Interoperability in Large-scale Distributed Information Delivery Systems

Ling Liu, Ling Ling Yan, and M. Tamer Özsu

University of Alberta, Department of Computing Science
Edmonton, Alberta, Canada T6G 2H1

Abstract. In this paper we address interoperability issues in large-scale distributed information delivery systems. Architecturally, we classify existing approaches and systems into two paradigms: *Multidatabase management-based* paradigm and *Mediator-based* information delivery paradigm, and analyze the techniques used in each. Technically, we describe a number of data delivery characteristics in terms of delivery protocols, delivery modes, and delivery frequencies. We further use these characteristics to discuss and compare several data delivery schemes. We argue that an advanced distributed information system must incorporate different types of information delivery so that the system can be optimized according to various criteria, such as network traffic and heterogeneity and constant evolution of online information sources. To illustrate the architectural and technical aspects of distributed information delivery systems, we review a number of research prototypes to demonstrate the various implementation approaches used in practice, and the different solutions to the interoperability issues addressed in the paper.

1. Introduction

In the past few years there has been an explosion in the amount and diversity of information available across networks. The proliferation of Internet and intranets and the ongoing advances in the World Wide Web (WWW or Web) have fueled the development of a wide range of data-intensive applications and information dissemination systems. Many new ways are being explored to deliver information contents to users in office, at home, and on the road.

A common problem facing many organizations and enterprise computing systems today is the uniform and scalable access of multiple, disparate information sources and repositories, including databases, object stores, knowledge bases, file systems, digital libraries, and information retrieval systems. It is widely recognized that information sources change constantly, and users are faced with the daunting challenges of navigating, collecting, evaluating, and processing data in this dynamic and open information universe. Decision makers often need information from multiple information sources but are unable to get and fuse information from multiple information sources in a timely fashion, not only due to the unpredictable state of networks and the contention at information sources, but also due to the heterogeneous and evolving nature of information sources.

An "advanced" distributed information system is an open and interoperable system, rather than a static data delivery system. Two immediate functional requirements of such a system is the support of extensible distributed object management and dynamic interoperability among diverse information sources and between information consumers and information producers.

In terms of object-oriented terminology, *interoperability* refers to the ability to exchange requests between objects and the ability to enable objects to request services of other objects, regardless of the language in which the objects are defined and their physical location (e.g., hardware platforms, operating systems, DBMS's). *Distributed interoperable objects* are objects that support a level of interoperability beyond the traditional object computing boundaries imposed by programming languages, data models, process address space, and network interface [Bet94]. The abstractions of distributed interoperable objects are captured and utilized by the distributed object management services to schedule and control the remote data access and delivery.

A key objective in providing interoperability and distributed object management in a dynamic and open information universe is the *scalability* and *effectiveness* of remote data access and delivery.

- Scalability refers to the ability of distributed object management services to scale the process of delivering information (in the form of objects) from a set of data sources to a typically larger set of consumers, with respect to both the heterogeneity in hardware platforms, process address spaces, operating systems, data models and programming languages, and the evolving nature of information sources.
- Effectiveness of remote data access and delivery refers to the ability of distributed object management services to incorporate the constant information evolution and explosion, the network traffic, and the availability of information sources or communication links, into the distributed query optimization and execution strategies.

Query optimization and evaluation strategies have long been studied in centralized, parallel, and tightly-coupled distributed environments. However, data access across widely-distributed and highly autonomous information sources imposes significant new challenges for distributed object management for a number of reasons. First, there are semantic and performance issues that arise due to the heterogeneous nature of the data sources. Second, the amount and diversity of online information available across networks is exploding. Users today are faced with increasing difficulty in collecting, processing, and integrating information effectively and in a timely fashion. As the scale and rate of change for online information continues to grow, the user-initiated, comprehensive searching is no longer sufficient as a dominant mode of information access and delivery. To improve the query responsiveness, remote data access and delivery must combine user-initiated, comprehensive searching with source-initiated dissemination of relevant information, and migrate from pre-established communication endpoints to anytime-access-anywhere on the globe. Finally, data access over wide-area networks today depends heavily on the specific data sources accessed and the current state of the network at the time that such access is attempted, including the availability of information sources, intermediate sites, and communication links.

In this paper we describe a number of techniques and strategies that are used or can be used to address these technical challenges. We also present a brief review of the state-of-the-art in research and development for accessing multiple and heterogeneous information sources. The reminder of the paper is organized as follows. We outline the architectural issues in supporting interoperability of multiple and heterogeneous information sources in Section 2., and outline a number of mechanisms for effective information delivery in distributed and interoperable information systems in Section 3.. In Section 4. we overview several representative systems or ongoing research projects that contribute to the issues of dynamic interoperability and scalable distributed object management. A few popular enabling technologies for deployment and implementation of interoperable distributed object management services is summarized in Section 5.. In Section 6. a brief overview of the AURORA project that we are currently developing for the electronic commerce domain is presented. We end the paper with a summary and some remarks on the role of database technology within the global information infrastructure, in particular the wide-range of data-intensive applications being deployed on the Internet and intranets.

2. Architectural Issues

Over the last decade, several approaches and paradigms have been proposed for information access and delivery from multiple heterogeneous information sources. To simplify the review, we classify the state-of-the-art of research and development into two paradigms: *Multidatabase management-based* paradigm and *Mediator-based* information delivery paradigm. In this section we concentrate on the architectural issues of data access and delivery with respect to these two paradigms. We discuss the techniques and challenges for delivering information effectively and responsively, in Section 3..

2.1 Multidatabase-Based Paradigm

Multidatabase management has evolved over years and through several stages. A classic approach [Ram91, She91] for multidatabase management relies on building a single global schema to encompass the differences among the multiple local database schemas. The mapping from each local schema to the global schema is often expressed in a common SQL-like language, such as HOSQL in the Pegasus system [Aea91] or SQL/M in the UniSQL/M system [Kea93]. Although the enforcement of a single global schema through data integration yields full transparency for uniform access, component databases have much restricted autonomy, scalability and their evolution becomes difficult.

The federated approach [SL90] improves the autonomy and the flexibility (composability) of multidatabase management by relying on multiple import schemas and the customized integration at various multidatabase levels. However, the integration of component schemas at each multidatabase level is enforced by the system. The integrated schema is static. The heterogeneity problems are resolved at the schema integration stage. This approach cannot scale well when new sources need to be added into an existing multidatabase system. Also the component schemas cannot evolve without the consent of the integrated schema.

The distributed object management approach [Mea92, ODV93] generalizes the federated approach by modeling heterogeneous databases of different levels of granularity as objects in a distributed object space. It requires the definition of a common object model and a common object query language. Recent activities in the OMG and the ODMG standard [Cea94], which extends the OMG object model to the database interoperability, are important milestones for distributed object management.

2.2 Mediator-Based Paradigm

Mediator-based information integration architecture has evolved from an initial proposal by Gio Wiederhold in [Wie92] and elaborated through the intelligent information integration (I^3) program [Wie95]. A mediator-based system consists of a network of mediators. *A mediator is a software module that exploits encoded knowledge about some sets or subsets of data to create information for a higher layer of applications* [Wie92]. Intuitively, each mediator offers a specific data service and acts as an information "broker" in a particular application domain. Mediators often have knowledge-based capabilities.

The most interesting features that distinguish the mediator-based paradigm from conventional multidatabase management approach are the following:

- On the information source side
  1. The information universe it addresses is large-scale, dynamic, and open in nature, rather than small-scale, static, and closed.
  2. Information sources considered in mediator-based systems include not only *structured* sources such as relational databases, object stores, and knowledge-bases, but also *semi-structured* information sources such as HTML files, WWW pages, or *unstructured* data sources such as plain text documents, images, video clips.
  3. Information sources are highly autonomous. The amount of information sources and applications available online is very large and grows rapidly. The content, the number, and the connectivity of information sources

change constantly. Heterogeneity problems become a natural and unavoidable consequence.

– On the interoperability management side

1. The set of functions for interoperability management are divided and packaged into two architectural tiers: mediator-tier and wrapper-tier. The mediator-tier is responsible for interfacing with applications and end-users. The wrapper-tier is responsible for interfacing with underlying information sources. A wrapper can be seen as a special type of mediator that deals with idiosyncrasies of the individual data sources such as translating a mediator tier request to executable instructions at the source.
2. There is no single global view or system that can serve for all applications. Different mediator-based systems may access shared information sources. All information sources accessed within a mediator system are wrapped using the mediator system-specific interface language. The same information source may be wrapped differently for different mediator systems. A wrapper serves as an agent or a delegate of a particular mediator system to communicate with the underlying (wrapped) information sources.
3. The scalability and extensibility issues become a major concern and an important evaluation criteria for interoperability management. Object-oriented design, programming, and development technology is a powerful and yet practical paradigm for building distributed information systems.
4. The role of ontology and classification hierarchies becomes increasingly important for dynamic interoperation and heterogeneity resolution between information consumers and information producers.

Architecturally, mediator-based systems are more flexible since wrappers are built independent of one another and are used to serve for all accesses to the corresponding data sources in the system.

There are many prototype systems and ongoing projects for developing mediator-based interoperability management systems. They vary in the way mediators and wrappers are built, functionality and capability that different mediators may provide, and ways of dividing functional components between mediator-tier and wrapper-tier. We will discuss some of the representative systems in Section 4..

3. Technical Issues

With the ongoing advance in WWW technology, everyone today can publish information on the Web at any time. The flexibility and autonomy of producing and sharing information on WWW is phenomenal. On the other hand, one has to learn to deal with the rapid increase of volume and diversity of online information and the constant changes of information sources in number, content, and location. Thus, queries to the current WWW search tools are mostly specified independent of the structure, location, or existence of requested information. One simply types in the keywords, the search tools will handle the request and find the sources that match the given keywords. However, the scalability is achieved at the price of effectiveness of queries, namely the quality and the responsiveness of the answers, for several reasons. First, responses returned by WWW search tools often contain too much irrelevant information (noise). Second, queries in network-centric information systems are more vulnerable to failure due to the congestion of networks, traffic at the intermediate sites and the contention at the sources. Thus frequently one needs information from multiple information sources but is unable to get and fuse the information from information sources in a timely fashion.

A practical optimization solution to these problems is to provide technologies that support a variety of data delivery schemes and allow the scheduling process of queries to be tuned at run-time according to the state of networks and the availability of intermediate sites and source sites. One example of such technology is to combine and interleave the user-initiated, comprehensive search-based data delivery with the server-initiated dissemination of relevant information.

In this section we describe a number of data delivery characteristics [FZ96] in terms of protocols, delivery modes, and delivery frequencies, and use these characteristics to discuss and compare several data delivery schemes.

3.1 Data Delivery Protocols

Data delivery is defined as the process of delivering information from a set of information sources (servers) to a set of information consumers (clients). There are several possible ways that servers and clients communicate for delivering information to clients, such as clients request and servers respond, servers publish what are available and clients subscribe to only the information of interest, or servers disseminate information by broadcast. Each way can be considered as a protocol between servers and clients, and has pros and cons for delivering data in an open and dynamic information universe.

3.1.1 Clients Request and Servers Response. The *request/response* protocol follows the data delivery mechanism that clients send their request to servers to ask the information of their interest, servers respond to the requests of clients by delivering the information requested.

Current database servers and object repositories deliver data only to clients who explicitly request information from them. When a request is received at a server, the server locates or computes the information of interest and returns it to the client. The advantage of the *request/response* protocol is the high quality of data delivery since only the information that is explicitly requested by clients is delivered. In a system with a small number of servers and a very large number of clients, the *request/response* mechanism may be inadequate, because the server communication and data processing capacity must be divided among all of the clients. As the number of clients continuous to grow, servers may become overwhelmed and may respond with slow delivery or unexpected delay, or even refuse to accept additional connections.

3.1.2 Servers Publish and Clients Subscribe. The *publish/subscribe* protocol delivers information based on the principle that servers publish information online, and clients subscribe to the information of interest. Information delivery is primarily based on the selective subscription of clients to what is available at servers and the subsequent publishing from servers according to what is subscribed.

As the scale and rate of changes for online information continues to grow, the *publish/subscribe* mechanism attracts increasing popularity as a promising way of disseminating information over networks. Triggers and change notifications in active database systems bear some resemblance to the *publish/subscribe* protocol based on point-to-point communication [AFZ97]. The *publish/subscribe* mechanisms may not be beneficial when the interest of clients change irregularly because in such situations clients may be continually interrupted to filter data that is not of interest to them. A typical example is the various online news groups. Another drawback is that publish/subscribe is mostly useful for delivering new or modified data to clients, but it cannot be used to efficiently deliver previously existing data to clients, which the clients later realize they need. Such data are most easily obtained through the request/respond protocol.

3.1.3 Servers Broadcast. The *broadcast* mechanism delivers information to clients periodically. Clients who require access to a data item need to wait until the item appears. There are two typical types of broadcasting: *selective broadcast* (also called *multicast*) and *random broadcast* [FZ96]. Selective broadcast delivers data to a list of known clients and is typically implemented through a router that maintains the list of recipients. Random broadcast, on the other hand, sends information over a medium on which the set of clients who can listen is not known *a priori*. Note that the difference between selective broadcast and *publish/subscribe* is that the list of recipients in selective broadcast may change dynamically without explicit subscription from clients.

The *broadcast* protocol allows multiple clients to receive the data sent by a data source. It is obvious that using broadcast is beneficial when multiple clients are interested in the same items. The tradeoffs of broadcast mechanisms depend upon the number of clients who have common interests and the volume of information that is of interest to a large number of clients [FZ96, AFZ97].

3.2 Data Delivery Modes

With the rapid growth of the volume and variety of information available online, combined with the constant increase of information consumers, it is no longer efficient to use a single mode of data delivery. A large-scale modern information system must provide adequate support for different modes of data delivery in order to effectively cope with the various types of communications between clients and servers to improve query responsiveness. Another benefit of providing different modes of data delivery is to allow the system to be optimized for various criteria according to different requirements of data delivery. In this section we identify three potentially popular modes of data delivery and compare them with the types of delivery protocols that can be used. They are client pull-only option, server push-only option, and client pull with server push combined option.

3.2.1 Pull-only Mode. In the *pull-only* mode of data delivery, the transfer of data from servers to clients is initiated by a client pull. When a client request is received, the server responds to it by locating the requested information. The *request/respond* style of client and server communication is *pull-only*.

The main characteristic of pull-based delivery is that the arrival of new data items or updates to existing data items are carried out at a server without notification to clients unless clients explicitly poll the server. Also, in pull-based mode, servers must be interrupted continuously to deal with requests from clients. Furthermore, the information that clients can obtain from a server is limited to when and what clients know to ask for. Conventional database systems (including. relational and object-oriented database servers) offer primarily pull-based data delivery.

3.2.2 Push-only Mode. In *Push-only* mode of data delivery, the transfer of data from servers to clients is initiated by a server push in the absence of specific request from clients. The main difficulty of push-only approach is to decide which data would be of common interest, and when to send them to clients (periodically, irregularly, or conditionally). Thus, the usefulness of server push depends heavily on the accuracy of a server to predict the needs of clients. *Broadcast* style of client and server communication is a typical *push-only* type.

In push-only mode, servers disseminate information to either an unbounded set of clients (random broadcast) who can listen to a medium or a selective set of clients (multicast) who belong to some categories of recipients that may receive the data. It is obvious that the push-only data delivery avoids the disadvantages identified for client-pull approaches such as unnoticed changes. A serious problem with push-only style, however, is the fact that in the absence of a client request the servers may not deliver the data of interest in a timely fashion. A practical solution to this problem is to allow the clients to provide a profile of their interests to the servers. The *publish/subscribe* protocol is one of the popular mechanisms for providing such profiles. Using publish/subscribe, clients (information consumers) subscribe to a subset of a given class of information by providing a set of expressions that describe the data of interest. These subscriptions form a profile. When new data items are created or existing ones are updated, the servers (information providers) publish the updated information to the subscribers whose profiles match the items.

3.2.3 Hybrid Mode. The hybrid mode of data delivery combines the client-pull and server-push mechanisms. The continual query approach described in [LPBZ96] presents one possible way of combining the pull and push modes, where the transfer of information from servers to clients is first initiated by a client pull and the subsequent transfer of updated information to clients is initiated by a server push.

The hybrid mode represented by continual queries approach can be seen as a specialization of push-only mode. The main difference between hybrid mode and push-only mode is the initiation of the first data delivery. More concretely, in a hybrid mode, clients continuously receive the information that matches their profiles from servers. In addition to new data items and updates, previously existing data that match the profile of a client who initially pull the server are delivered to the client immediately after the initial pull. However, in push-only mode, although new data and updates are delivered to clients with matching profiles, the delivery of previously existing data to clients that subsequently realize that they need it is much more difficult than through a client pull.

3.3 Data Delivery Frequency

There are three typical frequency measurements that can be used to classify the regularity of data delivery. They are

- *periodic*: Data is delivered from server to clients periodically. The period can be defined by system default or by clients using their profiles.
- *conditional*: Data is delivered from servers whenever certain conditions installed by clients in their profiles are satisfied. Such conditions can be as simple as a given time span and as complicated as sophisticated ECA rules.
- *ad-hoc* or *irregular*: A data delivery is requested at any time from clients to servers and the matched data items are sent to clients from servers within a reasonable response time.

3.3.1 Periodic Data Delivery. Both pull and push can be performed in periodic fashion. Periodic delivery is carried out on a regular and pre-specified repeating schedule. A client's weekly requests for the stock price of IBM is an example of periodic pull. Periodic pull is a simpler case of the request/respond protocols. An example of periodic push is when an application can send out stock price listing on a regular basis, say every morning. Using period push, a set of data items is sent out periodically according to a pre-defined schedule. Since the pattern repeats, a client who misses a data item in one period of the pattern can get it in the next one.

Periodic push is particularly useful for situations when clients might not be available at all times or might be unable to react to what has been sent, such as in the mobile setting where clients can become disconnected.

3.3.2 Conditional Data Delivery. Conditional delivery is mostly used in the hybrid or push-only based delivery systems. Using conditional push, data are sent out according to a pre-specified condition rather than any particular repeating schedule. An application that sends out stock prices only when they change is an example of conditional-push. An application that sends out the balance statement only when the total balance is %5 below a pre-defined balance threshold is an example of hybrid conditional push. Conditional push assumes that changes are critical to the clients and that clients are always listening and need to respond to what is being sent. Hybrid conditional push further assumes that missing some update information is not crucial to the clients.

3.3.3 Ad-Hoc Data Delivery. Ad-hoc delivery is irregular and is performed mostly in a pure pull-based system that uses traditional request/ respond protocol of data delivery. Data is pulled from servers to clients in an ad-hoc fashion whenever clients request it. In contrast, periodic pull arises when a client uses polling to obtain data from servers based on a regular period (schedule).

3.4 General Remarks

An advanced distributed information system must incorporate different types of information delivery so that the system can be optimized according to various criteria, such as network traffic and heterogeneity and constant evolution of online information sources. Based on a number of important characteristics of distributed information delivery discussed in the previous sections, we provide below a brief comparison of data delivery mechanisms.

The three protocols that are considered here are *request/respond*, *publish/subscribe* and *broadcast*. Table 1 shows a comparison of the use of different protocols or frequencies of data delivery with respect to the modes of data delivery.

Table 1. A combination of techniques applicable to different modes of data delivery

	request/ respond	publish/ subscribe	broadcast	periodic	conditional	ad-hoc
pull only	Y			Y		Y
push only		Y	Y	Y	Y limited	
hybrid	Y	Y	Y	Y	Y limited	Y

As we mentioned in Section 3.1, most of the conventional database systems are pull-based data delivery systems and use the request/respond protocol for data delivery, where clients pull servers by sending requests to the servers and servers respond by locating and computing the data items that match the requests and delivering the results to the corresponding clients. Clients can pull the servers anytime (including a periodic time span controlled by clients) whenever there is a need for information from servers. Since the pulling of servers is always initiated by users in an ad-hoc fashion, rather than monitored by a computer program automatically, neither broadcast, publish/subscribe nor conditional delivery is relevant.

The pure push style of data delivery is very useful when the volume of information of common interest is huge and/or the number of clients who are interested in the same amount of information is large. The push-only delivery can also be seen as an optimization strategy that reduces the traffic and server load when the volume of information of common interest at a particular server continues to grow. The publish/subscribe protocol is typically used for server-initiated information dissemination, where data is delivered automatically (push-only) or semi-automatically (hybrid mode). The frequency of such server push is either according to a periodic (regularly repeated) schedule or based on a conditional schedule. Ad-hoc delivery frequency does not make much sense in practice for push-only mode. The broadcast protocol is also a push-only delivery protocol. But it is mostly used for server-initiated periodic delivery. The publish/subscribe protocol can be seen as a specialization of the broadcast protocol by incorporating profiles of clients that specify what groups of information to subscribe, such that the servers only publish the information to the clients, which match their profiles.

4. Overview of Some Existing Projects

We discuss in this section a few projects that are either the well-known systems (such as Carnot), or mostly related to the mediator-based architecture discussed in Section 2. (such as TSIMMIS, DIOM, DISCO, InfoSleuth), or to the data delivery mechanisms outlined in Section 3. (such as Broadcast Disks project). Space limitation prevents us from being exhaustive.

4.1 Broadcast Disks Approach to Information Dissemination

The broadcast disks approach [AAFZ95, AFZ97, FZ96] to information dissemination uses a periodic push-only data delivery with broadcast protocol. The broadcast disks paradigm is based on a cyclic broadcast of objects (e.g., pages) and a corresponding collection of client cache management techniques. The main idea is to explore multi-level frequencies of disks and their relationship to cache management. For example, using broadcast disks, groups of objects (e.g., pages, disks) are assigned different frequencies depending on their probability of access. By broadcasting higher priority items more frequently, their access times can be reduced at the expense of increasing the latency for lower priority items. A key issue here is the generation of a broadcast schedule that can deliver the data items of different priority requirements most efficiently.

A broadcast schedule is generated based on the number of disks, the relative frequencies of each disk and the assignments of data items to the disks on which these items are to be broadcast [AFZ97]. Interesting to note is that the multi-levels are in fact superimposed on a single broadcast channel by interleaving the data items of the various levels in a manner that results in the desired relative frequencies. For instance, consider a simple broadcast of three objects (pages), say A, B, C, arranged on two disks, where A is on a disk that is spinning twice as fast as the disk where B, C are located. If we have two levels of disk frequency, say 2:1, data item A has a higher priority than data items B and C. Thus, a broadcast schedule would generate the following broadcasting pattern: "A, B, A, C, ..." [FZ96]. One of the difficulties in deciding how to structure a broadcast schedule is that the server must use its knowledge to reason about the needs of the clients who require disseminated data. Intelligent management of client caches is a key to solving this problem.

As we mentioned earlier, most of database systems and current Internet browsers are pull-only style of data delivery systems. The Broadcast Disks project developed a set of strategies and algorithms for a periodic push style of data delivery. Currently, an integrated dissemination-based information system is under development as a continuation to the Broadcast Disks project at Brown University and University of Maryland, aiming at supporting a wider range of ways to deliver data to clients [AFZ97].

4.2 Carnot

The Carnot Project [CHS91, WCH$^+$93] at MCC has developed and assembled a large set of generic facilities for managing integrated enterprise information. These facilities are organized into five sets of services: communication services, support services, distribution services, semantic services, and access services. The Carnot architecture is shown in Figure 1.

The communication services provide the user with a uniform method of interconnecting heterogeneous equipment and resources. These services implement and integrate various communication platforms that may occur within an enterprise. Examples of such platforms include ISO OSI session and presentation layer protocols running on top of ISO TP4 with CLNP, TCP/IP via a convergence protocol, or X.25. Additional platforms being considered are OSF's DCE and UI's Atlas.

The support services implement basic network-wide utilities that are available to applications and other higher level services. These services currently

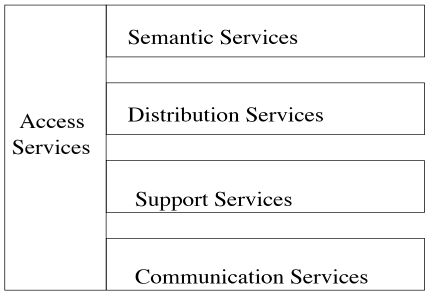

Fig. 1. Architecture of Carnot

include the ISO OSI Association Control (ACSE), ISO OSI Remote Operations (ROSE , CCITT Directory Service (X.500), CCITT Message Handling System (X.400), ISO Remote Data Access (RDA), and MIT Project Athena Authentication Service Kerberos). Work is progressing on support for additional services, such as ISO Transaction Processing, OMG's Object Request Broker (ORB) and Basic Object Adapter (BOA), interfaces to Information Resource Dictionary Systems (IRDS), and Electronic Data Interchange (EDI).

The distribution services layer supports relaxed transaction processors and a distributed agent facility that interacts with client applications, directory services, repository managers, and Carnot's declarative resource constraint base to build workflow scripts. The workflow scripts execute tasks that properly reflect current business realities and accumulated corporate folklore. The declarative resource constraint base is a collection of predicates that expresses business rules, inter-resource dependencies, consistency requirements, and contingency strategies throughout the enterprise. Carnot's Distributed Semantic Query/Transaction Manager (DSQTM), based on the work of OMNIBASE [Rus89], provides relaxed transaction processing services. It dynamically expands a query to include access to all semantically equivalent and relevant information resources, and also groups any updates to these resources as a set of independent transactions that interact according to a dynamically defined relaxed transaction semantics.

The semantic services provide a global or enterprise-wide view of all the resources integrated within a Carnot-supported system. This view, or portions of the view, can be compiled for use within the distribution services layer. The Enterprise Modeling and Model Integration facility uses a large common-sense knowledge base as a global context and federation mechanism for coherent integration of concepts expressed within a set of enterprise models. A suite of tools uses an extensive set of semantic properties to represent an enterprise information model declaratively within the global context and to construct bidirectional mappings between the model and the global context. There are two interesting features of semantic services: First, a key to coherent integration is Carnot's use of the Cyc common-sense knowledge base as a global context. Further, Carnot uses the knowledge representation language of Cyc to express both the information structures and the processes of an enterprise. The broad coverage of Cyc's knowledge enables it to serve as a fixed-point for representing not only the semantics of the formalisms, but also the semantics of the modeled domains. Second, the relationship between a domain concept from a local model and one or more concepts in the global context is expressed as an articulation axiom equivalence mapping. Enterprise models are then related to each other – or translated between formalisms – via this global context by means of the articulation axioms. As a result, each enterprise model can be integrated independently, and the articulation axioms that result do not have to change when additional models are integrated. This same technology can also be used to integrate database schemas and database views. Besides its common-sense knowledge of the world, Cyc knows about most data models and the relationships among them. This enables database transactions to interoperate semantically between, for example, relational and object oriented databases.

The access services provide mechanisms for manipulating the other four Carnot services. The access services allow developers to use a mix of user interface software and application software to build enterprise-wide systems. Some situations (such as background processing) utilize only application code and have no user interface component. In other situations, there is a mix of user interface and application code. Finally, there are situations in which user interface code provides direct access to functionalities of one or more of the four services.

4.3 DIOM

The DIOM project [LPL96, LP97] presents a concrete implementation of the mediator-based interoperability management infrastructure described in Section 2.2. Users may pose queries to DIOM on the fly, namely queries can be specified independently of the structure, the content, or the existence of information sources. The DIOM query mediation manager first filters the queries by creating user query profiles and then dynamically matches the queries to the information sources that are relevant at the time the query is processed. Additional features that distinguish DIOM from other systems include the dynamic query scheduling strategies, such as the dynamic query routing algorithms, the dynamic query execution planning strategies, and the set of result assembly operations. In DIOM application-specific mediators are created through specialization of meta mediator query processing framework, which includes DIOM metadata manager for managing user query profiles and source capability profiles, the distributed query scheduler, and the generic wrapper manager functions. Figure 2 presents a sketch of the DIOM system architecture currently developed using SunJDK version 1.1 and accessible from `http://ugweb.cs.ualberta.ca/d̃iom/`.

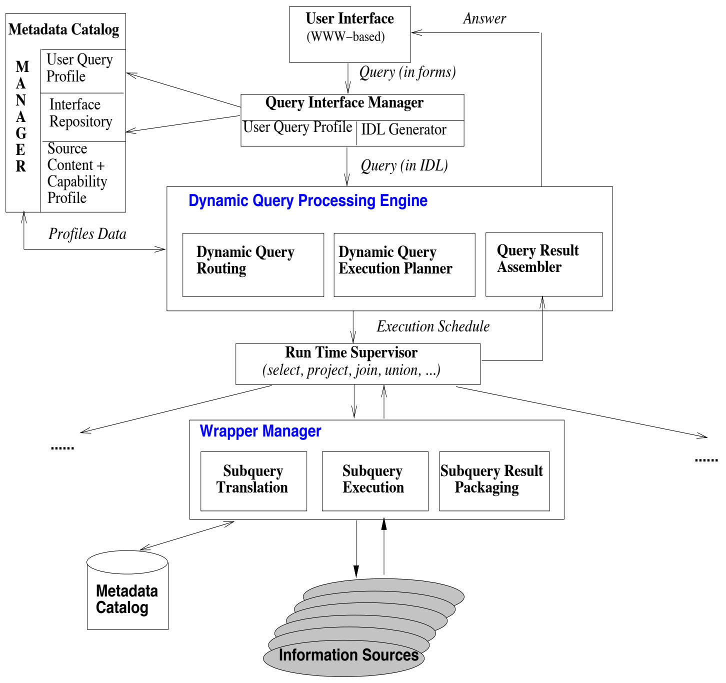

Fig. 2. The distributed query scheduling framework in DIOM

The main task of DIOM distributed query mediation manager is to coordinate the communication and distribution of the processing of consumer's queries between a mediator and wrappers to the relevant data sources. The processing at mediator layer includes query routing, query execution planning, and result assembly. The processing at wrapper layer involves the subquery translation and execution and the subquery result packaging.

Query routing is responsible for locating and selecting relevant information sources that can actually contribute to answering a query. *Query execution planner* is responsible for the decomposition of a user query into a set of subqueries, each targeting at a single source, and then generating a query execution schedule that is optimal in the sense that it utilizes the potential parallelism and the useful execution dependencies between subqueries to restrain the search space, minimize the overall response time, and reduce the total query processing cost. *Subquery translation and execution* process basically transforms each subquery into an executable program that can be executed at the source. The DIOM *query result packaging and assembly* process involves two semantic resolution phases for resolving the semantic variations among the subquery results: (1) packaging each individual subquery result into a DIOM object (done at wrapper level) and (2) assembling results of the subqueries in terms of the consumers' original query statement (done at mediator level). The semantic attachment operations and the consumers' query profiles are the main techniques that are used for resolving semantic heterogeneity implied in the query results [LP97].

4.4 DISCO

DISCO, the Distributed Information Search COmponent [TRV96, TRV97], adopts the general mediator-based paradigm, and can be seen as another concrete instance of the mediator-based approach. The DISCO mediator data model is based on ODMG-93 data model specification. It extends the ODMG-93 Object Definition Language (ODL) with two constructs:

1. *Extents.* An interface in DISCO may have a bag of extents. Each extent in this bag mirrors the extent of objects of a particular data source, associated with the interface. This extension is fully integrated into the ODMG model, the full modeling capabilities of the ODMG model are available for organizing data sources. DISCO evaluate queries on extents and thereby on data sources.
2. *Type mapping.* This extension associates type mapping information between a mediator type and the type associated with a data source.

In addition, DISCO defines two standard ODMG interfaces: `Wrapper` and and `Repository`. A DBA can add a new data source into DISCO in a few steps. First, the DBA creates an instance of the `Repository`:

```
r0 := Repository (host="rodin.inria.fr", name="db",
      address="123.45.6.7")
```

Second, the DBA locates a wrapper, implemented separately, for the data source. A wrapper is an object with a standard interface that identifies the schema and functionality of a source. It is also able to answer queries, such as `w0 := WrapperPostgress()`. In the third step, the DBA defines the type in the mediator which corresponds to the type of the objects in the data source.

```
interface Person {
     attribute String name;
     attribute Short salary;}
}
```

Finally, the DBA specifies the extent of this mediator type: `extent person0 of Person wrapper w0 repository r0;` This adds extent `person0` to the `Person` interface. The name `person0` is determined by the name of the data source in the repository, it could be a relation name or an object type name. To add a new data source containing `Person` objects, say `person1`, one needs to add an extent called `person1` to the mediator type `Person` explicitly, i.e., `extent person1 of Person wrapper w0 repository r1;` DISCO provides constructs to specify a name for the `Person` extent that includes all the sub-extents specified as above. Assume this extent is defined to be `person`. Then

DISCO is able to process queries such as `select x.name from x in person where x.salary > 10`.

Whenever a new source becomes available or a new relation or class type is added to an existing source, it is DBA's task to include the new object (class or relation or data source) into the existing mediator types by following the above procedure. This process can become expensive when a large number of mediators need to include this new class or this new source.

Query processing in DISCO is performed by cooperation between the mediator and the wrappers in two ways: (1) Determine whether a subquery can be evaluated by the wrapper; (2) Determine the cost of a query execution plan where the wrapper evaluates some subqueries. A partial query evaluation model where *the answer to a query is another query* is also studied, which defines semantics of accessing unavailable data sources [TRV96].

4.5 InfoSleuth

InfoSleuth [BBa97] is a mediator-based project at MCC that extends Carnot technology to meet the challenges presented by the World Wide Web. Although Carnot has developed semantic modeling techniques that enable the integration of static information resources and pioneered the use of agents to provide interoperation among autonomous systems, it was not designed to operate in a dynamic environment where information sources change over time and new information sources can be added autonomously and without central control.

InfoSleuth integrates the following new technological development in supporting mediated interoperation of data and services over information network:

- Agent Technology. Specialized agents that represent the users, the information resources, and the system itself cooperate to address the user' information requirements. Adding a new source implies adding a new agent and advertising its capabilities. This provides a high degree of decentralization of capabilities, which is the key to system scalability and extensibility.
- Domain models (ontologies). Ontologies give a concise uniform and declarative description of semantic information, independent of the underlying syntactic representation or the conceptual models of information bases.
- Information brokarage. Broker agents match information needs with currently available resources, so retrieval and update requests can be properly routed to the relevant sources.
- Internet Computing. Java and Java Applets are used to provide users and administrators with system-independent user interfaces, and to enable ubiquitous agents that can be deployed at any source of information regardless of its location or platform.

As shown in Figure 3, InfoSleuth is comprised of a network of cooperative agents communicating by means of the high level query language KQML.

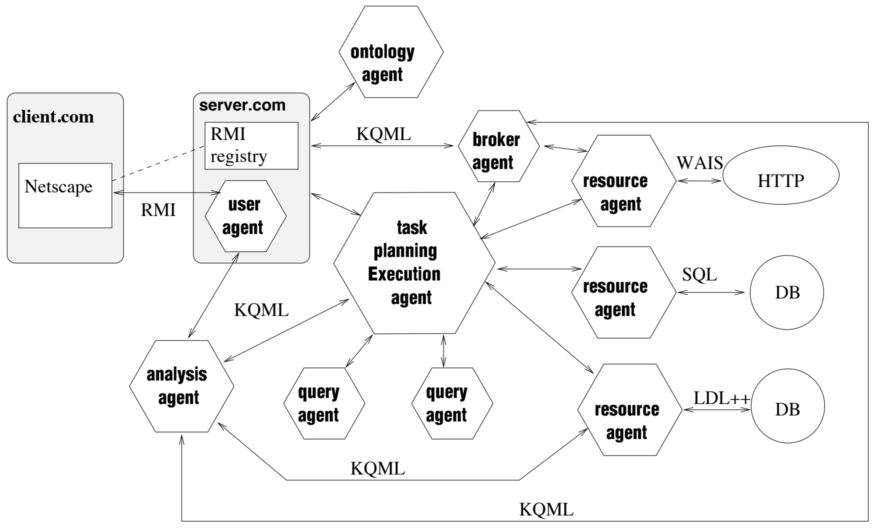

Fig. 3. Architecture of InfoSleuth

Users specify requests and queries over specified ontologies via applet-based user interfaces. The dialects of the knowledge representation language KIF and SQL are used internally to represent queries over specified ontologies. Queries are routed by mediation and brokage agents to specialized agents for data retrieval from distributed resources, and for integration and analysis of results.

Agents process requests either by making inferences based on local knowledge, by forwarding the request to a more appropriate agent, or by decomposing it into a collection of subrequests and then routing these to other agents and integrating the results. Decision on relevant source selection is based on the InfoSleuth ontology, a body of metadata that describes agents' knowledge and their relationships with one another.

4.6 MIND

The METU INteroperable DBMS (MIND) [DDO96] is a multidatabase system that supports integrated access to multiple heterogeneous and autonomous databases. MIND can access Oracle 7, Sybase, Adabas and MOOD, a home grown OODB developed at METU, Turkey. The canonical data model and query language of MIND are object-oriented. MIND differs from other multidatabase systems in that it uses CORBA as the model for managing the distribution and system level heterogeneities. MIND has a distributed and object-oriented architecture as shown in Figure 4.

The central components of MIND are two object classes: the Global Database Agent (GDA) class and the Local Database Agent (LDA) class.

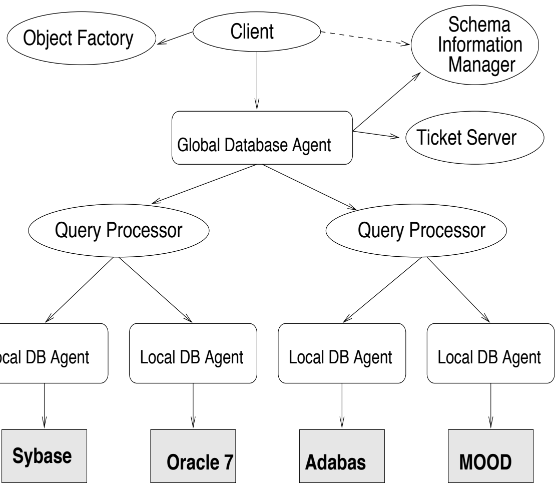

Fig. 4. MIND Architecture

Objects of these classes can be created by an object factory. A LDA object acts as a local DBMS driver, and is responsible for maintaining export schemas provided by the local DBMSs, and translating queries received in the canonical query language to the query language of the local DBMSs. A GDA object is responsible for parsing, decomposing and optimizing the queries according to the information obtained from the Schema Information Manager object. It also provides global transaction management that ensures serializability of multidatabase transactions without violating the autonomy of local databases. When a user wants to query MIND, a GDA object is created by the object factory. The location and implementation transparency for this object is provided by ORB. A GDA object contains an object of the Global Query Manager (GQM) class, which processes queries, and an object of the Global Transaction Manager (GTM) class which performs global transaction.

MIND views each participating database as an DBMS object registered with an ORB with a standard interface (but different implementation). Objects in individual databases are not registered with the ORB, that is, they are not accessible via the ORB; they are only accessible by the local DBMS where they reside. For example, consider a data source storing `Person` information. With the MIND approach, the interface of `Person` objects is not known to the ORB. The fact that MIND does not allow registration of fine-granularity objects makes MIND different from the distributed object management approach towards database interoperability as described in [Man92], where all objects in all databases form an object space that is accessible via the ORB. The way MIND uses CORBA is largely as a sophisticated communication backplane, and it has little impact on the major technical aspects of MIND, such as schema integration and query processing.

Schema integration in MIND is performed by DBAs using an object definition language that allows specification of interfaces of objects in the global schema and how they relate to objects exported by various data sources. MIND also develops some query processing techniques, especially in global query optimization [EDNO96, ONK+96], including (1) cost-based global query optimization in case of data replication. This technique deals with site selection issues in cases when a subquery can be executed at more than one site. (2) cost-based inter-site join optimization. This technique starts from a left-deep join tree and tries to transform this tree into a more bushy tree so that response time can be reduced by exploiting parallelism. (3) dynamic optimization of inter-site joins. This technique is still cost-based but is dynamic in that it uses partial results at run time, do some cost estimation and determine the next step. This approach reduces uncertainties in cost estimation.

4.7 TSIMMIS

The TSIMMIS project at Stanford [PGMW95] [PGGMU95] [PGMU96] [PAGM96] represents a big step away from most previous work. Rather than a semantically rich, structured data model, TSIMMIS uses a self-describing model, the *Object Exchange Model* (OEM) for expressing integration and for querying. OEM is an information exchange model; it does not specify how objects are stored, it only specifies how objects are to be sent and received.

In TSIMMIS, one does not need to define in advance the structure of an object and there is no notion of schema or object class. Each object instance contains its own schema. An OEM object consists of four fields: an *object id*, a *label* which explains its meaning, a *type* and a *value*. Mostly, the object ids are internal strings that are used for linking objects. The following OEM object describes a person Fred:

$< \mathrm{p1}$, *person-record, set*, $\{component_1, component_2, component_3, \} >$
$< component_1$, *name, string, "Fred"*$>$
$< component_2$, *office-number-in-building-5, integer, 333*$>$
$< component_3$, *department, string, "Toy"*$>$

Each data source to be accessed are viewed as a collection of OEM objects in the above form, with no predefined structure. Querying in OEM is via *patters* of the form <*object-id, label, type, value*>, where constants or variables can be put in each position. When a pattern contains constants in the label (value) field, it matches successfully only with OEM objects that have the same constant in their label (value). For instance, the following pattern would match successfully with person Fred given earlier:

<*person-record,* {<*name "Fred"*>, <*department "Toy"*>}>

Essentially, this pattern matches with all *person-record* that has a component *name* with value "Fred" and a component *department* with value "Toy". Notice that this pattern matching assumes no structure on the objects, as long as the object has the right label with the right value, it matches successfully. This effectively makes the labels (*person-record*, *name*, *office-number-in-building-5*, *department*) first-class citizens. Labels do not put any constraints on what type of queries are acceptable, rather, they can be queried themselves.

Queries and view specifications in TSIMMIS are also formed using patterns. The TSIMMIS Mediator Specification Language (MSL) is a rule-based language. For instance, the following rule defines a view *ToyPeople* that contains names of all people who work in the Toy department:

<ToyPeople, {<Name N>}>:- <person-record, {<name N>, <department "Toy">}>.

The following query finds all persons who have name "Fred":

FredPerson :- FredPerson:<person-record, {<name "Fred">}>

In this query, *FredPerson* is an *object variable*. The formula to the right of :- says that *FredPerson* must bind to all *person-record* with a sub-object by the label of *name* and value of "Fred". The symbol :- says that all such objects are included in the query result. Notice that the query result is potentially heterogeneous with objects with all sorts of structures, except that each object must have a label *person-record* and a *name* sub-object with value "Fred".

TSIMMIS wrappers must be built for every data source in the access scope. TSIMMIS provides a *wrapper implementation toolkit* to support fast generation of wrappers. These wrappers are indeed an OEM query processor. The wrapper implementer is required to (1) describe the types of OEM queries that the source can handle using query templates; and (2) map these query templates to local queries/actions at the data source.

Intuitively, OEM is so simple and flexible that it can represent data of any type, from unstructured random records, to relational data, to complex objects. After all types of data is represented in OEM, they can then be integrated using the set of techniques developed in TSIMMIS. The TSIMMIS approach uses logic rules that transform and merge OEM objects from various data sources to form a mediator view. This view can then be queried. Query processing in TSIMMIS leverages deductive database techniques; it includes view expansion and execution plan generation. In [PGMW95], various aspects of the OEM model are defined and discussed. In [PGGMU95], an approach for developing OEM wrappers for semi- or unstructured data sources is described. In [PGMU96], an OEM-based mediation language and its implementation is described. This language allows creation of integrated views in the mediator that removes various types of semantic conflicts. In [PAGM96], an approach for object matching (referred to as object fusion in this paper) using OEM is described. This approach allows resolution of instance level conflicts. An approach for global optimization of queries posed against these "fused" object is also described.

In the database community, OEM is also the representative of an emerging data model that is not constrained by database schemas. This feature alone removes a major representational heterogeneity among data sources. The labelled-tree structures like those in OEM can represent all sorts of data structures equally well and have a great potential in supporting integration of heterogeneous data. Query and manipulation language and optimization techniques are being developed for this new data model [BDHS96].

4.8 Comparison with respect to Data Delivery Capabilities

In previous sections we have described several projects or prototype systems in terms of the capabilities and mechanisms they use to process queries through mediators and wrappers and the support they provides for interoperability and scalable distributed object management.

In addition to Broadcast Disks project which implements a push-only data delivery system with broadcast protocol, all the others are pull-only data delivery systems, although the DIOM project has developed techniques for supporting continual queries [LPBZ96], which use hybrid mode of data delivery with publish/subscribe protocol. Table 2 shows a brief comparison of these systems with respect to the variety of data delivery capabilities.

Table 2. A comparison with respect to delivery capabilities

	request/ respond	publish/ subscribe	broadcast	pull	Push	hybrid
Broadcast Disk		Y	Y		Y	Y
Carnot	Y			Y		
DIOM	Y	Y		Y		Y limited
DISCO	Y			Y		
InfoSleuth	Y			Y		
MIND	Y			Y		
TSIMISS	Y			Y		

5. Enabling Technology for Interoperability

5.1 CORBA

The CORBA (Common Object Request Broker Architecture) technology enables object-oriented computing in distributed heterogeneous environments.

The main features of CORBA include the ORB Core, the Interface Definition Language (IDL), the Interface Repository (IR), the language mappings, the stubs and skeletons, dynamic invocation and dispatch (DII and DSI), the object adaptors and the inter-ORB protocols. The general structure of an ORB is illustrated in Figure 5.

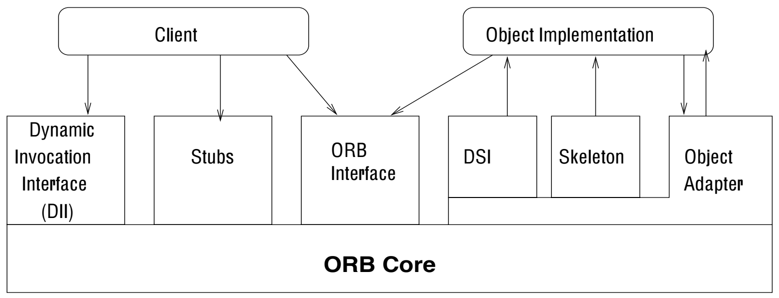

Fig. 5. Common Object Request Broker Architecture

The ORB core delivers requests to objects and returns any responses to the callers. The key feature is to facilitate transparent client/object communication by hiding object location, implementation, execution states, and communication mechanisms. To make a request, the client specifies the target object by using an *object reference*. These are created upon CORBA object creation; they are immutable and opaque references that only ORB cores know how to manipulate.

Before a client can make requests on an object, it must know the types of operations supported by the object. The Interface Definition Language (IDL) is used to define an object's *interface*. IDL is a *language independent*, declarative language, not a programming language. It forces interfaces to be defined separately from object implementations. Objects can be constructed using different programming languages and yet communicate with one another. *Language Mappings* determine how IDL features are mapped to facilities of a given programming language. OMG has standardized language mappings for C, C++ and Smalltalk. IDL language mappings are where the abstractions and concepts specified in CORBA meet the "real world" of implementation.

A CORBA-based application must have a way to know the types of interfaces supported by the objects being used. The CORBA Interface Repository (IR) allows the OMG IDL type system to be accessed and written programmatically at runtime. IR is itself a CORBA object that has a standard interface. Using this interface, an application can traverse an entire hierarchy of IDL information. IR is usually used together with the CORBA dynamic invocation interface and as a source for generating static support code for applications.

OMG IDL language compilers and translators generate client-side *stubs* and server-side *skeletons*. These are interface specific code segments that cooperate to effectively exchange requests and results. A stub is a mechanism that effectively creates and issues requests on the client's behalf. A skeleton is a mechanism that delivers requests to the CORBA object implementation. Communicating through stubs and skeletons is often called *static invocation* because the stubs and skeletons are built directly into the client application and object implementation. For clients, a stub is an *adapter* that adapts the function call style of its language mapping to the request invocation mechanism of the ORB. To object implementation, the skeleton is a *proxy* that handles translation issues in passing request in the right format to an object implementation and also passing result back to the ORB. In addition to the static invocation via stubs and skeletons, CORBA supports two interfaces for dynamic invocation: *Dynamic Invocation Interface* (DII) and *Dynamic Skeleton Interface* (DSI). DII allows clients to invoke requests on object without stubs. DSI allows servers to be written without skeletons.

The Object Adaptor (OA) is the "glue" between CORBA object implementation and the ORB itself. OA is an object that adapts the interface of another object to the interface expected by a caller. It is an object that uses delegation to allow a caller to invoke requests on an object even though the caller does not know the object's true interface.

Prior to CORBA 2.0, one of the biggest complaints about commercial ORB products was that they did not interoperate. Lack of interoperability was caused by the fact that earlier CORBA specification did not mandate any particular data formats or protocols for ORB communications. CORBA 2.0 specifies an interoperability architecture based on the *General Inter-ORB Protocol* (GIOP), which specifies transfer syntax and a standard set of message formats for ORB interoperation over any connection-oriented transport. CORBA 2.0 also mandates the *Internet Inter-ORB Protocol* (IIOP), which is an implementation of GIOP over TCP/IP transport. With IIOP, ORBs can interoperate with one another over the Internet.

5.2 COM/OLE

This is the alternative standard to CORBA that is developed by Microsoft. COM (Common Object Model) is similar in functionality to CORBA ORB, while OLE (Object Linking and Embedding) is the complete environment for componentization for handling compound documents. COM/OLE is less well-defined than CORBA and since it is a single vendor proposal, its contents are fluid and changing. Currently, it is a single machine environment with distribution to come with the release of *Cairo*.

COM object model is quite different than CORBA's; COM objects are really not "objects" in the commonly accepted sense of the term. The main differences with CORBA object model are the following:

– A COM (or OLE) object is one which supports one or more interfaces as defined by its class. Thus, there could be multiple interfaces to an object.
– All objects support one interface called **IUnknown**.
– COM objects have no identifiers (OIDs).
– There is no inheritence defined among object classes. The relationship among them is defined by means of containment/delegation and aggregation.
– COM objects do not have state; applications obtain a pointer to interfaces that point to the methods that implement them.
– There are two definition languages: IDL for defining interfaces and ODL for describing object types.

Clients access COM objects by means of the interfaces defined for each object. This is accomplished by indirection through an *Interface Function Table* each of whose entries points to an interface implementation inside the COM object. There is one Interface Function Table per each interface that the object supports. The client obtains a pointer to the Interface Function Table that corresponds to the particular interface that it wishes to access and invokes the interface functions contained therin. This method isolates clients from interface implementations.

A *COM server* performs a number of functions. It encapsulates a COM object and a class factory. In addition to the COM object interfaces that it supports, it provides an **IClassFactory** interface to interact with the class factory. The functions that the server performs are the following: (a) it implemetns a class factory interface, (b) it registers the classes that it supports, (c) it initializes the COM library, (d) it verifies that the library version is compatible with the object version, (e) it implements a method for terminating itself when no clients are active, and (f) it terminates the use of the library ewhen it is no longer needed.

There are three types of COM servers. An *in-process server* is one that shares the same process space as the clients that connect to it. A *local server* runs on the same machine as the clients, but in a different process space. The interconnection between the clients and the COM server in this case is by means of lightweight RPC. Finally, a *remote server* is one that runs as a separate process on a separate machine. In this case, the connection between the clients and the server is by means of DCE RPC.

6. AURORA and its Application to Electronic Commerce

The AURORA project at the University of Alberta, in collaboration with IBM Canada, is developing a collection of mediators that can be used for constructing integrated (read only) access to heterogeneous and autonomous data sources. The goal of the project is to make such access scalable and efficient. The target application driving the research in AURORA is electronic commerce.

6.1 A Motivating Example: Virtual Catalogs in Electronic Commerce

A virtual shopping mall is a typical electronic commerce (EC) application. A key component in a virtual mall is a catalog system. Companies organize their catalogs differently. This gives rise to a set of heterogeneous and autonomous catalogs. When the number of participating catalogs is large, it is difficult for a shopper to perform cross-catalog searching or comparative shopping in order to locate items of interest. One possible solution to this problem is to require all vendors to re-organize their catalogs in a common format and merge all the catalogs into a central catalog database which allows customers to perform sophisticated searching without dealing with individual catalogs. This requires re-engineering of existing catalogs. In general, vendors want to participate in the central catalog without making changes to their existing catalogs. One solution is to create a *virtual catalog* that has the look and feel of a central catalog but holds no physical catalog information. Upon a customer request, this catalog retrieves relevant information from (multiple) individual catalogs and assemble an answer. Such a virtual catalog should satisfy the following requirements:

- It is up-to-date but does not violate the autonomy of the participating catalogs.
- Its search performance does not degrade as the number of participating catalogs increases.
- It allows easy inclusion of new catalogs and integrates with other EC applications.
- It is easy to construct. Tools should be provided to assist in construction.

The AURORA project potentially allows construction of such virtual catalog. AURORA adopts the mediator paradigm as described in [Wie92]. It has two main themes: (1) a scalable mediation model; and (2) the enabling techniques.

6.2 Homogenization and Integration Mediators: a 2-tier Mediation Model

A *mediation model* describes how heterogeneities among data sources are perceived and handled. The AURORA mediation model, shown in Figure 6, is a 2-tier model. It is 2-tier because it models mediation as a 2-step process: homogenization followed by integration, performed by respective mediators. The original mediator framework proposed by [Wie92] encourages specialization of mediators but using specialized mediators for accessing heterogeneous data sources has not been explored before. Most previous mediator systems are 1-tier, providing a single type of mediator. AURORA's 2-tier mediation model is designed to allow scalable mediation.

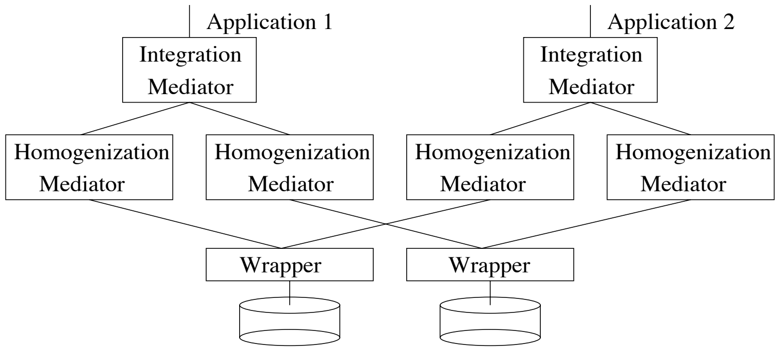

Fig. 6. The AURORA Mediation Model

6.2.1 AURORA's 2-tier Mediation Model. We distinguish between two categories of heterogeneities among data sources: *schematic mismatches* that arise when the same application domain is modeled differently; and *instance level conflicts* that arise when inconsistent values on the same real world object are recorded. Schematic mismatches must be resolved first, otherwise there is no basis for further integration. The process of resolving schematic mismatches is referred to as *homogenization*. In AURORA, specialized mediators, the *homogenization mediators*, support this process. The task of homogenization is to map a common application model onto a participating data source that models the same application (differently and partially) by constructing a *homogenizing view* on top of it. The *integration mediator* "glues" a large number of data sources together by combining all the homogenizing views into an integrated view. In this process, instance level conflicts must be resolved.

6.2.2 The 2-tier Mediation Model and Scalability. To include a new data source into the access scope, one must resolve two issues:

1. *Communication.* It must be possible to "talk" to the data source. A wrapper is used to remove idiosyncrasies of the data source.
2. *Semantic integration.*

Scalability requires that both steps be performed rapidly. For 1, this requires rapid, if not automatic, wrapper generation. Various enabling techniques have already been developed [PGGMU95]. Much is known about *how* 2 can be done; the scalability aspect of it has not been well addressed. The traditional approach is to *derive* an integrated view from all the data sources in a monolithic integration specification. This approach does not favor scalability because adding or removing a data source potentially requires the integration specification to be modified. There has been no methodology that prescribes incremental modifications. When the number of data sources involved is large, the specification is difficult to modify or reconstruct.

AURORA's 2-tier mediation model requires that data sources be *homogenized* before being integrated. The scalability issue in AURORA is reduced to the issues of rapid homogenization of individual data sources and the rapid integration of multiple data sources. Homogenization can be performed in parallel among sources. It is a process that concerns single data sources; it has good potential to be fast. AURORA provides tools to assist in this process. Integration handles multiple but homogeneous (although still autonomous) sources. To achieve scalability of this step, AURORA integration mediator assumes that the integrated view is pre-defined by the application requirement. Let this view be pre-defined as V_W. To integrate objects in each data sources (these data sources now appear as homogenization mediators) into V_W, AURORA requires these objects to be *registered* as *fragments* of objects defined in V_W. Removing a source from the access scope only requires the relevant registrations to be cancelled. This way the integration can be performed in a plug-and-play fashion.

6.3 AURORA Mediators as Distributed Components

AURORA does not restrict the canonical data model to a specific one that is deemed to be most "suitable". Rather, AURORA provides mediators and wrappers in two popular data models, the relational data model and the ODMG object data model. Necessary facilities are provided to allow the two data models to coexist seamlessly. AURORA mediators are classified along two dimensions: the canonical data model and the specialty, homogenization or integration. Figure 7 shows this classification.

Canonical Data Model / Mediator Type	Relational	Object-Oriented
Homogenization	AURORA-RH	AURORA-OH
Integration	AURORA-RI	AURORA-OI

Fig. 7. AURORA Mediator Classification

AURORA designs and develops three types of software components: the wrappers, the homogenization mediators and the integration mediators. These are provided as distributed components that communicate and cooperate via an Object Request Broker (ORB), as shown in Figure 8. AURORA wrappers and mediators support pre-defined interfaces, as shown in Figure 9. These are the only interfaces via which a wrapper/mediator can be accessed by the application or by other AURORA mediators.

A middleware that facilitates integrated access to multiple heterogeneous data sources can be constructed by using a network of mediators that cooperate with one another to provide an integrated data service. With AURORA,

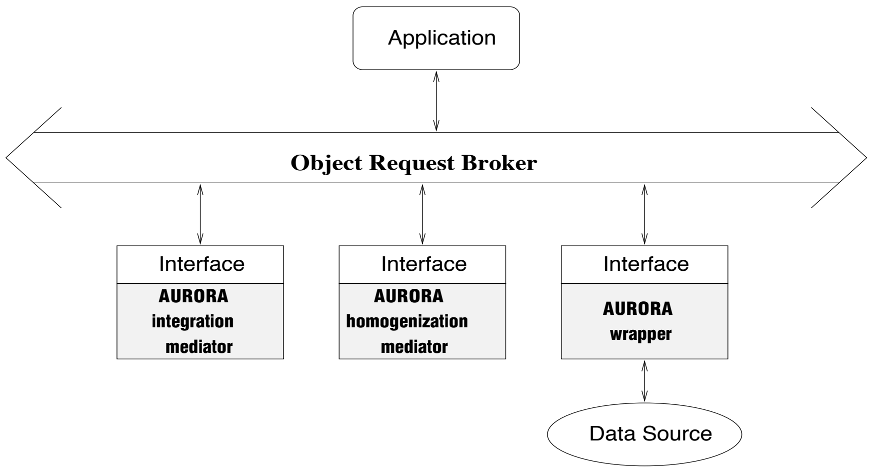

Fig. 8. AURORA Mediators/Wrappers and ORB

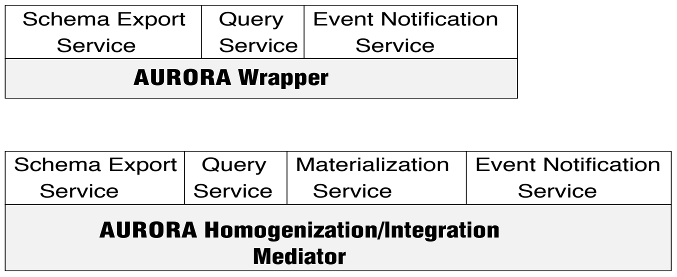

Fig. 9. The AURORA Mediator Interfaces

one can choose between relational and object-oriented components. The use of AURORA mediators in building middleware is best illustrated by Figures 10 and 11. AURORA-O mediators have the built-in capability of accessing AURORA-R components.

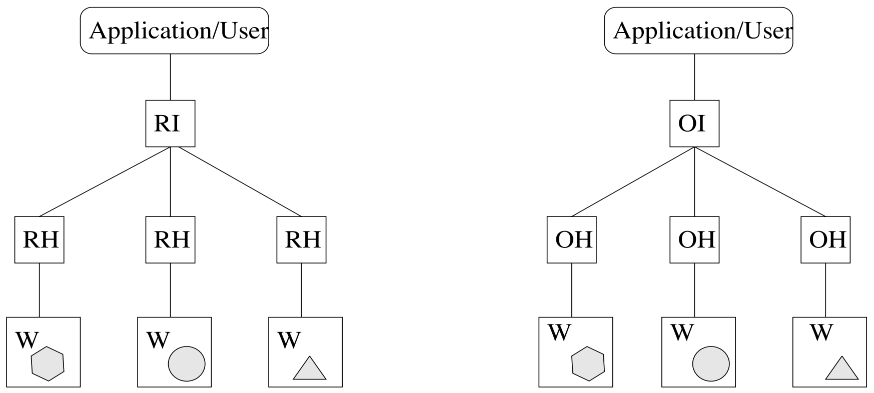

Fig. 10. Application Using AURORA Mediators (Uniform)

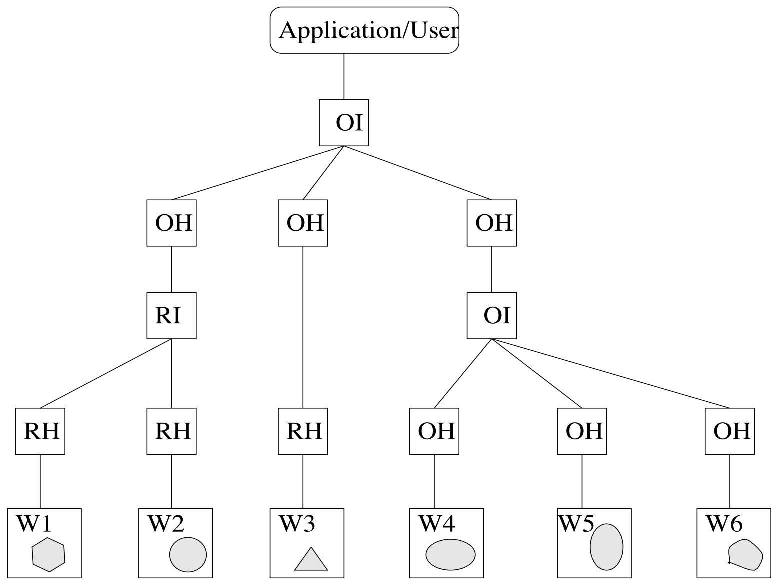

Fig. 11. Application Using AURORA Mediators (Mixed)

6.4 AURORA Mediator Development Workbench

Each AURORA mediator is a *mediator development workbench* consisting of a mediator skeleton and a toolkit named MAT.

Mediator Skeletons. The most important components in a mediator are an integrated view over (multiple) data sources and a query processor that answers queries posed against this view. Building a mediator means building the view, the query processor, and software modules that support other standard services. In AURORA, mediators are constructed from *mediator skeletons* that have all the necessary components of a mediator except for the view.

Mediator Author's Toolkits (MATs). In AURORA, a *mediator author* chooses a mediator skeleton, identifies heterogeneities among the sources, and defines views into the mediator skeleton to resolve the heterogeneities. AURORA MATs assist the mediator authors in performing such tasks. This scenario is shown in Figure 12. A MAT has two main functionalities: it mandates a *mediation methodology* and it provides *Mediation Enabling Operator*s (MEOs).

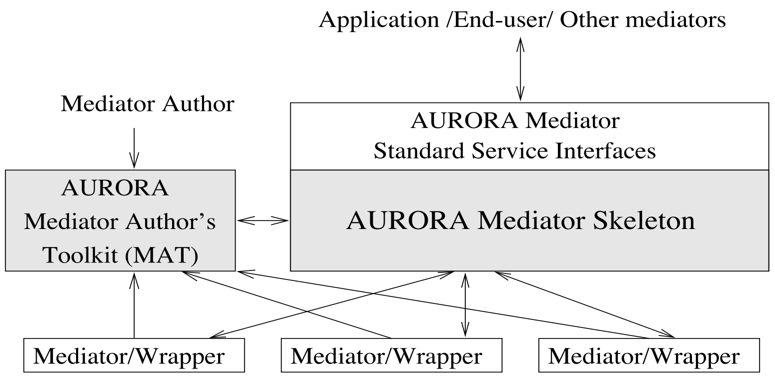
Application /End-user/ Other mediators
Mediator Author
AURORA Mediator
Standard Service Interfaces
AURORA Mediator Author's Toolkit (MAT)
AURORA Mediator Skeleton
Mediator/Wrapper
Mediator/Wrapper
Mediator/Wrapper

Fig. 12. General Form of AURORA Workbenches

6.5 Homogenization Mediators

The architecture of AURORA-RH is shown in Figure 13. **MAT-RH** is a toolkit that assists a mediator author in constructing a homogenizing view. It provides a set of MEOs and mandates a homogenization methodology. Each tool in the toolkit allows specification of *transformations* and *domain mappings*. Transformations are expressions consisting of the AURORA-RH MEOs and the usual relational operators. Domain mappings are arbitrary mappings. This information is captured in the **View Definition Repository** and are used for query processing. **AURORA-RH Primitives** are

MEOs; they extend the relational algebra to form a *Mediation Enabling Algebra* (MEA), MEA-RH. **AURORA-RH Query Processor (AQP)** processes mediator queries posed against the target database. It rewrites such a query into a (optimal) set of queries over the source database, sends these queries for execution and assembles the answer to the mediator query from the returned data.

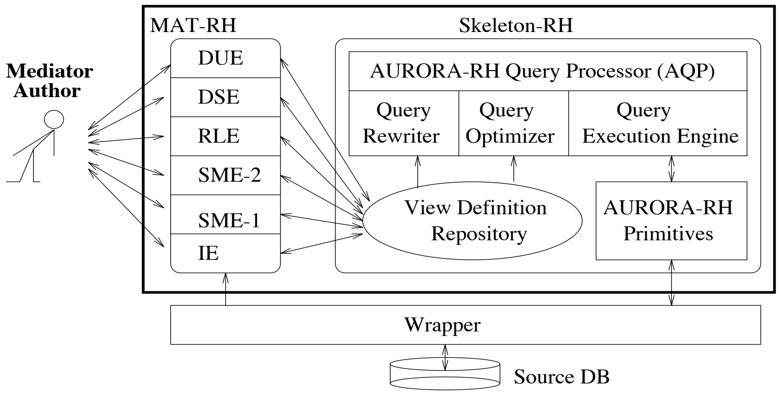

Fig. 13. AURORA-RH Workbench

6.5.1 MAT-RH and AURORA-RH Query Processing. MAT-RH identifies a wide range of domain and schema mismatches and mandates a methodology to resolve them systematically. It also provides constructs for expressing such resolutions. The types of mismatches considered by MAT-RH are the following:

Cross-over schema mismatches. A **type 1** cross-over mismatch happens when a concept is represented as data in M but as relations in B. A **type 2** cross-over mismatch happens when a concept is represented as data in M but as attributes in B.

Domain structural mismatches. This mismatch happens when a domain in M corresponds to a domain with a different data type or several data domains in B.

Domain unit mismatches. This mismatch happens when a domain in M assumes different unit of measurement from the corresponding domain(s) in B.

Domain population mismatches. This mismatch happens when a domain in M assumes different population from the corresponding domain(s) in B.

MAT-RH mandates a 6-step methodology for homogenization. It supports each step by a specialized tool (environment) that accepts certain types of transformations and domain mappings. This is shown in Figure 13 and is further described below:

1. Construct an import schema. The supporting environment is the Import Environment (IE).
2. Resolve type 1 schema mismatches. The supporting environment is the Schema Mismatch Environment 1 (SME-1).
3. Resolve type 2 schema mismatches. The supporting environment is the Schema Mismatch Environment 2 (SME-2).
4. Link relations. The supporting environment is the Relation Linking Environment (RLE). RLE mandates that for each relation in the homogenizing view, a prototype relation must be specified. This relation is close to the target relation modulo domain mismatches.
5. Resolve domain structural mismatches. The supporting environment is the Domain Structural Environment (DSE). DSE allows specification of *domain structural functions*.
6. Resolve domain unit/population mismatches. The supporting environment is the Domain Unit Environment (DUE). DUE allows specification of *domain value functions*.

Detailed description of these environments and their application are given in [YOL97].

Query processing in AURORA-RH is based on MEA-RH. MEA-RH extends the relational algebra with operators specially designed for homogenization. These new operators are *retrieve*, *pad*, *rename*, and *deriveAttr*. Homogenizing views are defined using these operators. When processing a mediator query, AQP uses these view definitions to rewrite the query into an algebraic formula in MEA-RH, transforms this formula into an "optimal" form, and evaluates it with the participation of a relational wrapper. Details of AQP are described in [YOL97].

6.6 AURORA Integration Mediators

AURORA integration mediators, AURORA-RI and AURORA-OI (Figure 7), are responsible for integrating a large number of *homogenized* sources. Since the sources are homogenized, the types of heterogeneities that the integration mediator must handle is limited. First, lets give a closer look at the 2-tier mediation model and the meaning of homogenization.

6.6.1 A Plug-and-Play Integration Mechanism. Lets assume that the application view consists of a single relation R_g. A data source is said to be *homogenized in regard to* R_g if:

1. It is structurally homogenized. It contains a single relation R_s that is a *fragment* of R_g, that is, $ATTR(R_s) \subseteq ATTR(R_g)$.
2. It is semantically homogenized. Each attribute in R_s is the *same* as that in R_g with the same attribute name.

Now the following plug-and-play integration mechanism can be imagined: To "plug" a data source into the the integration mediator, we first homogenize this data source in regard to the application view, that is, construct a *homogenizing view* on top of the data source. We then *register* each relation in this view with the integration mediator as a fragment of a global relation. To "unplug" a data source from the integration mediator, we remove all fragments from this source. When a particular source is down, all fragments from this source are considered to be empty.

6.7 Constructing Virtual Catalogs with AURORA

Consider a number of vendors each having an autonomous catalog database. A virtual catalog effort can be initiated by a third-party broker who seeks to offer value-added services. The broker first design a common catalog structure, its data model and query language. To include a vendor into the virtual catalog, the broker first homogenizes the vendor's catalog using an AURORA homogenization mediator. This process maps the vendor catalog structure and semantics into that in the common catalog. After homogenization, it should be straightforward to "plug" a catalog into an AURORA integration mediator that supports the common catalog. While homogenization is a more complex process, the broker can hire a few people to homogenize individual vendor catalogs in parallel. Integration mediator is where large number of virtual catalogs merge but the integration is a simple mechanism. Overall, construction of the virtual catalog is scalable.

7. Conclusion

We have discussed several interoperability issues in large-scale distributed information delivery systems from both architectural and technical perspective. We present two representative architectural paradigms: *Multidatabase management-based* paradigm and *Mediator-based* information delivery paradigm. The former contributes to the interoperability research in terms of techniques for resolution of semantic heterogeneity and techniques for distributed query optimization in tightly-coupled and somewhat closed distributed environments [She91, SL90]. The later contributes to the interoperability research in terms of techniques for handling both structured data sources and semi-structured or unstructured data sources, and techniques for providing scalable and adaptable data delivery services in loosely-coupled and open distributed environments.

Technically, we identify a number of data delivery characteristics in terms of (1) *delivery protocols*, such as Client Request/Server Response, Server Publish/Client Subscribe, and Server Broadcast, (2) *delivery modes*, such as Pull-only, Push-only, and Hybrid, and (3) *delivery frequencies*, such as periodic, conditional, and ad-hoc. We analyse several data delivery schemes using different combinations of these characteristics (Section 3.4). For example, the publish/subscribe protocol is typically used for server-initiated information dissemination, where data is delivered automatically (push-only) or semi-automatically (hybrid mode). The frequency of server push can be either periodic or conditional. Ad-hoc delivery frequency does not make much sense in practice for push-only mode. We argue that an advanced distributed information system must incorporate different types of information delivery so that the system can be optimized according to various criteria, such as network traffic and heterogeneity and constant evolution of online information sources.

We illustrate the architectural and technical aspects of distributed information delivery systems through a review of several existing research prototypes and demonstrate the different implementation approaches used in practice, and the various solutions to the interoperability issues. We also report our research and experience with AURORA, a mediator-based information integration project, in collaboration with IBM Canada, and its application to electronic commerce domain.

We believe that interoperability in large-scale distributed information delivery systems is one of the most critical functional requirements for many enterprise-wide cooperative applications, such as business workflow automation, computer-aided software engineering (CASE), computer-aided design (CAD) and manufacturing (CAM), and interactive programming environments.

References

[AAFZ95] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: Data management for asymmetric communications environments. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, San Jose, CA, May 1995.

[Aea91] Ahmed and et al. The pagasus heterogeneous multidatabase system. *IEEE Computer*, 24(12), 1991.

[AFZ97] S. Acharya, M. Franklin, and S. Zdonik. Balancing push and pull for data broadcast. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Tucson, Arizona, May 1997.

[BBa97] R. Bayardo, W. Bohrer, and R. Brice and et al. Semantic Integration of Information in Open and Dynamic Environments. In *SIGMOD'97*, 1997.

[BDHS96] P. Buneman, S.B. Davidson, G.G. Hillebrand, and D. Suciu. A Query Language and Optimization Techniques for Unstructured Data. In *SIGMOD 96*, pages 505–516, 1996.

[Bet94] Mark Betz. Interoperable objects: laying the foundation for distributed object computing. *Dr. Dobb's Journal: Software Tools for Professional Programmer*, October 1994.

[Cea94] R. Cattell and et al. *The Object Database Standard: ODMG-93 (Release 1.1)*. Morgan Kaufmann, 1994.

[CHS91] C. Collet, M. Huhns, and W. Shen. Resource Integration Using a Large Knowledge Base in Carnot. *IEEE Computer*, 24(12):55–62, December 1991.

[DDO96] A. Dogac, C. Dengi, and M.T. Ozsu. Building Interoperable Databases on Distributed Object Management Platforms. *Communication of ACM (to appear)*, 1996.

[EDNO96] C. Evrendilek, A. Dogac, S. Nural, and F. Ozcan. Query Optimization in Multidatabase Systems. *Journal of Distributed and Parallel Databases (to appear)*, 1996.

[FZ96] M. Franklin and S. Zdonik. Dissemination-based information systems. *IEEE Bulletin of the Technical Committee on Data Engineering*, 19(3):20–30, September 1996.

[Kea93] W. Kim and et al. On resolving semantic heterogeneity in multidatabase systems. *Distributed and Parallel Databases*, 1(3), 1993.

[LP97] Ling Liu and Calton Pu. An adaptive object-oriented approach to integration and access of heterogeneous information sources. *DISTRIBUTED AND PARALLEL DATABASES: An International Journal*, 5(2), 1997.

[LPBZ96] Ling Liu, Calton Pu, R. Barga, and T. Zhou. Differential evaluation of continual queries. In *IEEE Proceedings of the 16th International Conference on Distributed Computing Systems*, Hong Kong, May 27-30 1996.

[LPL96] L. Liu, C. Pu, and Y. Lee. An adaptive approach to query mediation across heterogeneous databases. In *Proceedings of the International Conference on Coopertive Information Systems*, Brussels, June 19-21 1996.

[Man92] F. Manola et al. Distributed Object Management. *International Journal of Intelligent and Cooperative Information Systems*, 1(1), March 1992.

[Mea92] F. Manola and et al. Distributed object management. *International Journal of Intelligent and Cooperative Information Systems*, 1(1), March 1992.

[ODV93] T. Ozsu, U. Dayal, and P. Valduriez. *Distributed Object Management.* Morgan Kaufmann, 1993.

[ONK+96] F. Ozcan, S. Nural, P. Koksal, C. Evrendilek, and A. Dogac. Dynamic Query Optimization on a Distributed Object Management Platform. In *Proceedings of Fifth International Conference on Information and Knowledge Management (CIKM)*, Maryland, USA, 1996.

[PAGM96] Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object Fusion in Mediator Systems. In *VLDB 96*, Bombay, India, September 1996.

[PGGMU95] Y. Papakonstantinou, A. Gupta, H. Garcia-Molina, and J. Ullman. A Query Translation Scheme for Rapid Implementation of Wrappers. In *International Conference on Deductive and Object-Oriented Databases*, 1995.

[PGMU96] Y. Papakonstantinou, H. Garcia-Molina, and J. Ullman. Medmaker: A Mediation System Based on Declarative Specifications. In *ICDE 96*, pages 132–141, New Orleans, February 1996.

[PGMW95] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object Exchange Across Heterogeneous Information Sources. In *ICDE 95*, pages 251–260, Taipei, Taiwan, March 1995.

[Ram91] Sudha Ram. *Special Issue on Heterogeneous Distributed Database Systems.* IEEE Computer Magazine, Vol. 24, No. 12, December 1991.

[Rus89] M. Rusinkiewciz et al. OMNIBASE: Design and implementation of a multidatabase system. In *Proceedings of the 1st Annual Symposium in Parallel and Distributed Processing*, pages 162–169, Dallas, May 1989.

[She91] Amit Sheth. *Special Issue in Multidatabase Systems.* ACM SIGMOD Record, Vol.20, No. 4, December 1991.

[SL90] A. Sheth and J.A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, Vol. 22, No.3 1990. 183-236.

[TRV96] A. Tomasic, L. Raschid, and P. Valduriez. Scaling Heterogeneous Databases and the Design of Disco. In *Proceedings of the International Conference on Distributed Computer Systems*, 1996.

[TRV97] A. Tomasic, L. Raschid, and P. Valduriez. A Data Model and Query Processing Techniques for Scaling Access to Distributed Heterogeneous Databases in Disco. *IEEE Transactions on Computers, special issue on Distributed Computing Systems*, 1997.

[WCH$^+$93] D. Woelk, P. Cannata, M. Huhns, W. Shen, and C. Tomlinson. Using Carnot for Enterprise Information Integration. In *Second International Conference on Parallel and Distributed Information Systems*, pages 133–136, January 1993.

[Wie92] Gio Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, pages 38–49, March 1992.

[Wie95] Gio Wiederhold. I3 glossary. *Draft 7*, March 16 1995.

[YOL97] L L. Yan, T. Ozsu, and L. Liu. Accessing Heterogeneous Data Through Homogenization and Integration Mediators. In *Second IFCIS Conference on Cooperative Information Systems (CoopIS-97)*, Charleston, South Carolina, USA, June 1997.