The Identification and Resolution of Semantic Heterogeneity in Multidatabase Systems

Douglas Fang Joachim Hammer Dennis McLeod

Computer Science Department
University of Southern California
Los Angeles, CA 90089–0782
USA

Abstract

This paper describes several aspects of the **Remote–Exchange** project at USC, which focuses on the controlled sharing and exchange of information among autonomous, heterogeneous database systems. The spectrum of heterogeneity which may exist among the components in a federation of database systems is examined, and an approach to accommodating such heterogeneity is described. An overview of the **Remote–Exchange** experimental system is provided.

1 Introduction

Consider an environment 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. We shall refer to this as the (interconnected) autonomous heterogeneous database environment. Such environments are extremely common in various application domains, including office information systems, computer-integrated manufacturing systems (with computer-aided design as a subset), personal computing, business and financial computing, and scientific research information bases. The trend towards decentralization of computing that has occurred over the past decade has accentuated the need for effective principles, techniques, and mechanisms to support the information sharing and exchange among the component data/knowledge base systems, while maximally retaining autonomy for the components.

Traditional research on "distributed databases" (see, e.g., [CP84]) assumed a common, integrated database specification (conceptual database schema). While some of the research results obtained in this general area of endeavor are applicable in the autonomous heterogeneous database environment, such approaches generally assume a single conceptual database which is physically distributed. Work on "multi-databases", "superviews", and "virtual databases" has stressed the need to provide a unified, perhaps partial, global view of a collection of existing databases [DH84, LA86, MB81]. Techniques for database integration [BLN86], which are often primarily considered for the design of a single database system based upon a number of application subsystems, can also be brought to bear on the problem of partially integrating existing heterogeneous databases. Notably, techniques for multi-databases and database integration all focus on conceptual schema level diversity. "Federated database" architectural issues and techniques for information sharing at the conceptual schema level have also been specifically addressed in the interconnected, autonomous database environment [HM85, LM84].

In this paper, we examine the **Remote–Exchange** project at USC. A major focus of this project is to devise and experimentally implement techniques and mechanisms to support the controlled sharing of information among a collection of autonomous, heterogeneous database systems. We shall not attempt here to examine the thrusts of this research project in detail, but rather examine three of its principal distinguishing aspects.

First, the **Remote–Exchange** approach adopts a more general view of sharing among autonomous database systems than previously supported. In particular, we examine information sharing and coordination beyond the conceptual schema level; we note that sharing may be at a number of levels of abstraction and granularity, ranging from specific information units (data objects), to meta-data (structural schema specifications and semantic integrity constraints), to behavior (operations), to database model and supporting system (e.g., database management system). Our choice of an object-based common data model is a key to accommodating this sharing. Second, objects from external database systems are manipulated transparently by user and application level processes of the local database system. Here, integration of the *remote* objects into the local database system plays a vital role, as does the ability to compare objects at various levels of granularity and abstraction. Third, we employ the notion of an "intelligent" advisor to assist non-expert database system users in establishing and refining sharing patterns with external database systems.

2 The Spectrum of Heterogeneity

As a basis for our analysis of and approach to the various kinds of diversity that may exist in the interconnected autonomous database environment, consider a federation of components, which are individual data/knowledge base systems. A component contains both structural and behavioral information. Structural information includes information units (viz., objects) and their inter-relationships, at various levels of abstraction; these represent both specific data facts and higher-level meta-data specifications. Behavioral information includes operations to manipulate information units and their inter-relationships; these operations can be invoked either by users or by the system itself; also included are services that a component may provide to itself or to other components in the federation.

In this context, we can consider a spectrum of heterogeneity. That is, the heterogeneity in the federation may be at various levels of abstraction:

- **Meta–data language (conceptual database model):** The components may use different collections of and techniques for combining the structures, constraints, and operations used to describe data.

- **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).

- **Object comparability (database):** 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 [Ken89]. 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 (naming).

- **Data form / 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 or description).

- **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.

3 Accommodating Heterogeneity in a Federation

The **Remote–Exchange** project focuses primarily on the middle three kinds of heterogeneity described above: meta-data specification, object comparability, and data form/format. A cornerstone of our approach to sharing in such a heterogeneous environment is the capability to incorporate remote objects directly into a component's local database. This allows users to use the local data manipulation facilities to access remote objects, and to combine local and remote objects. This results in substantial location transparency[1] .

[1] In some sense this can be viewed as an aspect of data independence.

Operationally, we view the steps in establishing sharing between components as first, the *discovery* of relevant information from remote components for importation into the local database. Having found these remote objects, the next step is to *integrate* them with the local component's objects. Both these processes will need to compare objects at different levels of granularity and abstraction, thereby requiring some notion of *object equivalence*. Finally, remote objects along with local objects are manipulated using the component's local data manipulation language/mechanism. Below we explain specific issues involved in this framework and the **Remote–Exchange** mechanisms which address them.

3.1 Object–Based Common Data Model

In order for any sharing to take place among heterogeneous components, some common model for describing the shared data must be established[2]. This model must be *semantically expressive* in order to capture the intended meanings of conceptual schemas which may have been designed from different perspectives using different models. To this end, we have chosen a Kernel Object Data Model (KODM) as the common data model for describing the structures, constraints, and operations on the shared data.

At the same time, due to autonomy issues, a component may wish to exercise control over information that it exports by isolating the sharing to a fixed set of operations. This situation can also occur when the component does not support a general purpose data manipulation language/mechanism (e.g., a file system). KODM addresses this problem by allowing components to *encapsulate* the functionality of the shared objects.

Another benefit of using an object-based common data model is that it is *extensible*. In an environment where dynamic changes in component sharing patterns occur naturally over time, this capability is indispensable. It allows components to gracefully adapt to new and unplanned operations on shared data.

A final aspect of KODM that we exploit in this project is *object uniformity*. Meta-data, specific data, and operations are represented uniformly as objects. This allows the unification of various concepts (e.g., object equivalence), as described in what follows.

[2]Another alternative would be to deal with the heterogeneity on a component pairwise basis which would require n^2 translators. We feel that the scaling problems associated with the translators renders this approach impractical in general.

3.2 Remote Database Transparency

Traditionally, database management systems (DBMSs) support logically centralized repositories of information. Multiple users access this information in a uniform, controlled manner using tools provided by the DBMS. With the advent of networks, this basic architecture has been extended to support physically distributed databases, and to accommodate the client/server model. In the latter, users (clients) are physically distributed among different machines. Yet, the fundamental asymmetry in the client/server relationship remains. Data is stored and managed centrally. The servers also dictate the interface users must use for accessing and manipulating the data.

In the environment we are considering, the users may once again be physically distributed, however, they also maintain their own local database managed by their own local DBMS. These local databases may be very large or quite small. This results in two major issues:

1. The data is no longer managed by a single centralized agent.

2. Each local DBMS may have its own interface for accessing and manipulating data.

Our approach allows the incorporation of objects from remote databases into a user's local database. In so doing, remote objects should appear transparent to users of the local database. This means that the user manipulates remote data the same way s/he manipulates local data. In order to achieve this transparency, we exploit the *encapsulation* and *object uniformity* features of KODM to form a generalized framework for the execution of (remote) operations [Sha89].

In what immediately follows, we discuss sharing of data and of behavior. We then describe a unified mechanism for handling remote methods which supports both kinds of sharing.

3.2.1 Sharing Data

In the **Remote–Exchange** environment, there is a spectrum of kinds of object sharing that may be desired. At one extreme, a *copy* of the remote object can be made in the importing local database. At the other extreme, a *handle* representing the remote object can exist in the local database. References to the remote handle are retrieved for each reference (as needed). In between these two extremes, there are varying degrees of consistency (e.g., periodic updates).

Remote–Exchange supports a large range of sharing patterns. To illustrate our approach we consider here a critical kind of sharing: sharing by *reference*. In order to achieve sharing of this kind, we require the following:

- **Local handles for remote objects**: Local surrogate objects correspond to the remote instances. These local surrogates can serve as handles for accessing the remote instances. Among other things, surrogate objects must contain the object identifier (OID) of the remote object. This mechanism obviates the need for maintaining a global OID space. Each component is free to independently use its own OID generation algorithm for creating objects.

- **Remote methods on (meta)data using RPC**: Since in KODM the actual state of an instance is *encapsulated* by the methods that can be performed on it, all that is needed to obtain the state of the remote object is the ability to remotely execute the methods which encapsulate (i.e., serve as the interface for) that object. Use of the remote procedure call (RPC) paradigm supports this requirement. (Note that both "class" objects and "instance" objects can be shared, since class objects are also viewed as "instances" of the class "class". They are encapsulated by a set of methods which can be executed similarly using the RPC mechanism.)

- **Dynamic binding and overloading of methods**: This feature of object-based models allows us to re-define methods on the surrogate local objects so that they invoke the corresponding RPC, instead of erroneously trying to execute the method locally.

3.2.2 Sharing Behavior

In addition to simply sharing "data" objects, it is also possible to share methods. In effect, components will then have access to services which their own local systems do not support. The key to sharing methods lies in the interpretation and resolution of parameters to and from the method (i.e., the method's signature). In particular, a given parameter can either be an actual literal value (e.g., string or integer) or an OID. The first case of literals is simple and is completely handled by existing RPC implementations. The second case, however, is more common in our database environment. We plan to to use a "call-back" mechanism which allows the remote method to retrieve any relevant state it needs to service the request by "calling back" the requester and obtaining the necessary state information.

3.2.3 Generalized Execution of Remote Methods

In both cases above, the sharing of data and sharing of methods, the ability to execute methods across component boundaries is critical. The two independent factors were *where the method* was executed and *where the state* of the object actually resided. In the case of sharing objects, we saw that both method and state are located remotely. In the case of sharing methods, the method is located remotely, but the state of the object is located locally. In general, there is a total of four combinations of location of method and state. To illustrate this, suppose there are the two methods, Remote_method() and Local_method(). Also suppose there are two objects, Remote_obj and Local_obj, then the four possibilities are as follows:

- Remote_method(Local_obj): This corresponds to the sharing of methods.

- Remote_method(Remote_obj): This corresponds to the sharing of data objects.

- Local_method(Local_obj): This corresponds to the case where there are no remote objects.

- Local_method(Remote_obj): This should eventually come down to the first case, in order to retrieve part of the state of the remote object to complete the method.

3.3 Discovery

The preceding discussion has in a sense assumed that the objects being shared were already known. The *discovery* process pertains to finding out what information should be shared in the first place. At the most abstract level, the remote objects that would probably have most interest to the user of a local database are those objects whose concept domain overlaps with the concepts of his/her local database. Depending upon the type of information which a user is interested, s/he may wish for a large intersection or a small one. Large intersections would correspond to concepts that are very similar to the concepts in his/her database (e.g. Person and Employee). Small intersections would correspond more to "related" type of information (e.g., House and Owner), or completely disjoint information.

To assist users in discovering, establishing and refining these sharing patterns with external database systems, we utilize the notion of an *intelligent* **Sharing Advisor**.

3.3.1 The Sharing Advisor

The **Sharing Advisor** of **Remote–Exchange** assists users in discovering, establishing, and refining sharing patterns with external database systems. In particular, the **Sharing Advisor** exploits the following techniques:

- *Structural and behavioral equivalence*: By using structural and behavioral equivalent techniques, we can identify semantically related objects. This applies to similarities among exported objects and similarities between exported objects and the requested information (see the section on Object Equivalence below).

- *Constraint analysis*: Constraint analysis is utilized to identify semantically related objects; the knowledge of constraints among remote objects suggests their potential relevance and relationships with local objects [UD88].

- *Heuristics*: We utilize a set of heuristics to help in identifying semantically similar objects, and in estimating the initial probability relevance of each object. Heuristics can be dynamically added from time to time due to the feedback data from users.

We note that the **Sharing Advisor** is an interesting tool, in the sense that it utilizes input from the user in applying the above techniques. In order to provide the functionality described above, the **Sharing Advisor** must have access to information which describes (meta)data and possible relationships among them in the various components. This information is stored in the **Semantic Dictionary**.

3.3.2 The Semantic Dictionary

The semantic dictionary is simply another component database. When a component wishes to export objects for sharing purposes, the **Semantic Dictionary** is augmented. Similarly, when it wishes to terminate the sharing, this information is removed from the **Semantic Dictionary**. These two activities characterize the **Semantic Dictionary** and can be executed as remote operations from the other components.

The issue that remains is how to organize the exported meta-data. Since two different processes need to access information in the **Semantic Dictionary**, two possibilities exist: organize the **Semantic Dictionary** based on the discovery process, or organize it based on the integration process. We are investigating both possibilities and hope to incorporate both approaches into our system.

3.4 Object Equivalence

One of the fundamental goals of **Remote–Exchange** is to automate the sharing of information objects among a collection of database components. In general, sharing is possible at many different levels of abstraction and granularity, ranging from specific information units (data objects), to meta-data (structural schema specifications and semantic integrity constraints), to behavior (methods/operations). In light of such diversity, the ability to compare these various objects is required.

Recall that the purpose of the discovery phase is to determine the presence of relevant non-local information among the exporting schemas of the participating databases. In order to perform the integration and semantic resolution of several schemas, it is necessary to determine the correspondence of this information with the local information. **Remote–Exchange** must be able to compare data objects with each other and determine if and to what degree they represent the same real world object. In order to compare objects, we need a notion of object equivalence and a mechanism to detect it. While it is nearly impossible to completely automate such a procedure, we are currently using two different approaches that can help in this task. Each approach by itself may be incomplete but when applied in combination they provide a reliable detection mechanism that works in most situations.

The first and most widely used approach is to determine if two objects are structurally equivalent. As an example, consider a local object L and a remote object R that are being compared[3]. In the case where both objects are atomic, then the comparison is trivial. We can simply apply some sort of "eq" or "equal" semantics and we are done. Otherwise, we need to make further comparisons. This includes comparing information such as the class name, the instances that are currently defined, the subclasses and so forth. In addition, attribute information such as value types, missing attributes, and mapping constraints, for example, can provide useful information about structural characteristics. The more commonalities there are with respect to the above criteria the higher the correlation between L and R.

The second approach is to look at the operations that are defined on L and R in order to establish a behavioral equivalence[4] between the objects. The basic idea behind this approach is to run all operations that are applicable to L on object L and compare the results with those obtained when running the same methods on R.

In order to establish equivalence between two objects, we will make use of a so-called *Correspondence Function*: given two objects as input, the correspondence function determines the degree of equivalence between the two. Since equivalence also depends on the level of granularity at which the objects are being examined, the correspondence function will need to take this as additional information into consideration. It is important to note that this is just a conceptual description of such a procedure and more work needs to be done on this in order to build this correspondence function.

The success of this approach largely depends on how well our architecture supports the remote execution of procedures and there are several difficulties that have to be addressed before this approach can be realized in our prototype. While the concept of structural object equivalence is well established and has been widely used in the content of schema integration [BLN86], behavioral object equivalence has not yet been adequately explored.

3.5 Integration/Resolution

When a component wishes to import information from other participants of the federation, it first consults with the sharing advisor to establish a sharing pattern and locate possible sources of information. Once the relevant data has been discovered, the local component may add the remote (meta)data to its own local schema through the integration process [KG81, BLN86, Mot87, HY90].

Adding (meta)data to an already existing schema is a two step process. First, all conflicts (e.g., naming, structural, scaling) between the local database objects and the external objects must be resolved. Second, these objects must be integrated into the local schema as gracefully as possible. Since the objects being imported can represent data values, meta-data, or operations, this task can be difficult. To help in this process, the integration process needs to access information in the **Semantic Dictionary** described earlier for information about external objects. The first part of this two step process has already been discussed. We now focus in greater detail on the resolution and integration process.

Assuming the component has consulted with the sharing advisor and established a "sharing plan", it is ready to start the integration process. The purpose of the sharing plan is to explore sharing patterns and record the location(s) of (meta)data in the collection of participating components that the component wishes to import. Upon importing the (meta)data, structural conflicts with existing classes in the component's local class hierarchy may arise. Several possibilities exist, and we demonstrate our approach with the help of several examples. The examples differ in the complexity of the schemas to be integrated, starting with the most simple case: the integration of a single isolated class object. The second example examines the case when a component wishes to import one or more classes which are inter-related. The classes to be integrated are usually part of a more complex class hierarchy, hence we also need to be concerned with inheritance issues. The last example is a special case of the previous one and focuses on the relationship(s) between the objects to be integrated. In this case, it is possible to construct a scenario in which a component wishes to import objects that are connected in a complex manner involving one or more classes not present in the local schema. Although it is difficult to propose a useful measure of the "completeness" of this enumeration, we shall attempt to use the examples presented here as a foundation for our work, structure them, and argue that they are sufficient to illustrate the feasibility of this approach.

In the first scenario that we are going to present, we assume that a component, named *C1*, wishes to

[3] In this and the following paragraphs the term object can refer to a class object as well as an instance object.

[4] This requires the ability to compare methods and their results when executed in different environments.

import a single object class *T1*:

- In the case where *T1* does not exist in *C1's* local schema, the new class *T1* can be added to the local schema without further modifications. It is important to note that "adding" a class requires some additional work in the sense that there may be problems with value classes for new attributes.
- In the case where *T1* already exists as some local class *LT1* we propose to make the imported class *T1* a superclass of *LT1* and add the necessary attributes to both the superclass as well as the subclass. It is important to note that *T1* will be integrated as the superclass of *LT1* in order to handle remote execution of methods that are associated with the imported class *T1*.
- The case where *T1* and *LT1* are identical, is a simplification of the above scenario and only requires the integration of the class instances in which *C1* is interested.

In our second scenario, a component *C1* wishes to import (meta)data consisting of several, interrelated classes. For simplicity, we will work with only two different classes, namely *T1* and *T2*, but the following discussion can be adapted to situations where more than two classes are involved in the integration process:

- If *T1* and *T2* do not exist in the local schema, add the classes and all class instances as required by *C1*.
- If one of the classes is present in *C1's* local schema, say *T1* as *LT1*, several things need to be done[5]. First we propose to create a new superclass *LST1* for *LT1* in order to hold the imported instances from *T1*. We then add *T2* as *LT2* to *C1's* local schema including all of the instances. Finally, we create new attributes relating *LST1* and *LT2* [6].
- Both classes *T1* and *T2* are present in *C1's* local schema. In this case, all instances can be imported into the already existing local classes.

As a final example, consider the case where component *C1* wishes to import classes whose relationships are modeled differently then the corresponding class relationships in *C1's* local schema (e.g., *LT1* and *LT2* are related through a ternary relationship that includes an additional class *LT3* whereas the corresponding remote classes, *T1* and *T2*, are related directly through attributes and their inverses). Using the example stated above, two new superclasses for *T1* and *T2* are necessary. Then, for each instance pair that *C1* imports, a new instance for *LT3* has to be created which relates the imported objects. There are many variations to this general scenario but it is possible to show that they can all be treated in a similar manner.

[5] The following is also true if *T2* were present as *LT2* instead of *LT1*.

[6] This is true provided *T1* and *T2* are related outside of *C1's* schema.

4 Conclusions and Research Directions

This paper has examined the problem of supporting dynamic patterns of sharing among a collection of autonomous, heterogeneous databases. We have explored the spectrum of heterogeneity that may exist in this environment, and have described a top level architecture and approach to sharing. The experimental **Remote–Exchange** system under development explores a number of architectural and implementation issues, as well as providing a basis for the refinement of the sharing model and mechanism.

The principal current focus of research centers on the following issues:

- Detailed design and implementation of the remote transparency mechanism, and the transparent sharing of behavioral objects among components;
- Efficient (remote) execution of these behavioral objects;
- Refinement of the kernel object database model, and experience with the use of the extensible features of the model;
- The content and organization of the **Semantic Dictionary**;
- The collection of heuristics that form the basis for the operation of the sharing advisor;
- Studies of the tradeoffs involved in centralized vs. decentralized control and management of coordination information;
- Access control (a related project is currently examining this);

- Utilization in specific application domains (viz., collaborative design environments and computer-integrated manufacturing); and
- Employing **Remote–Exchange** as a framework for the interconnection of heterogeneous database management systems.

References

[BLN86] C Batini, M. Lenzerini, and S. Navathe. A comparative analysis of methodologies of database schema integration. *ACM Computing Surveys*, 18(4):323–364, 1986.

[CP84] S. Ceri and G. Pelagatti. *Distributed Databases: Principles and Systems.* McGraw Hill, 1984.

[DH84] 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.

[HM85] D. Heimbigner and D. McLeod. A federated architecture for information systems. *ACM Transactions on Office Information Systems*, 3(3):253–278, July 1985.

[HY90] R. Hull and S. Yoshikawa. Ilog: Declarative creation and manipulation of object identifiers. Technical report, USC, 1990.

[Ken89] W. Kent. The many forms of a single fact. In *Proceedings of the IEEE Spring Compcon.* IEEE, February 1989.

[KG81] 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.

[LA86] W. Litwin and A. Abdellatif. Multidatabase interoperability. *IEEE Computer*, 19(12), December 1986.

[LM84] P. Lyngbaek and D. McLeod. Object management in distributed information systems. *ACM Transactions on Office Information Systems*, 2(2):96–122, April 1984.

[MB81] A. Motro and P. Buneman. Constructing superviews. In *Proceedings of the ACM SIGMOD International Conference on Management of Data.* ACM SIGMOD, April 1981.

[Mot87] A. Motro. Superviews: Virtual integration of multiple databases. *IEEE Transactions on Software Engineering*, 13(7), July 1987.

[Sha89] M. Shan. Unified access in a heterogenous information environment. *IEEE Office Knowledge Engineering*, 3(2):35–42, August 1989.

[UD88] S.D. Urban and M.L. Delcambre. Constraint analysis: A tool for explaining the semantics of complex objects. In *Advances in Object-Oriented Database Systems*, pages 156–161. Springer-Verlag, 1988.