification to the query processor or any other aspect of the local architecture. Other than the basic requirements of an MODM interface and support of a local lexicon, each component retains autonomy over its database. Furthermore, through the export schema it can specify at any given time which objects are sharable and which objects should remain private.

An experimental prototype of our mechanism is currently under development. This initial prototype, which we are using to demonstrate, evaluate, and refine our approach, is based upon a testbed consisting of Omega [16] DBMS components. Omega was chosen for two reasons. First, by using an exiting DBMS we were able to focus our attention on implementing the resolution and sharing mechanism. Second, the Omega database model contains most of the modeling constructs needed to implement MODM.

Acknowledgements

This research was supported in part by NSF grant IRI-9021028. The authors are particularly grateful to the referees for the very useful recommendations on the content and organization of this paper. We would also like to acknowledge the very useful insights of Mike Papazoglou and Hans Schek, as well as other researchers involved in Remote-Exchange, including K.J. Byeon, Doug Fang, Shahram Ghandeharizadeh, and Antonio Si.

References

[1] H. Afsarmanesh and D. McLeod. The 3DIS: An Extensible, Object-Oriented Information Management Environment. *ACM Transactions on Office Information Systems*, 7:339–377, October 1989.

[2] M. Atkinson, et al. The Object-Oriented Database System Manifesto. In *Proceedings of the 1st Intl. Conf. on Deductive and Object-Oriented Databases*. Kyoto, Japan, December 1989.

[3] C. Batini and M. Lenzerini. A Methodology for Data Schema Integration in the Entity Relationship Model. *IEEE Transactions on Software Engineering*, 10(6):650–664, 1984.

[4] C. Batini, M. Lenzerini, and S. Navathe. A Comparative Analysis of Methodologies of Database Schema Integration. *ACM Computing Surveys*, 18(4):323–364, 1986.

[5] E. Bertino, G. Pelagatti, and L. Sbattella. An Object-Oriented Approach to the Interconnection of Heterogenous Databases. In *Proceedings of the Workshop on Heterogenous Databases*. NSF, December 1989.

[6] D. G. Bobrow and T. Winograd. An Overview of KRL, a Knowledge Representation Language. *Cognitive Science*, 1(1):10–29, 1977.

[7] S. Ceri and G. Pelagatti. *Distributed Databases: Principles and Systems*. McGraw Hill, 1984.

[8] T. Connors and P. Lyngbaek. Providing Uniform Access to Heterogeneous Information Bases. In *Proceedings of 2nd International Conference on Object Oriented Database Systems*. Bad Münster am Stein Ebernburg, Germany, September 1988.

[9] U. Dayal and H. Hwang. View Definition and Generalization for Database Integration in Multibase: A System for Heterogeneous Distributed Databases. *IEEE Transactions on Software Engineering*, 10(6):628–644, 1984.

[10] D. Fang, J. Hammer, and D. McLeod. An Approach to Behavior Sharing in Federated Database Systems. In M.T. Özsu, U. Dayal, and P. Valduriez, editors, *Distributed Object Management*, pages 334–346. Morgan Kaufman, 1993.

[11] D. Fang, J. Hammer, D. McLeod, and A. Si. Remote-Exchange: An Approach to Controlled Sharing among Autonomous, Heterogenous Database Systems. In *Proceedings of the IEEE Spring Compcon*. IEEE, San Francisco, February 1991.

[12] P. Fankhauser and E. Neuhold. Knowledge Based Integration of Heterogeneous Databases. Technical report, Technische Hochschule Darmstadt, 1992.

[13] A. Ferrier and C. Stangret. Heterogeneity in the Distributed Database Mangagement System SIRIUS-DELTA. In *Proceedings of the International Conference on Very Large Databases*. VLDB Endowment, 1983.

[14] D. Fishman, D. Beech, H. Cate, E. Chow, T. Connors, T. Davis, N. Derrett, C. Hoch, W. Kent, P. Lyngbaek, B. Mahbod, M. Neimat, T. Ryan, and M. Shan. Iris: An Object-Oriented Database Management System. *ACM Transactions on Office Information Systems*, 5(1):48–69, January 1987.

[15] K. Frenkel. The Human Genome Project and Informatics. *Communications of the ACM*, 34(11):41–51, 1991.

[16] S. Ghandeharizadeh, et al. Design and Implementation of OMEGA Object-based System. Technical Report USC-CS, Computer Science Department, University of Southern California, Los Angeles CA 90089-0781, September 1991.

[17] M. Hammer and D. McLeod. On Database Management System Architecture. In *Infotech State of the Art Report: Data Design*, volume 8 of *Infotech State of the Art Reports*, pages 177–202. Pergamon Infotech Limited, Maidenhead, United Kingdom, 1980.

[18] S. Hayne and S. Ram. Multi-User View Integration System (MUVIS): An Expert System for View Integration. In *Proceedings of the 6th International Conference on Data Engineering*. IEEE, February 1990.

[19] D. Heimbigner and D. McLeod. A Federated Architecture for Information Systems. *ACM Transactions on Office Information Systems*, 3(3):253–278, July 1985.

[20] M. Huhns, N. Jacobs, T. Ksiezyk, W. Shen, M. Singh, and P. Cannata. Enterprise Information Modeling and Model Integration in Carnot. Technical Report Carnot-128-92, MCC, 1992.

[21] R. Hull and R. King. Semantic Database Modeling: Survey, Applications, and Research Issues. *ACM Computing Surveys*, 19(3):201–260, September 1987.

[22] R. Hull and S. Widijojo. ILOG: Specificational Language for Generating OIDs. Technical Report RJ3325, ISI, February 1990.

[23] G. Jacobsen, G. Piatetsky-Shapiro, C. Lafond, M. Rajinikanth, and J. Hernandez. CALIDA: A Knowledge-Based System for Integrating Multiple Heterogeneous Databases. In *Proceedings of the 3rd International Conference on Data and Knowledge Bases*, pages 3–18, June 1988.

[24] R. Katz and N. Goodman. View Processing in Multibase – A Heterogeneous Database System. In *An Entity-Relationship Approach to Information Modelling and Analysis*, pages 259–280. ER Institute, 1981.

[25] W. Kent. The Many Forms of a Single Fact. In *Proceedings of the IEEE Spring Compcon*. IEEE, February 1989.

[26] W. Kent. Solving Domain Mismatch Problems with an Object-Oriented Database Programming Language. In *Proceedings of the International Conference on Very Large Databases*, pages 147–160. IEEE, September 1991.

[27] W. Kent, R. Ahmed, J. Albert, M. Ketabchi, and M. Shan. Object Identification in Multidatabase Systems. Technical report, Hewlett-Packard Laboratories, 1992.

[28] W. Kim, J. Banerjee, H. T. Chou, J. F. Garza, and D. Woelk. Composite Object Support in an Object-Oriented Database System. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 118–125, 1987.

[29] J. Larson, S.B. Navathe, and R. Elmasri. A Theory of Attribute Equivalence and its Applications to Schema Integration. *IEEE Transactions on Software Engineering*, 15(4):449–463, April 1989.

[30] C. Lecluse, P. Richard, and F. Velez. O_2, an Object-Oriented Data Model. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Chicago, Ill., June 1988. ACM SIGMOD.

[31] Q. Li and D. McLeod. Object Flavor Evolution in an Object-Oriented Database System. In *Proceedings of the Conference on Office Information System*. ACM, March 1988.

[32] V. Linnemann, et al. Design and Implementation of an Extensible Database Management System Supporting User Defined Data Types and Functions. In *Proceedings of the International Conference on Very Large Databases*, pages 294–305, Los Angeles, Ca., 1988.

[33] W. Litwin. An Overview of the Multidatabase System MRSDM. In *Proc. of the ACM National Conference*, pages 495–504. ACM, October 1985.

[34] W. Litwin and A. Abdellatif. Multidatabase Interoperability. *IEEE Computer*, 19(12):10–18, December 1986.

[35] P. Lyngbaek and D. McLeod. Object Management in Distributed Information Systems. *ACM Transactions on Office Information Systems*, 2(2):96–122, April 1984.

[36] D. Maier, J. Stein, A. Otis, and A. Purdy. Development of an Object-Oriented DBMS. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 472–482. ACM, 1986.

[37] F. Manola, S. Heiler, D. Georgakopoulos, M. Hornick, and M. Brodie. Distributed Object Management. *International Journal of Intelligent & Cooperative Information Systems*, 1(1):5–42, 1992.

[38] A. Mehta, J. Geller, Y. Perl, and P. Fankhauser. Computing Access Relevance to Support Path-Method Generation in Interoperable Multi-OODB. In *Proceedings of the International Conference on Very Large Databases*, pages 119–139. IEEE, August 1992.

[39] A. Motro. Superviews: Virtual Integration of Multiple Databases. *IEEE Transactions on Software Engineering*, 13(7), July 1987.

[40] A. Motro and P. Buneman. Constructing Superviews. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Ann Arbor, Mich., April 1981. ACM SIGMOD.

[41] S. B. Navathe, R. ElMasri, and J. Larson. Integrating User Views in Database Design. *IEEE Computer*, 19(1):50–62, 1986.

[42] M. Papazoglou, S. Laufmann, and T. Sellis. An Organizational Framework for Cooperating Intelligent Information Systems. *International Journal of Intelligent & Cooperative Information Systems*, 1(1):169–202, 1992.

[43] B. Raphael. A Computer Program for Semantic Information Retrieval. In M. Minsky, editor, *Semantic Information Processing*. MIT Press, Cambridge, Mass., 1968.

[44] M. Rusinkiewitz, R. Elmasri, B. Czejdo, D. Georakopoulous, G. Karabatis, A. Jamoussi, K. Loa, and Y. Li. OMNIBASE: Design and Implementation of a Multidatabase System. In *Proceedings of the 1st Annual Symposium in Parallel and Distributed Processing*, pages 162–169. IEEE, May 1989.

[45] A. Savasere, A. Sheth, S. Gala, S. Navathe, and H. Marcus. On Applying Classification to Schema Integration. In *Proceedings of IEEE 1st International Workshop on Interoperability in Multidatabase Systems*, pages 258–261. Kyoto, Japan, April 1991.

[46] M.H. Scholl, C. Laasch, and M. Tresch. Update Views in Object-Oriented Databases. In *Proceedings of the 2nd International Conference on Distributed Object-Oriented Databases*, December 1991.

[47] A. Sheth and J. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[48] A. Sheth, J. Larson, A. Cornelio, and S. B. Navathe. A Tool for Integrating Conceptual Schemata and User Views. In *Proceedings of the 4th International Conference on Data Engineering*, pages 176–183. IEEE, February 1988.

[49] D. Shipman. The Functional Data Model and the Data Language DAPLEX. *ACM Transactions on Database Systems*, 2(3):140–173, March 1981.

[50] A. Silberschatz, M. Stonebraker, and J. Ullman. Database Systems: Achievements and Opportunities. *ACM Sigmod Record*, 19(4):6–23, December 1990.

[51] J. Smith, P. Bernstein, U. Dayal, N. Goodman, T. Landers, K. Lin, and E. Wong. Multibase: Integrating Heterogeneous Distributed Database Systems. In *Proceedings of the National Computer Conference*, pages 487–499. AFIPS, June 1981.

[52] T. Templeton, et al. Mermaid: A Front–End to Distributed Heterogenous Databases. In *Proceedings Intl' Conf. on Data Engineering*, pages 695–708. IEEE, 1987.