A Scalable Architecture for Autonomous Heterogeneous Database Interactions

Steve Milliner and Athman Bouguettaya and Mike Papazoglou
School of Information Systems
Queensland Univ. of Technology
Brisbane, QLD 4001, Australia
{steve,athman,mikep}@icis.qut.edu.au

Abstract

With the exponential proliferation of databases and advances in wide area networking, interest in worldwide database interoperability has gained momentum. Scalability and language support for this new environment remain open questions. We propose a scheme where database nodes are dynamically clustered around current *areas of interest*. Data sharing is then pursued, with any relationship information discovered being fed back for re-clustering. In order to achieve scalability, the proposed architecture sub–divides both the *relationship* and *information* spaces.

Keywords:
Interoperable Heterogeneous Autonomous Databases, Database Clustering, Distributed Database Language, Scalable Architecture for Interoperability.

1   Introduction and Motivation

Sharing information among autonomous heterogeneous databases has been researched extensively. In essence the problem has been to make component databases *interoperable* despite their different platforms (software and hardware). In addition, there has been a proliferation of database systems to handle ever

increasing volumes of information. These systems tend to be developed in isolation, and this results in structural and semantic heterogeneity. A major assumption in resolving heterogeneity is that component databases have *a–priori* knowledge of remote schemas. However, this is only reasonable provided the number of participating databases (and global information) is small. Recent advances in communications technology have led to expectations of large scale, world wide database interoperability.

There are various fundamental difficulties associated with large scale database interoperability. These include scale, autonomy and heterogeneity [BK93]. The case of the failed 3 1/2 year $ 125 million CONFIRM project clearly demonstrates the limits of current technology. This project entailed the linking of organizations spanning only three general areas: hotel, airlines and car rentals. The final reason given for failure was "technical difficulty" associated with constructing interfaces between the systems [Oz94].

In large collections of autonomous distributed resources the question of finding appropriate information becomes paramount. This is further complicated in the database case by the complexity of *implicit* inter–relationships of information items between database nodes, and the complexity of queries which may be formed based upon these inter–relationships. There are thus two distinct types of *space* that must be explored one concerns the search for appropriate information – the *information space*, the other concerns the search for relationships between nodes – the *relationship space*. To promote scalability, a two level architecture is introduced which segments both of these spaces. If the information space is not sub–divided exhaustive searching must be performed. If exhaustive interactions between database nodes are permitted, then communications and processing bottlenecks will occur. Hence, in the case of large numbers of au-

tonomous databases, an organization and segmentation of database nodes needs to be introduced to filter interactions, accelerate information searches, and allow for the sharing of data in a tractable manner.

System size and complexity precludes the static *a-priori* sub–division of the relationship and information spaces. Thus, in both levels of the architecture, an incremental building and sharing of inter–relationship meta–information [Bou94, MP94, PLS92] is pursued. A main objective is to let databases know dynamically (as opposed to statically) how they relate to remote sites, and what these databases contain. Relationship information discovery, and data sharing, must be driven by the states of the individual database nodes as the system executes. In particular, the aim here is to consider how a large collection of database nodes may be organized so that information resources may be easily identified and shared. This work is concerned with integration at the schema and model levels, as well as interoperation at the language and database application levels. Heterogeneity resolution of lower levels such as the file and operating systems, the DBMS and transaction management levels, are outside the scope of this paper.

The organization of this paper is as follows. In section 2 an overview of related work is presented. Section 3 provides an overview of the proposed architecture, while sections 4 and 5 describe the initialization and execution of the system. The conclusion is presented in the final section 6.

2 Related Work

Related work has mainly been carried out in the areas of multidatabases, federated databases, information retrieval, and distributed system naming.

Multidatabases

Most multidatabase systems provide resource sharing through a global schema [TBC+86, NW82, LBE+82] which is usually obtained by integrating multiple schemas. Global queries are then executed against this global schema. The major problem associated with this technique is the translation of local languages and schemas into a global format. No automatic translation, schema update and integration has been performed so far. The main difficulty concerns how one might map an understanding of entities from one schema to another [BLN86]. In existing systems, translations and integration are done in an ad hoc fashion. In addition autonomy is sacrificed and decentralized decision making is not achieved. Nodes are required to reveal details of their schemas so that a central schema, designed by a single schema administrator, can be formed. Because of the scale of systems being considered here. Global schemas are not consid-

ered, as they lead to serious design bottlenecks.

Global Naming

Another area of related research is global naming [Wat81]. In this scheme, the system views resources as simple entities. The name service is in charge of mapping the name of an entity into a set of properties, each of which is a string. The search is usually instance based rather than type based. The data involved belongs to a small set of basic types. Hence, little or no semantics are attached to the data. Most services use one single hierarchy to cope with extensibility [BLNS82]. These hierarchies are meant to provide means for better organizational management. [Sch93] presents an interesting model for finding resources in a network of computer systems. In this project, resources are typically unstructured text. As the research was conducted from a system point of view, databases issues were simplified. The most interesting idea in this latter paper is the stress on the *separation* of concerns between resource providers and resource consumers.

Information Retrieval

In most information retrieval systems, the emphasis is usually on how to build an indexing scheme to efficiently access information given some hints about the resource [SM83]. Most of the distributed information retrieval systems are designed to work in a homogeneous environment. There has been some work to extend schemes to a network of heterogeneous information retrieval systems [Sim88]. In [ABC89, SA89], an approach is described that relies on *external indexing* for finding information in a network of information systems. Each node of the index contains a network address along with a set of condensed descriptions called *skeleton*. Resource providers are added to the index using *knows–about* relationships. This approach tends to centralize the search as a single index is used for the actual resource discovery. Potentially (if users make queries about all existing information space), all nodes would have the same index. It is not clear from the above references how the system behaves if several nodes can answer a given query. There is also no reference of how the actual node selection is performed.

Federated Databases

A work that shares some philosophical commonalities with ours and is of particular interest to our research, is the federated approach. In federated databases [HM85, SL90], a certain amount of autonomy for individual database systems is maintained. In this approach, information sharing occurs through *import* and *export* schemas. All databases are registered in a federal dictionary. The federated approach as described in [HM85] does not address the issue of how the federal dictionary is to be designed in presence

of a large number of databases. The federated approach implicitly assumes that only a few databases participate in sharing information. In this framework, no incremental sharing is implied. Instead, databases share actual data in one single step. We believe that any viable approach to sharing information in a large network of databases has to rely on defining a flexible architecture, a variable–grained information sharing, and a more user oriented centered sharing (versus database administrator–centered sharing). It should be borne in mind that the federated databases approach was not specifically designed to address the issue of large networks of databases. The purpose has been to provide a better alternative to the global integration approach and to a large extent, has succeeded in doing so.

Academic and Commercial Information Retrieval Databases

There are several products that offer access to a wide variety of databases. Some of these products are commercial while others are academic. The basic assumption underlying most of these products is that the participating databases have little or no structure at all (i.e. text–based) [ODL93, Sch93]. Academic products like Archie [ED92], WAIS [KM91], World Wide Web (or WWW) [BLCL+94], and Gopher [McC92] use indices and browsing or a combination thereof to access freestyle documents. The architecture centers around servers providing information to clients. In the case of Gopher and WWW, servers are connected in a graph–like fashion. In the case of WWW, hypertext links constitute the graph.

An interesting commercial product that is somewhat more sophisticated is the French *Minitel* service [Con92]. This service is provided by the state run PTT. Hundreds of databases are hooked together to provide a wide range of information services. However, this service requires a uniform interface and access can only be through specialized terminals. All databases hooked to this service are text–based.

In summary, a constant across all the systems described above is that the problem of database autonomy is not a concern. In most cases, databases are actually plain text files. In these cases, no interaction with a database management systems is assumed, thus no problem of database heterogeneity arises. However, these issues are fundamental to addressing the problem of data sharing in database systems. Data sharing in databases requires that, before data can be accessed, we first need to understand *schemas* and *inter-relationships* between databases.

3 Overview of the Relationship and Information Space Architectures

In a single centralized database, inter–relationships between information resources are specified using for instance, relational or object oriented organization. When a centralized database is segmented further, organizational constructs must be introduced to maintain these relationships between the now remotely distributed information components. In the case of large numbers of autonomous databases, there is no explicit pre–existing organization as such – thus one must be introduced. Because of the size and complex nature of the interoperability problem, explicit static specification of relationships is not appropriate. In addition it seems unlikely that every possible present and future inter–relationship could be identified before the system begins functioning. Thus, any organizational architecture must allow for an incremental building up and dynamic sharing of meta information.

This meta information relates to two basic aspects of the distributed database system, that is, the information and relationship search spaces. In order to find and share data within the data space, dynamic database interactions must be organized and maintained, and remote schemas understood. Once established, this organization also provides a means for searching of the system based upon these established inter–database node relationships.

In order to sub–divide the relationship space, a high level "context abstraction" is used. This is to be implemented through the use of dynamic meta objects termed *global concepts* (GCs). GCs are based upon "descriptions" (meta data) of each local database's domain, and correspond to the current areas of interest within the universe of discourse [PLS92]. Database nodes then form links to each GC, and associate an updatable weight with each of these. This results in a *clustering* of database nodes [Bou95] around the various GCs (as shown in Figure 1).

By clustering database nodes, the relationship space is sub–divided, and searching is implemented via reference to the link weights. While this structure may appear "flat", the link weights actually impose a dynamic ordering and structure upon the GCs and the database nodes clustered around them. When several database nodes all link strongly to the same GC (eg. weight 10/10), a dynamic cluster A is formed. However, each of these same database nodes will also link less strongly (eg. weigh 8/10) to other GCs which will have their own associated clusters B, C, D and so on. Database nodes in cluster A will therefore overlap with all other clusters to various degrees.

Link weights updates are performed locally at each database node. The organization is based upon in-
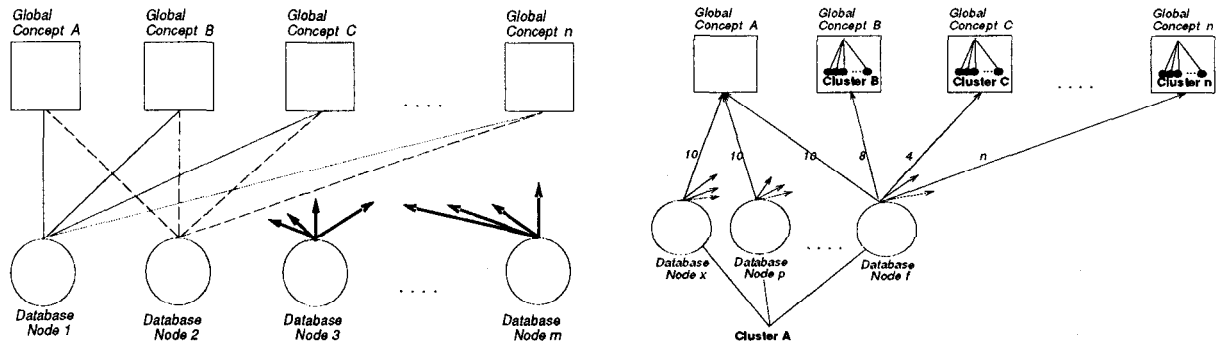
Figure 1: Database Nodes Linked to Global Concepts and Clusters

dividual database node "states", and thus control remains logically distributed. Consequently, the organization is inherently flexible, and avoids the short comings of strict hierarchical and object oriented organizations noted in [KS94, Sch93]. Search granularity is determined by the number of GCs and the number of nodes clustered around them.

In order to establish, and maintain a dynamic subdivision of the information space, the *Tassili* language was developed [Bou94]. The primitives of this language are used to query and extract information, and to educating users about the information space. Linking of database nodes to one another results in the formation of database clusters. It should be noted that these "information space" clusters are different to those formed in the relationship space. Clusters formed in the information space allow for the exchange of structural information between database nodes, as a prelude to data sharing. In addition, relationships may be formed between two clusters, or between a single database node and a cluster. In the relationship space, clustering provides high level logical associations, and does not result in the sharing of physical information. By grouping database nodes in the information space, meta level relationship *types* are specified. Their formation is driven by the system, and they represent how database nodes physically relate to each other. Formation and alteration of information space relationships represent an alteration in current areas of interest. Hence, information regarding relationships is fed back into the relationship space for link weight and GC adjustment. A meta-information type is a unit which describes the structure and behavior, of semantically related portions of database schemas. Two basic forms of meta-information types are used; one which arises from a cluster, and a second which is instantiated by relationships between two databases, a database and a cluster, or two clusters.

Consider for instance, a second-hand car dealer's database node. Initially it will be strongly linked (eg. weight 10/10) to a GC representing the area of in-

terest "sales", and more weakly linked (eg. weight of 3/10) to other GCs dealing with the area of interests relating to cars. Assume the dealer database node requires continual information from several remote sites regarding car registration, car insurance and car spare parts which are initially strongly linked to GCs representing these three areas. The dealer database node will examine GCs it is less strongly linked to, and will locate these remote sites. If the information at these remote sites is found to be appropriate, then the dealer database links to the remote GCs is increased (as shown in the Figure 2 insert). Links pictured as lines in the Figure 2 insert, do not represent relationships, but overlaps between clusters representing the current areas of interest. Database nodes which now share physical data, form an information space clustering. Figure 2 depicts the information space subdivision, and will be referred back to throughout the remainder of the paper. The information space clustering is implemented by having the dealer database node join with "Vehicle Sales" cluster. Information space relationships are then formed between the "Vehicle Sales" cluster and the "Vehicle Spare Parts", "Insurance" and "Vehicle Registration" clusters (shown as solid lines in the figure). In this way, the relationship space sub-division creates a platform for the resulting information space sub-division. As time progresses seven information space clusters and their associated relationships are formed.

Consider the database owned by the *CheapRent* car rental company in Figure 2. It will initially be strongly linked to the GC representing rentals – and less strongly connected to GCs representing other areas of interest (such as cars). After exploration of the relationship space, its database contacts remote sites and eventually joins the *rentals* information space cluster – this is depicted in Figure 2 by the solid bold line leading from the CheapRent database symbol to the collection of databases comprising the *rentals* cluster. CheapRent also enters into a relationship with the *vehicle spare parts* cluster. This is depicted in Figure 2 by
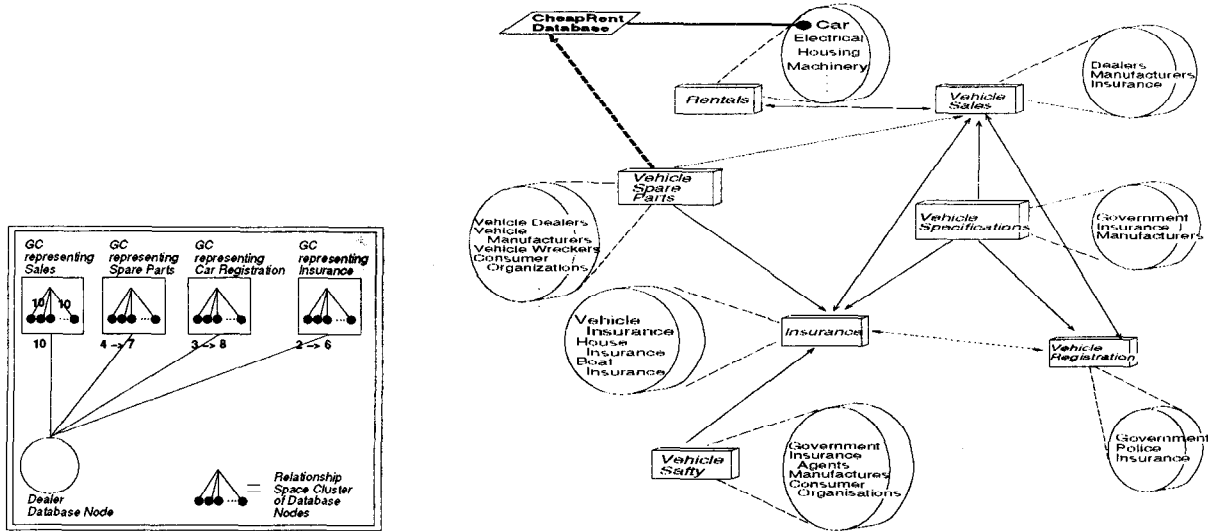
518

Figure 2: A Partial View of a Network of Relationships and Clusters

the broken bold line between the CheapRent database symbol and the collection of databases comprising the *vehicle spare parts* cluster. Once these information space associations have been established, this relationship information is fed back into the relationship space level architecture, causing the link weight updating shown.
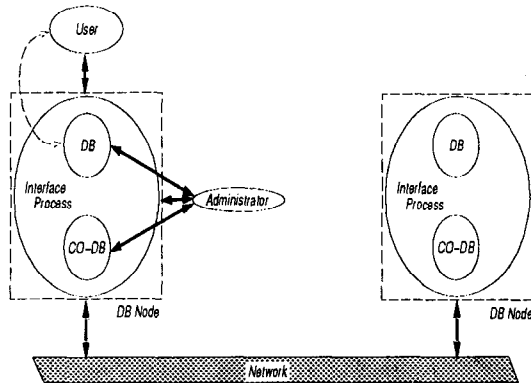

Figure 3: Database Nodes Connected Over a Network

Figure 3 depicts how database nodes may be connected over a network, and how general users and administrators interact with the system. General users are permitted to interact with the local database (as indicated by the broken arrow). To access remote systems however, these users must interact with the interface process. Co-databases (CO–DB) are object oriented databases where information and knowledge of the relationship space and information space subdivisions are stored. General users may also suggest updates to local co-databases, however, persistent updates are performed only by the administrators (as

indicated by the arrows in the figure).

There are two basic phases in the lifetime of a distributed system organized around the proposed architecture. These are the *initialization* phase and *execution* phase. During initialization GCs are created and link weights to them are established. This is essentially a pre–clustering process, where groupings of database nodes are established on a tentative basis only. During execution the system operates on four levels: an *architectural* level, an *interoperability* level, an *interaction/negotiation* level, and an *exploration* level. These are discussed in section 5.

4 System Bootstrapping

The initialization phase consists of three basic activities: presentation of database node descriptions, global concept instantiation and link weight initialization.

4.1 Database Node Descriptions

Initial clustering is based upon a description which each database node forwards to a central location. This step is performed only once, and should be viewed as pre–clustering of the system. As information space clusters are formed, inaccuracies in the pre–clustering phase are eliminated and link weight adjustment results in re–clustering of the system. In addition to the database node description, other descriptive categories may be included to further filter database nodes involved in the information space clustering process. For example, geographical location and corporate structure may be included, and so on. Although it is assumed that descriptive categories are defined statically at initialization, there is no intrinsic reason why further categories cannot be added or existing categories

519

deleted.

4.2 Pre–Clustering and Global Concept Instantiation

The "topic" based clustering of database nodes, is depicted in Figure 4. After being clustered, each database node forms a weighted link with the other clusters. In Figure 4, each node in *cluster#1* forms a weighted link with *cluster#2*, *cluster#3* and so on. Database nodes will be most strongly connected (eg. have a weight of 10/10) to the cluster they were initially assigned to. Weights for the other links are automatically initialized by having each node calculate the ratio of similarity of all database nodes in its cluster to all database nodes in the remote cluster it is forming a link to. This process will occur for each cluster set, to produce a series of distinct fully logically connected groupings. These are subject to user verification and are updated over time.

The outcome is a collection of abstract global concepts around which nodes can cluster. The set of GCs which forms a basis for relationship space by :

1. Sub–dividing the nodes to produce a reasonably uniform distribution and,

2. Providing an appropriate degree of search space granularity.

Initial clustering may not result in uniform size groupings. Hence, an additional sub–division of groups may be required [Eve80]. A *tuning* phase may also be introduced if node descriptions do not provide an adequate basis for the clustering process (eg. when a small proportion of clusters contain a large percentage of the clustered items). This situation corresponds to an over generalization of key features and can be resolved by employing a subdivision algorithm which refines and decomposes over generalized GCs.

5 The Execution Phase

During execution, activities are characterized by four levels: an *architectural* level, an *interoperability* level, an *interaction/negotiation* level, and an *exploration* level. In the architectural level a logical interconnection of nodes is pursued, and clusters are formed in the relationship space. In the interaction/negotiation level, the GC organization is used as a platform for information space clustering. Information concerning physical relationships formed in the information space, is then fed back into the first layer for adjustment of the relationship space structure (via link weight and GC updates). Together these two layers allow organization of interactions between the system's database nodes. Remote information must be understood, this
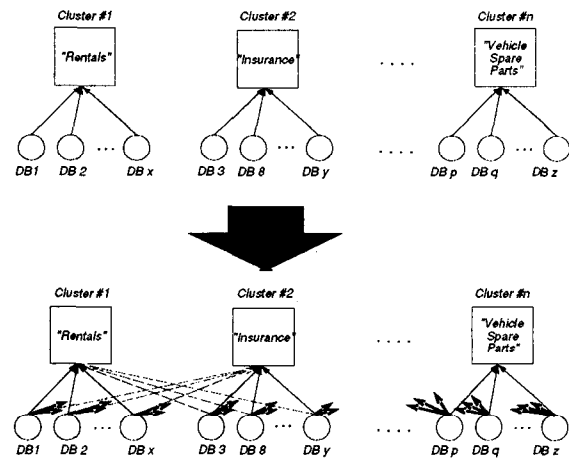

Figure 4: Creation of Links After Initial Clustering

is achieved in the interoperability level through the use of *demonstrations* which are described in section 5.3. Lastly, the appropriate information must be located. This is achieved in the exploration level using relationship space GC clusters, in conjunction with the Tassili language and existing information space clusters.

5.1 The Architectural Level

This level supplies an architecture for data sharing between database nodes, using the clusters formed in both the relationship and information spaces. By forming clusters, a means is provided for the synchronizing of both relationship formation and data sharing activities. By adjusting link weights locally, each database node itself decides which relationship space clusters it will participate in. By joining an information space cluster, database nodes implicitly *agree* to work together (and thus sacrifice their autonomy to some degree). However, nodes retain control locally and join or leave clusters based upon local considerations. Consequently, maximum local autonomy is maintained and both, inter–relationship space and information space control, remains distributed. In addition, inter–node relationships continue to be dynamically updated, formed, and dissolved.

Because database nodes are fully *logically* interconnected, via GCs and link weights, any information type defined by a particular clustering of nodes is potentially accessible. The system thus allows the sharing among databases of *all* information types available. Once inter–node relationships have been initiated, a platform is provided for the interoperability process and data sharing. If this dynamic subdivision is not performed the information space remains huge and the interoperability problem becomes intractable.

5.2 The Interaction/Negotiation Level

There are several database node interaction phases. The first form of node interaction involves searching of the inter-node relationship space by means of search heuristics. Previously, a pre-clustering of nodes has been performed. Once the appropriate nodes are identified, access to the remote information may be negotiated to determine which tasks get delegated to which remote servers. This latter process is performed through the Tassili language, and may result in updates to both the information space clusters, as well as relationship space clusters (i.e. GC link weights, and GCs themselves).

The second form of node interaction results in the discovery of interoperability facts which are stored in the node co-database. In order to extract/explore (or export) information a node may *contact* and interact with the relevant nodes in a information space database cluster. Once again, this interaction is performed using Tassili language primitives.

5.2.1 Information Space Cluster Construction and Update

The ability to form, and control joining and leaving of clusters, is restricted to selected users. The primitives for these interactions and semi-automated negotiation sessions are provided by the Tassili language, and result in the creation and maintenance of an object oriented information space schema.

In some instances, users may ask about information that is not in the local database domain interest. If these requests are small and not persistent, a mapping to a remote information type will suffice. If the number of requests remains high, re-clustering may ensue. The database will then either share the meta-information type represented by the remote "popular" databases, or a new information type will be formed.

In the CheapRent car rental company case, information retrieval begins by determining which remote databases should be contacted. After examining the relationship space, and/or the available information space cluster information, the CheapRent administrator joins the *rentals* cluster. That is, after identifying the appropriate "area of interest", joining of the information space cluster and sharing of data ensues. In Tassili this negotiation process is performed using the the following primitive:

Inquire at GM–Spare–Parts *With Message* "Wish to establish a relationship. What are the main attributes of your resource?"

A message is sent along with the query to explain what is expected. If the query fails, a diagnostic is returned.

Its co-database is then loaded with information regarding relationships between the rentals cluster and other cluster/databases, as well as information regarding databases in the cluster itself. The CheapRent administrator will now, for instance, have knowledge of the *vehicle sales* cluster, and indirect knowledge of the *vehicles spare parts* cluster for example – because of the relationship between the sales and spare parts clusters. A general user can thus investigate this latter cluster in the hope of resolving an av–cost–of–parts query for instance. If this investigation proves fruitful a short term interaction with this cluster site will ensue. In order to obtain remote structural information, the following Tassili primitive is used:

Send to CheapRent *Object* GM–Spare–Parts.template.

This query is used by the target (representative) database to send information about the information requested to the servicing database or cluster. If the specifications meets the servicing needs, no further action is taken. However, if the specifications do not meet the servicing entity specifications, an *Inquire* query is sent back to the target entity for further refinement. This process of negotiation ends whenever the involved entities decide so. If the *Send* query is successful, nothing is returned. If the query fails, a diagnostic of the failure is returned. Other primitives exist to create data structures at local co-databases to implement the relationship abstraction, and to end the relationship. This results in formation and/or modification of the object oriented schema.

When the car rental company joins the *rental* cluster a information space schema update occurs. A new class (representing the rental company node) is instantiated via an existing cluster member. This update is then propagated to the other cluster members. These changes are achieved using the primitive:

Instantiate Class Rentals *With Object* Car *With* Name = CheapRent.

The rental database manager may then choose to allow remote access to certain local information. For example, the CheapRent node may wish to allow access to "rental–rate", "model" and "year" attributes which are contained within its database. This is achieved by

521

adding methods to its class via the Tassili primitive:

Add Method Rental–price *With Body*

if date.month >= Oct and date.month <= Jan
   return(1.2 * base–price)
else
   return(base–price)

To Class Rentals.

Other Tassili primitives exist to remove methods and objects (ie. when a node relinquishes access to local information or leaves a cluster), and to alter methods or objects. There are also more basic primitives which are used to establish a cluster, and propagate and validate changes. Each operation has to be validated by the participating database administrators.

For any database to enter or leave a certain cluster, it has to fulfill the requirements set by that cluster, and provide some information about the data it would like to share as well as information about itself. The administrator of that cluster will then decide how the information space schema is to be augmented if the database is accepted as a new member. During this informal exchange, many parameters need to be set. For instance, a threshold for the minimum and maximum number of cluster members is negotiated and set. Likewise, a threshold on the minimum and maximum number of relationships with databases and cluster is also set. In the example shown in Figure 2, the CheapRent administrator has decided that long term connection is required with the *vehicle spare parts* cluster. Thus, a relationship is entered into with the *vehicle spare parts* cluster's nodes (ie. information flow from the *vehicle spare parts* cluster to the CheapRent database).

Initially, a database administrator creates the root class of the cluster schema. Once this is done, the root of the schema is sent to every participating database for validation. If the operation is not validated, the rejecting node sends an edited version of the object to the creator of the object. Based on this feedback, the creator will decide whether to change the object or not. This process will continue until there is a consensus. Changes are only made at a single site until consensus is achieved – at which time the change is made persistent and propagated to the appropriate database nodes.

If existing classes/methods are to be updated responsibility lies with the database that "owns" it. Prior to any changes, the database owner tells 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, and propagates the update.

A database cluster is dismantled by deleting the whole corresponding subschema in every participating database schema, along with all objects belonging to the classes of that cluster. All clusters with which there is a relationship are notified that the cluster no longer exists. Local schemas are updated by their administrators. The decision to dismantle a cluster is reached by a consensus of participating databases, or when only one node remains in the cluster. The update of co–databases resulting from relationships changes is practically the same as defined for clusters. The only difference being that changes in the cluster case obey a stricter set of rules.

After CheapRent creates a relationship connection with the *vehicle spare parts* cluster, both it and the cluster members must update their co–database. Similarly, when CheapRent joins the *rentals* cluster co–databases must be updated to reflect the new inter–node relationship structure. In this way CheapRent builds up its local knowledge of remote sites and forms appropriate relationships with those containing applicable information. Thereby filtering the amount of information which must be assimilated at the car rental node. Further relationships and clusters are contacted by CheapRent users as they attempt to access required information. The rental database administrator may then choose to enter into/create further relationships and clusters based upon users needs. When no information space meta–types exist, or when there are a huge number, it becomes necessary for users to explore the information space using the underlying information space GC sub–division. That is, when there are no information space clusters, or when there are too many, the underlying relationship space sub–division may be utilized to restrict the search space [MP94].

5.3 The Interoperability Level

Before information can be accessed inter–relationships between databases must be organized *and* schemas must be understood. Understanding of remote information is handled through the introduction of *documentations*. These are sample behaviors or structures of certain classes of information, which are attached to each information type that is shared with the outside world. Their aim is to explain/define that information type. Even if two databases contain the same information, local usages will vary and different behaviors may be exhibited.

In the example depicted in Figure 2, documentation is required by CheapRent to make sense of remote node methods *advertised* in the various clusters it must interact with. Confusion may arise when interacting

with databases in the *vehicle spare parts* cluster because of differences in price listings (eg. one database may include sales tax, others may not), differences in domains (eg. a database may only stock parts for one type of car), differences in the types of businesses (eg. a car wrecker may only display part availability, not price) and so on. Demonstrations may also provide details of information needed by the remote node to access the required data. For example, a vehicle's year and make will be required to obtain a spare part's price.

Similarly, in the case of interpreting *insurance* cluster members, demonstrations may be included to differentiate policies offered by various insurance agents. This may include a simple policy statement, or include graphical/video/audio data comparing one companies policy with another. For example, one company may provide cover for both the vehicle and driver, others may not. Different companies may also pay out different market value prices if a car is damaged beyond repair, and so on. In the *vehicle sales* cluster databases may list prices of models. Video and audio demonstrations may be included to highlight vehicle features. A similar demonstration format may be included by a government database in the *vehicle safety* cluster to explain attributes in a safety standards database. Documentation is offered by the providing databases and is not an integral part of the system, and may include several of the following features: Structured text description, graphical descriptions (e.g. diagrams, video etc.), and audio information. If the documentation relies on a programming tool, databases are encouraged to provide many platforms of implementation so that the maximum number of users can use the documentation. Databases are also encouraged to provide documentation that may run under different operating systems. The user is prompted for a choice of operating systems to choose from. Last and not least, the documentation runs on a multitude of hardware.

Demonstrations thus provide a way of converting data (eg. price of parts versus price with sales tax), evaluating data (eg. the "best" insurance policy – cost benefit analysis), translating data (eg. demonstrations may be offered in several different languages), and interpreting data (eg. what is the definition of a "safe" car). Note also that demonstrations on the same topic will vary from database to database. For instance, safety features in a manufacturer's database (meant for public consumption) may not equate with safety features in a government vehicle safety database (meant for expert consumption). Thus, the local "meaning" of attributes/objects is defined, through the Tassili query language.

5.4 The Exploration Level

Once a number of information space clusters have been established, exploration of the universe of discourse may be pursued via the relationship or information space organization representations. Although an explosion in the number of information space clusters may limit information space based searches. The syntax specifications of Tassili queries provide constructs to educate users about the available information space organization, as well as connecting databases and performing remote queries. The information type name, structure, behavior, and graphical representation are used as a handle for identifying the appropriate information resources. Node co–databases maintain schemas and it is to co–databases that all information space queries (local and remote) are directed.
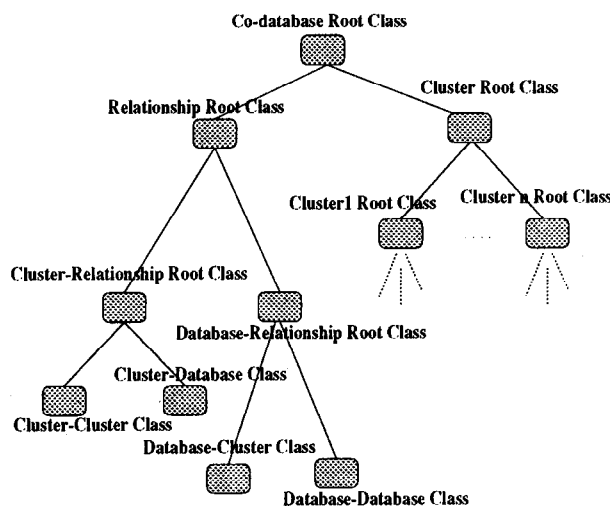

Figure 5: Skeleton of a Typical Co–Database Schema

A co–database information schema consists of two subschemas as depicted in Figure 5. Each subschema represents either a cluster or a relationship and contains 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 *car parts*). A graphical interface has been implemented so users may navigate through the information space. The relationship subschema (left side of Figure 5) consists of two subschemas, the first depicts relationships that clusters (it is member of) have with other databases and clusters. The second is a subschema of relationships the database in question has with other databases and clusters. Each of these subschemas, in turn, consists of two subclasses that describe relationships with databases and clusters. The cluster subschema (right side of Figure 5) consists of one or more subschemas, each of which represents a cluster the database in question is a member of.

Cluster relationship descriptions include information about points of entries and contact with those clusters that are involved in a relationship with. 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 relationships providing clusters will be the same for all databases that are members of the providing cluster.

A set of databases, containing a certain information type is represented by a class in the schema. Every class contains a description of the participating databases and the information type(s) they contain. Some attributes describe the information type while the remaining attributes describe the databases that contain this information type. Database descriptions include information about the data model, operating system, query language, etc. A description of the information type includes its general structure and behavior (if applicable). Since databases may have different views on the same information type, only the common parts of the view are represented in the class. These descriptions differ from demonstrations in that they only offer general structural information. Using this structural information and various Tassili query primitives, a user at the CheapRent company can begin to resolve complex queries. The CheapRent co-database is a member of the *rentals* cluster and has a relationship with the *vehicle spare parts* cluster. Because of these relationships CheapRent has knowledge of the *vehicle sales* cluster stored in its co-database. This information is accessed by a user through the Tassili primitives:

Display Clusters of CheapRent.

which provides a list of clusters CheapRent is a member of, while

Display Cluster Relationships Rentals.

provides a list of clusters the CheapRent database and *rentals* cluster has a relationship with (eg. *vehicle sales* and *vehicle spare parts*). A similar query may be used to examine the relationship links of the *vehicle spare parts* and *vehicle sales* clusters. In this way CheapRent users can discover the clusters: *vehicle specifications, insurance* and *vehicle registration*. Lastly, by examining the relationship links to the *insurance* cluster, the *vehicle safety* cluster can be found.

Primitives also exist which display the classes of relationships and clusters, and allow connections to remote clusters/database relationships. Users may utilize other primitives and stipulate an "information name" – for example "part prices" – which returns a list of possibly remote clusters/relationships that have a corresponding class name. This process requires that

nodes maintain an appropriate list of thesaurus terms. If no match is made a null list is returned. In this case a user may either submit another information name term, attempt to navigate through the system and discover the information "manually", or make use of the underlying relationship space search structure. In the last case three basic search heuristics have been proposed in [MP94], based on link weight relaxation.

In the car rental case the relationship between the CheapRent database and the *vehicle spare parts* may have been returned by stipulating the information name *spare parts* or simply by listing relationships. Once an appropriate cluster/relationship has been found, Tassili provides primitives for investigating its classes. For example:

Display SubClasses vehicle–spare–parts

will display all the subclasses of the *vehicle spare parts* cluster.
The query:

Find Information part–prices

will display information associated with the subclass *part prices* within the cluster *vehicle spare parts*.
Lastly, the query:

Find   Information   With   Attributes   parts   : *"exhaust", model : "JaguarXJS", year : "1990".*

will display actual information. Clearly these queries require increasingly greater knowledge of the cluster schema. Thus both expert and novice users are catered for.

Further Tassili primitives are available for exploring objects more fully (as required in the first two of the three class queries above). For example, the "Display SubClass" query will return the list : cars, trucks, motorbikes and so on. The user then issues the query:

Display Instances By Attribute Of Information Car.

By leaving the "instance" term blank all instances associated with the sub–class "Car" are displayed. Similarly, Tassili allows attributes of instances to be displayed. Finally, Tassili allows for further understanding of an information type using the primitive:

Display Documentation of Instance Ford–spare–parts *Of Information* part–price.

The answer can be in textual or graphical form (as noted previously). If there is no behavioral capability or no suitable environment is present, the query fails. Otherwise, documentation is displayed.

6 Conclusion

This paper addresses issues relating to large scale interoperation. In particular, we propose an architectural framework for organizing interactions among a large number of autonomous disparate database nodes. The proposed form of interaction results in the creation of dynamic clusters of databases centered around subject areas of interest or expertise called global concepts. Global concepts are high level abstract meta-schema objects which essentially represent centroids of a database cluster pertaining to an area of interest. Database nodes then form weighted links to global concepts in a way which reflects their own interests. Thus they organize the inter–database information and relationship search space.

Linguistic tools are also provided in order to allow application programmers to establish and maintain the subdivision of the inter–database information space. Moreover, the system provides facilities for querying and extracting information regarding the inter–cluster relationships and the information space in general. The proposed approach allows for a high degree of flexibility as it allows relationship information discovery and data sharing to be driven by the states of the individual database nodes as the system executes.

The proposed approach fulfills three fundamental tasks: it provides a common framework (the global concepts) to which participant databases contribute; it specifies a relatively small set of databases/nodes for interaction (viz. database clusters) thereby accelerating information searches; and implicitly provides local nodes with an abstract model of other clusters and database nodes. Other virtues of this approach are its simplicity, dynamic nature, and extensibility in addition to the the fact that it retains local autonomy because nodes provide and control their own classification.

Acknowledgments

References

[ABC89] R. Alonso, D. Barbara, and L. L. Cova. Data sharing in large heterogeneous information networks. *Workshop on Heterogeneous Databases*, Dec. 1989. IEEE-CS Technical Committee on Distributed Processing.

[BK93] A. Bouguettaya and R. King. Large multidatabases: Issues and directions. In *IFIP DS-5 Interoperable Database Systems (Editors: D. K. Hsiao, E. J. Neuhold, and R. Sacks-Davis)*. Elsevier Publishers, Lorne, Victoria, Australia, 1993.

[BLCL+94] T. Berners-Lee, R. Calliau, A. Luotonen, H. F. Nielsen, and A. Secret. The world-wide-web. *Communications of the ACM*, 37(8):76–82, August 1994.

[BLN86] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):324–364, Dec. 1986.

[BLNS82] A. D. Birell, R. Levin, R. N. Needham, and M. D. Schroeder. Grapevine: An exercise in distributed computing. *Communications of the ACM*, 25(4):260–274, April 1982.

[Bou94] 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, January 1994. Global Publications Services.

[Bou95] A. Bouguettaya. On-line clustering. *To appear in the IEEE Transaction on Data and Knowledge Engineering Journal*, 1995.

[Con92] W. W. Conhaim. Maturing french videotext becomes key international tool. *Information Today*, 9(1):28, January 1992.

[ED92] A. Emtage and P. Deutsch. Archie: An electronic directory service for the internet. *Proc. Winter 1992 Usenix Conf.*, pages 93–110, 1992.

[Eve80] B. Everitt. *Cluster Analysis*. Halsted Press, 1980.

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

[KM91] B. Kahle and A. Medlar. An information system for corporate users: Wide area information servers. *Connexions - The Interoperability Report*, 5(11):2–9, November 1991.

[KS94] V. Kashyap and A. Sheth. Semantics-based information brokering: A step towards realizing the infocosm. Technical Report DCS-TR-307, Rutgers University, March 1994.

[LBE+82] 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, pages 311–343, Amsterdam, 1982. North-Holland Publishing Company.

[McC92] M. McCahill. The internet gopher protocol: A distributed server information system. Connexions - The Interoperability Report, 6(7):10–14, July 1992.

[MP94] S. Milliner and M. Papazoglou. Reassessing the roles of negotiation and contracting for interoperable databases. In International Workshop on Advances in Databases and Information Systems, pages 169–202, March 1994.

[NW82] E. Neuhold and B. Walter. An overview of the architecture of the distributed database system POREL. In Distributed Databases, pages 247–290, Amsterdam, 1982. North-Holland Publishing Company.

[ODL93] K. Obraczka, P. B. Danzig, and S. H. Li. Internet resource discovery services. IEEE Computer Magazine, 26(9):25–35, September 1993.

[Oz94] E. Oz. When professional standards are lax. Communications of the ACM, 37(10), 1994.

[PLS92] M. Papazoglou, S. C. Laufmann, and T. K. Sellis. An organizational framework for cooperating information systems. International Journal of Intelligent and Cooperarting Information Systems, 1(1):169–202, March 1992.

[SA89] P. Simpson and R. Alonso. A model for information exchange among autonomous databases. Technical Report, Dept. of Computer Science, Princeton University, May 1989.

[Sch93] M. F. Schwartz. Internet resource discovery at the University of Colorado.

IEEE Computer Magazine, 26(9):25–35, September 1993.

[Sim88] P. Simpson. Query processing in a heterogeneous retrieval network. 11th International Conference on Research & Development in Information Retrieval, pages 359–370, June 13-15,1988.

[SL90] A. P. Sheth and J. A. Larson. Federated database systems and managing distributed, heterogeneous, and autonomous databases. ACM Computing Surveys, 22(3):183–226, September 1990.

[SM83] G. Salton and M. J. McGill. Introduction to modern information retrieval. In McGraw Hill Computer Science Series, New York, 1983. McGraw Hill Book Company.

[TBC+86] M. Templeton, D. Brill, A. Chen, S. Dao, and E. Lund. Mermaid - experiences with network operation. Proc. 2nd Data Engineering Conference, 292-300:292–300, Feb. 1986.

[Wat81] R. Watson. Identifiers (naming) in distributed systems. In B. W. Lampson, M. Paul, and H. J. Siegert, editors, Lecture Notes in Computer Science: Distributed Systems - Architecture and Implementation, pages 191–210. Springer Verlag, New York, 1981.