Pergamon

Information Systems Vol. 20, No. 7, pp. 557–577, 1995
Copyright© 1995 Elsevier Science Ltd
Printed in Great Britain. All rights reserved
0306-4379/95 $9.50 + 0.00

0306-4379(95)00030-5

ON BUILDING A HYPERDISTRIBUTED DATABASE†

ATHMAN BOUGUETTAYA[1], MIKE PAPAZOGLOU[1] and ROGER KING[2]

[1]School of Information Systems, Queensland University of Technology, Brisbane, Qld 4001, Australia
[2]Department of Computer Science, University of Colorado, Campus Box 430, Boulder, Colorado 80309, USA

(Received 15 July 1994; in final revised form 14 June 1995)

Abstract — Sharing data among disparate databases has so far mostly been achieved through some form of *ad-hoc* schema integration. This approach becomes less tractable as the number of participating database increases. Therefore, the complexity of making autonomous heterogeneous databases interoperate is dependent on adequately addressing the autonomy and heterogeneity issues. In this paper, we describe a prototype that implements an approach which addresses these issues in the context of large multidatabase systems. In particular, we describe a scheme that builds a *Hyperdistributed Database* using a two-staged approach. We also describe how conglomerations of databases are formed, modified, and evolved.

Key words: Multidatabases, Federated Databases, Interoperable Heterogeneous and Autonomous Databases, Query Languages

1. INTRODUCTION

The past few decades have witnessed an explosion in the number of databases. With this increase, there has been a renewed interest in sharing information across heterogeneous platforms. Unfortunately, many problems are hampering the achievement of this goal. Although one may *potentially* access a large number of databases, in reality this is an almost intractable task due to various fundamental problems related to scale, autonomy and heterogeneity [7]. In this information age, the need to share data across autonomous heterogeneous databases has become a necessity. Organizations across continents rely on a wide variety of databases to conduct their everyday business. In many instances, databases are designed from scratch if none is found to meet the organization requirements. This has led to a proliferation of databases obeying different set of organizational requirements and modelling.

Interoperability among heterogeneous databases has been researched extensively [25] [22, 36, 28]. In that regard, the *global schema integration* approach has been used to achieve interoperability among heterogeneous databases. This approach is known to be extremely tedious and error-prone. It becomes less tractable as the number of schemas grows. An important aspect in the integration process is the need to understand data semantics for the integration to take place [32, 20, 18]. This is the most challenging problem database researchers have been grappling with. The *federated* approach was recently proposed to address the issue of database autonomy. Through import/export schemas, it makes integration more tractable in larger multidatabase systems. However, this approach becomes less tractable as the network size grows [15, 33]. We introduce the *hyperdistributed* approach that is intended for very large environments of heterogeneous autonomous databases [7]. In this approach, site autonomy is respected and heterogeneity bridged with a reasonable amount of overhead. To this end, *coalitions*, *service links*, and *co-databases* are used [6]. Coalitions are a means to provide more information exchange at a higher overhead while service links are a means to provide sparser information exchange at a lower overhead. Overhead is measured in terms of the amount of information *details* that has to be exchanged. Co-databases are used to implement the coalition and service link concepts. The research presented in this paper builds on previous research and refine many of the ideas presented in [7, 6, 8]

This paper is structured as follows. In section 2, we provide an overview of the related work. In section 3, we describe the hyperdistributed database framework. In section 4, we provide an

†Recommended by Peri Loucopoulos

example depicting the use of FINDIT which implements the idea presented in this paper. In section 5, we describe the formation and evolution of the hyperdistributed database. In section 6, we give an overview of the system implementation. We finally overview the paper in section 7.

2. RELATED WORK

The federated architecture [15] [33] is the first approach to succeed in providing a reasonable amount of autonomy for individual database systems. The federated approach, however, is best suited for a small to medium number of databases because of its intrinsic architectural design. For further discussion about autonomy in distributed systems, we refer the reader to [13] and [33]. There has been a consensus in [31], [9], [12], and [35] that the above observations are the major open problems that stand against achieving interoperability among a large number of autonomous heterogeneous databases.

2.1. Autonomy and Heterogeneity

We first need to define the criteria used to compare the different approaches. For that purpose, we distinguish several types of autonomy and different levels of heterogeneity. In what follows, we define autonomy and heterogeneity.

2.1.1. Autonomy

There exists four main types of autonomy in heterogeneous autonomous databases [33] [11] [13]. The first type of autonomy is called *design autonomy.* The second type is called *communication autonomy.* The third type is called *execution autonomy.* The last type is called *association autonomy.*

Design Autonomy

Design autonomy is related to the way data is organized and accessed. In other words, a complete design autonomy is achieved whenever data owners are the sole authority that decide how the schema should be organized and accessed. Ideally, in an interoperable environment we would like to keep this autonomy intact. In this case, data owners make decisions on the organization and access regardless of the environment it is supposed to fit in. This is obviously unrealistic. In that regard, a compromise is necessary where some autonomy is sacrificed to allow a greater degree of data sharing.

Communication Autonomy

Communication autonomy is related to whether a database can independently decide to establish communications with other databases. This ability enables databases to make decisions as to how and when to handle a request from a foreign database. The communication autonomy also refers to the fact that participating databases are sovereign in deciding what information to provide, when, and where to.

Execution Autonomy

The execution autonomy is related to whether a component database can execute local operations without any interference from external operations [33] [11]. This also includes the ability to decide in which order an external operation is to be executed locally. Further, the execution autonomy provides the ability for component databases to abort and rollback external operations if these violate its consistency constraints. Local databases are not under any obligation to let foreign databases know about the execution order of their operations. This in fact provide an operational insulation from the outside environment.

Association Autonomy

Association autonomy is related to whether a component database has the freedom to associate or disassociate itself from a group of networked databases. This assumes that the grouping's minimal functioning is not dependent on any single component database. This is a very important aspect of *Open Systems Interoperability.* Meeting this condition is ideal but not realistic. Therefore, we expect that the association and disassociation will come after some form of negotiation had taken place.

It is important to keep in mind that though total autonomy is desirable, we cannot realistically expect that sharing could be achieved if some form of autonomy is not relinquished. However, we would like to make sure that only a minimal amount of autonomy is relinquished.

2.1.2. Heterogeneity

There are several levels of heterogeneity [2] [31] [37] [29]. The first level occurs at the physical level where different file systems are used. The second level occurs at the operating system level. These types of heterogeneity are dealt with within these areas of research. The third level of heterogeneity to overcome is at the DBMS level. At this level transaction management systems are the hardest to interoperate. The fourth level occurs at query optimization level. The fifth level of heterogeneity to overcome is integrating different schemas of the same or of different models. The sixth level of heterogeneity to overcome is making different query languages interoperate. The seventh level of heterogeneity to overcome is making different database applications (or views) interoperate.

2.2. Global Schema Integration

Most multidatabase systems to date provide data sharing through a global schema [36] [39] [38] [28] [25] [40] [24]. This schema is usually the result of integrating multiple schemas in a tight and static manner. In the schema integration approach, a global schema is necessary for executing global queries. Usually, it means translations of languages, schemas, or both. The global schema approach has two major advantages. On the one hand, queries that span many databases may be executed, and on the other hand transparent access to target schemas is provided. This is usually accomplished through a translation from the global schema to individual target schemas and by providing means for aliasing. So far, no automatic translation in schema update and integration has been successful. The main hindrance has been understanding and interpreting what different entities mean and somehow translate an understanding from one schema to another [3]. Another hindrance is the ability to maintain global schemas in the face of frequent component schema evolutions. In existing systems, translation and integration are done on an ad hoc fashion. Since databases were mostly in-house, autonomy concerns were not important. There was therefore no provision for database autonomy. Indeed, in order to participate in such a tightly coupled databases, individual databases have had no choice but to reveal their schema. Since there is one single central schema, decision making was centralized and performance bottlenecks were introduced. Figure 1 depicts the global integration approach.

The integration of schemas is a very tedious and highly manual job. It is also error prone. An error in the design could dramatically compromise the consistency of the databases. In addition, any modification is one of the participating databases schemas could trigger a major change in the global schema which in turn could make the participating databases unavailable for an unacceptable amount of time. Adding or removing a database schema may amount to redesigning a global schema from scratch which may be prohibitively expensive. The mapping between the global schema to the component schemas is also problematic. A change in the global schema may trigger unacceptable changes in component schemas. At the present, there is no automatic way of propagating these changes top down or bottom up. This difficulty stems from the inherent complexity of data semantics. Although this approach makes it easy for users to simultaneously use several databases, the overhead incurred to get homogeneity is high.

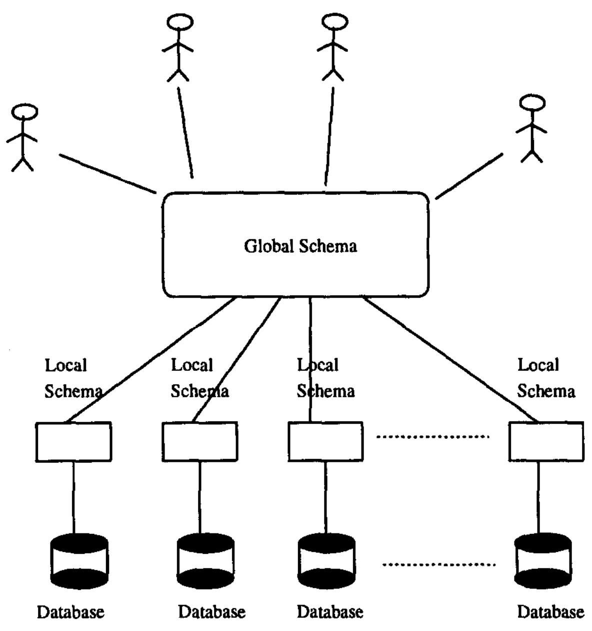


Fig. 1: The Global Schema Approach

In a small network of heterogeneous databases and within the same organization, schema integration is still a viable and reasonable solution. In this environment users are expected to know where the information is located within the global schema. This is best suited for those environments where all participating databases are owned by a single organization and where schema updates are not frequent. In this approach, views are good tools to secure data and customize it to users' needs. From users' point of view, it may be that it is unreasonable to assume users know the entire schema. It is worth mentioning that queries against the global schema cannot always deterministically (and automatically) be mapped into subqueries that span multiple component schemas. This problem is similar to the view update problem [17].

2.3. Federated Configuration

In federated databases [15] [33] [1], a certain amount of autonomy for individual database systems is maintained. In this approach, information sharing occurs through *import* and *export* schemas. A particular database exports part of or the whole schema depending on which database it is exporting to. The importing database has to do any needed integration locally. All databases and their available schema portions are registered in a federal dictionary. In the current state of federated databases, locating information is achieved in two steps. First, the requesting database consults the federal dictionary for existing databases and available schemas, and second, imports all known schemas (whenever possible) and browses through them for a requested information type. Once this is done, a negotiation is initiated with the exporting database to actually query this information type.

There are some problems with this approach. First, finding the right information in a large unstructured network of data dictionaries is not feasible. Second, databases have to show all pieces that they export to the federation by storing them in the federal dictionary, which obviously violates database autonomy. In addition, in case a database agrees to share its schema, the importing database has to understand the intrinsic organization (i.e. semantics) of the imported schema. If there are tens (not even hundreds or thousands) of such schemas, this scheme is at best impractical for locating information. It should be noted that the federated approach has been designed as a better alternative to the global integration approach and, to a large extent, has succeeded in doing so. The idea was to provide some flexibility in terms of integration and autonomy. For instance, integration is done *on demand* instead of being decided regardless of component database needs and concerns. Because of this flexibility, this environment enables more databases to share data. Figure 2 depicts a visual representation of the federated approach.

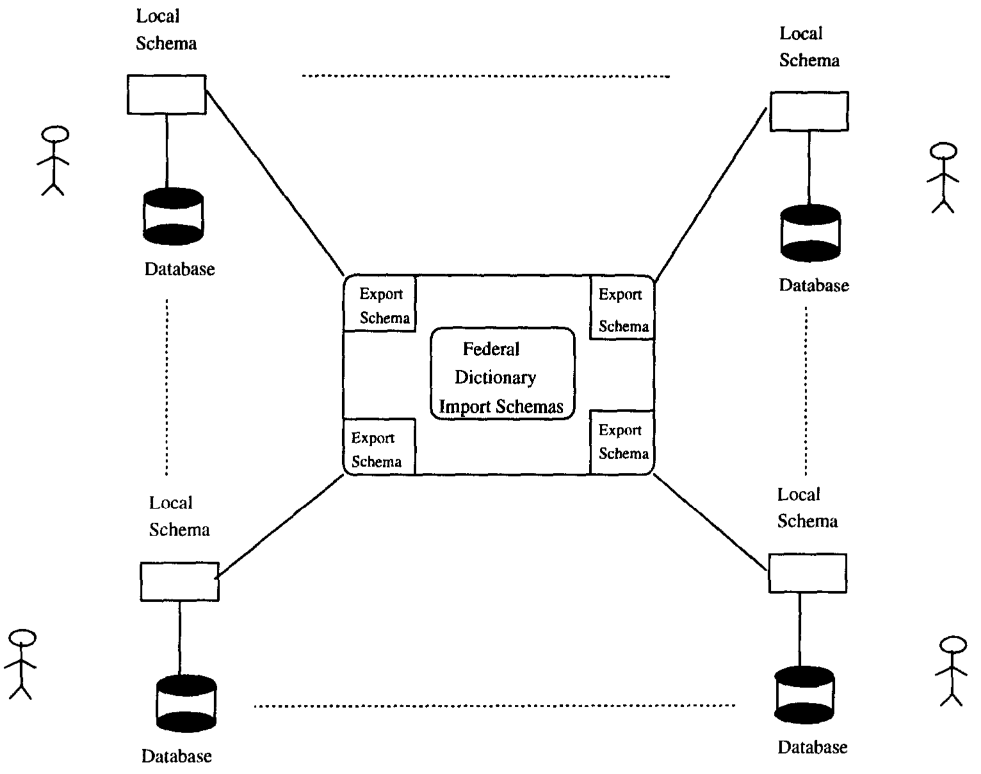


Fig. 2: The Federated Approach

On the downside, this approach makes it difficult for users to access all databases participating in the federation. Instead, they can access at most, few databases at any point in time. This hindrance is due to the intrinsic structure of the federated architecture. The import/export schema implies that the integration can only take place between pairs of databases. This would mean that this point to point architecture may not be scalable. While the *incremental* and *on demand* schema integration increases flexibility, it makes scalability more difficult to achieve. In fact, one–to–one (or point–to–point) schema integration increases the difficulty of achieving global sharing as compared to global integration. Further, the federated approach does not seem to provide a framework for data sharing among *already* integrated schemas. In the current state of research, every data sharing endeavor may have to start from scratch [15].

Information related to import/export schemas is centralized. Thus, database administrators are forced to relinquish some control over part of their schema to the benefit of a central authority. In a large environment, this is obviously highly unlikely to happen. Papers describing the federated approach do not give details about the federated dictionary and its intrinsic architecture [33] [15]. It is our assumption that from the services the federal dictionary provides, it can only be a special purpose database. This means that every participating database has to first be integrated with the federal dictionary (database) before any other integration could take place.

From users' point of view, information is more easily manageable. Integration is done upon their request and on–demand basis. This means that views may be constructed more easily. Users are not faced with a large space of information that they would have to sort out themselves. Once, users know what information they need, they could readily query the integrated schema. However, whenever users need a piece of information, there is a need to query the federated dictionary. If the component database is found, a request is sent for a possible integration with the local schema. If the answer is positive, the actual integration then takes place. In a very large federation, this would hardly be tractable.

3. THE HYPERDISTRIBUTED DATABASE FRAMEWORK

We present a two–level approach to bridge heterogeneity and respect database autonomy as much as possible and yet conceptually build the Hyperdistributed Database out of existing databases. Users are incrementally and dynamically educated about the available information space without being overwhelmed with all available information. The two–level framework provides participating databases with a flexible means of sharing information. The system that implements this approach is called FINDIT.

The two–level approach we suggest in this research corresponds to *coalitions* (first level) and *service links* (second level). Coalitions are a means for databases to be strongly coupled whereas service links are a means for them to be loosely connected. *Co–databases* † are introduced as a means for implementing these concepts and as an aid to inter–site data sharing. In Figure 3, we present an example where databases are grouped into coalitions and service links. For instance, the coalition *Chinese Restaurants* groups four databases that share description about this type of information. This same coalition provides a service link to another coalition, *Restaurants in the US*, that could be used by this coalition under the terms and agreements between both coalitions.

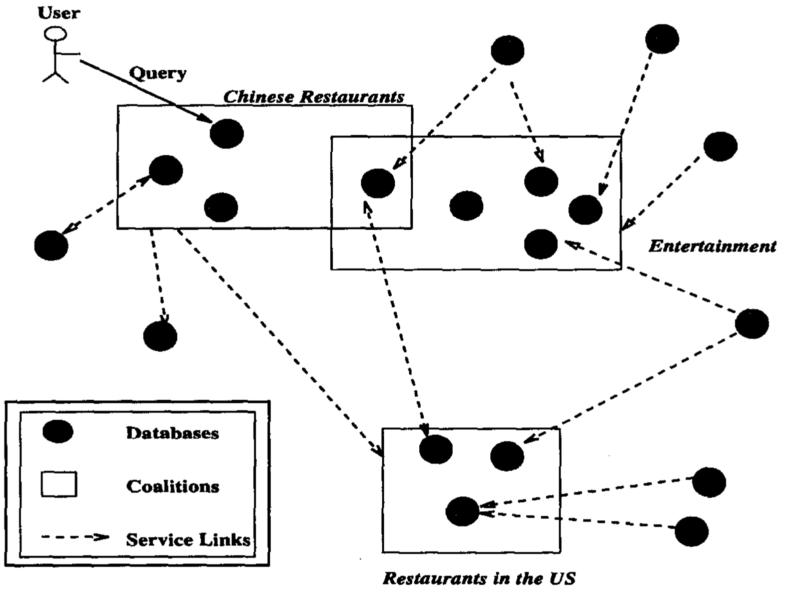


Fig. 3: Example of Coalitions and Services

3.1. Coalitions

Coalitions are groupings of databases that share some common interest. This consists of an interest in an information meta–type (e.g. *Restaurants*). In that context, databases will share descriptions about this type of information. A close analogy to our concept of coalition is how political systems work in western democracies. In many countries, political parties obeying different ideologies agree on a minimum set of objectives for a limited period of time. The result is a coalition. Therefore, coalitions are based on a short term interest. In political coalitions, members keep their autonomy intact while being committed to a set of rules that are agreed upon. Our concept of coalitions is therefore very close to the concept of political coalitions. On the other hand, the concept of federation is more like the federation of states where the set of rules are long term and where states enjoy a reasonable, though limited, amount of autonomy.

Another example of grouping close to the concept of coalitions is the Internet [30]. The Internet is a computer network that consists of subnetworks that are connected to each other. Every

†We select to name this database a co–database because it is an accessory to the actual database like a co–processor (FPU) is to a processor.

subnetwork has its own set of standard protocols for communicating. Although, unlike FINDIT, all subnetworks provide almost the same set of information, the idea of a cooperative environment exists. Within every subnetwork, the participating sites obey a set of rules that govern communications among themselves. Subnetworks are usually set up to serve a certain purpose within some geographical boundaries. For instance, NSFnet is a network whose purpose is to link major research institutions in the US (geographic boundary) to conduct research using supercomputers (purpose). Another instance of a subnetwork is DECnet where the boundary is the company (organizational boundary) and the purpose is to provide a framework for their researchers to share information. Every network is linked to other networks mainly to exchange electronic mail.

3.2. Services

Service links are a simplified way for databases to share information. They allow sharing with low overhead. Service links usually entail some binding agreements that are similar to *contracts* [16]. In terms of data sharing, the amount expected to be exchanged in a service link will typically involve a minimal amount of information. For instance, only the name of the information with few synonyms and information about the database that is exporting this information are expected to be shared. In that respect, service links incur a low overhead as compared to using coalitions.

Service links may occur between any two entities. Service links can only be of one of three types. The first type involves a service link between two coalitions to exchange information. The second type involves a service link between two databases. The third type involves a service link between a coalition and a database. A service link between two coalitions involves providing a general description of the information that is to be shared. Likewise, a service link between two databases also involves providing a general description of information that databases would like to share. The third alternative is a service link between a coalition and a database. In this case, the database (or coalition) provides a general description of the information it is willing to share with the coalition (or database). The difference between these three alternatives lies in the way queries are resolved. In the first and third alternative (when the information provider is a coalition), the providing coalition takes over to further resolve the query. In the second case, however, the user is responsible for contacting the administrator of the providing database to know more about the information.

Whenever databases are reluctant to show too many details of the information type they contain, they have a choice to join in a service link with other databases or coalitions. In essence, service links are means for databases to be loosely connected to other databases (or coalitions). This provides a framework where databases exchange a minimum amount of data about information they would like to share. Only general information about actual information to be shared and information about the servicing databases are exchanged. It should be stressed that as in any service link, there is a service to be provided by one of the entities involved, i.e., coalitions or databases.

3.3. Why Coalitions and Services?

Coalitions and service links are two mechanisms by which databases participate in sharing information in FINDIT. It is important to understand how these two concepts differ. We will start by giving a philosophical explanation and then continue with a more technical description of the differences.

Technically, the nature and environment of coalitions and service links are different. For instance, the establishment of a coalition may involve an arbitrary number of databases, whereas the establishment of a service link involves only two entities (either a database or a coalition). Also, any participating database in a coalition must share a certain information type with all other databases that are members of the coalition. In a service link, the exchange of information is directional. This means that a database that is member of a service may be a provider or a consumer but not both at the same time. Another major difference between service link and coalitions is the importance of geographic proximity. This is an essential factor in coalitions (in addition to information relatedness) where it makes more sense to form a coalition within defined geographic boundaries. This condition is not so important for service links where the flow of information is probably not as important as in coalitions.

Because of the closeness of the research goals, we compare the coalition and service link constructs to those provided in the federated approach. The federated architecture main constructs are the import and export schemas along with the federated dictionary. The import and export schemas rely on the federated dictionary as a means to sharing data. The unit of data sharing is the schema. Databases record their export schemas in the federal dictionary. It is clear therefore that the federated approach implicitly requires a substantial amount of cooperation, and hence overhead, for sharing information. More importantly, the size problem is implicitly ignored. As a consequence, issues related to educating users about available information and information sources were simply not addressed. These problems are fundamental towards achieving an interoperable environment in a large number network of databases.

In contrast, coalitions and service links are constructs that use as basic unit of sharing *single* information types like *Restaurants*, *Movie Theaters*, etc. These constructs are designed to respect database autonomy by providing a higher level abstraction for data sharing. This in fact means that sharing takes place at the meta–data level. While federations goal is general (sharing *all* information types), coalitions and service links goals are sharing *specific* meta–information. Further, the federated approach requires centralized bookkeeping while service links and coalitions do not. Moreover, keeping track of export schemas requires the agreement of *all* participating databases while the FINDIT architecture limits the agreements to coalition members.

3.4. Documentation

One of the major problems that solutions to heterogeneity have not adequately succeeded in addressing, in our opinion, is the problem of understanding the different structure and behavior of information as represented in various databases. A critical problem has been to determine how an entity from a schema fits in the globally integrated schema. This obviously requires understanding the structure and semantics of the different entities. At this current state of research, the only reasonable alternative has been an *ad-hoc* integration. In this approach, database administrators are responsible for understanding the different schemas and then mapping that understanding into a model well understood by local users. It is important to note that this process could be best performed for a few number of schemas that are not very dissimilar. Recently, there has been a renewed interest in solving semantic heterogeneity [21] [20] [34].

Our approach tries to incrementally help users make decisions about the information they need. In particular, we use the concept of *data demonstration* to help users select the information that meets their requirements. In a nutshell, users would likely be satisfied with just a *demonstration* about the information of interest. We believe that this approach combined with other traditional help approaches is the key to educating users about the space of information available to them.

3.5. Co–databases

Co–databases are object–oriented databases attached to every participating database. Object–orientation has proved to be an excellent modelling paradigm for complex structure and behavior. Inheritance and encapsulation have proven to be critical in designing reliable and easily maintainable software systems [26].

A co–database schema consists of two subschemas. Each subschema represents either a coalition or a service. Each sub–schema consists of a lattice of classes. Each class represents a set of databases that can answer queries about a specific type of information (e.g. queries about *Restaurants*). The service subschema (left side of Figure 4) consists of two subschemas, the first is a subschema of services the coalitions it is member of have with other databases and coalitions and the second is a subschema of services the database has with other databases and coalitions. Each of these subschemas, in turn, consists of two subclasses that respectively describe services with databases and services with other coalitions. The coalition subschema (right side of Figure 4) consists of one or more subschemas where each one of them represents a coalition.

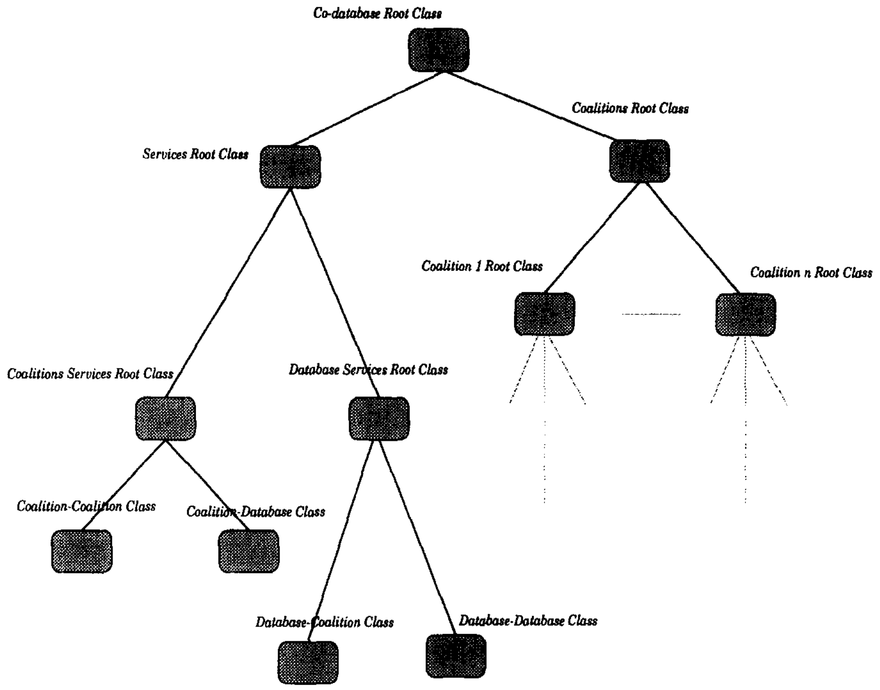


Fig. 4: A Skeleton of a Typical Co–database Schema

Description about coalition servicers includes information about points of entries and contact with those coalitions. Other descriptions provide information to local databases so the best point of contact can be chosen. It should be noted that the subschema representing the set of coalition servicers will be the same for all databases that are members of the servicing coalition.

4. A FINDIT EXAMPLE

Figure 5 depicts an example of the hyperdistributed approach. In this example, there are 5 coalitions: *Research in Industrialized World*, *World Religions*, *Databases*, *Public Software*, and *Commerce between the US and other Countries*. For instance, the databases participating in the coalition *Research in Industrialized World* share descriptions of the information type *Research in Industrialized World*. These descriptions are stored in co–databases of each participating database. Service links are also depicted in this example. For instance, the service link between coalition *World Religions* and the coalition *Commerce between the US and other Countries* is an agreement between these two coalitions. In this instance, Coalition *World Religions* provides a minimal description of the information type *World Religions* to coalition *Commerce between the US and other Countries*. This description is also stored in the co–database of the importing database. This figure also shows that a database may be part of more than one coalition. For instance, there is a database that participates in both coalition *Research in Industrialized World* and coalition *World Religions*.

Data and database locations are easily determined due to the two–level approach of the hyperdistributed scheme. In this approach, databases keep track of coalitions they are member of as well as service links they are part of. This enables a dynamic and incremental approach in determining data and database location. Likewise, users locate relevant information using the object–oriented hierarchy within a co–database and using remote co–databases that are related through a coalition or service link. For instance, a user of coalition *Research in Industrialized World* may submit a query about public domain software. In that case, the coalition, after a local search is performed, would query its service links about this type of information. In this example, a service link exists between this coalition and coalition *Public Software*. In this instance, the query is resolved there.

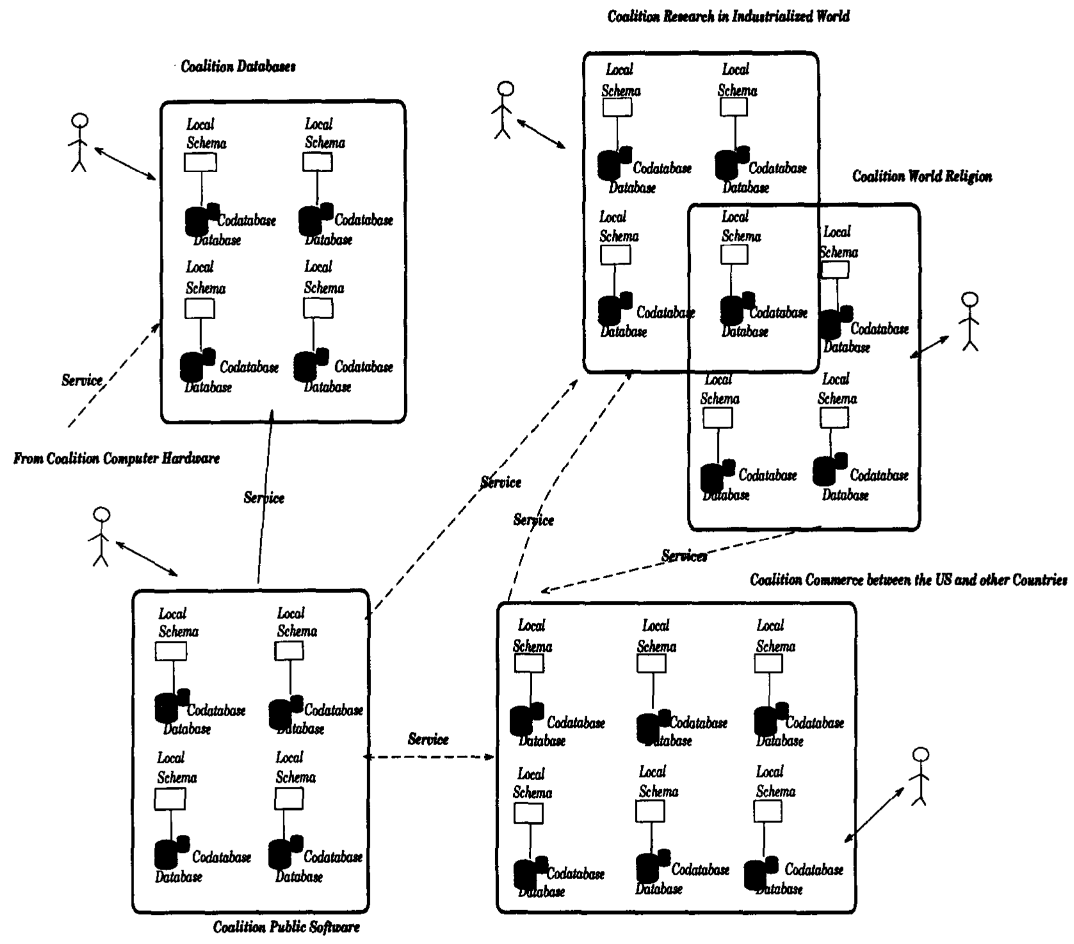
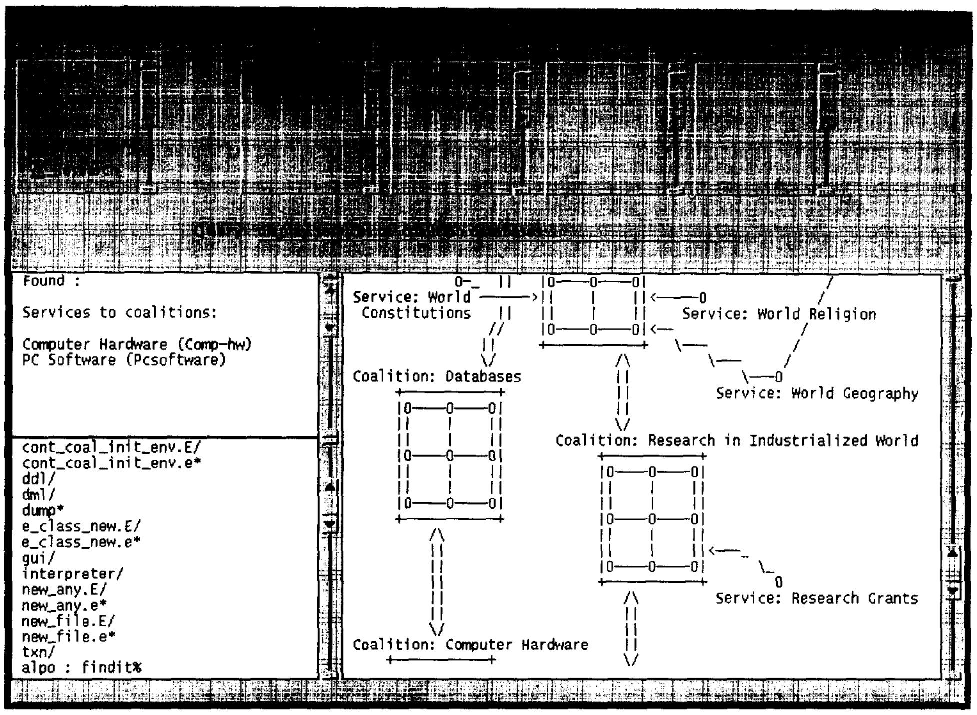

Coalition Research in Industrialized World
Coalition Databases
Coalition World Religion
From Coalition Computer Hardware
Service
Local Schema
Codatabase
Database
Coalition Commerce between the US and other Countries
Coalition Public Software

Fig. 5: The Hyperdistributed Approach

Figure 6 shows a FINDIT window box. An example of a FINDIT query is also shown. The inset shows a partial picture of "nearby" coalitions and databases. The query is a request to display all service links that are linked to the coalition named *Databases*. FINDIT finds two service links to coalitions. These service links are to coalitions *Computer Hardware* and *PC Software*.

The decentralized and flexible approach that the hyperdistributed approach uses, i.e., the two stage concept (coalitions and service links), enables databases to dynamically know about, and form flexible groupings with, other databases. This approach also allows databases to participate in this scheme with relatively minimal overhead. Databases can either group into coalitions or service links, depending on the overhead that can be tolerated. Joining a service link or a coalition does not entail any significant overhead given the complete decentralization of the hyperdistributed approach. In addition, prospective databases are offered a flexible way to join. They can either be part of a coalition or a service link (or both) without much penalty to the coalition/service link or the joining database. It is very reasonable to assume that a database can be part of coalition *Research in Industrialized World* and *World Religion* as it may store data about research conducted in industrialized world and world religions.

Understanding remote information is is central to the concept of sharing. *Documentation* tries to address this specific aspect. It is a means through which databases understand what foreign information means locally . In this case, the data model in which this information is represented is irrelevant. For instance, pieces of information may be represented using the same data model but still be heterogeneous in their semantic content. This is usually the case as no data model can (exactly) semantically represent an entity in any universe of discourse. Documentation is, therefore, a powerful concept that has the important advantage of adjusting the meaning of foreign


Service: World Constitutions
Service: World Religion
Coalition: Databases
Service: World Geography
Coalition: Research in Industrialized World
Service: Research Grants
Coalition: Computer Hardware

```
Found :

Services to coalitions:

Computer Hardware (Comp-hw)
PC Software (Pcsoftware)
```

```
cont_coal_init_env.E/
cont_coal_init_env.e*
ddl/
dml/
dump*
e_class_new.E/
e_class_new.e*
gui/
interpreter/
new_any.E/
new_any.e*
new_file.E/
new_file.e*
txn/
alpo : findit%
```

Fig. 6: An Example of a Textual Query

information to local understanding.

Data sharing is done at the meta-schema level and no actual data mapping is performed. Databases only share descriptions of information types with other databases. In addition, a database shares only the information that it wishes to share. More precisely, since we are using an object-oriented model, the participating databases have to only contribute a description about *one* single type of information (in the object-oriented sense) to each coalition they participate in. Therefore, only the bare minimum of information descriptions is mapped. We feel that this is an optimal way of sharing information. This flexible and incremental approach provides a mechanism by which heterogeneity is dealt with on the databases own terms. For example, databases participating in the coalition *World Religions* would share the descriptions of the types of information they store. These, for instance, could include language of the movies, countries, ratings, etc.

In the hyperdistributed scheme, a coalition of databases may opt to implement a complete schema integration while another coalition may opt to implement a federation while yet another may want to share information at the meta-schema level (as described above). It is interesting to note that the global schema and federated approaches are *both* implemented using the hyperdistributed approach. The hyperdistributed scheme is therefore powerful enough to model these two data sharing approaches.

For instance, databases in coalition *World Religions* must divulge relatively elaborate information descriptions and share these with the other participating databases. Furthermore, these databases must share information pertaining to the databases themselves that may include the local query language, DBMS, data model, and other descriptions. In contrast, a service link would require less overhead (in terms of amount of descriptions shared). For instance, the service link between coalition *Commerce between the US and other Countries* and coalition *World Religions* may involve giving just the name of the information type and its synonyms and an internet address to access the coalition *World Religions*. This has the potential benefit that very little information is divulged. The combination of these two constructs provide a flexible approach for databases to share information.

5. FORMATION AND EVOLUTION OF A HYPERDISTRIBUTED DATABASE

In this section, we describe in some details how the parts (coalitions and services links) of the Hyperdistributed Database are formed and evolved. The Hyperdistributed Database is not directly formed or evolved but rather its coalitions and service links are the atomic entities that get updated. In particular, the Hyperdistributed Database evolves whenever new databases become part of a coalition or a service link. In this case, changes are introduced in the different co–databases. This scheme does not require any central decision for new databases to join coalitions or service links. Instead, decision making is highly decentralized as new databases do not need the approval of all database members or any central authority. They only need the approval of those databases that are directly concerned.

As pointed out earlier, coalitions are meant to provide a mechanism by which databases can "unite" based on a short term interest. Therefore, any database can change its decision based on its proper evolving needs and interests. In this respect, a database can resign from FINDIT altogether or just resign from one of the coalitions it is member of. A database can also change its status from a member of a coalition to a service provider of a certain database or coalition. Another change of status might be a database changing its status from a service provider to a member of a coalition or both.

If a query returns the empty set, this answer works as a trigger to evolve the Hyperdistributed Database. This is a positive result since query failures work as a means for FINDIT to increase its knowledge about other information providers. It is likely for users to ask about information that are not in the local database domain of interest. If these requests are small, a mapping between a set of information types to a set of database (coalition) service provider is enough to resolve the query. If the number of requests start to get large on a certain of information that is foreign to the local database domain, the database may wish to invite those "popular" databases that contain the information to join in an existing or new coalition. This database may also initiate a negotiation with other database members of the same coalition to establish a service link with an existing coalition.

5.1. Rules for Joining and Leaving the Hyperdistributed Database

FINDIT offers a flexible way for databases to join and leave the system. A database joins the system by either being a service provider or a member of one or more coalitions. No central authority is required to decide upon the membership of a database. Likewise, a database resigning from the system does not require any central authority approval either.

For any database to enter or leave a certain coalition, it has to fulfill the requirements set by that coalition. In this respect, every coalition in FINDIT has its own set of requirements that defines the rules governing membership. If a database wishes to become a member of a coalition, it must to provide an agreed upon amount of information about the data it would like to share in addition to information about the database itself (like type of DBMS, data model, query languages, internet address, etc). The administrator of that coalition will then decide how the coalition schema is to be changed if the new database is accepted as a new member. One of the pre–conditions that a new member has to fulfill is to install the whole FINDIT package before it starts operating. Also, if a database member would like to resign from a coalition, it has to notify the coalition administrator so that to carry out possible changes in the coalition schema. Every coalition sets a membership threshold (i.e., number of participating databases) as a condition for the continuation of its existence. Below this threshold, the coalition is dissolved and other coalitions that it provides a service to are notified.

Because a database may belong to more than one coalition, membership to or resignation from a coalition does not necessarily influence membership in FINDIT. The reason being that a database resignation from a coalition does not mean a resignation from all other coalitions it is a member of. Likewise a database becomes a member of FINDIT by joining a coalition the first time. A database that is not a member of any coalition is either loosely connected to FINDIT (part of one or more service links) or not a member at all.

It is worth to note that the Hyperdistributed Database does not have a single "origin." Instead coalitions are formed and dissolved independently from each other. This is an important aspect of FINDIT as this offers a great deal of flexibility. This means that coalitions can form independently of the overall state of FINDIT. Therefore, there is no need to keep track of FINDIT's overall state as a requirement for system evolution. However, in many instances, it may be desirable to do so for various fine tuning and administrative purposes. Not having this information should in no way influence the function of the system. Therefore, at any point in time, FINDIT may consist of coalitions that are not connected to each other in any way whatsoever. Within this framework, two coalitions might deal with the same information type and still be unconnected for reasons that have to do with geographical location and convenience among other reasons.

5.2. Coalition Formation and Update

We describe some of the important protocols by which coalitions are formed and updated. The formation of FINDIT itself is indirectly accomplished through the formation of coalitions. In this respect, coalitions are formed concurrently without requiring prior arrangements or agreements from other coalitions. Therefore, FINDIT may consist at one point in time of coalitions and service links of databases that are mutually exclusive. Those protocols represented by functions provide a high level abstraction to clarify the functionality that FINDIT is supposed to perform. Therefore, they do not necessarily directly correspond to the implementation and functionality that is provided to users.

5.2.1. Formation

The design of a new coalition obeys a set of guidelines which provide a framework for sharing information. These guidelines are in no way set to dictate a particular protocol for coalition establishment. As a measure of flexibility and a respect of component autonomy, databases have the latitude to decide which mechanism is best for achieving the establishment of a coalition. The set of operations that come with FINDIT are there to provide a default set of protocols for coalition formation.

Initially, a database administrator is chosen to create the root class of the schema. Once this is done, the root of the schema is sent to every participating database administrator for validation. If the operation is not validated, the validator sends the edited object to the creator of the object. Based on this feedback, the creator will decide whether to change the object. If the feedback requires some changes, the originator of the operation will change the object and send a message for validation again. This process will continue until there is a consensus among the database administrators. The next operation to be performed is also decided based on a consensus among the participating databases. The originator of an operation is determined using a predefined order. In case an object is created at a site, all changes are made persistent on that site. Once operations on that object are complete, the site manager will propagate the new object to other participating databases. Therefore changes to an object are not immediately propagated to other databases. Instead it is only when an object is ready that propagation takes place. In the process of establishing a coalition, database administrators are responsible for designing the initial schema that represents their coalition. Once the schema and its extension are defined, all participating databases in a coalition will have an exact copy of the schema representing that coalition. Since a database may belong to more than one coalition, its co–database schema may consist, in this case, of copies of schemas from different coalitions it is member of. In this instance, the local database administrator is responsible for creating a global schema that would contain all coalition schemas. As stated earlier, each coalition may have service links with other coalitions. In this case, a schema to describe those coalitions is created. This schema is exactly the same in all member co–databases.

There are some important assumptions that are made with regard to establishing coalitions. For instance, any formation of a coalition assumes that participating databases know the existence of each other. All participating databases are also assumed to be accessible through some common computer network. There should also be a prior understanding of what information type is to be shared. All participating databases are assumed to have the FINDIT system installed before any formal negotiation takes place. During this informal exchange, many parameters need to be set. For instance, a threshold for the minimum and maximum number of member databases is discussed and set. Likewise, a threshold on the minimum and maximum number of service links with databases and coalitions is also set. All operations are performed in a synchronous manner. This means that proper receipt of operations is assured. This has the disadvantage that it will increase the number of messages by the number of participating databases. Since creation of coalitions happens only once and frequency of updates is not expected to be high, we feel that the overhead incurred is kept at a reasonable level. It should be noted that these operations are privileged and as such are only accessible to database administrators.

Add(Coalition, Class, {Super—Sub}, Set_of_Classes, Originator)

This operation takes a coalition and adds a class to it. This class is a subclass or superclass of a set of classes. This operation will also call the create and the validation operations before it is propagated to the participating databases.

Create(Coalition, Originator)

This operation creates the root class of the schema representing a certain coalition. It takes two parameters: a coalition name, and the name of the database originating this operation. The originator defines the structure and the behavior of the class based on his understanding. The originator then submits it for the approval of the other database administrators.

Edit(Coalition, Object, Originator)

This procedure uses an editor to make changes to an object (class or instance). If it is required, the originator will then submit those changes to the creator of the object for approval. Based upon a predefined consensus, the changes will either be partially or completely implemented, or just discarded. If there is no requirement for third party approval, the changes are implemented.

Instantiate(Coalition, Class, Object, State, Originator)

This operation takes a class and instantiates it. This object is then initialized with the state that defines the values that this object will take. In most cases, this will not require any validation as most likely the originator would be the database described by this object. It has only to propagate the new object to other databases.

Propagate(Coalition, Object, Originator)

This operation is executed after each creation or modification of an object, be it a class or an instance of a class. This operation is an internal mechanism that is invisible to users. Object updates are propagated to sites that have the same objects.

Remove(Coalition, Object, Originator)

This operation takes the name of an object and removes it from the coalition. This object can either be a class or an instance of a class. Unless the originator is the database described by the object, this process is subject to validation from the other database administrators. Once there is a consensus, this update is propagated to other databases.

Set_Members(Coalition, Originator, Text, Add_Member)

This operation is initially sent to a set of database administrators. The parameters are the coalition name, the name of the database where the message originated, text explaining the goal of this message, and a set of participating databases. The originating database is in charge of collecting the participating database names and initiating the negotiation for forming the coalition.

Validate(Coalition, Operation, Alternative_Operation, Status, Originator)

This operation takes the prospective operation and returns a status that is either affirmative or negative. If it is affirmative, no further action is required. Otherwise, the disagreeing party will either do nothing or provide an alternate operation.

Each operation described above has to be validated by all participating database administrators. There is an exception with the instantiation operation where the database described by an object is the one that decides what the object state should be. If there is disagreement in the validation process, the originator of the operation will choose the course of action to be taken.

5.2.2. Coalition Update

An instance of coalition update occurs whenever a new database joins. The coalition schema may be updated intensionally or extensionally or both. If the update is extensional, the update consists of a class instantiation (object creation). In this case, the instantiation is initiated by a database coalition member. After the object has been created, this database sends a message to every participating database. This message contains a request for a certain class instantiation and provides the values of every attribute of that class. Creating a class instance automatically triggers sending a message to the other participating databases.

The other update (intensional) involves the subschema that describes coalition service providers. As with the schema representing database members, this schema is also subject to evolution changes. In what follows, we only describe the changes introduced upon the schema representing the database members of a coalition. These changes are similar to those that are introduced upon the schema representing the coalition service providers.

Throughout its lifetime, a coalition experiences many changes, including addition and deletion of members, and internal changes. Database members may also change their corresponding object in the schema. Another major change that may occur is the decision to dismantle it altogether. In the first case, any type of change to the structure of the coalitions is performed upon members' agreement. In this instance, one agreed upon database member introduces the changes in its local schema, then broadcasts them to the other members. During this process, only the sub–schema representing the coalition in question is locked. The agreed upon database member first sends a request to other database members to lock their local sub–schema that corresponds to the coalition in question. It then performs the changes in its local schema and sends those changes to the member databases.

The second kind of update involves a state change. In essence, every database is responsible for its corresponding object in the coalition. Any changes made to an object are the responsibility of the database that "owns" it. Prior to any changes, the database owner sends a message to every participating database to lock the object to be changed. After an acknowledgment from those databases, the local database administrator proceeds with implementing the changes. The update is then sent to every participating database.

In the third case, a coalition is dismantled by deleting the whole corresponding subschema in every participating database schema and notifying all coalitions with which there is a service link that the coalitions no longer exists. Local schemas are updated by their administrators. The decision to dismantle a coalition is made by the participating databases whenever there is no need for that coalition any longer. In the same process all objects that belong to the classes of that coalition are also deleted.

5.3. Service Formation and Update

The update of co–databases resulting from service link changes is practically the same as defined for coalitions, the only difference being that changes in coalitions obey a stricter set of rules. For instance, a service link between a database and another database or a coalition can easily be discontinued with no prejudice to the responsible party. Instead, the servicing entity is the one that assume all changes resulting from this service link discontinuation and accordingly change its co–database. Therefore, we will not describe in details how service links are updated. It is important to mention that service links that are established between two entities entail some conditions that both entities have to implement. For instance, an entity may not want to have the name and/or the description of the information type it is sharing to be disclosed to any other entity other than the one it is exporting it to. This would be a binding condition and the receiving entity must implement this condition in its co–database.

5.3.1. Service Formation

Database–Database Service

In this type of service link, at least one of the databases has to be a member of FINDIT (coalition or service link). Otherwise, this service is considered to be outside the scope of FINDIT. If the service link is unidirectional and the direction of information is from a database member of a coalition to a database that is not a member of FINDIT, the negotiation framework for information exchange is left to the discretion of the databases. However, if the databases are both members of FINDIT and therefore have the system installed, the negotiation is simply a series of *inquire* and *send* messages that describe the nature of the information to be shared. Once the negotiation is complete, the database administrator implements this new information in its co–database.

Database–Coalition Service

If the database is not a member of FINDIT and the information exchange is from the database to the coalition, the negotiation protocol is determined by the two parties. Once there is an agreement on the information to be exchanged, the information description is broadcast by the coalition representative to the other participating databases. If the direction of information exchange is from the coalition to the database, this transaction is outside the scope of FINDIT. If the database is a member of FINDIT, a negotiation takes place between the two parties. Like the first type of service link, the negotiation taking place between the database and the coalition (through its representative database) is a series of *inquire* and *send* protocols. Once the information description is agreed upon, it is implemented in the database that plays the role of coalition representative and later distributed to the other databases. Before then, the coalition representative has to make a decision how to change the coalition schema with this new information.

Coalition–Coalition Service

The negotiation process takes place between two databases representing two coalitions. Only one direction of information exchange is negotiated at one time until it is complete. As in the first and second type of service link, the negotiation protocols are a series of *inquire* and *send* messages. Once the negotiation process is complete, the database disseminates the necessary changes to the other databases.

5.4. Data Sharing

Although the described hyperdistributed approach does not enable sharing of actual data instances, it provides the mechanisms and hooks for this to be implemented. For instance, once the type of information and coalition (or service link) are located, a functional approach enables users access and construct views from databases [41]. A wide leverage is left to the participating databases to organize the way data should be accessed. For instance, a coalition may implement this direct data sharing through a local *global schema* integration. Another coalition may implement it using a local *federation.* The architecture of the hyperdistributed approach is meant to be a default that databases could start with but that could later be discarded and replaced by a more convenient approach. This could be done in isolation of any major side effects on the hyperdistributed database.

In the hyperdistributed approach, autonomy is substantially respected with little loss of data sharing quality. The different level of data sharing granularity (types, schemas, actual data) added to the flexible architecture using coalitions and service links takes the meaning of autonomy a little further. Sharing of information is based on the database own terms and interests. The other major advantage of the hyperdistributed approach is the respect of autonomy. Databases are no longer faced with the dilemma of sharing either everything or nothing. In being "forced" to share, databases must give up part or even all of their autonomy. This autonomy violation may take several forms including autonomy of decision, autonomy of data organization, and autonomy of manipulation. Concerns about site autonomy are of prime importance and take a new dimension when databases belong to different organizations. Therefore, any reasonable framework must acknowledge that the overhead incurred ought to be proportional to the degree of sharing.

6. SYSTEM IMPLEMENTATION

The first prototype of FINDIT is currently operational. This prototype is implemented on a local area network of Sun sparcstations. The implementation language is Eiffel [27] although *C* was used for implementing the low level modules. Although the Eiffel language lacks features to directly perform schema updates, its extensive library along with its intrinsic simplicity and flexibility proved to be an overwhelming advantage for the rapid prototyping of FINDIT. The graphical interface was also implemented using Eiffel libraries for *X* windows. The crucial library that we felt was missing from Eiffel's collection was a concurrency control library. Despite these shortcomings, Eiffel proved to be an excellent choice as a language of implementation.

The co-databases design of FINDIT follows the same philosophy as the underlying Eiffel constructs. This was chosen out of our belief that Eiffel provides a good object-oriented paradigm. The syntax and semantics of classes and instances of the co-databases are the same as defined in Eiffel. Since Eiffel was not meant to be a database programming language, we had to use "unorthodox" methods to get around, among other things, difficulties related to dynamic schema changes and active behavior.

Some classes in the co-databases will have some predefined attribute names to make the search for certain information reasonably easy to conduct. For instance, the root classes of coalitions and services will contain a predefined set of attributes to be used by the browser. These attributes include an attribute for synonyms for a certain class name, an attribute for services of a certain class, and so on.

The FINDIT architecture is divided into eight main modules as shown in Figure 7. These modules are the graphical interface, the interpreter, the browser, the transaction management system, recovery, access control, schema evolution controller, and the DDL and DML primitives. Some of the modules use primitives defined in the data manipulation and definition primitives module.

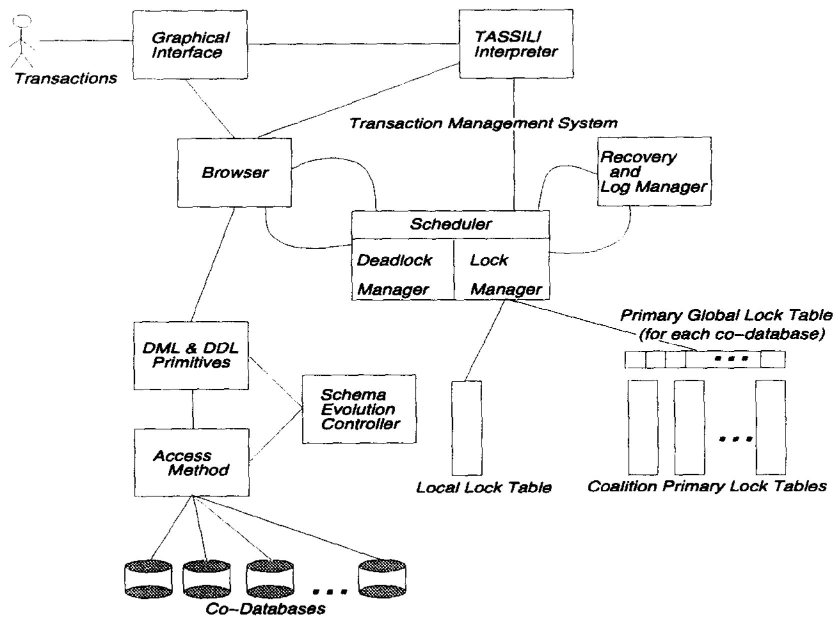


Fig. 7: The FINDIT Implementation

6.1. *Graphical Interface*

The graphical interface is implemented on top of FINDIT. Its purpose is two-fold. The first goal is to provide a user-friendly interface to FINDIT. The second goal is to provide users with iconic queries as an alternative or complement to textual queries. The graphical interface is designed to run on workstations using the X windows protocol. The interactive part consists of a series of windows providing the following functions:

1. Control the use of iconic queries.
2. Choose an object to display (be it a class or instance).
3. Display an object.
4. Operate a command line query.
5. Create and store a script corresponding to an icon.
6. Display on-line help.
7. Display the current or history of, query execution.

The graphical interface is a toolkit, written in Eiffel, that accomplishes tasks such as iconic queries, creation of scripts, and the display of FINDIT data. A user opens a session which is designated by the user's name and a session window is created. From this window, the user can use Tassili to query FINDIT. Along with the session window, a window manager will pop up along with an iconic manager from which users can make iconic queries. Any queries that users make, textual or iconic, are submitted to the Tassili interpreter. The interpreter calls the graphical toolkit display routines corresponding to Tassili display instructions. Each display window in the toolkit is interactive so that the user may select any piece of desired information. Display class windows show users the various attributes of a class and allows further investigation of information types. Display class also displays subclasses for that class. The display coalitions lists all coalitions that a database is a member of or has service with and allows users to choose a coalition and view it. This is similar to displaying services. The documentation windows offer a choice of displaying information textually, visually, audibly, or any combination thereof.

6.2. *Transaction Management in FINDIT*

Most transactions in FINDIT are likely to be **Read** transactions. **Write** transactions are performed by privileged users, i.e , the system administrators. There will thus be many more **Read** than **Write** transactions in FINDIT.

6.2.1. *Concurrency Control*

Concurrency control is achieved using a locking mechanism instead of a timestamping algorithm. This is done for two main reasons: simplicity (allowable because the number of **Read** operations heavily outweighs **Writes**); and because the overhead in keeping track of timestamps and synchronizing clocks across sites becomes a factor in decreasing efficiency and hence concurrency.

We distinguish two kinds of situations for managing co-database transactions. The first case is when a coalition is being formed. In this instance, no problem arises as **Write** transactions are only performed one at a time. After each round of negotiation, one **Write** is performed. The second case is when a co-database schema is being updated by a privileged user while being read by one or more users. Here a two-phase locking algorithm is adopted with the requested object being locked in all participating co-databases. Changes are introduced and then propagated to all participating co-databases. Hence, no write contention can occur. In fact the only contention that may arise is between a **Read** and a **Write**. Since co-databases follow an object-oriented data model and are replicated, we use a version of the multigranularity protocol combined with the two-phase locking algorithm, as a means to control concurrency [14, 19, 4, 10]. Further, since the database is replicated, we use the primary copy algorithm to deal with data replication [4]. In FINDIT, all co-databases follow the same data model and operate under the same DBMS. The only difference between the different co-databases lies in the different pieces of schemas and data they maintain. This means that existing databases do not have to introduce any changes.

6.2.2. Scheduler

As mentioned before, the scheduler uses strict two–phase locking, along with the primary copy protocol to address data replication. Since the co–databases are object–oriented, these protocols are combined with the multi–granularity protocol [14]. The scheduler takes a transaction as an input and outputs a list of locks for each operation. It unlocks all locks after the transaction commits or aborts. The transaction process starts with lock requests on all subclasses that have more than one superclass of the class to be locked. It then requests intention locks on one chain of superclasses of the class to be locked. Depending on the lock mode, the suppliers or clients of the class to be locked may also be locked. This procedure is conducted for every class to be locked. The last object to be locked is the class and, if applicable, instances of that class. If an instance is a composite object, the approach used by ORION [19] to control concurrency is adopted. Since we are dealing with a distributed and replicated database, we use a variation of the primary site algorithm [4]. In order for a transaction to get a lock on an object, it first has to get a lock on the copy located in the primary site. Once a lock is granted no other transaction can get it until it is relinquished by the current owner. This approach was chosen because of its simplicity and the low number of messages needed to get a lock. This basic method makes no provisions for site crashes. Thus, the primary site of an object is migrated from time to time, ensuring only temporary unavailability of sites. This extension only incurs extra overheads when lock tables are updated during the migration of lock responsibility.

6.3. Recovery Management

FINDIT can be termed a "replicated distributed database", and is thus susceptible to various system crashes ranging from network failure to site and communications failures. Since several co–databases may contain the same piece of schema and data, an update must be propagated to all other co–databases. The problem of concurrent update does not exist, as at any point in time at most one user is updating a given object. Consistency problems will not occur in the presence of a network partition with **Read** only transactions, where transactions are accessing information that is not being replicated, or where transactions require information that does not reside in their partition [23].

As mentioned above, there may be several causes for failure. The first of these is the transaction failure. This occurs when there is an error in the transaction and a deadlock is detected. Secondly, system failure may happen if there is software or hardware failure at a site. A third type of failure occurs when secondary storage fails because of hardware or software crashes (i.e. media failure). Lastly, communication failures arise when links between component databases fail because of software or hardware failure [4, 5]. The three first failures are common to centralized and distributed databases, and in FINDIT we are implementing algorithms taken from the literature. However, network crashes will cause unique problems in the FINDIT environment and require specifically designed crash recovery handling routines [8].

7. CONCLUSION

With the emergence of global networks, the explosive growth of information sources, and the need for organizations to make their databases interoperate to strive in a highly competitive environment, new approaches are needed to address the new environments and challenges. So far research has focused on problems related to small to medium scale database environments.

We propose a novel approach to dealing with problems related to large scale interoperation. In this respect, we described an approach based on meta–schema sharing (description of schemas) as a unit of sharing. This approach scales nicely and offers a flexible architecture that enables different levels of sharing while respecting autonomy. We have shown that the hyperdistributed approach can and does encompass both the global schema integration and federated approaches. In this regard, its versatility allows several data sharing configurations to co–exist within the boundaries of the same architecture.

Acknowledgements — The first and second authors' work was partly supported by an Australian large ARC grant number 95-7-191650010.

REFERENCES

[1] R. Alonso and D. Barbara. Negotiating data access in federated database systems. In *IEEE Conference on Data Engineering*, pp. 56–65 (1989).

[2] R. Alonso, D. Barbara, and L. L. Cova. Data sharing in large heterogeneous information networks. In *Workshop on Heterogeneous Databases*, Chicago. IEEE-CS Technical Committee on Distributed Processing (1989).

[3] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. In *ACM Computing Surveys*, volume 18(4), pp. 324–364 (1986).

[4] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley Publishing Company, Reading, MA (1987).

[5] P. A. Bernstein and N. Goodman. Concurrency control in distributed database systems. In *ACM Computing Surveys*, volume 13(2), pp. 187–221 (1981).

[6] A. Bouguettaya. Large multidatabases: Beyond federation and global schema integration. In R.Sacks-Davis, editor, *Proceeedings of the Fifth Australasian Database Conference*, Christchurch, New Zealand. Global Publications Services (1994).

[7] A. Bouguettaya and R. King. Large multidatabases: Issues and directions. In *IFIP DS-5 Semantics of Interoperable Database Systems (Editors: D. K. Hsiao, E. J. Neuhold, and R. Sacks-Davis).* Elsevier Publishers (1993).

[8] A. Bouguettaya, R. King, D. Galligan, and J. Simmons. Implementation of interoperability in large multidatabases. In *Third International Workshop on Research Issues on Data Engineering: Interoperability in Multidatabase Systems*, pp. 18–20, Vienna, Austria (1993).

[9] M. L. Brodie. The promise of distributed computing and the challenges of legacy information systems. In *IFIP DS-5 Semantics of Interoperable Database Systems (Editors: D. K. Hsiao, E. J. Neuhold, and R. Sacks-Davis).* Elsevier Publishers (1993).

[10] M. J. Carey and M. Livny. Conflict detection tradeoffs for replicated data. In *ACM Transactions on Database Systems*, volume 16(4), pp. 703–746. ACM (1991).

[11] W. Du, A. Elmagarmid, and W. Kim. Effects of local autonomy on heterogeneous distributed database systems. In *MCC Technical Report ACT-OODS-EI-059-90* (1990).

[12] J. C. French, A. K. Jones, and J. L. Pfaltz. Summary of the final report of the NSF workshop on scientific database management at the university of virginia on march 12-13, 1990. In *SIGMOD RECORD*, volume 19(4), pp. 32–40. ACM (1990).

[13] H. Garcia-Molina and B. Kogan. Node autonomy in distributed systems. In *International Symposium on Databases in Parallel and Distributed Systems*, pp. 158–166, Austin, TX. IEEE (1988).

[14] J. F. Garza and W. Kim. Transaction management in an object-oriented database system. In *ACM SIGMOD*, pp. 37–45, Chicago, Illinois. ACM (1988).

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

[16] M. Huhns, M. Papazoglou, and G. Schlageter. Intelligent and cooperative information systems. In *Proceedings of the 1st International Conference on Intelligent and Cooperative Information Systems*, Rotterdam, Holland (1993).

[17] A. M. Keller. The role of semantics in translating view updates. In *SOFT*, pp. 63–73 (1986).

[18] W. Kent. Solving domain mismatch and schema mismatch problems with an object-oriented database programming language. In *Proceedings of the 17th VLDB International Conference*, pp. 147–160. Morgan Kaufmann Publishers, Inc. (1991).

[19] W. Kim. *Introduction to Object-Oriented Databases.* The MIT Press, Cambridge, MA, Computer System Series (1990).

[20] W. Kim and J. Seo. Classifying schematic and data heterogeneity in multidatabase systems. In *IEEE Computer*, volume 24(12), pp. 12–18. IEEE (1991).

[21] R. Krishnamurthy, W. Litwin, and W. Kent. Language features for interoperability of databases with schematic discrepancies. In *SIGMOD*, Denver, Colorado. ACM (1991).

[22] T. Landers and R. Rosenberg. An overview of multibase. In *Distributed Databases*, pp. 153–184, Amsterdam. North-Holland Publishing Company (1982).

[23] L. Lilien. Partitioning and quasi-partitioning in distributed database systems. In *Distributed Processing Technical Committee Newsletter*, volume 10(2), pp. 63–72. IEEE Computer Society (1988).

[24] W. Litwin and A. Abdellatif. An overview of the multi-database manipulation language MDSL. In *Proceedings of the IEEE*, volume 75(5), pp. 621–632 (1987).

[25] W. Litwin, J. Boudenant, C. Esculier, A. Ferrier, A. M. Glorieux, J. La Chimia, K. Kabbaj, C. Moulinoux, P. Rolin, and C. Stangret. SIRIUS system for distributed data management. In *Distributed Databases*, pp. 311–343, Amsterdam. North-Holland Publishing Company (1982).

[26] B. Meyer. *Object-Oriented Software Construction.* Prentice Hall, New York, Prentice Hall International Series in Computer Science (1988).

[27] B. Meyer. *Eiffel: The Language.* Prentice Hall, New York, Object-Oriented Series (1992).

[28] E. Neuhold and B. Walter. An overview of the architecture of the distributed database system POREL. In *Distributed Databases*, pp. 247–290, Amsterdam. North-Holland Publishing Company (1982).

[29] C. Pu, A. Leff, and S. F. Chen. Heterogeneous and autonomous transaction processing. In *IEEE Computer*, volume 24(12), pp. 64–72. IEEE (1991).

[30] J. S. Quarterman and J. C. Hoskins. Notable computer networks. In *Communications of the ACM*, volume 29(10), pp. 932–971. ACM (1986).

[31] P. Scheuermann, C. Yu, A. Elmagarmid, H. Garcia-Molina, F. Manola, D. McLeod, A. Rosenthal, and M. Templeton. Report on the workshop on heterogeneous database systems. In *SIGMOD RECORD*, volume 19(4), pp. 23-31. ACM, Held at Northwwestern University, Evanston, Illinois, December 11-13, 1989 (1990).

[32] A. Sheth and V. Kashyap. So far (schematically) yet so near (semantically). In *IFIP DS-5 Semantics of Interoperable Database Systems (Editors: D. K. Hsiao, E. J. Neuhold, and R. Sacks-Davis).* Elsevier Publishers, Lorne, Victoria, Australia (1993).

[33] A. P. Sheth and J. A. Larson. Federated database systems and managing distributed, heterogeneous, and autonomous databases. In *ACM Computing Surveys*, volume 22(3), pp. 183–226. ACM (1990).

[34] M. Siegel and S. E. Madnick. A metadata approach to resolving semantic conflicts. In *Proceedings of the 17th International Conference on Very Large Data Bases*, pp. 133–145. Morgan Kaufmann Publishers, Inc. (1991).

[35] A. Silberschatz, M. Stonebraker, and J. F. Ullman. Database systems: Achievements and opportunities. In *Communications of the ACM*, volume 34(10), pp. 111–120. ACM (1991).

[36] J. M. Smith, P. A. Bernstein, U. Dayal, N. Goodman, T. Landers, K. W. T. Lin, and E. Wong. Multibase-integrating heterogeneous distributed database systems. In *AFIP, National Computer Conference*, pp. 487–499 (1981).

[37] M. Templeton. Research areas in heterogeneous distributed DBMS. In *Distributed Processing Technical Committee Newsletter*, volume 10(2), pp. 29–34. IEEE Computer Society (1988).

[38] M. Templeton, D. Brill, A. Chen, S. Dao, and E. Lund. Mermaid - experiences with network operation. In *Proceedings 2nd Data Engineering Conference*, pp. 292–300 (1986).

[39] M. Templeton, D. Brill, S. K. Dao, E. Lund, P. Ward, A. L. P. Chen, and R. MacGregor. Mermaid - a front-end to distributed heterogeneous databases. In *Proceedings of the IEEE*, volume 75(5), pp. 695–708 (1987).

[40] K. K. Wong and P. Bazex. MRDSM: A relational multidatabases management system. In *Distributed Data Sharing Systems*, pp. 77–85, Amsterdam. North-Holland Publishing Company (1985).

[41] K. Zhao. Panorama: Dynamic view construction in large multidatabase systems. In *PhD Thesis*, Dept of Computer Science, University of Colorado at Boulder (1992).