Comput Syst Sci & Eng (1995) 4: 195-206

Use of middleware facilities in interoperable databases

Mike P Papazoglou*, Alex Delis† and Bernd J Krämer‡

*School of Information Systems, Queensland University of Technology, Brisbane, QLD 4001, Australia (email: mikep@icis.qut.edu.au)
†Department of Computer and Information Science, Polytechnic University, Brooklyn, NY 11201, USA (email: delis@photon.poly.edu)
‡Data Processing Technology, Fern Universität Hagen, 58084 Hagen, Germany (email: bernd.kramer@fernuni-hagen.de)

Many contemporary inter-networked databases simply co-exist and act in isolation. The participating systems are typically heterogeneous since they have been developed under different usage models, data modeling formalisms and platforms. Yet there is an ever increasing demand to create a unified database system in such a way that users can have efficient access to distributed information resources. In this paper we present an architectural framework that can assist in building such coherent aggregates of database systems. Key features of this framework find their origin in distributed environments such as the OSF's DCE, OMG's OMA, and the ANSAware.However, we augment these features in order to facilitate intricate inter-database operations and negotiation.

Keywords: open distributed databases, distributed development environments, information systems interoperability

1. INTRODUCTION

To function effectively, large organizations have to handle huge volumes of information. To this end, automated information servers have been deployed at an ever increasing rate within organizations as the most effective way of dealing with this task. Typically, large companies (such as banks, corporate organizations, or multi-nationals) have developed over time independent databases for production management, personnel, sales, marketing, research and development. Although being interconnected, such databases merely co-exist with each other and are incapable of working together.

Increasingly the productivity of large scale enterprises is limited not because of labour or capital investments but rather by a lack of suitable information. For example, getting new products and services rapidly to market requires numerous, interdependent decisions by several individuals, ranging from managers and senior executives to production engineers and workers, who might be located far apart at corporate headquarters, engineering centers, production plants, suppliers and sub-contractors. Decision making under these conditions is a very slow, onerous, and constantly error-prone process.

Most contemporary inter-networked databases simply co-exist and act in isolation. The component databases in this network are typically heterogeneous – meaning that they are developed under different data modeling formalisms and platforms – and are obviously distributed, implying that corporate-wide data/information do not reside at the same physical site or processor. Yet there is an ever increasing demand to host a larger unified database on several pre-existing databases of different sizes and provide capabilities in such a way that users can have efficient access to information resources distributed throughout a local or worldwide network. In such settings, each database is not only able to communicate with other systems but it can also use the entire functionality of other existing databases to interact among each other. Redevelopment of these systems in an integrated fashion has proved too costly in terms of funding, time and organizational disruption. Making effective use of such a heterogeneous network-wide information infrastructure will require interoperability mechanisms that cope with the crucial technical problem of reusing existing databases and application software (i.e. legacy systems and applications) by means of a high level collaboration between the system components with no detrimental impact on the organizations' current use of databases. By interoperability we mean the ability of two or more systems, within a distributed network, to work together and execute tasks jointly despite having been developed under different data modeling formalisms. In such environments, what is needed to resolve the various mismatches is a distributed infrastructure within which pre-existing systems may be retrofitted and an information model which describes the component global structures and semantics and provides appropriate mappings between them.

Several of these issues have been partially addressed by distributed computing environments which aim at transforming a group of independent networked computers into a single, coherent computing resource which is open to the dynamic addition or modification of services. Research activities concentrate on providing a layer of software that masks differences among a variety of interconnected computing nodes with diversities in hardware components, programming languages, operating systems, communication protocols and security services. This enables the development of distributed applications that can tap into a network's latent power, using widespread resources such as CPUs, storage devices and software programs. To facilitate interactions among these heterogeneous systems, efforts have concentrated on providing the necessary infrastructure within which the development of distributed applications would be made possible. This infrastructure is known as *middleware* and is typically manifested in a series of integrated software components that are added on top of the existing operating systems. They typically include interprocess communication mechanisms and basic services, such as naming, trading, security, time or management services that are shared between diverse incompatible systems. Currently a number of middleware platforms are being developed by standardization groups, including the Open Software Foundation (OSF), and the Object Management Group (OMG). The most well known middleware platforms are at this stage the OSF's Distributed Computing Environment (DCE)[1], the OMG's Object Management Architecture (OMA)[2] and ANSAware[3] from Architecture Projects Management Ltd.

The goal of this paper is to critically assess the services provided by these middleware platforms and determine their suitability for supporting the interoperation of autonomous inter-networked databases and applications. The paper is organized as follows: Section 2 outlines the challenges for advanced interoperability and talks about the needed underlying technology. Section 3 describes various approaches taken and presents an architecture for interoperability. In Section 4, we discuss the additional requirements that this architecture has to address in order to work in a diversified and distributed environment of database systems.

2. ADVANCED INTEROPERABILITY

2.1 Challenges

The dominant paradigm for the next generation of database systems will involve a large number of heterogeneous information brokers (brokers for short) distributed over the nodes of a common communication network. Brokers are computerized assistants performing complicated information-intensive tasks with or without human intervention and guidance. Each broker may support a clearly discernible work task or job function, automating what it can and working synergistically with other brokers by exchanging information, expertise and negotiating on how to solve parts of the common objective. Work tasks will be executed by brokers acting autonomously, or collaboratively, depending on the resources required to complete the task.

This scenario takes into account that information flows and work processes may cover the totality of enterprises of a large organization. It aims at information systems which support users in effectively performing complicated information-intensive tasks efficiently and transparently, using the most appropriate information and computing resources (e.g. processing, knowledge and data) available in large computer networks. A clear requirement is to provide the appropriate architectural and management support for allowing diverse autonomous databases to collectively operate as a unified whole in such a way that the fact of distribution and heterogeneity among the component databases can be made transparent to applications and users. In such a system each database is not only able to communicate with other databases but it can also use the entire functionality of other databases to interact with each other on a fine grain level. Potential contributions to the paradigm of interoperable databases from individual fields are considered in subsequent sections.

2.2 Contributing technology

In recent years, object-oriented technology has been used with considerable success in the design and implementation of many application contexts which are exceedingly demanding as regards their representational power and modeling versatility. These include Artificial Intelligence, Computer-Aided Design and Manufacturing, CASE tools, user interfaces, office information and multi-media systems. Any time that an application involves not only the accurate structural but also the behavioral representation of a set of entities, object-oriented programming is the natural candidate development style.

We view the object-oriented approach as the central ingredient facilitating the selective fusion of the various enabling technologies into a coherent architectural framework for distributed processing[4]. Here, we mention briefly the reasons that support this viewpoint. The next generation distributed database systems go far beyond the experiences gained in various other fields (such as software engineering, object-oriented programming languages, object-oriented database, and multidatabase systems) in at least two ways:

- emphasis is placed on the semantic aspects of interoperable systems
- the immediate goal is the direct use and subsequent integration/interoperation of existing legacy systems (i.e. existing systems often developed using obsolete technology) in the context of distributed computing rather than examining the reusability issues within the confines of the object-oriented paradigm.

Object-oriented technology facilitates interoperation by providing a natural model for the development of dis-

tributed platforms. Distributed components can only communicate with each other using messages addressed to well defined interfaces, and distributed components can only communicate by means of locally defined procedures which enable them to intercept messages sent to and understood by them. Object-oriented programming, distributed or not, carries this model down to the level at which the components of a distributed system are seen at different levels of granularity and abstraction. Objects at a coarse granularity level may range from representing individual application abstractions to systems or nodes in a distributed environment; whereas fine-grained objects may represented individual data items[4].

The main benefits of the use of object-oriented technology in distributed information system design are that objects represent a natural framework for system modularization, accommodation of the construction of complex entities with shared subcomponents, and provide flexible module interconnection capabilities beyond the structures attainable in conventional data models. In fact, the characteristics of distributed object data management are not only natural but also orthogonal to the object-oriented paradigm[5]. In summary, object-oriented database technology provides several important features for integrating heterogeneous components to form multidatabase systems in which the individual components retain their autonomy but can also be interoperable.

In this context a number efforts have taken place to produce generic technology that provides flexible, efficient and transparent interoperability between a variety of heterogeneous interconnected component databases systems using the object-oriented paradigm. Such configurations are normally referred to as *multidatabase* or *federated* database systems. Current advances in multidatabase technology have shown the applicability of object-oriented data models for the purpose of integrating diverse component data sources and supporting interoperability[6, 7]. In fact, the technical approach to object-oriented multidatabase interoperability is based on an amalgamation of object-oriented database concepts and distributed data management methodologies.

3. MIDDLEWARE FACILITIES FOR DISTRIBUTED COMPUTING

Most successful architectural approaches to distributed computing are based on the popular client/server model[1–3, 8]. In this model a server provides some set of services on one machine which can be invoked by a variety of clients which may run on different machines. The client initiates a distributed activity which the server carries out as if it were local to the client program. Distributed file service is a typical example of the client/server model whereby a set of servers provide access to a collection of files for clients which need to operate on these files.

3.1 Distributed computing approaches

The aim of distributed environments is to provide the infrastructural support, e.g. tools, utilities, languages and libraries, that is required to integrate a wide range of computer system types and support the development and operation of distributed applications. Currently a number of standards organizations and vendors have concentrated on developing the required infrastructure and services. The most important activities include the OSF/DCE[1], the ANSAware[3], and the OMG/CORBA[2]. Below, we provide a brief description of each of these platforms and then we examine them in the light of the infrastructure required to support the management of distributed information.

DCE is based on the client/server paradigm and provides a set of services that support the development, maintenance and use of applications in a heterogeneous distributed computing environment. Communication between distributed application components is realized through synchronous remote procedure calls (RPC) spanning several address spaces. Within a single address space concurrency is provided by means of multi-threading. A directory service and a distributed file service provide data access transparency through distributed name management. A security service manages authentication and authorization. Clock synchronization across a network is enabled by a distributed time service. The current components it offers include remote procedure calls (RPC) that programmers can use for structuring client/server systems using point-to-point communication. Other features include thread service, a directory service, cell support, a time service, a security service and a distributed file system service. The DCE's RPC support encapsulation in that a server's data is accessible only via a specific set of operations, i.e. the server's interface.

ANSAware is an architecture which aims to enable application components to work together despite diversity of programming languages, hardware, operating systems and network protocols. ANSAware is based on the client/server model and utilizes RPCs for communication. The ANSA computational model places emphasis on the principles of abstraction and encapsulation. ANSAware uses the concept of trading to pass information about services from service providers to potential service consumers. Other features include thread service,a distributed processing language for interacting with processes, a security service and a distributed file system service. It is expected that both OSF/DCE's and ANSAware's long term plans will include support for object-oriented environments.

The OMG's CORBA architecture specifies an application integration framework that provides interoperating object-oriented tools and services. The overall architecture consists of a backplane, called the object request broker, which provides the basics for object distribution and a collection of 'plug-in' object services. The object request broker provides the mechanisms by which objects make and receive requests and responses, while the basic services include a repository for type information, support for object creation and deletion as well as a persistent object store.

3.2 Underlying infrastructure

As databases serving different communities are designed, built and commissioned without much regard for other such systems it would be impractical to assume complete integration at the schema level. The proposed approach is to allow for information sharing of related data objects on the basis of partial schema unification without the need for a global view of the data that is stored in the different components. Partially unified schemas comprise schema elements from component database systems distributed across the network. Consequently, a distributed information environment comprises an evolving collection of partially unified schemas spread across the network and managed in a completely decentralized manner. Information sharing and exchange is achieved by a common minimal object-oriented data model describing the structures, constraints and operations on the shared data (and consequently the various partially unified schemas in the network) and directly supporting higher-level information units and the inter-relationships across system boundaries.

To achieve this, databases are wrapped with an object 'shell' providing an interface supporting the common data model. The interface is automatically generated from object-oriented type definitions defining the common data model, and the translation from the common to the local data model, and vice versa, is achieved by structure-driven mappings relying on an object-oriented extension of syntax-directed translation schemes[9]. In addition the functionality of each database is extended to that of a real information broker able to accept client requests for services and locate the endpoint where these requests can be executed in a synergistic manner. In this way independent autonomous databases are made able to converse with one another, while new applications may be developed around the information broker programming metaphor[10, 11]. Object wrappers provide virtual homogeneity as well as ease of inter-broker communication and increased modularity.

Internetworking is achieved between clusters comprising logically closely coupled schema parts, e.g. products, customers, distributors, etc, and their associated application programs. Each cluster (or database coalition) is fairly independent of the others and is organized to best suit the constraints placed upon it. Data (i.e. object) sharing can be achieved by each cluster choosing the facilities to export to other interconnected clusters; these clusters in turn select the facilities to use (import). This type of federated system promotes peer to peer rather than hierarchical structures. Moreover, the concept of selective export and import of data and services provides autonomy for scaling[8].

The main component of a client/server-oriented multidatabase (CSOMB) architecture is the information broker as shown in Figure 1. Its responsibility is to mediate communication between client databases and servicing databases and applications. Each database may fulfil multiple roles, i.e. it could be both the client as well as the provider of data and services. The client interface can be a library of routines which allows a client database to interact with server databases and applications through the broker. This interaction is normally in the form of a request, e.g. query or transaction. The application interface at the server side configures the server environment (object implementation) to the broker environment to allow methods invoked by clients to be bound to operations of servers. As the broker has to coordinate all the processing transparently, other components such as repositories, directories or dictionaries are required in the architecture. They provide information such as the identity and interfaces of the server database and applications in order to allow the broker to locate and invoke operations on the applications.

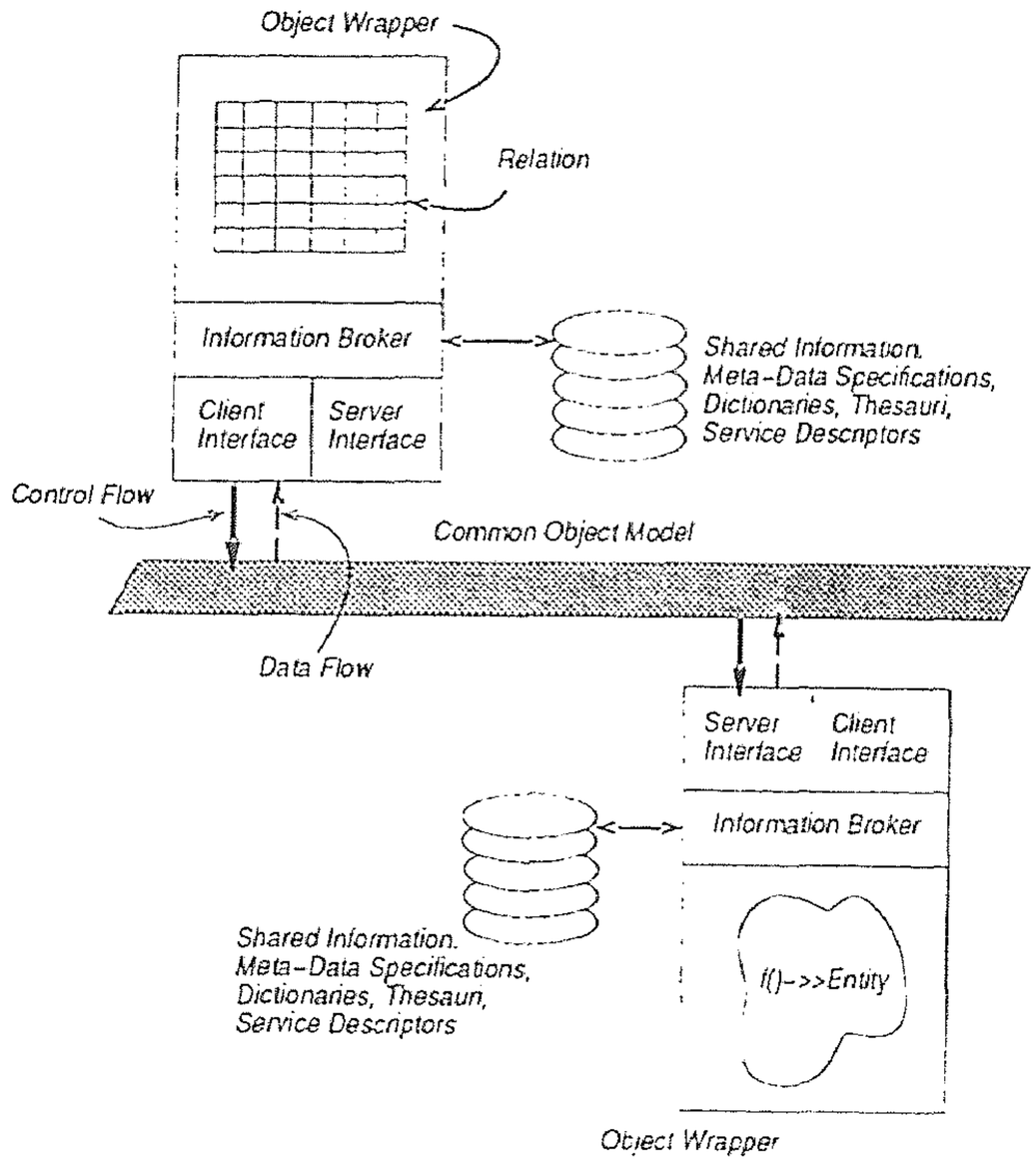

Figure 1 Information broker

The following items are provided by the conventional platforms introduced above. In this section we will review them in some detail within the broader context of client/server object management architectures required for the management of distributed database systems.

3.2.1 Communication Mechanisms

Conventional architectural approaches allow clients and servers to communicate with each other by using Remote Procedure Calls. RPCs channel client requests to the appropriate server, convert data to and from forms required by the different kinds of hosts and manage all the low-level aspects of communication by providing network transport independence, network connection management and some form of recovery from server failures. To accelerate performance RPC server stubs may use threads to allow the creation, management and synchronization of multiple threads of control within a single process to handle multiple client requests simultaneously.

RPCs are provided by both the DCE and ANSAware distributed computing frameworks. In a sense these two environments are based on an object model as servers encapsulate (computational) resources accessible through a restricted set of operations. A process holding a reference to an object becomes then the client of that object. Clients hold an opaque reference to an object and use the interface of the object to invoke operations applicable to that object. The internal structure of the object consists of its local state, operations executing on the local state and the run-time environment for that object and is provided by the server.

The above approach is slightly different from that supported by the Common Object Request Broker Architecture (CORBA) of the OMG which provides a collection of generic message passing mechanisms which allow objects to exchange messages across networks. At the communication level, message passing can be performed by RPC-like location independent object invocations with objects residing on different nodes.

In contrast to RPCs which require that value parameters are passed to the call, interoperable databases are based on the concept of message passing. Message passing allows references from client objects to be passed to server objects remotely. Parameters may range from single objects to object aggregations. For example, the ORB services requested to achieve this type of functionality are completely transparent to the client. Clients do not need to know where the objects reside on the network, or how they communicate or are implemented.

3.2.2 Interface Definitions

Interface definitions are used in both DCE and ANSAware to tie the server and client application code together. Interface definitions are written in an Interface Definition Language (IDL) used to describe the operation signatures that the interface offers and data types occurring as parameters and results of interface operations. CORBA follows a similar approach by requiring that all object interfaces are described in a declarative interface definition language with a syntax resembling C++. In fact, all three approaches support similar IDL semantics. Their differences are purely syntactic.

Although these environments provide facilities for the automatic conversion of data formats across different different hardware platforms they do not cater for type conversions between related data types at the client and server levels. Interoperability in the context of distributed databases requires that mutual understanding not only at program or system level but also at the data-type level[12]. The ORB comes somehow closer with the concept of an object adaptor which tries to align various types of object implementations. This concept has to ultimately be extended to allow for legacy systems and applications to be part of a network information system and will be examined in a later section.

Database systems rely on the existence of some form of IDL to provide interoperation among a variety of diverse data models. This form of IDL should include not only structural but also semantic information to alleviate the problems introduced by the heterogeneity of data models. Of particular interest is the work conducted in developing meta-models for describing existing schemas from diverse data models[13–15]. In Papazoglou *et al.*[15], a strategy is outlined for developing an intermediate model used to capture meta-schema information for a given database schema in a format that is independent of the underlying technology. An instance of the intermediate model meta-classes can be constructed for a specific schema with some form of enrichment provided from a domain expert. The intermediate model is capable of representing high level concepts such as containment, inheritance, aggregation, association, various types of references and constraints. The major thrust of the intermediate model is to capture sufficient semantic detail of the elements of a database schema to facilitate automatic transformation of this schema into an equivalent schema in a different data model.

3.2.3 Naming and Trading Facilities

Naming refers to the way the various entities within a complex of computing systems are referenced. In DCE the directory service allows distributed applications to store and find information about resources available in a distributed computing environment. The directory service provides a hierarchical naming scheme whereby a cell directory service (white pages method) provides the naming facilities of nodes grouped by some administrative domain – called cells – and a global directory service to control the naming environment between different cells.

ANSAware's approach to naming is provided by means of the trading facility. The ANSAware trader (i.e. locator and broker of services) allows clients to locate the services that they request. The trader stores information about service providers in a structure which is amenable to queries by potential service consumers. A trader has the task not only to enable clients to locate appropriate servers but also to bind a client to the server's interface. A yellow page scheme is used by the trader to describe the functions of the services.

DCE and ANSAware have a different approach to binding. Clients communicate with traders via import requests while traders are updated by servers via export requests. In DCE the directory service locates the node that offers the interface, then the node identifies the service to be used and binding can occur. In ANSAware the trader combines the ability to select an interface with a node and endpoint information required to establish a binding[16]. The trader's import operation is normally used for this purpose. The CORBA also provides for naming facilities which are managed by the ORB. They are similar to the facilities provided by the ANSAware traders. The main differences are that the ORB differentiates between static and dynamic interfaces. The ORB also manages the binding between client and server objects.

All of the above approaches suffer from the same limitation. They all assume that complete description of the services available throughout a distributed environment are specified *a priori* and do not normally change. This is an unrealistic assumption when considering a network

with large numbers of databases each contributing a large number of data items. Accordingly, interoperable databases should extend the approaches taken in the above systems. A request for interoperable database operations has to be resolved by identifying all the candidate systems to participate in the materialization of the requests. Recently work has been conducted in creating dynamically clusters of database nodes, in a database network, centered around areas of interest. Searches are not based upon simplistic 'string-matching' but rather on taking into account numerous types of inter-relationships between individual database systems[17].

Assuming that the client/server model is followed, requests are dispatched in a naming server facility that will be able to resolve resource references partially. The remaining references may have to be delegated to other appropriate name server facilities (based on how their areas of mutual interest are inter-related) throughout the network. In this manner, naming requests have to be propagated according to a policy (i.e. to the logically closer name server) and once this cycle of operations ends the initiating server will know the addresses of the object resources throughout the operational network (see Figure 2).

3.2.4 Static and Dynamic Invocation Support

One important point to make is that the DCE and ANSAware offer a static invocation model in the sense that their entities are assumed to be permanent and are published within the directory/trader. In this way their IDL declarations compile into stub files which allow clients to invoke stub routines on known target entities. However, in several situations applications may need to be able to make calls to objects without necessarily having compile-time knowledge of their interfaces. This results in a form of dynamic invocation interface whereby a client names the request's target object and relies on a request broker, such as the ORB, to supply the necessary arguments to the request at run-time. Such dynamic types of interface are provided only by CORBA. Again interoperable databases come far closer to the approach offered by CORBA as they require both static and dynamic object management. Type checking of objects is also required in this environment and should be managed in a manner similar to that of CORBA which offers an interface repository for this purpose.

The combination of the above facilities results in providing the basic communication protocol and interface mechanisms which shield the users from the heterogeneities of multi-vendor systems (at both the hardware and software levels) and allows them to work on a higher level of abstraction or on a logical as opposed to the physical level. However, programmers must still deal with intricate details of RPC services, synchronizing threads and providing for the recovery of data. Arguably although the current type of middleware as offered by DCE, ANSAware and CORBA provides some broad communication services it does have lots of limitations and requires several improvements, which will be covered in the remainder of this paper.

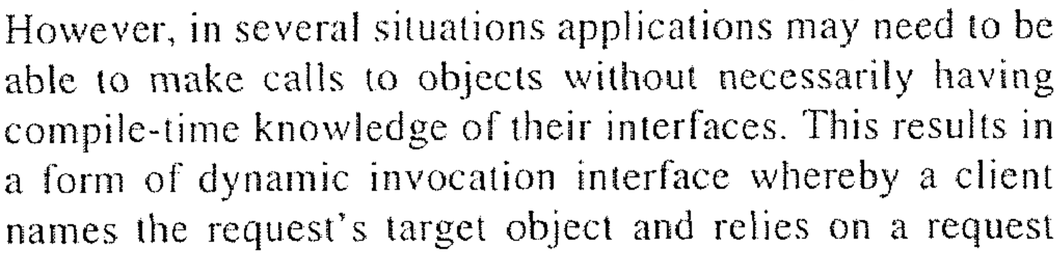

Figure 2 Inter-server organization

3.3 Core facilties for the management of distributed objects

We use the term nucleus to denote the core of a distributed object-oriented environment in which non-traditional applications can be developed, and autonomous and heterogeneous database systems can be integrated. An important feature of the nucleus is that it is independent of any supported database system and programming language. The most salient features of this nucleus for distributed objects include the management of persistent storage; management of nucleus object memory; network communications; the control of distributed computations and concurrent access to objects; as well as some form of basic object protection (Figure 3). Its purpose is to redefine applications, services and databases, to an object space larger in scope than the environment in which they were originally developed. This system provides the mechanisms that prepare an object to participate in its environment such as defining objects, locating objects, handling messages between objects, and binding methods to objects.

The nucleus is a distinct layer on top of the operating

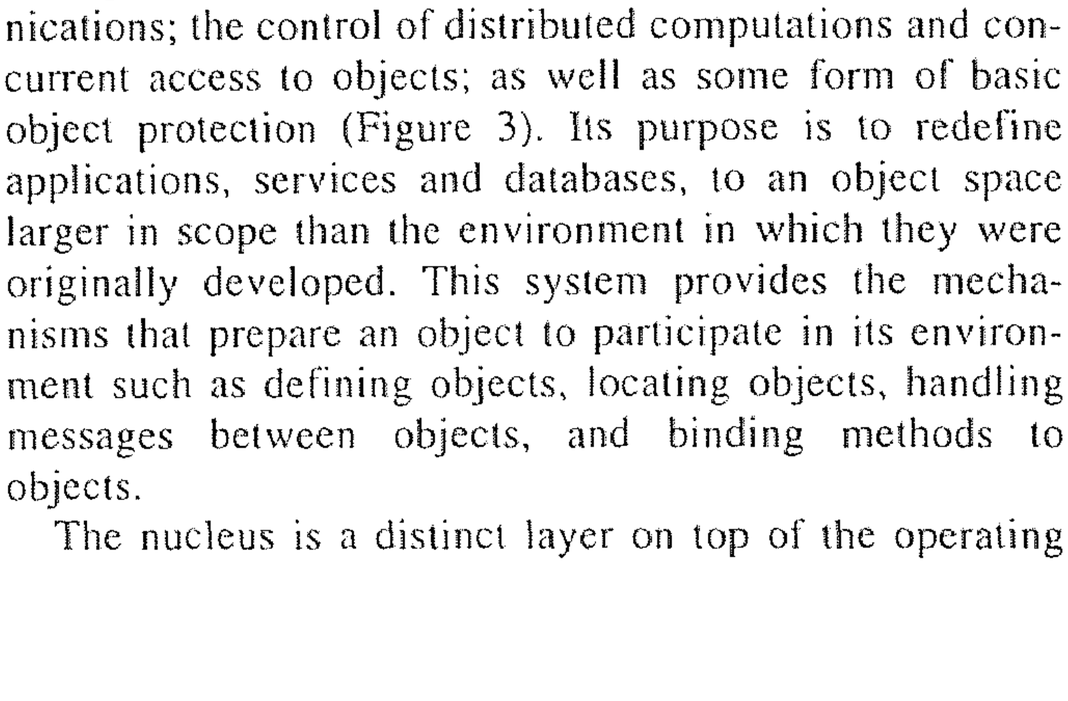

Figure 3 Core functionality

system which allows applications (objects), running under different operating systems, to communicate with each other. In effect, the interfaces to these applications (objects) become a higher level operating system. For instance, an application can be limited to the local environment as well as transcended to the nucleus environment by using the nucleus' representation of interface definitions. The interface definitions shield other users from having to know about the implementation details and the operating environment of the application.

Some of the features of the nucleus have already been described in the previous sections. In this subsection we extend the basic functions discussed above with additional features specific to database systems interoperation currently not provided by the the OSF/DCE, ANSAware and CORBA.

Object Persistence

With the application centered approaches to distributed computing such as DCE, ANSAware and CORBA the lifetime of objects – with the exception of files – is bound by the duration of the programs that create and run them. In interoperable databases, it is a requirement that several objects outlive the programs that created them. Such objects are called persistent objects and when not in use are stored in secondary memory. Accesses to these objects result in automatically fetching them from storage and making them available in virtual memory in an appropriate mode. Accordingly, the virtual object memory component of the nucleus is responsible for implementing transparent access to to distributed persistent objects and for all the other aspects of object management including object creation, low-level object naming, object location, remote invocations and the mapping and unmapping of objects to and from the secondary storage. This approach has been suggested by the Commando distributed application platform[18].

Management of Data Replicas

It is often the case that data are partially replicated due to the pre-existence of the component systems. It is always possible that a virtual object may comprise several parts some of which may be replicated. The virtual object memory component should therefore be in a position to manage replicated objects (or object portions) in a consistent manner by enforcing simultaneous updates to all the data replicas.

Both read and write operations have to happen in a transparent manner and the users should not be aware of the actual sites that are involved in the processing of their requests. DCE offers a distributed file system that could constitute the basis for building interoperable databases in the framework of a DCE cell. However, pure filing structures are far from sufficient in such environments and need to be enhanced with layers of higher database functionalities. ANSAware maintains strong emphasis on the distributed processing and not distributed data.

A server resident CORBA core could maintain information about correlated and replicated data in its vicinity in terms of its naming services. This would provide a fundamental manner of accessing and updating dispersed data. Issues of exact update propagation methods, unification of fragments (destruction), creation of new fragments and appending data are yet to be designated with specific methods.

Concurrent Access to Objects

To improve distributed application performance through parallelism the concept of threads is used by both DCE and ANSAware. Threads provide for concurrent processing within a shared distributed address space. Threads are used by RPC server stub routines for handling multiple asynchronous non-blocking remote client invocations. However, thread programming is quite intricate as programmers should protect shared resources (such as pieces of data), schedule and synchronize activities and recover from errors. The notion of threads is currently not supported by CORBA. Threads can provide for extended flexibility as they may allow ORB objects to accept several invocations.

In a database environment the execution of an activity may result in a nested invocation of methods on objects. Each invocation may take place on any node in the system. At each invocation the referenced object is located and, if necessary, loaded into virtual memory at some node. An activity may create one or more parallel subactivities within the same application. Therefore it is required that two levels of concurrency are provided: (i) between activities belonging to the same application; and (ii) between execution threads belonging to independent applications. Such support is currently not provided by any of the distributed computing platforms that we examined.

4. ADDITIONAL DATABASE INTEROPERATION REQUIREMENTS

Higher level cooperation among distinct pre-existing databases involves a number of intertwined issues. These include additional functional as well as organizational requirements. Two important organizational requirements that the middleware facilities must satisfy are autonomy and openness. The term *autonomy* characterizes the ability of a component database system to continue executing local operations and running already developed applications unaffected from any external functionality – preserving, thus, investments in application software. This implies that each component system has complete authority over its local operations and existing applications chooses the local services that other systems may invoke and does not rely on an external database system for successful operation. Openness implies the ability to extend or reconfigure the system by adding new or removing existing components. In the following subsections, we identify a number of issues that have to be addressed if flexible and open interoperable systems are to be realized.

As an example consider a large, geographically distributed multinational car manufacturing corporation. The corpo-

ration is composed of a large number of regional and market-based operating divisions, which have historically been relatively autonomous. Over time, these operating divisions have developed and deployed a number of systems customized in both structure and content for their specific needs. These systems and their contained data are 'owned' by the operating divisions, which are sensitive to any changes, either in functionality or ownership, proposed by the corporate organization. The corporate organization, however, has its own set of information needs, such as setting and meeting long-term corporate goals and ascribing accountability for various activities. The information required by the parent organization is available in the operating divisions' collective database systems, but it is neither homogeneous nor directly controllable. Dynamic, ongoing tasks like tracking markets are difficult at the corporate level when a large number of individual reports from operating divisions must be compiled and merged into a cohesive statement of corporate activity. Often this information is time-critical, and by the time the report is assembled manually the information no longer has value. The information thus may have a single owner and retention strategy, it is useful in multiple ways, and is accessed via differing retrieval mechanisms.

4.1 Distributed query processing

In response to a client request the middleware system locates the desired information and executes the appropriate type of function (i.e. query or update) by translating, interpreting and synthesizing the individual results which it presents to the user. Each node in the network may originate and submit global queries or transactions (queries or transactions with both a local and remote component) to the middleware platform, and receive and answer queries from remote nodes in the system.

The objective of the object query facility of the middleware is to provide efficient access to a variety of distributed objects on the basis of their structural and behavioral properties. Therefore, an important property of the query language is that it is a natural extension of an object-oriented language and treats objects uniformly no matter whether they are local or distributed. In other words the data language for distributed applications should provide uniform syntax and semantics of query and programming language statements.

Based on our running example a meaningful distributed query may be to obtain the total product-related revenues for *small business customers in Germany, The Netherlands, Belgium, France and the UK* over the past year, reported according to the relative sizes of cities and towns without inclusion of service revenues or other income related sources.

4.2 Virtual integration of data

During query execution a distributed database system provides its client systems with the impression of a single logically integrated database system, where in fact there may be many data sources with a high degree of data redundancy and data replication.

A central issue in supporting interoperability is achieving compatibility among the different data types supported by the diverse database systems in the network. This will enable remote data objects and their associated procedures used in one database to be shared by others despite differences in data representation. The objective is to be in a position to partially unify, or combine, schema types from diverse database systems.

DCE and ANSAware utilize mostly the notion of marshaling as offered by the Remote Procedure Calls to allow for limited format conversion. Thus, these systems hide such differences as byte orders, floating point precisions or array accessing mechanisms. This type of format conversion is not sufficient in an environment of diversified database systems.

Interoperable database systems cross organizational and technological boundaries. At these boundaries not only names, but data types and queries will have to be translated and checks should be made on the validity of interactions. It is important that the middleware uses a transparent addressing and language scheme in such a way that technology boundaries remain invisible to operations. We use the term technology boundaries to imply the existence of different data modeling facilities and formalisms, e.g. relational, hierarchical, CODASYL, object-oriented, etc, which underlie the various systems in the network.

Middleware facilities should provide type compatibility at the specification level thereby alleviating differences in representation for abstract as well as simple data types. In this way when two interoperating component systems communicate or share a common data object they do so by having consistent views of the properties of the object they mutually rely upon. Type conversions are required for translating the heterogeneous data objects to and from a single universal representation. In this regard, only the CORBA specifications provide a mechanism to assist in data conversion/integration, viz. the object adaptor. The additional requirement for interoperable databases is partial data schema translation and the return of error codes so that users do have an understanding of underlying problems. Issues of exporting and importing objects become relatively straightforward by having the servers indicate rules of bridging the various data discrepancies. Using the running example this may mean that that the operating divisions of our multinational corporation in France and Italy may use different systems such as relational vs. CODASYL which requires data type, schema, and language conversions. They also use different monetary units which also implies that the necessary conversions should be made.

An additional problem that must be solved before one can successfully unify schemas is that of semantic incompatibilities. In contrast to structural heterogeneity, this type of heterogeneity refers to the differences in the meaning and use of data that make it difficult to identify the various implicit associations that exist between similar or related data objects in different component systems. For example, although the database systems in operating

divisions in Germany and Italy may use the term 'luxurious car' they may have different classifications and interpretations. Recently, a lot of work has been conducted in this area and some results have been already reported[19, 20].

4.3 Scalability

Over time, the operation of multiple databases is expected to demonstrate dynamic behavior with individual systems joining or departing from server clusters. Obviously, these meta-data changes have to be reflected on the information repositories of the server sites. DCE internals allow for update of cell directory information and can deal with such changing situations. ANSAware's Node Manager also provides mechanisms for modifications of the services provided by the node. CORBA's core through its dynamic invocation interface may offer alternative services to the ones already compiled for a particular core. However, this falls short of the needs of an ever changing and adapting interoperable database environment.

We consider that the set of static interfaces in the CORBA core should be possible to be updated dynamically over time to cater for dynamic database attachments. In this way, it should be possible to alter the repositories on the fly – and not at initiation time – and furnish the elementary mechanisms to carry out such operations at the servers.

4.4 Negotiation

Given that the knowledge that is contained in each component system is incomplete and that component systems may have conflicting goals, negotiation is required to alleviate conflicts and formulate partial solutions in a collective manner. Negotiation has been proposed as a scheme for resolving inter-agent conflicts and exchanging pieces of information among agents. Negotiation is the mechanism used by autonomous systems to resolve inconsistent views and reach agreement on how they can work together in order to cooperate effectively[21, 22]. Both DCE and ANSAware offer only very low-level facilities in support of negotiation. Actually what they offer are facilities enabling clients to choose appropriate servers without prior knowledge of the server objects. This is achieved mainly by the use of a centralized directory service. The DCE environment provides no explicit negotiation mechanism among the participating nodes other than the yellow pages of the local directory. ANSAware provides the mechanism of the *trader* for furnishing service interfaces to sites that are not aware of the existence of these interfaces. The trading happens via export requests qualified with type names, property and value pairs, and types imports. The CORBA core offers no such service explicitly and lets the developers specify their own in the core.

The sharing and exchange of information in interoperable databases is a two staged approach. First the information providers should be located. This is normally the task of an information broker which may use a combination of directory services, yellow pages and thesauri to facilitate the finding of the relative information. Then once the relative information sources have been identified, sharing and exchange of information will eventuate. As it is not possible to store global information about an entire application at every site, negotiation is required when the system has no knowledge how to handle parts of a request. This implies that negotiation in interoperable databases is a necessity due to the large amount of data involved and its corresponding retrieval and processing overheads. Negotiation has to be an integral part of the organization of servers since it can be an excellent vehicle for achieving cost effective operations through cooperation. Negotiating requests are finally delegated to servers that have access to all pertinent data for a specific request. Figure 2 reflects this situation.

Consider, for example, the situation where a query has been made in the operating division in Germany and requests information about prices of mufflers and differentials of Italian origin. It is reasonable to assume that such information is not available locally, in which case the database client in Germany would start a negotiation phase by asking the server in Italy to provide it with the right type of information. The information that will be passed between two nodes must not only contain structural descriptions but also the context of use of the individual data items as well as access privileges (i.e. if they can be further broadcasted, changed and so on).

It is now commonly accepted that negotiation should be restricted to only a relatively small set of systems; requires a common language in which the negotiations can be couched; requires a common framework, i.e. an abstraction to the solution, to which the participants contribute; and also requires models of other systems and a unified negotiation protocol[23].

4.5 Management of distributed transactions

Most of the database requests bear a transactional nature and they require substantial management across the various distributed resources. An assumption made often is that every site that maintains data run a local transaction manager that has to be coordinated with various other managers involved in the realization of a transaction.

Most of the work in the area of multidatabase systems has relied on the existence of conventional (short) transactions; assumes the existence of a two-level nested transaction model for the processing of remote data; and adheres strictly to the classical ACID (Atomicity, Consistency, Isolation, Durability) paradigm for network-wide transaction management[24–26]. This model of multidatabase transaction processing introduces several limitations. Often multidatabase transactions result in long-lived transactions which may lock local database resources for unacceptably long periods of time delaying significantly the termination of conventional short transactions submitted at these sites (and which are outside the scope of the multi-databases).

These deficiencies have been pointed out and solutions have been suggested in the research work conducted on the area of non-conventional database transaction management for MDBS and distributed object-oriented systems[27–29].

To achieve database interoperability at the transaction level it is desirable to have more flexible and fine-grained transaction models that have to accommodate along with the traditional jobs, nested and long transactions. Nested transactions extend the flat transaction structure by allowing a transaction to invoke atomic transactions as well as atomic operations. They allow the potential internal parallelism to be exploited. They also provide finer control over failures by limiting the effects of failures to a small part of the transaction. This implies that if a sub-transaction is unnecessarily delayed it may be rescheduled or one may opt for executing an alternate transaction with similar or near similar effects. Consider again, for example, the situation where a transaction has been made in the operating division in Germany and requests among other information about automobiles information about prices of mufflers and differentials of Italian origin. If the execution of this transaction detects that the Italian site's database is not available, it might be sensible to issue an alternate subtransaction requesting prices of mufflers and differentials of French origin in the hope that the user may eventually resort to French differentials. This model of transactions provides added flexibility for a client/server-oriented multi-databases and we consider it as an integral part of interoperable systems. It is our strong belief that future interoperable database systems will rely on the use of special purpose scripting languages to support transaction programming.

Equally important is to cater for recovery mechanisms that are going to help the overall system's recovery. Log-oriented methods may not be the best option to be deployed in the servers and, after all, not all resources may demonstrate any recovery capabilities. Thus flexible mechanisms for recovery management should also be investigated. Both DCE and ANSAware provide very limited support for transaction management while the CORBA-core specifications do not deal with this issue at any significant level, since it is believed to be level higher than that of the core.

4.6 View management

An interoperable database system is a highly sophisticated environment comprising a large mass of complex and inter-related data items and a series of partially unified schemas. In fact the data in the system is composed of aggregations of data items some of them at a fine grained granularity. The user or developer of this environment will be more effective if they deal with global abstractions of the large complex set of distributed data. The concept of view allows the construction of global abstractions that describe a unified structure and behavior over a large set of objects, some of which may be distributed[30, 31].

Most conventional approaches to object-oriented databases suggest either the use of query language expressions for defining views or the usage of special language features. Queries can be used to define virtual classes populated by selecting existing objects from schema classes and by creating new objects. The extent of these view classes is usually not stored explicitly but rather computed from the view defining query upon request. Some approaches treat the virtual classes as stand alone objects or attach them directly to the root of object-oriented schema. Other more ambitious approaches either classify the virtual classes derived by query language statements into one schema or reorganize the database by introducing hierarchies of virtual classes created through generalization or specialization. In all cases, views are semantics preserving since they introduce only new information as computed attribute values, e.g. by merging existing attributes, or by hiding attributes[32, 33] and importing existing schema types[32].

View management is an important aspect of interoperable databases as it allows users and developers to ignore a huge object space and focus only in these parts of an application which are of interest to them. For example, if a user of any of the operating division in the multinational corporate organization of our running example is continuously interested in information about mufflers and differentials ignoring other parts of automobiles it is natural to assume that a view extracting these two products from their surrounding environment possibly combining them into a single entity may be created.

4.7 Security and authorization

The RPC communication in DCE is based on a security service which provides authentication and authorization services to help protect resources against illegitimate access. Application clients and servers run a distributed authentication protocol in conjunction with security servers which validate their identity. ANSA and CORBA have not yet introduced any security concepts into their environments.

Authentication is a must in a diversified environment of multiple cooperating systems. Certain users can access selectively resources throughout the network. There must be mechanisms in place to ensure that user access legitimacy. ANSAware provides no authentication and security while DCE offers authentication and authorization mechanisms. In DCE an access control list is maintained for all the clients that are authorized to access the various cell resources. The CORBA-core specification makes no explicit provision for security and authorization although such mechanisms could be built within the core.

5. SUMMARY

The next generation of database systems are characterized as dynamically reconfigurable aggregations of multicomponent systems and applications. As these heterogeneous open databases grow and expand their functionality, effi-

cient integration capabilities and trading functions are needed to keep track of available information and processing services and match client requests with appropriate service offerings. The architectural paradigm for such systems will concentrate on client and server components comprising current generation systems and pre-existing applications collaborating over high bandwidth communications media.

Integration concerns will focus around providing an expandable set of software modules and services that can be used for building systems that use diverse problem-solving methods and incorporate various pre-existing components of independently or complementary developed technologies to collaboratively solve a problem. The ability to efficiently develop interoperable databases from a software engineering point of view relies significantly on our ability to reuse or reengineer existing application code, i.e. programs and schema definitions, database services, as well as the development of environments and high level tools for composition and integration of these parts.

The activity aims at designing generic concepts and core mechanisms that enable the integration of heterogeneous information resources and processing components to interoperable databases while obviating the need to modify the body of individual component systems. A central component of this research consists in the design of a distributed object management system to underlie the development of an open, distributed architecture. The core technology necessary for achieving transparent interoperability is based on client/server architectures for improved performance, and makes use of object-oriented technology to attain wide-scale interoperation. The broader goal of the distributed object management system is to provide application inter-operation through a common approach to transaction, naming, sharing and service trading. Interoperability will be provided by means of object-oriented extensions to heterogeneous systems which are capable of exchanging messages, requesting services and coordinating receipt of responses. Particular emphasis has to be placed on important software design and engineering issues such as reusability of existing systems and applications software,adaptability of component systems, and system extensibility.

REFERENCES

1 OSF-DCE, *Introduction to OSF-DCE*, Revision 1.0, Prentice Hall, Englewood Cliffs, NJ (1992)

2 *The Common Object Request Broker: Architecture and Specification*, OMG TC Document Number 91.12.1, Revision 1.1 (December 1991)

3 *ANSA Reference Manual*, Release 1.01, APM Ltd., Cambridge UK (July 1989)

4 **Manola, F *et al.*** 'Distributed Object Management', *Int. J. Intelligent & Cooperative Infor. Syst.*, Vol 1 No 1 (March 1992) pp 5-42

5 **Fong, E, Kent, W, Moore, K and Thompson, C (eds)** *X3/SPARC/DBSSG/OODBTG Final Report* (September 1991)

6 **Jenq, B P, Woelk, D, Kim, W and Lee, W** 'Query Processing in Distributed Orion', *EDBT 90: Int. Conf. on Extending Database Technology* (May 1990)

7 **Kim, W, Ballou, N, Garza, J F and Woelk, D** 'A Distributed Object-Oriented Database System Supporting Shared and Private Databases'', *ACM Trans. Infor. Syst.*, Vol 9 No 1 (January 1991)

8 **Delis, A and Roussopoulos, N** 'Performance and Scalability of Client-Server Database Architectures', *Proc. 18th Int. Conf. Very Large Data Bases (VLDB '92)*, Vancouver, BC, Canada (August 1992)

9 **Krämer, B and Schmidt, H W** 'Object-oriented development of integrated programming environments with ASDL', *IEEE Software* (January 1989)

10 **Papazoglou, M P** 'Knowledge-Driven Distributed Information Systems', *14th Int. Computer Software & Applications Conf.: COMPSAC*, Chicago, IL (October 1990)

11 **Collet, C, Huhns, M N and Shen, W** 'Resource Integration Using a Large Knowledge Base in Carnot', *IEEE Computer*, Vol 24 No 12 (December 1991)

12 **Wileden, J C, Rosenblatt, W and Tarr, P L** 'Specification-Level Interoperability', *Comm. ACM*, Vol 34 No 5 (May 1991)

13 **Barsalou, T and Gangopadhyay, D** 'An Open Framework for Interoperation of Multimodel Multidatabase Systems', *Proc. Data Eng. Conf.* (March 1992) pp 218–227

14 **Urban, S** 'A Semantic Framework for Heterogeneous Database Environments', *Proc. 1st Int. Workshop on Interoperability in Multidatabase Systems* (April 1991) pp 156-163

15 **Papazoglou, M P, Tari, Z and Russel, N** 'Object Technology in Interschema Transformations', in *Object-Oriented Multidatabase Systems*, A. Elmagarmid, O. Bukhres (eds), Prentice-Hall (1995)

16 **Beitz, A, King, P and Raymond, K** 'Comparing two Distributed Environments: DCE and ANSAware', *Workshop on Distributed Computing Environments*, Springer-Verlag, Lecture Notes on Computer Science no. 731, A. Schill (ed)

17 **Milliner, S, Bouguettaya, A and Papazoglou, M** 'A Scalable Architecture for Autonomous Heterogeneous Database Interactions', *Proc. 21st Conf. Very Large Databases (VLDB)*, Zurich, Switzerland (September 1995)

18 **Cahil, V *et al.*** 'The Comandos Distributed Application Platform'', Springer-Verlag, Research Reports ESPRIT (1993)

19 *RIDE: Research Directions in Interoperability IEEE Workshop Proceedings*, eds. H. Scheck, A. Sheth (1993)

20 **Hammer, J and McLeod, D** 'An Approach to Resolving Semantic Heterogeneity in a Federation of Autonomous, Heterogeneous Database Systems', Vol 2 No 2 (March 1993) pp 51–84

21 **Conry, S E, Meyer, R A and Lesser, V R** 'Multistage Negotiation in Distributed Planning', *Readings in Distributed Artificial Intelligence*, Morgan Kaufmann, San Mateo CA (1988)

22 **Cammarata, S, McArthur, D and Steeb, R** 'Strategies of Cooperation in Distributed Problem Solving'', *Proc. IJCAI* (also in Readings in Distributed Artificial Intelligence, Morgan Kaufmann, San Mateo, CA) (1988)

23 *AAAI-92 Workshop on Cooperation Among Heteroge-*

neous Intelligent Systems, San Jose, CA (July 1992)

24 **Breitbart, Y and Silberschatz, A** 'Multidatabase update issues'. *Proc. ACM SIGMOD Int. Conf. Manage. of Data* (June 1988)

25 **Pu, C** 'Superdatabases for Composition of Heterogeneous Databases', *IEEE Proc. 4th Int. Conf. Data Eng.* (February 1988)

26 **Elmagarmid, A and Du, W** 'A Paradigm for Concurrency Control in Heterogeneous Distributed Database Systems', *IEEE Proc. 6th Int. Conf. Data Eng.* (February 1990)

27 **Buchmann, A, Ozsu, M T, Hornick, M, Georgakopoulos, D and Manola, F A** 'A transaction Model for Active Distributed Object Systems', In A. Elmagarmid, editor, *Transaction Models for Advanced Database Applications*, Chapter 5, Morgan-Kaufmann, San Mateo, CA (February 1992)

28 **Levy, E, Korth, H F and Silberschatz, A** 'An Optimistic Commit Protocol for Distributed Transaction Management', P*roc. ACM SIGMOD Int. Conf. Manage. Data* (May 1991)

29 **Rusinkiewicz, M, Krychniak, P and Cichocki, A** 'Towards a Model for Multidatabase Transactions', *Int. J. Intelligent and Cooperative Infor. Syst.*, Vol 1 No 3 (September 1992)

30 **Abiteboul, S and Bonner, A** 'Objects and Views', *SIGMOD-91*, Denver, CO (May 1991) pp 238–247

31 **Schilling, J and Sweeney, P** 'Three Steps to Views: Extending the Object-Oriented Paradigm', *Proc. OOPSLA '89* (October 1989) pp 353–361

32 **Abiteboul, S and Bonner, A** 'Objects and Views', *Proc. SIGMOD-91* (May 1991) pp 238–247

33 **Scholl, M, Laasch, C and Tresch, M** 'Updatable Views in Object-Oriented Databases', *Proc. 2nd DOOD Conf.* (December 1991) pp 189–207