An Intelligent System For Identifying and Integrating Non-Local Objects In Federated Database Systems*

Joachim Hammer, Dennis McLeod and Antonio Si

Computer Science Department
University of Southern California
Los Angeles, CA 90089-0781, USA

Abstract

Support for interoperability among autonomous, heterogeneous database systems is emerging as a key information management problem for the 1990s. A key challenge for achieving interoperability among multiple database systems is to provide capabilities to allow information units and resources to be flexibly and dynamically combined and interconnected while at the same time, preserve the investment in and autonomy of each existing system. This research specifically focuses on two key aspects of this: (1) how to discover the location and content of relevant, non-local information units, and (2) how to identify and resolve the semantic heterogeneity that exists between related information in different database components. We demonstrate and evaluate our approach using the Remote-Exchange experimental prototype system, which supports information sharing and exchange from the above perspective.

1 Introduction

The past several decades have witnessed the emergence of powerful general-purpose database management facilities, as well as the proliferation of databases constructed with those facilities. A central current problem in database research is to support the effective sharing and exchange of information among various database systems, while at the same time respecting the autonomy of those systems. Further, recent advances and wide acceptance of communication networks have provided foundational mechanisms to support the interconnection of related existing database systems. Such a collection of cooperating but heterogeneous, autonomous database systems may be termed a *federated database systems* or *federation* for short; each individual database system in a federation is termed a *component database system* or *component.*

While the interoperability problem is largely addressed in the communication network environment [22], it only provides limited support for the interoperation of heterogeneous database systems. A main reason for this limitation stems from the difficulties in overcoming the problem of unifying heterogeneous information. This is commonly known as the *semantic heterogeneity problem* [9].

In light of this observation, various architectures have been proposed to address the database interoperability problem. Such architectures range from tightly-coupled composite approaches in which component databases are integrated into a centralized global database [2], to loosely-coupled federated environments wherein information is shared among component database systems while retaining their autonomy [11]. A key challenge for all these environments is to provide capabilities to allow information units and resources to be flexibly and dynamically combined and interconnected, while at the same time preserving the investment in and autonomy of each individual database component.

The *Remote-Exchange* federated architecture and experimental system extends the traditional federated architecture in supporting two key aspects of this: (1) how to discover the location and content of related, non-local information units (simple objects, types of object, units of behavior, etc.), and (2) how to identify and resolve the semantic heterogeneity that exists between related information in different database components. The goal of this paper is to demonstrate the architecture of this system and explain key implementation issues faced. Our approach to the two key problems cited above can be utilized by a variety of intelligent and cooperative information systems (ICISs) [17, 19].

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, wherein individual components can share and exchange information units (objects). Section 4 describes the architecture of the Remote-Exchange experimental system in detail and illustrates the ideas of our approach to information sharing. In Section 5, we examine key issues in the design and implementation of the Remote-Exchange prototype using a collection of Omega-based database systems [8]. Finally, Section 6 contains concluding observations with a critical evaluation of our results and their potential impact on future work.

*This research was supported in part by NSF grant IRI-9021028.

2 Related Work

The term "heterogeneous databases" was originally used to distinguish work that included database model and conceptual schema heterogeneity from work on "distributed databases[1]" which addressed issues solely related to distribution [1]. Recently, there has been a resurgence in research in the area of heterogeneous database systems (HDBSs). Work in this area can be characterized by the different levels of integration of the component DBSs and by different levels of global (federation) services. In Multibase [2], 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 expensive and difficult, and tends to be hard to change.

The federated architecture proposed in [11], which is similar to the multidatabase architecture of [16], 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 approach.

One common approach to schema integration is to reason about the meaning and resemblance of heterogeneous objects in terms of their structural representation. In Larson et al. [15], 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 have been developed include heuristics to determine the similarity of objects based on the percentage of occurrences of common attributes [10, 23]. More accurate techniques use classification for choosing a possible relationship between classes [21].

Whereas most of previous methods primarily utilize schema knowledge, techniques utilizing semantic knowledge (based on real-world experience) have also been investigated [5, 12]. These approaches usually assume the existence of a real world knowledge base which serves as a global schema to which every local schema is mapped. However, we believe that a useful approach is for the centralized knowledge base to only contain information actively used to support sharing within the federation and thus, as illustrated in our mechanism, should be incrementally built tailored to the federation. Furthermore, these approaches lack the ability of tailoring the identification process to the context of user request.

A different approach proposed by Kent [14] uses an object-oriented database programming language to express mappings among different similar concepts that allow a user to view them in some integrated way. It of course 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. [18] uses so-called path-methods to explicitly create inter-component and inter-object mappings between source and target object 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 and/or dynamic sharing patterns.

3 The Federated Database System Context

Consider the following scenario involving a group of molecular biologists whose databases form a collaborative **F**ederation **O**f **M**acromolecular **D**atabanks (FOMD), as depicted in Figure 1. The goal of FOMD is to share and exchange macromolecular information between individual institutes and researchers [7, 20]. For instance, a researcher could efficiently reuse protein data that had already been sequenced (decoded) by other components for his/her experimental work.

Figure 1 also shows a snapshot of the information stored in FOMD at a given point in its lifetime. Since each macromolecular database is managed by a different organization (institute), the contents of their databases reflect the different foci and interests. We can see, for example, that institute B is the only component with information on both genetic and protein sequences. All other components maintain either genetic or protein information with various levels of detail.

A difficulty in finding a solution to the problem of achieving interoperability in FOMD and similar federations stems from the conflicting nature of sharing and autonomy. On the one hand, an institute would like to share information with other components of FOMD. On the other hand, the same component would also like to exercise autonomy over its own database with respect to organization, administration and sharing, e.g., control over the information it is willing to "export" to the other components[2]. In consequence, we assume that when a component agrees to join the federation, information to be made available to other institutes is stored in a specially "marked" schema called an *export schema*.

The level of interoperability that can be achieved within such a federation depends largely upon two key

[1] 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.

[2] This capability of exporting only a specific portion of a component's database is especially important in the FOMD environment where a research institute would probably not willing to release any information that has not been published or fully validated.

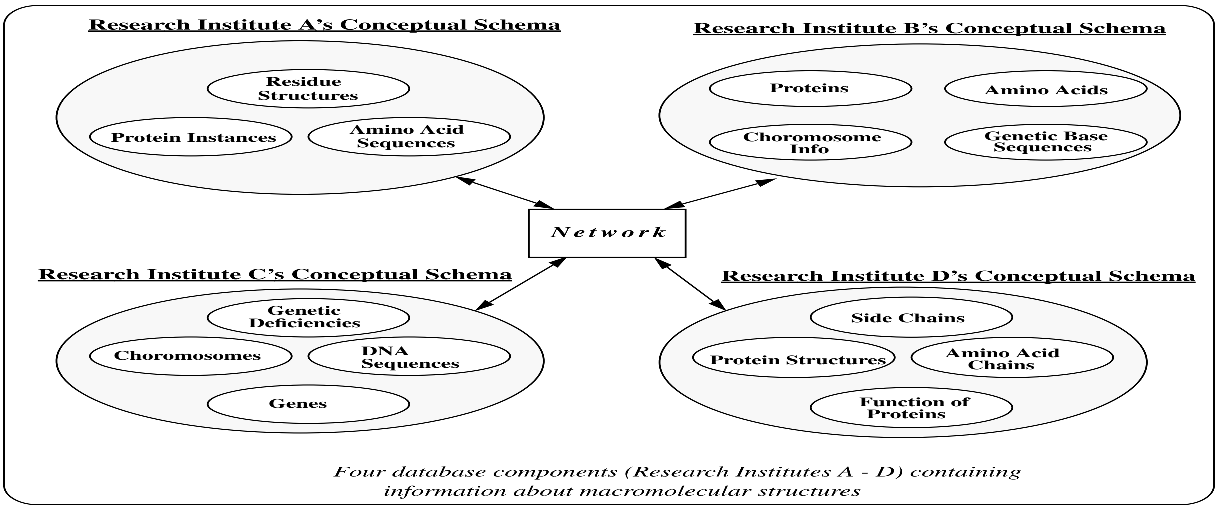

Figure 1: A conceptual overview of FOMD and its components

capabilities:

1. the ability of a component to identify and locate potentially appropriate non-local information with respect to its needs (the *discovery problem*); and

2. as requested remote information is identified and located, the ability to fold it into the local system framework (the *unification problem*).

The focus of this work is to address the above two problems; several other important issues such as security (access control) and automatic update of shared data are not directly addressed here.

4 Achieving Interoperability in Remote-Exchange

In order for any collaboration to take place among the heterogeneous components of a federation, some common model for describing the sharable data must be utilized. One may of course argue as to the nature of this "lingua franca". We believe that this model should be semantically expressive enough to capture the intended meanings of conceptual schemas which may reflect essential kinds of heterogeneity (as will be discussed below). Further, this model must be simple enough so that it can be readily understood and implemented. To this end, we have chosen to use a **M**inimal **O**bject **D**ata **M**odel (MODM) as the common database model for describing the structure, constraints, and operations for sharable data.

4.1 MODM

MODM is a generic functional object database model, which supports the usual object-based constructs. In particular, it draws upon the essentials of functional database models, such as those proposed in Daplex, Iris, and Omega [8]. MODM contains the basic features common to most semantic and object-oriented models [13]. The model supports complex objects (aggregation), type membership (classification), subtype to supertype relationships (generalization), inheritance of properties (attributes) from supertype to subtypes, and user-definable functions (methods).

In the context of MODM, interoperation among components, generally, is possible at many different levels of abstraction and granularity, ranging from factual information units (data objects), to meta-data (conceptual schema), 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) are preliminarily examined in [4].

4.2 Remote-Exchange Experimental System

Figure 2 illustrates the architecture of the Remote-Exchange experimental system. In this figure, the *sharing advisor* is a special component that manages knowledge about existing type objects that components export; this knowledge resides in the *semantic dictionary*. The sharing advisor provides four "intelligent" services to the components of the federation: *Registration*, *Discovery*, *Semantic Heterogeneity Resolution* and *Unification*. These services address different aspects of the problem of type-level sharing, ranging from detecting similarity during the registration process to determining precise relationship between objects during the unification process. While it is nearly impossible to completely automate these services, our approach provides substantially useful functionality.

4.3 Registration

Registration allows a (new) component to inform the sharing advisor about any information it is willing to share with other components in the federation. In a sense, it establishes an initial sharing context within

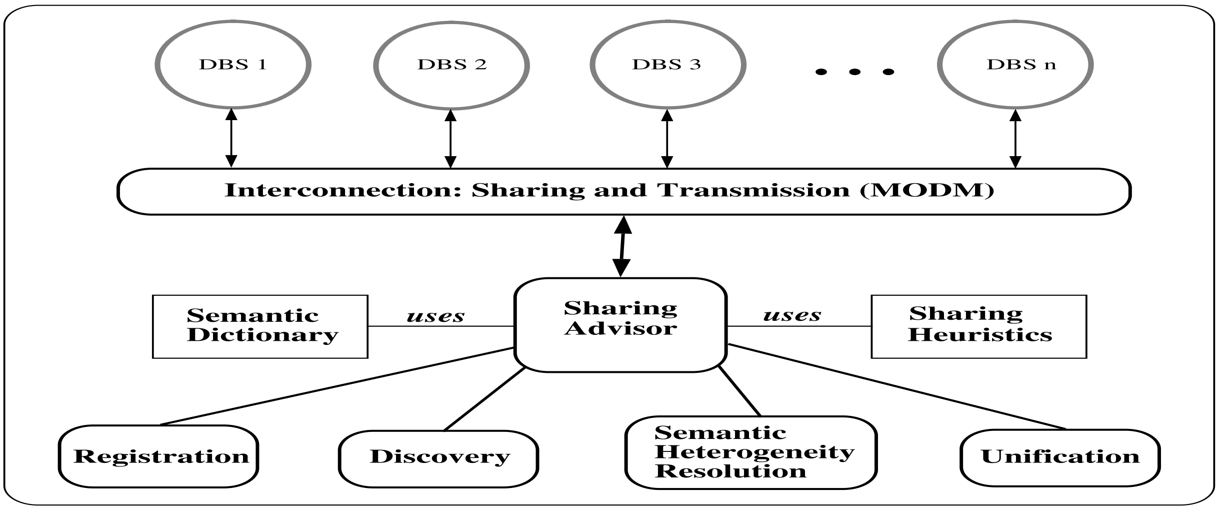

Figure 2: The Remote-Exchange architecture

the federation by logically connecting the exported information to the semantic dictionary via the sharing advisor. Incremental registration allows a component to augment its export schema with new information.

Figure 3 shows two partial conceptual schemas of component A and component D, respectively. Let us assume that component A has already registered the type objects **Protein Instances** and **Amino Acid Sequences** with the sharing advisor. This is reflected in the semantic dictionary shown in Figure 4a. When component D registers type object **Protein Structures**, the sharing advisor can use its existing knowledge (in this case the information obtained from component A) to determine if **Protein Structures** has any similarities with previously registered types, i.e., **Protein Instances** and **Amino Acid Sequences**. User interaction may be necessary to instruct the sharing advisor in case the information in the semantic dictionary is insufficient to detect (dis)similarities among type objects automatically. The newly acquired knowledge and the newly registered information are stored in the semantic dictionary for future consultation; see, for example, contents of the semantic dictionary in Figure 4b after the registration of the type object **Protein Structures**. In a sense, the semantic dictionary represents a dynamic *federated knowledge base* about sharable information in the federation.

In the remainder of this section, we illustrate how the sharing advisor can effectively utilize the semantic dictionary as a means of organizing various registered type objects in order to accommodate information sharing. We also introduce a set of sharing heuristics that guide the sharing advisor through registration.

4.3.1 The Semantic Dictionary

In the semantic dictionary, types determined to be similar by the sharing advisor are classified into a collection called a *concept*, within which subcollections called *subconcepts* can be further classified. This generates a *concept hierarchy*. Naturally, the relationships expressed in this concept hierarchy can only be approximations of the true, real-world relationships that exist between the exported types of different components; additional mechanisms are needed to establish more exact relationships (see Section 4.5).

Figure 4 shows two snapshots of the concept hierarchy in the semantic dictionary taken at different times during the life-time of our example federation. Figure 4a indicates the concept hierarchy after types **Protein Instances** and **Amino Acid Sequences** of component A have been registered. Figure 4b shows the corresponding hierarchy after component D has registered type **Protein Structures**. The hierarchy in Figure 4b also indicates that **Protein Instances** and **Protein Structures** are representing similar information, since they belong to the same classification, called **Protein Information**. In addition, types **Protein Instances** and **Protein Structures** have properties that distinguishes one type from another. Hence, they are also created as subconcepts of **Protein Information** in order to express their dissimilarities. By contrast, **Amino Acid Sequences** is similar to neither **Protein Instances** nor **Protein Structures**, and thus appears as a separate concept in the hierarchy. Similarity and dissimilarity of this kind is detected by the sharing advisor based upon sharing heuristics (described below), with user input as required.

The advantage of organizing type objects in a concept hierarchy is that a hill climbing technique can be used to place newly registered type objects. For example, consider a type object being registered which is determined to be unrelated to members of concept **Protein Information** in Figure 4b; in this case, no further comparisons with members of subconcepts of **Protein Information** are necessary.

Generally, a type object represents a specific view

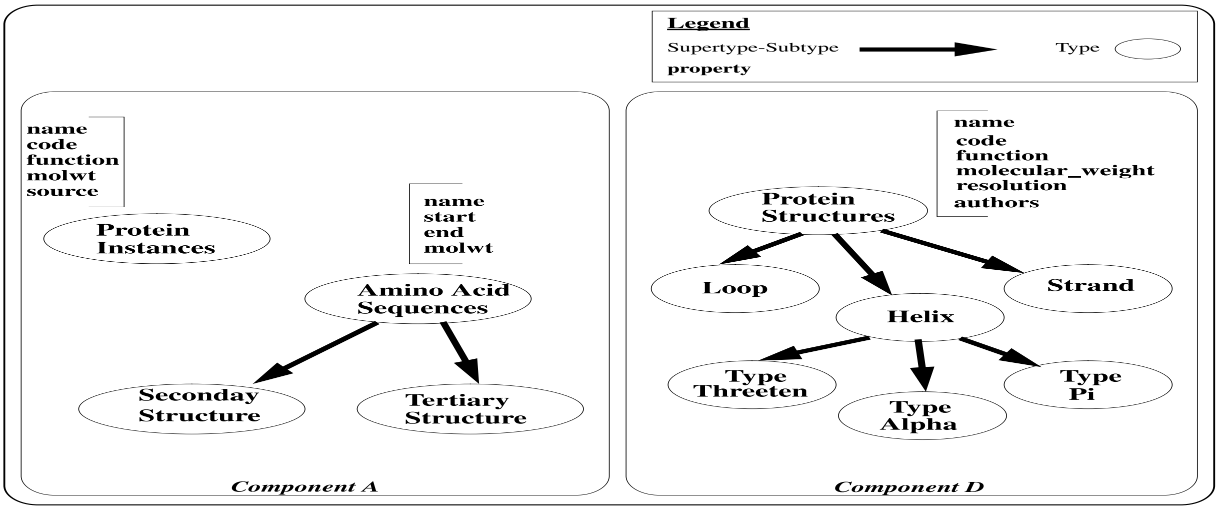

Figure 3: Partial conceptual schemas of two macromolecular databases

of a corresponding real world concept, and is tailored to the focus and interest of the database component; therefore, the set of properties associated with the type object can be viewed as a subset of those associated with the real world concept. In Figure 3, type **Protein Instances** of component A, and **Protein Structures** of component D indicate two different views on the real world concept **protein**. In order to properly merge various views on a similar concept, the semantic dictionary is established in a bottom up fashion with the set of properties belonging to a concept at a particular level represented as the union of the properties of all its subconcepts. This is illustrated by concept **Protein Information** in Figure 4b. Using a concept hierarchy to organize exported types will incrementally establish a *federated view* of all export schemas in the federation.

Note that the concept hierarchy introduced is not static. It is dynamic and evolves incrementally depending on the knowledge and information received by the sharing advisor. Consider the following scenario where a total of N type objects are registered with the sharing advisor. On one extreme where all N types are similar enough to belong to one single concept, there will be $N+1$ objects in the semantic dictionary (N type members + 1 concept); on the other extreme where all N types are completely unrelated, there will be $2N$ objects in the semantic dictionary (N type members + N concepts).

4.3.2 Sharing Heuristics

The sharing heuristics employed draw upon the incremental clustering paradigm of machine learning, as described in [6]. The idea behind these heuristics is to assess the extent of distinguishing capability of a property with respect to a concept; this allows the sharing advisor to determine if the meaning of a type object being registered can be determined based upon its properties, or whether further assistance from users is necessary. The distinguishing capability with respect to a concept is based upon the *inter-concept dissimilarity* between concepts and the *intra-concept similarity* within a concept. As an example, consider property **code** of concept **Protein Information** in Figure 4b. This property has a high inter-concept dissimilarity, as no other concept at the same scope/level possesses such a property. On the other hand, **code** also has a high intra-concept similarity with respect to **Protein Information**, since this property is associated with all concept members of **Protein Information**, for example, **Protein Instances**. However, when looking at the subconcepts of **Protein Information**, property **code** has a low inter-concept dissimilarity with respect to each subconcept of **Protein Information**, since this property is possessed by all concepts within this same scope of a common superconcept, **Protein Information**. Note that a property does not necessarily possess the same extent of distinguishing capability across different levels.

Inter-concept dissimilarity and intra-concept similarity values are estimated via statistical analysis based on previously registered type objects. A statistical heuristic-based approach offers a degree of error resilience, allowing the accuracy of the distinguishing capabilities to be gradually improved over a period of time. A limitation of this approach is its potentially oscillating nature during early stages of a federation; however, this limitation becomes less significant as the number of components and type objects increases, damping the oscillation to a steady state.

Information that is utilized by the sharing advisor to interrelate similar properties comes from three different sources. (1) The property itself: in the situation where a property is defined as an atomic data type such as INTEGER or STRING, only the property name is utilized by the sharing advisor; this is because the value type of the property does not provide additional useful semantic. On the other hand,

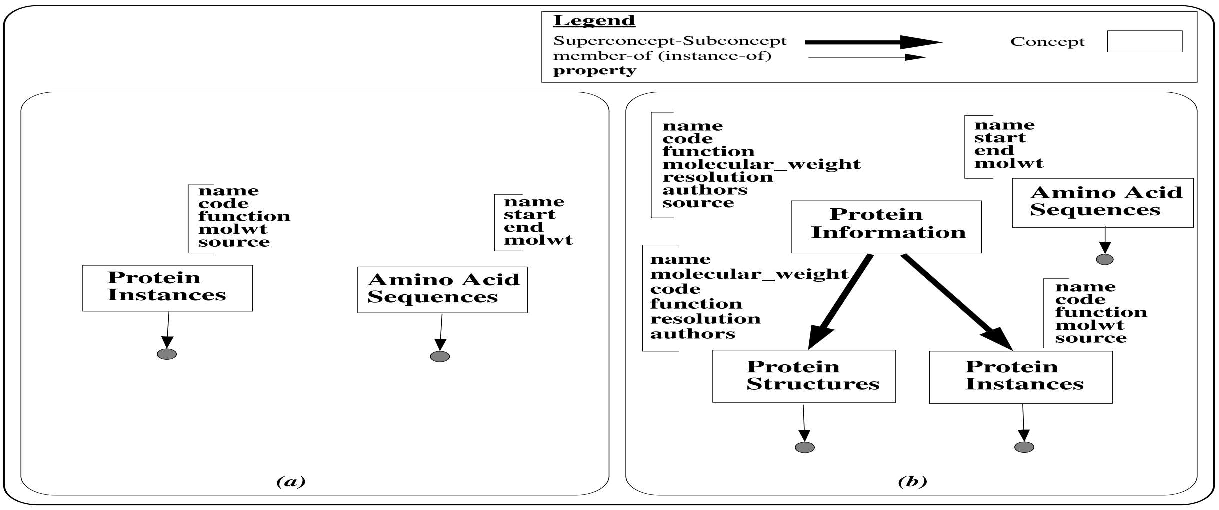

Figure 4: Evolution of concept hierarchies in the semantic dictionary

if a property is defined as a user defined type, both property name and the value type of the property will be considered by the sharing advisor as both of these information provide real world semantics to the meaning of the property. (2) Previously acquired knowledge: a list of property terms correspondence is incrementally maintained in the semantic dictionary as a result of user instruction. This list is consulted by the sharing advisor whenever properties correspondence information are needed. (3) User consultation: in the situation where the semantic dictionary does not provide adequate knowledge to the sharing advisor to properly interrelate similar properties, user is consulted.

4.4 Discovery

The purpose of discovery is to identify appropriate information relevant to the request of a component initiating a sharing procedure. Although we have proposed a methodology to detect "similarities" among different types, it is not adequate in a federated environment where the goal is to integrate specific non-local information into the environment of a local component. This is because even though a remote object is similar to a particular local object, it might not be relevant within the intended context of a local component. Therefore, it is imperative that user characteristics of the local components be taken into consideration when locating relevant information. For this purpose, we have categorized three basic kinds of discovery requests, which when combined, allow a component to discover a wide variety of non-local information:

- Type 1—Similar Concepts.
  In this kind of discovery request, a component user is interested in locating type objects in remote components that are conceptually similar/related to a particular type object in the local component. All type objects belonging to the portion of the concept hierarchy in which the local type object resides (in the semantic dictionary) are appropriate to this request. For example, in the concept hierarchy of Figure 4b, **Protein Structures** of component D is a proper candidate to the request by component A for related information on **Protein Instances**.

- Type 2—Complimentary Information.
  In this kind of discovery request, a component user is interested in discovering additional information about a local type object. This may occur when component A is interested in additional information on **Protein Instances**, for example. All type members with different sets of properties belonging to the same portion of the hierarchy in which the local type object resides are proper candidates for this request. For example, **Protein Structures** of component D would also satisfy a request for additional information of **Protein Instances** issued by component A.

- Type 3—Overlapping Information.
  This kind of discovery request arises when a component is interested in locating non-local type objects that overlap in their information content with a component's local type. For example, component A would like to display all "protein"-like information using its own three dimensional viewing program which works on members of type **Protein Instances**. All types with similar properties as **Protein Instances** that belong to the sub-hierarchy rooted at **Protein Instances** are proper candidates for this request. According to the concept hierarchy of Figure 4b, there is no candidate type object that would satisfy such a request at this time.

4.5 Semantic Heterogeneity Resolution

After identifying relevant non-local information objects, a component may wish to fold them into its local information framework. However, the problem is to determine how this information can be unified with its own local data due to semantic discrepancies that may exist between related concepts in different components. Such semantic heterogeneity is a natural consequence of the independent creation and evolution of autonomous databases which are tailored to the requirements of the applications they serve [9]. The purpose of our approach to semantic heterogeneity resolution is to resolve these discrepancies between the relevant non-local type object(s) and the local meta-data context (conceptual schema). This is a prelude to unifying the remote information into the local context. Our mechanism for resolving semantic heterogeneity employs a *(local) lexicon* for each component, which specifies its perspective on the precise relationship between its local types and a global set of commonly understood concepts. Specifically, knowledge is represented as a static collection of facts of the simple form:

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

A term on the left hand side of a relationship descriptor represents a local type object (i.e., the unknown), which is described by the term on the right side. We have an initial set of descriptors, which is extensible; we also anticipate collections of descriptors that are tailored to given application domains. The following is a list of basic conceptual relationship descriptors initially supported in Remote-Exchange:

Descriptor	Meaning	Example
`Identical`	Two types are the same	**Protein** `Identical` ***Protein***
`Equal`	Two types are equivalent	**DNA Strand** `Equal` ***Genetic Base Seq.***
`Compatible`	Two types are transformable	**RNA** `Compatible` ***Protein***
`KindOf`	Specialization of a type	**Protein** `KindOf` ***Molecule***
`Assoc`	Positive association between types	**Gene** `Assoc` ***Protein***
`CollectionOf`	Collection of related types	**Protein** `CollectionOf` ***Amino Acids***
`Common`	Characteristic of a collection	**DNA** `Common` ***Structure=Double Helix***
`Feature`	Descriptive feature of a type	**Protein** `Feature` ***Molecular Structure***
`Has`	Property of all instances	**Chromosome** `Has` ***Genetic Map***

For example, we may have the following in the local lexicon of component A:

Term	Relationship Descriptor	Key concept	Textual description (not part of lexicon)
Protein Instances	`KindOf`	***Protein Information***	A protein instance is a certain kind of protein information.
Protein Instances	`CollectionOf`	***Proteins***	It is a collection of all proteins.
Protein Instances	`Has`	***Components***	Each instance of Protein Instances consists of (amino acid) building blocks.

The terms that are used to describe unknown concepts are taken from a dynamic list of concepts drawn from the semantic dictionary, characterizing the commonalities in a federation. Since interoperability only makes sense among concepts 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. The semantic dictionary contains partial knowledge about all the terms in the local lexica in the federation, suggesting possible relationships between different terms. Utilizing this knowledge, each local lexicon describes the precise meaning of all type objects that the component exports. In our example (see Figure 4b), the set of commonly understood concepts at a particular moment could be:

$\mathcal{C}$ = {*Amino Acid Sequences*, *Authors*, *Proteins*, *Protein Information*, *Protein Instances*, *Protein Structures*}

The essential idea of using a local lexicon is to represent the semantics of the shared terms in a more expressive and complete manner than in the conceptual schema. The additional semantic information is important for the following reason: as noted earlier, the results of the discovery process are not always conclusive enough to determine the exact meaning of a term, or more precisely, how two similar terms in different components are related. Similarly, it is not possible to "derive" the meaning/usage of terms by looking at their structural representation in the conceptual schema of a local component. In order to determine the relationships between objects in a federation, we realize that not one single method but a combination of several different but complimentary approaches (i.e., Registration, Discovery, Semantic Heterogeneity Resolution) taken together is highly promising. For example, there is an important connection between local lexica and semantic dictionary. Local lexica contain only semantic information and no knowledge about any relationships among its objects; information that is necessary to solve the semantic heterogeneity problem. This kind of information is provided by the semantic dictionary (through registration) which contains partial knowledge about the relationships among all the terms in the local lexica in the federation. Note of course that the lexica and semantic dictionary are both dynamic, i.e., they grow (and shrink) as the amount of shared data in the federation increases (decreases).

The basic problem addressed by the semantic heterogeneity resolution mechanism may now 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 (conceptual schema information) and the (known) relationships that exist between common, global concepts in set $\mathcal{C}$ 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 $\mathcal{C}$ and creating the local lexicon) rather than during.

4.6 Unification

At this point, the non-local object(s) can be unified with the corresponding local object(s). In some cases, the local meta-data framework (conceptual

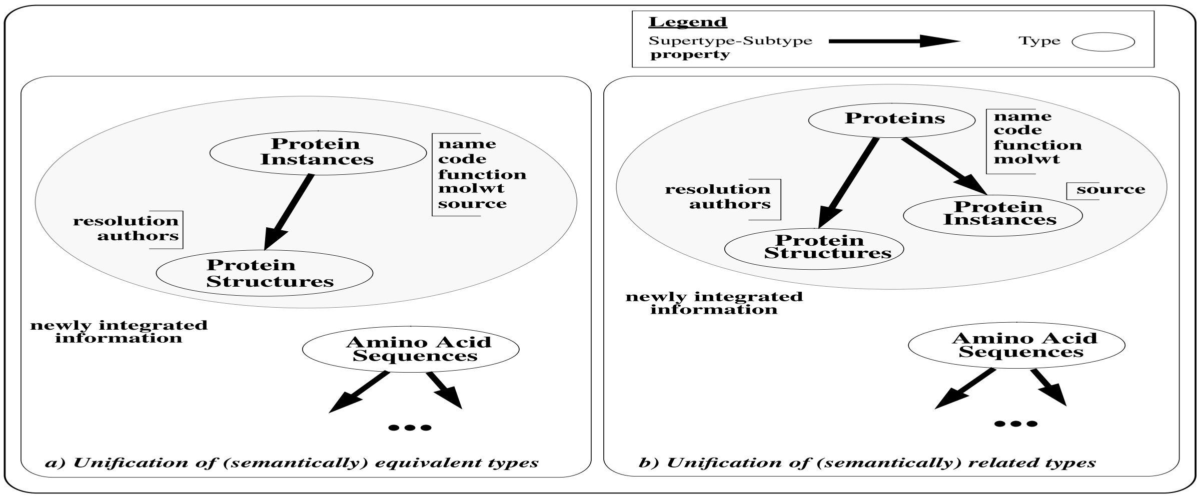

Figure 5: A's schema after resolution and unification of "protein"-like information.

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. In [9], we enumerated in detail various conflicting possibilities that can arise while importing non-local meta-data into a local schema. Here, we illustrate our mechanism with the simple example of importing type **Protein Structures** of component D into A's schema. The following two possibilities exist:

- **Protein Structures** is (semantically) equivalent to **Protein Instances** in A's schema.
  In this case, we make **Protein Structures** a subtype of **Protein Instances**. Properties that previously belonged to **Protein Instances** remain there. Properties that belong to **Protein Structures** but not to **Protein Instances** are added to the new subtype. In those cases where **Protein Instances** has additional properties that do not exist for **Protein Structures** special null values must be assigned to all its instances (see Figure 5a). Note that "subsetting" is considered to be the basis for accommodating multiple user perspectives on comparable types used by most methodologies [2]. The case in which **Protein Structures** is identical to **Protein Instances** is a special case and only requires the importation of the type instances in which A is interested.

- **Protein Structures** is related to **Protein Instances** in A's schema.
  In this situation, a new supertype called **Proteins** is created that contains only the properties common to both types **Protein Instances** and **Protein Structures**. The properties in which **Protein Instances** and **Protein Structures** differ are associated with two subtypes which inherit the properties from their common supertype (see Figure 5b). Together, the new supertype and its two subtypes contain the same information as **Protein Instances** and **Protein Structures** in separate components before the unification [2].

Note that whether **Protein Structures** is related to **Protein Instances** or is equivalent to **Protein Instances** depends upon the perspective of the importing component, as specified in the local lexicon.

5 Experimental Implementation

The current Remote-Exchange prototype consists of a collection of Omega database components[3] [3]. Each component consists of an Omega database for maintaining local information, and two communication interfaces called an *importer* and an *exporter* which run on top of Omega. The purpose of these interfaces is to handle communication requests to and from remote components using the RPC message passing paradigm. This section briefly describes the key issues we faced when implementing our prototype.

As noted, our mechanism requires the ability to extract meta-data information from a database component. Since the original Omega database does not support this functionality, we introduced so-called *meta-functions* in the exporter[4], which return meta-data information about objects in its local Omega

[3]The current Remote-Exchange environment also supports Iris database components to some degree, and we plan to incorporate other database systems in the future.

[4]We have an alternative of implementing such functionality inside the Omega database system; however, it would require modification of the kernel of the database system which is not acceptable; additional modifications would also be necessary

database. An initial set of meta-functions is the following:

Meta-Function	Description
ShowAllT ypes()	Returns a list of all types that can be shared
HasStored Functions(t:T ype)	Returns a list of all stored functions defined on t
HasComputedFunctions(t:T ype)	Returns a list of all computed functions defined on t
HasInstances [a](t:Type)	Returns a list of all instances defined on t
HasV alue(i:Instance, f:Function),	Returns the value of the stored function f on i
HasV alueT ype(f:Function)	Returns the value type for a stored function f
HasDirectSubT ypes(t:T ype)	Returns a list of all direct subtypes of type t
HasDirectSuperT ype(t:T ype)	Returns the direct supertype of type t

[a]Some primitive types like STRING or INTEGER do not have this functionality

In effect, meta-functions serve as an inter-component communication protocol.

In the design and implementation of the sharing advisor, we model it as another Omega component. To avoid modifications of the Omega system, each service provided by the sharing advisor (registration, discovery, resolution and unification) is supported again through its exporter. The semantic dictionary and the concept hierarchy are accordingly modeled by the corresponding Omega database as follows: Each concept in the concept hierarchy is modeled as a type object in the semantic dictionary while each type member of the concept hierarchy is modeled as an instance object in the semantic dictionary.

6 Conclusions and Future Directions

We introduced a framework and mechanism for identifying and integrating type objects from diverse information sources. The mechanism is built on an object based model containing features commonly found in most existing object-based database systems and hence, our mechanism can be implemented in these systems without modification to existing DBMS software. We also demonstrated the feasibility of our mechanism in the FOMD environment, representing an area of tremendous and growing interest and importance [20].

An important characteristic of our mechanism stems from the fact that our approach separates the discovery issue from the integration issue; prior works largely mix the two, with the discovery issue implicitly a part of integration. We observed that different kinds of knowledge are needed in the discovery and integration processes; by identifying the knowledge needed in the different processes, a component is more likely to be able to access the most appropriate non-local information in the most natural way.

As noted, the ability of the sharing advisor to determine the similarity among type objects is critical to the success of our mechanism. As such, we are in the process of experimenting with the sharing advisor by measuring its behavior on different sets of testing data of diverse nature. The measurements are based upon the following two metrics: (1) *completeness*, which measures if all the similar objects are being identified by the sharing advisor, and (2) *precision*, which measures if all the identified objects are similar.

In this paper, we have focused on the ability to access and utilize appropriate non-local information in a natural way; performance efficiency issue was not explicitly considered. We are currently investigating the impact of the so-called "*lazy evaluation*" paradigm on reducing the amount of overhead involved. Briefly, the "lazy evaluation" paradigm shifts the burden on determining the similarity among type objects from the registration process to the discovery process. In this way, the similarity among type objects is never determined unless the type object is queried by a component, thereby eliminating the unnecessary overhead on analyzing the type objects that are never queried by components.

whenever new meta-functions are introduced.

Acknowledgements

The authors would like to acknowledge the valuable comments of Shahram Ghandeharizadeh and K.J. Byeon of USC, as well as those of the referees. Special thanks to Kathryn Stuart who helped us understand the molecular biology that is used in our example scenario.

References

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

[2] 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.

[3] D. Fang, S. Ghandeharizadeh, D. McLeod, and A. Si. The Design, Implementation and Evaluation of an Object-Based Sharing Mechanism for Federated Database Systems. In *International Conference of IEEE Data Engineering*, 1993.

[4] D. Fang, J. Hammer, and D. McLeod. An Approach to Behavior Sharing in Federated Database Systems. In M.T. Ozsu, U. Dayal, and P. Valduriez, editors, *International Workshop on Distributed Object Management*, pages 66–80. Morgan Kaufman.

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

[6] D. Fisher. Knowledge Acquisition Via Incremental Conceptual Clustering. *Machine Learning*, pages 139–172, 1987.

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

[8] S. Ghandeharizadeh, et al. Omega: A Parallel Object-Based System. Technical Report USC-CS 92-517, Computer Science Department, University of Southern California, Los Angeles CA 90089-0781, May 1992.

[9] J. Hammer and D. McLeod. An Approach to Resolving Semantic Heterogeneity in a Federation of Autonomous, Heterogeneous Database Systems. *International Journal of Intelligent and Cooperative Information Systems*, 2(1):51–83, March 1993.

[10] 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.

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

[12] 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.

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

[14] 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.

[15] 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.

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

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

[18] 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.

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

[20] J. Saldanha and J. Eccles. The Application of SSADM to Modelling the Logical Structure of Proteins. *CABIOS*, pages 515–524, 1991.

[21] 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.

[22] M. Schwartz, A. Emtage, Brewster K., and B. Neuman. A Comparison of Internet Resource Discovery Approaches. *Computing Systems*, August 1992.

[23] 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.