# An Adaptive Approach to Query Mediation across Heterogeneous Information Sources

Ling Liu **, Calton Pu †, Yooshin Lee **

** Department of Computing Science
University of Alberta
email: lingliu@cs.ualberta.ca

† Dept. of Computer Science and Engineering
Oregon Graduate Institute
email: calton@cse.ogi.edu

**Abstract**

*We propose a query mediation framework to support customizable information gathering across heterogeneous and autonomous information sources. Instead of an integrated (and static) global schema, we propose an adaptive approach to interoperability which allows information consumers to represent their queries based on the customized personal view rather than a system-defined integrated view. The query mediation framework consists of five steps: query routing, query decomposition, parallel access plan generation, subquery translation and execution, and query result assembly. Concrete examples illustrate the challenges arising from heterogeneity in these five steps and how the framework scales up as the number of information sources grow and evolve.*

# 1 Introduction

Large-scale cooperative information systems often require information consumers and information producers to cooperate across departmental, organizational, and national boundaries. The goal of cooperation is to increase productivity, and reduce the production and business transaction costs. Information sources provided by autonomous information producers are often heterogeneous due to discrepancies in data models, DBMSs, and legacy applications. Information consumers often have diverse customization needs due to the varying business objectives and business rules. Our goal is to build a cooperative information system capable of interconnecting information consumers and information producers, providing transparent and customizable information access across multiple heterogeneous and autonomous information sources (including databases, knowledge bases, flat files, or programs). We have captured the requirements of these systems as the USECA properties [14]: Uniform access to heterogeneous information sources, Scalability to the growing number of information sources, Evolution and Composability of software and information sources, and Autonomy of participants, both information consumers and information producers.

We have proposed the Distributed Interoperable Object Model (DIOM) [14] in the context of the Diorama project as a concrete information mediation architecture to support the cooperation in a network of specialized mediators. The distinctive features of the DIOM architecture include: the support for USECA properties in the development of distributed object query services [14], and the flexible cooperation of a network of mediators for transparent query processing. In contrast, much of previous work (summarized in Section 2) addresses only one or a subset of the USECA properties, and the combination of all USECA properties remains a challenge.

The main contribution of this paper is the development of a DIOM query mediation framework to support adaptive and customizable information gathering, while preserving the USECA properties. Advantages of our query mediation framework include:

- the use of interface abstractions in DIOM IDL to promote the reusability and the robustness of consumer's domain query model against the evolution of the system;
- the flexibility in expressing queries using IQL such that information consumers wishing to access remote information may interact with the appropriate mediator rather than directly with the source(s);
- the systematic development of the query mediation framework and the procedure of each query processing step from query routing, query decomposition, parallel access planning, query translation to query result assembly.

The rest of the paper proceeds as follows. First, we summarize the related work for background. Second, we present the DIOM approach to heterogeneity problems, including a running example used throughout the paper.

Third, we describe how a mediator handles an information consumer's query request and decompose the consumer's query into a set of source queries. Fourth, we discuss how the information requests are efficiently processed. Fifth, we introduce the metadata attachment techniques to handle the semantic and representational heterogeneity issues during the query result assembly process. Finally, we conclude with a comparison to the related work, a summary of the paper and the work that remains to be done.

# 2 Related Work

Several approaches have been proposed for information gathering from multiple heterogeneous information sources. A classic approach [19, 20] 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 [1] or SQL/M in the UniSQL/M system [11]. 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 [21] 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 from the integrated schema.

The distributed object management approach [16, 17] 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 [5], which extends the OMG object model to the database interoperability, are the important milestones for the distributed object management.

Another approach, called the intelligent information integration ($I^3$) mediation [23] can be seen as a generic system architecture for information integration, with several projects funded by ARPA in the $I^3$ program [24]. For example, TSIMMIS [8] (The Stanford-IBM Manager of Multiple Information Sources) is one of the best known systems, which implements the mediators-based information integration architecture through a simple object exchange model. Another example is the Context Exchange project at MIT [9, 22]. It uses context knowledge in a context mediator to explicitly define the meaning of information provided by a source and that required by a receiver. Similarly, the Intelligent Agent Integration at University of Maryland at Baltimore County uses the Knowledge Query and Manipulation Language (KQML) to integrate heterogeneous databases. The SIMS project [3, 2] uses the LOOM knowledge representation system as the data model and query language to implement the agent-based integration. Other examples include University of Maryland [6, 7] and SRI [18], which in different ways integrate heterogeneous data and knowledge bases using multiple F-logic object schema, via the KIF knowledge interchange logic.

Among these projects, only a few have dealt with different aspects of the query processing and query optimization issues. For example, TSIMMIS query processing is based on pre-defined query pattern matching. The query reformulation and simplification are performed by using logic-based rewriting rules against the pre-defined query patterns. The query processing in the information mediation project at University of Maryland and SRI uses F-logic to define a set of heterogeneous object equivalence rules to facilitate the mapping between the multidatabase and the local schemas. The query processing in SIMS is done through query reformulation and query optimization. The LOOM reasoning module is used for query optimization in SIMS. However, their results on query reformulation and simplification have not addressed all the requirements of the USECA properties in a systematic manner, which is the goal of our approach.

# 3 The DIOM approach

The DIOM approach to the heterogeneity problems promotes the full respect for the autonomy of information producers' sources and of information consumers' query models. No integration is imposed unless it is required explicitly by the information consumers in their user profiles or their query specifications. The main techniques used to achieve this objective are (1)the flexible cooperation of a network of information mediators, (2)the adaptive techniques used in the design of DIOM-IDL and in metadata catalog construction and maintenance, (3)the delaying of resolution of heterogeneity problems to the query result assembly phase, and (4)the utilization of consumer's domain usage model and query profile to distinguish the desirable heterogeneous information from the undesirable ones.

## 3.1 A Running Example

In this section, we briefly examine the potential heterogeneity problems in a distributed multiple information source environment. The running example we use in this paper is described in Figure 1.

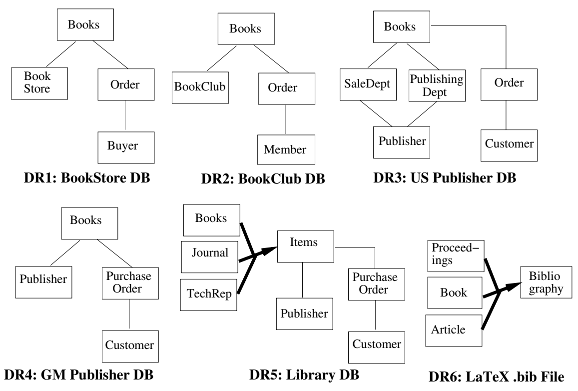


Figure 1: Examples of information sources

This example can be seen as an enterprise system that deals with book inquiry, sale, and purchase business in a distributed and heterogeneous environment such as the Internet. For concreteness, we assume that the relevant information sources that are currently available include BookStore database (Oracle), book club database (Sybase), publisher database (Informix), library database (ObjectStore), and bibliography file (in LaTeX `.bib` format). We refer to these information sources as component information sources of the given enterprise system. The relations or classes are described as boxes and the links indicate the primary key and foreign key relationships or object reference relationships. The arrow links are depicted as superclass and subclass relationships. Fragments of the export schemas of these information sources are given in [13]. Obviously, some of these information sources have similar data contents, but their representations are different. This is generally true for most of the information sources that are created and managed by separate organizations (information producers) with different business objectives or using different types of data repository managers. Moreover, different heterogeneous factors at information source models may have different effects on multidatabase query processing strategies:

- **Heterogeneity in data model types**.
  Examples are relational models (Oracle, Sybase, Informix), network/IMS models, object-oriented models (ObjectStore, GemStone), simple record-based data model (e.g., file system). The heterogeneity in data models may further incur the heterogeneity in query languages, query formulation, query optimization methods, and query result representations.
- **Heterogeneity in domain requirements**.
  This type of heterogeneity comes from the variation in business rules and business objectives. For example, export schema in DR2 includes a business rule/constraint that only the member can order books from his/her book club; while this constraint may not be relevant in other information sources such as DR1, DR3, and DR4.
- **Heterogeneity in schema designs**.
  This type of heterogeneity includes three typical categories:
  - *Structural heterogeneity* is mainly caused by the varying number of object types or relations used by different database schemas to model the same domain of applications. For instance, the publisher data source DR3 utilizes six object types to organize all the data elements while the publisher data source DR4 uses only four relations. Also, the number of attributes used to model the same entity in the real-world may differ from schema to schema.
  - *Naming heterogeneity* may occur at both attribute/relationship level (e.g., `book#` in DR1 and `bookID` in DR2) and object type level (e.g., the supplier information is modeled using `BookStore` in DR1, `BookClub` in DR2, and `Publisher` in DR3.
  - *Semantic heterogeneity* includes different underlying semantics for the same named attributes or relationships. For instance, the `price` attribute Πin different databases may be represented in several different scales Π(US$, CDN$, DM, JapaneseYan, etc.).

All these heterogeneity factors make it difficult to effectively retrieve data from various component information sources. For example, the same query *find the cheapest price and the supplier name of the book titled "The Mathematical Universe"* may need to be written differently for each relevant information source because of the varying logical data structures, the heterogeneity in attribute naming, and the different semantic meanings in each information source schema design. The situation becomes even worse when the number of information sources is large (tens and hundreds versus three or four), growing fast, and each may evolve dynamically.

## 3.2 Overview of DIOM

In DIOM environments, the information sources may differ by their organization, such as traditional databases, knowledge bases, and flat files; by their content, such as HTML hypertext, relational tables, and objects of complex class structures; and by the many browsers and graphical user interfaces (GUIs) as well as query languages used for information access. Figure 2 shows an example network of information mediators collaborating through the DIOM mediator network architecture. This network includes simple wrapper-based mediators such as a wrapper to BookStore data repository which only provides information about and

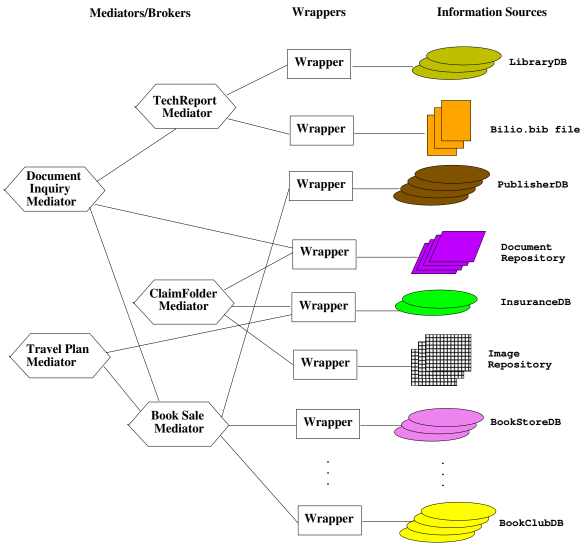


Figure 2: The DIOM system architecture

access to the BookStore repository. This wrapper-based mediator is in turn used to construct a BookSale mediator. The BookSale mediator is again used to build both a document inquiry mediator and a travel plan mediator. This recursive construction and organization of information access have the following important features. First, the individual mediators can be independently built and maintained. Each specialized mediator represents a customized personal view of a particular information consumer over the large amount of information from the growing number of information sources. Second, these mediators can be generated automatically or semi-automatically by using the DIOM IDL/IQL interface specification language and the associated incremental compilation techniques. These features also make it possible to scale the DIOM architecture not only to the large and growing number of information sources but also to the varying information customization needs from diverse information consumers.

To build a network of specialized information mediators, we need an architecture for a single mediator (or so called meta mediator) that can be instantiated to build multiple specialized mediators. The ultimate goal of developing such a mediator architecture is to provide a uniform interface description language and the associated query processing services that can be utilized and customized by the number of application-specific mediators to facilitate the access to heterogeneous and autonomous information sources. Each DIOM mediator contains a detailed query model with its domain of expertise and models of the information sources that are available to it. The DIOM query mediation architecture can be generalized into five-phase process:

1. Information consumers issue a query using the DIOM interface query language IQL or based on the predefined IQL query patterns.
2. Accept a query, determine the appropriate set of information sources to answer the query.
3. Reformulate the original IQL query into a set of appropriate IQL subqueries or procedures, each targeted at one information source.
4. Translate each IQL subquery into a query expression that is executable at the source level.
5. Obtain results form the information sources, perform appropriate translation, filtering, and merging of the information, and return the assembled result to the information consumers (users or applications).

The basic idea is to use a uniform DIOM interface description language (DIOM-IDL) to describe the information producers' source models and use the DIOM interface query language (DIOM-IQL) to represent information consumers' query models. The extensibility and the abstractions provided in DIOM IDL/IQL make it possible to scale the query routing module, the query decomposition mechanisms, the query translation methods, and the result assembly operations to the growing number of information sources and the evolution of information sources. Before going into details of the query mediation steps, we first briefly describe the metadata model and the features of DIOM IDL and IQL.

## 3.3 The DIOM Metadata Model

The DIOM metadata model is developed in the Diorama project for construction of a USECA view of data from disparate information sources [14]. It consists of an interface description language (IDL) and an interface query language (IQL). Rather than inventing yet another object-oriented data model, DIOM adopts the ODMG-IDL and ODMG-OQL object database standard [5, 10] as the base model with two major extensions:

1. DIOM IDL adds a number of interface abstractions to allow new interfaces to be constructed in terms of existing ones, and to provide better support for the USECA properties. These incremental interface construction facilities include: *import mechanism, interface aggregation, interface generalization*, and *interface specialization*.
2. DIOM introduces base interface and compound interface concepts to separate the interface descriptions for information producers' source models and the interface descriptions for information consumers' domain usage models (personal views). Such a separation serves dual purposes: (a)It enables the system to automate the specification of base interfaces. (b)It allows information consumers to build dynamic and personal views, rather than monolithic and integrated

snapshots, over the growing number of information sources.

The term *metadata* refers to both the data types and the data semantics. The DIOM metadata model consist of two main components: the description of information consumer's domain usage model (consumer's domain interface schema) and the description of the portions of the information source models (the export schemas) that are relevant to the information consumer's domain model. We use the term "*mediated metadata*" to distinguish the description of the metadata, defined by information consumers, at multidatabase level from the metadata defined at information source level. A detailed discussion on DIOM interface description language (IDL) may refer to [14].

In the DIOM query mediation framework, an information consumer may build his/her special-purpose information mediator using either DIOM-IDL or DIOM-IQL and the associated query services. A typical application-specific information mediator contains two levels of interface descriptions:

- an information consumer's *domain query model* (personal view), describing the usage and expertise of the domain, and
- a number of information producers' *source data models*, each providing relevant information contents that are of interest to the information consumer.

Every mediator defined in terms of DIOM-IDL is specialized to a specific application domain and provides access to the available information sources that are relevant to that domain.

Since this paper's focus is on query processing in a large heterogeneous environment, we only outline the relevant metadata management services here. Details may refer to our technical report [13]. For this paper, we assume that the distributed metadata management system in DIOM is capable of representing and maintaining a mediator's knowledge. The basic services needed include (i)an information source catalog to store the information producers' source registration profiles, (ii)an interface repository to store information consumer's domain usage model, query profile, and the mediator's domain knowledge, and (iii)the implementation repositories, each is managed by the source-specific wrapper to store the information producer's data source model, the semantic correspondence between the mediator's domain model and the source model, and the source-specific knowledge.

## 3.4 Informal Overview of DIOM-IDL

An information consumer's domain model is defined as an IDL interface schema. Each interface schema consists of one or more IDL compound interface descriptions, one connected with another in terms of the DIOM interface construction mechanisms such as generalization, specialization, aggregation, and import. When a consumer is only interested in some portions of an information source, a compound interface to this information source model may be explicitly defined in terms of the DIOM `import` mechanism to specify the objects of interest. For example, in a *book inquiry* application, the objects of type `Journal` and `TechRep` in the information source `DR1:LibraryDB` may not be relevant. Thus a compound interface to the source `DR1:LibraryDB` can be defined by explicitly specifying the objects of interest using the `IMPORT` mechanism. The following example illustrates how a compound interface is defined in DIOM-IDL.

```
CREATE INTERFACE DiomDR5
{  FROM LibraryDB
   IMPORT TYPE Books, Library, BorrowRequest, Reader;
}

CREATE INTERFACE Book
( extent books
  key    Book->ID )
{   GENERALIZATION OF
      select interface-name from Interface-Repository
      where  description = '*Book' OR description = '*Books';
    ATTRIBUTES
      String ID,
      String title,
      Supplier supplier,
      String medium,
      Double price,
      Order orderNo;    }

CREATE INTERFACE Supplier
( extent suppliers
  key    Supplier->ID )
{  GENERALIZATION OF
     select interface-name from Interface-Repository
     where  description CONTAINS
            ['BookStore','BookClub','Publisher','Supplier'];
     ATTRIBUTES
       String ID,
       String name,
       String addr;      }

CREATE INTERFACE Customer
( extent customers )
{  GENERALIZATION OF
     select interface-name from Interface-Repository
     where description CONTAINS
      ['Buyer','Client','Customer','Reader','Member'];
        ...
}
```

The consumer's interface schema defined in DIOM-IDL is different from the conventional global schema approach to multidatabase integration in at least three ways:

1. Unlike in the tightly-coupling approach to schema integration [21] where equivalent attributes are merged into one attribute, the DIOM mediated attributes are only given in a logical form. That is, a mediated attribute is solely a semantic correspondence to, rather than an absolute integration/generalization of, the equivalent attributes in component information

sources. Difference in domain and operations do not matter since no attributes are merged in the DIOM interoperable system. Thus, the implementation details about coding and data structure are not necessary.

2. Each DIOM interface schema only represents an information consumer's personal view of the relevant information sources. We call it the domain query model. There is no global schema that is created by the system itself for all the multidatabase users. For each consumer's domain model, the DIOM metadata naming convention is used to denote the consumer-defined interface types, attributes and relationships. For example, the attribute naming convention used to denote the mediated attributes is to associate each mediated attribute with its interface type name through arrow notation (e.g., `Book->Price` and `Supplier->Name`).

3. The interface abstraction mechanisms supported by DIOM-IDL ensures not only the seamless extensions to the consumers' interface schemas, but also the scalability of IDL specification. Each consumer-defined IDL interface schema is maintained as a dynamic view, rather than a static "snapshot", of the global information. Upon arrival of a new information source, an incremental recompilation of the relevant consumer IDL interface descriptions will be performed transparently to ensure a seamless incorporation of the new information sources into the global query planning and dynamic source selection process. From information consumers' point of view, no manual interface schema modification is required upon the arrival of new information sources.

In short, the DIOM metadata model establishes a foundation for the extensible (and scalable) query processing framework to be discussed in Section 3.5.

### 3.5 Informal Overview of DIOM-IQL

The DIOM interface query language (IQL) is designed as an object-oriented extension of SQL. The basic construct in DIOM-IQL is a SQL-like `SELECT-FROM-WHERE` expression. The syntax is:

```
[TARGET <Source-expressions>]
SELECT <Fetch-expressions>
FROM   <Fetch-target-list>
[WHERE  <Fetch-conditions>]
[GROUP BY <Fetch-expression>]
[ORDER BY <Fetch-expression> ]
```

The detailed syntax description of an IQL expression is provided in [13]. The IQL design follows a number of assumptions: First, IQL by itself is not computationally complete. However, queries can invoke methods. Second, IQL is truly declarative. It does not require the users to specify the navigational access paths or the join conditions. Third, IQL also deals with the representational heterogeneity problem. For example, different information sources may have different structural and naming conventions for the same real world entity. These information producers' representation may not fit to the information consumer's preference. Using IQL, the information consumer may specify a query in terms of his/her own IDL specification or express a query without having a pre-defined IDL schema. The system is responsible for mapping a consumer's query expression into a set of information producer's query expressions. Furthermore, IQL allows the use of object type names of information sources in the FROM clause by prefixing the source name with the dot notation. It is also possible to use the information source names directly in the FROM clause of a query. The result of an IQL query is by itself an object of the special interface type `QueryResult`. The main objective of these IQL extensions is to increase the accessibility of information consumers and to improve the quality of global information sharing and exchange.

For example, the query $Q_1$: "find titles, prices and supplier names of all the books that are related to 'multimedia' applications and are published in CD-ROM" can be expressed in IQL as follows:

```
SELECT Book->title, Book->price, Supplier->name
FROM   Book, Supplier
WHERE  Book->medium = 'CD-ROM'
AND    Book->title CONTAINS 'multimedia';
```

Figure 3: An example query

First, in representing this query, the query writer does not need to concern about the different naming conventions and different structural design from `BookClub` to `BookStore` to `Publisher`'s sale department. Instead, she can write a query using the terminology that is preferable in her own application domain. Second, the query writer need not to specify any "join" conditions in her query expression. The IQL preprocessor will automatically insert the necessary connection paths to relate different object types involved in the query with each other, since such connection semantics should be easily derived from the metadata maintained in the DIOM interface repository and implementation repository. Note that, unlike the conventional SQLs, in IQL the `FROM` clause may allow zero or more target objects, and the "joins" between target objects may not be specified at the top level `WHERE` clause of the query .

Another interesting feature of the DIOM query model is the automatic creation of result type for each consumer query. It means that, for every consumer query expressed in DIOM-IQL, a compound interface will be generated as the result type of the query, and maintained in the DIOM interface repository. This functionality is particularly important for preserving the closure of the DIOM object model and for assembly of multidatabase query results in terms of consumer's preferred representation. Consider the query given in Figure 3. the following compound interface description will be generated as the result type of this query:

```
CREATE INTERFACE QueryResult<Q1>
( extent result_Q1_objects )
{
    AGGREGATION OF Book, Supplier;
    ATTRIBUTES
        Book->title,
        Book->price,
        Book->medium,
        Supplier->name;
}
```

If this query is installed as a continual query [15], its result type is persistent. We use the basic techniques described in [12] for automatic generation of query result type for an IQL query expression.

# 4 Query Mediation Process

## 4.1 The Framework

The main task of DIOM distributed query mediation manager is to coordinate the communication and the distribution of the processing of information consumer's query requests among the root mediator and its associated mediators or wrappers. A mediator here can be seen as a software component at middle ware layer with services that facilitate integration and access of heterogeneous information. A wrapper is a tool that, on the one hand, translate the mediator query expressions into the known repository query statements or repository commands, and on the other hand, convert the objects returned from the local repository to the objects that are represented and understandable by the mediator interface.

The general procedure for DIOM query mediation process consists of the following five steps:

1. **Query routing**
   The main task of query routing is to select relevant information sources from available ones for answering the query. This is done by mapping the domain model terminology to the source model terminology, by eliminating null queries, which return empty results, and by transforming ambiguous queries into semantic-clean queries. In general, the choice is made such that the number of different information sources used to answer a query is minimized.
2. **Query decomposition**
   This is done by decomposing a query expressed in terms of the domain model into a collections of queries, each expressed in terms of a single source model. The useful dependencies between subqueries are captured during query decomposition phase and will be used in parallel access planning and query result assembly.
3. **Parallel access plan generation**
   The goal of generating a parallel access plan for the groups of subqueries is to obtain the maximum parallelism and the best quality of cooperation in searching for query answers from multiple information sources. A parallel query plan is constructed for the modified query resulted from query decomposition. The plan is composed of single-source queries (each posed against exactly one local export schema), move operations that ship results of the single-source queries between sites, and the postprocessing operations that assemble the results of the single-source queries in terms of the information consumer's query request expressions.
4. **Subquery translation and execution**
   The translation process basically converts each subquery expressed in terms of an IDL source model into the corresponding information producer's query language expression, and adds the necessary join conditions required by the information source system.
5. **Query result assembly**
   The result assembly process involves resolving the semantic variations among the subquery results. The semantic attachment operators and the consumers' query profiles are the main techniques that we use for resolving semantics heterogeneity implied in the query results.

Note that the performance of query mediation process will be affected when global query planning and subquery translation are synchronized with updates to the DIOM metadata catalog. Both locking and optimistic approaches can be used for synchronization between query decomposition, query translation and the DIOM metadata updates. However, with the optimistic approach, all the query planning steps that are affected by the update have to be redone when inconsistency is detected.

## 4.2 Query Routing

The first step in answering an IQL query is to select the appropriate information sources. The ultimate goal for query routing is to minimize the number of information sources to which the query is broadcasted.

To identify the set of candidate information sources that are needed to answer a query, the mediator applies a set of catalog mapping operators to map each compound interface type defined in the consumer's domain model into the base interface types at information source level. We summarize these catalog mapping operators in Figure 4. These mapping functions utilize the metadata information maintained in the information source catalog and the interface repository (for an example see [13]).

In the design of the query routing Algorithm, the application of operators **A-map**, **G-map**, or **S-map** is recursive. The recursion ends when all the compound interface types defined in terms of interface generalization/specialization and interface aggregation are replaced by the corresponding base interface types. Then the operator **D-map** is used to find the correspond information source name. Due to the space limitation, the algorithm for selecting candidate information sources is omitted here. We will illustrate the flavor of our algorithm by examples. Readers may refer to [13] for detail.

Consider the query $Q_1$ given in Figure 3. The interface types involved in the query $Q$ are `Book` and `Supplier`. The query routing algorithm discover the relevant sources by applying the catalog mapping operators to these interfaces. For example, by applying the operators **G-map** and **D-map** to the compound interface type `Book`, the set of candidate information sources selected contains `DR1, DR2, DR3, DR4, DR5, DR6`, whereas by applying **G-map** and **D-map** to the compound interface type `Supplier`, the information sources selected are `DR1, DR2, DR3, DR4`. Thus, the minimal set of sources we need to answer this query would be {`DR1, DR2, DR3, DR4`}.

When more detailed information about individual sources are available at the mediator level, we may further reduce the number of candidate information sources needed to answer the query. We refer to this step as the *candidate information source filtering*. For example, by applying the attribute-to-source mapping (A-to-S-map) operator, which maps each mediated attribute in the *fetch condition* of the original query $Q$ to the corresponding attributes defined at information source level to identify the relevant information sources, only two information sources `DR2` and `DR3` are actually relevant to this query, because based on the attribute correspondence maintained in the metadata catalog, only the source `DR2` and `DR3` have attribute that corresponds to the mediated attribute `Book->medium`. Thus, the set of the candidate information sources is now reduced to {`DR2, DR3`}. The query $Q_1$ is reformulated as follows:

```
TARGET DR2, DR3
SELECT Book->title, Book->price, Supplier->name
FROM   Book, Supplier
WHERE  Book->medium = 'CD-ROM'
AND    Book->title CONTAINS 'multimedia'
AND    Book->supplierID = Supplier->supplierID;
```

For the presentation convenience, a keyword `TARGET` is used to lead to the selected information source names. It is only used as an internal representation. Users will have no knowledge of it. Also, the algorithm for candidate information source filtering is omitted here due to the space limitation. Readers may refer to [13] for detail. An information consumer may also specify the target information sources of interest explicitly in an IQL query using the `TARGET` clause. When the `TARGET` clause is used, the query routing module will bypass the candidate information source selection algorithm and directly run the candidate information source filtering algorithm to determine the final selection of relevant information sources for answering the query.

Besides, in the DIOM query mediation framework, only the *fetch-condition* in an IQL query will be used to eliminate the irrelevant candidate information sources. We allow the *fetch-expressions* of IQL queries to be partially applicable to any given individual source to support incomplete query answering (see the query example in Section 4.5).

## 4.3 Query Decomposition

The goal of query decomposition is to break down a multi-information source query into a collection of subqueries, each targeted at a single source. The query decomposition module takes as input the IDL query expression modified (reformulated) by the dynamic query source selection module, and perform the query decomposition in two phases:

1. **target split**. This is done by decomposing the query into a collection of subqueries, each targeting at a single source. These subqueries can be either independent of or dependent upon each other. The procedure of target split consists of the following five steps:

   (a) For each compound interface involved in the query,

      - if it is defined in terms of aggregation abstraction, then an aggregation-based join over the component interfaces is used to replace this compound interface;
      - if the compound interface is defined in terms of generalization abstraction, then a distributed-union (D-union) over the specialized interfaces is used to replace this compound interface.

   (b) Move the information sources specified in the target clause downward and closer to the leaves of the query tree. Metadata in the interface repository are used to lead the moving of each information source to an appropriate leaf node of the query tree.

   (c) Group the leaf branches of the query tree by target source.

   (d) Use conventional query processing tactics, such as moving selection down to the leaf nodes, perform selection and projection at individual sources, and attach appropriate subset of the projection list to each subquery before executing any inter-source joins and inter-source unions.

   The process of target split results in a set of subqueries and a modified query expression that is semantically equivalent to the original query, and that

| Operator | Operand | ReturnResult | Purpose |
|---|---|---|---|
| **Direct-map** (D-map) | base interface | source name | Return the information source that corresponds to the base interface type. |
| **Aggregation-map** (A-map) | compound interface | a set of component interace types | Replace a compound interface defined in terms of aggregation with an appropriate set of component interface types |
| **Generalization-map** (G-map) | compound interface | a set of specialized interface types | Replace a compound interface defined in terms of generalization with an appropriate set of specialized interface types |
| **Specialization-map** (S-map) | compound interface | a generalized interface type | Replace a compound interface defined in terms of specialization with a generalized interface type |

Figure 4: Operators for candidate information source selection

presents the operations of how the subqueries should be combined to produce the answer for the original multi-information source query.

2. **Useful subquery dependency recording**. This is done by recording all the meaningful subquery dependencies with respect to the efficiency of the global query processing.

   For each consumers' query, if the number of subqueries resulted from query decomposition phase is $k$, then the number of possible synchronization alternatives is $C_k^1 + C_k^2 + \ldots + C_k^k$ ($k \geq 1$). Not surprisingly, many of the possible synchronization alternatives will never be selected as the parallel access plan for the given query. For example, assume that, for any two subqueries, the independent evaluation of a subquery subQ$_1$ is more expensive than the independent evaluation of subquery subQ$_2$. Thus, the synchronization scheme that executing subQ$_1$ first and then ship the result of subQ$_1$ to another source to join with the subquery subQ$_2$ is, relatively speaking, an undesirable access plan because of the poor performance it presents. The following simple heuristics may be used to rule out those undesirable synchronization alternatives.

   (a) The selectivity level of a *Fetch* condition with one of the following operators "$<=, <, >, >=$" is higher than the selectivity level of a *Fetch* condition with the operator "$=$", but lower than the selectivity level of a *Fetch* condition with "$<>$" operator.

   (b) Subqueries of the lower level of selectivity are executed first. If there are more than one subqueries of lowest level of selectivity, then they should be executed in parallel.

   (c) Subqueries of the highest level of selectivity are executed last.

   The idea of applying these simple heuristics is to use the obvious difference in the selectivity levels of different query predicates (*fetch conditions*) to estimate the cost difference among various subqueries. When the selectivity level of a subquery is higher, the size of the subquery result tends to be relatively small. Thus it is always a recommended tactic to execute those subqueries of high selectivity level earlier.

It is well known that the quality of a query processor relies on the efficiency of its query processing strategies and the performance of its query execution plans. Furthermore, the performance of a distributed query processing plan is not only determined by the response time of the local subqueries but also affected by the synchronization scheme chosen for synchronizing the execution of subqueries. A good synchronization scheme may utilize the results of some subqueries to reduce the processing cost of the other subqueries whenever it is beneficial.

Consider the example query reformulated at the end of Section 4.2. Let X denote `Book->title, Book->price, Supplier->name`. Let F denote `Book->title CONTAINS 'multimedia' AND Book->medium = 'CD-ROM' AND Book->supplierID = Supplier->supplierID`. The query decomposition process first generates the query tree graph as shown in Figure 5(a). Then the **target split** module is invoked. The procedure of target split starts by repeatedly replacing the compound interface based on generalization by the distributed union over the corresponding base interfaces as shown in Figure 5(b). To replace the generalization-based interface `Book`, only `DR2.Books` and `DR3.Books` are used since by the query routing step, `DR2` and `DR3` are the only sources that are relevant to this query. The next step is to move the target downward close to the leaf nodes, and group the leaf nodes by target source. Now the query is modified into the query tree shown in Figure 5(c). By applying the conventional distributed query processing tactics, such as moving selection down to the leaf level, performing selection and projection earlier, performing local joins before moving the data among sites for inter-source joins, the query is reformulated as shown in

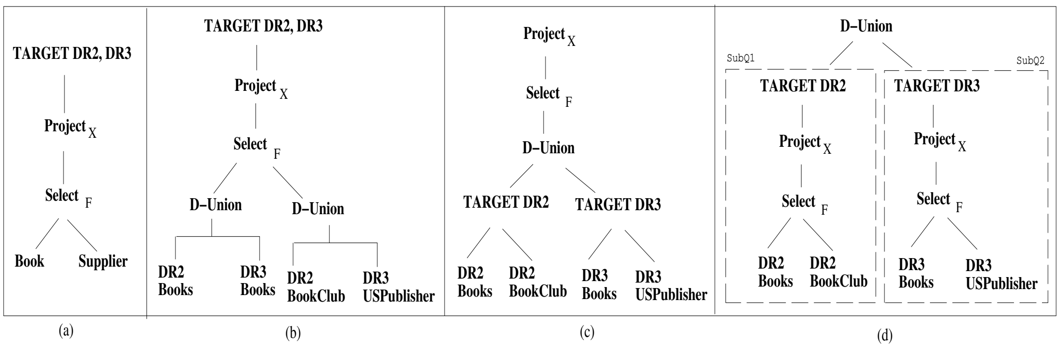


Figure 5: An example of query decomposition

Figure 5(d). As a result, this example query is decomposed into two subqueries, each targeting at a single source, as shown below.

```
SubQ1:
  TARGET DR2
  SELECT Book->title, Book->price, Supplier->name
  FROM   Book, Supplier
  WHERE  Book->medium = 'CD-ROM'
  AND    Book->title CONTAINS 'multimedia'
  AND    Book->supplierID = Supplier->supplierID;

SubQ2
  TARGET DR3
  SELECT Book->title, Book->price, Supplier->name
  FROM   Book, Supplier
  WHERE  Book->medium = 'CD-ROM'
  AND    Book->title CONTAINS 'multimedia'
  AND    Book->supplierID = Supplier->supplierID;
```

The original query is now transformed into an semantically equivalent DIOM distributed query algebraic expression: *D-Union*(`SubQ1, SubQ2`). The operation *D-Union* is a distributed union operator which adds the necessary semantic attachments to the result of `SubQ1` and the result of `SubQ2` to make them union-compatible before applying union operation to combine the two operands. A detail discussion is provided in Section 4.5.

The second phase of query decomposition is the recording of useful subquery dependencies. According to the result of target split (Figure 5(e)), the original query in Figure 3 is transformed into an equivalent IQL query expression D-Union(`SubQ1, SubQ2`). Thus the execution of the original query $Q_1$ can be reduced to the executions of the two subqueries and the distributed union of the two subquery results.

There are three alternative execution sequences for these two subqueries:

1. `SubQ1` $\|$ `SubQ2`;
2. `SubQ1` $\mapsto$ `SubQ2`;
3. `SubQ2` $\mapsto$ `SubQ1`.

The synchronization operator $\|$ is called *parallel dispatch* operator. The dependency expression `SubQ1` $\mapsto$ `SubQ2` indicates the sequence that `SubQ1` is executed before `SubQ2`. Obviously, the selectivity of `SubQ1` is at the same level as the selectivity of `SubQ2`. Thus it is unlikely that the result of `SubQ1` is useful to the query processing of the subquery `SubQ2` and vice versa. Therefore the parallel execution of `SubQ1` and `SubQ2` is the only useful dependency with respect to the performance of subquery processing. This dependency annotation specifies that the submission and execution of the subqueries `SubQ1` and `SubQ2` should be carried out in parallel, the results returned can be merged directly in terms of the DIOM annotation-based union operator.

Obviously, in order to make the above subqueries executable over the corresponding information source, each of the subqueries must be translated into the correct query statements expressed in the specific source query language. This is one of the tasks that will be carried out at the wrappers level since the subquery translation and execution can be seen as a source-specific task.

We have outlined the query routing and query decomposition strategies and provided an example to illustrate the flavor of query routing and query decomposition process. Since the global optimization of mediated queries is beyond the focus of this paper, we will not discuss in this paper the detail on the selection of useful subquery dependencies, the design of the DIOM distributed object algebra operators, and the parallel access plan generation.

## 4.4 Subquery Translation

After query routing, query decomposition and parallel access plan generation, the subqueries in IQL expressions will be passed from the mediator to the corresponding wrappers for subquery translation and execution. The wrapper will first convert each IQL subquery into a query expression that is executable over the information source to which the wrapper is associated. When the repository is a RDBMS, say Oracle, the wrapper will map an IQL expression into an Oracle/SQL query statement. If the data source is an OODBMS (say ObjectStore), the wrapper will map each IQL expression into an ObjectStore query method that is bounded to an ObjectStore class dictionary. When the data source is stored and managed solely by a file server, say in the form of HTML, then the wrapper will map the IQL expression into, for example, a C module or a Perl module that scans the source data and returns the matching information.

We are currently investigating the generic wrapper architecture and the generic conversion functions for transforming DIOM-IQL queries to different types of data repositories. Our first implementation is done through a Netscape based Diorama Query Supervisor, which currently runs over three types of wrappers: an Oracle wrapper for structured data sources, a HTML wrapper for accessing unstructured or semi-structured web data sources (e.g., WWW pages, NetNews articles) and a wrapper for bibtex files.

The subquery translation module is in charge of translating a subquery in IQL into a source-specific query in the source query language or a piece of program that can be executed over the corresponding source. This is done in three steps: (1)Attribute conversion and type replacement. (2)Completion of the subquery expression. (3)Subquery refinement. Due to the space limitation, an example is used to illustrate these steps. For detail see [13].

Consider the *book sale* scenario. Recall Section 4.3, the query $Q_1$ was decomposed into two subqueries `SubQ1` and `SubQ2`. The procedure of subquery translation proceeds as follows:

- **(1) Attribute conversion and type replacement** The procedure of subquery translation starts with attribute conversion and type replacement . For each subquery, the query decomposition module searches the source-specific implementation repository in the metadata catalog, and replace the mediated attributes and interface types by the corresponding source attributes and entity types defined at information source model level. For instance, by using the metadata maintained in the implementation repository of DR2 [13], we may find that the mediated attribute `Book->price` has the matching source attribute `Order.price` in the source DR2. Following a similar approach, all the mediated attributes are replaced by the corresponding source attributes. The two subqueries generated at the query decomposition module (Section 4.3) are further modified as follows:

```
SubQ1:
TARGET DR2
SELECT Books.description, Order.price, BookClub.name
FROM   Books, BookClub
WHERE  Books.medium = 'CD-ROM'
AND    Books.description CONTAINS 'multimedia'
AND    Books.clubID = BookClub.clubID;
```

```
SubQ2:
TARGET DR3
SELECT Books.title, Books.price, Publisher.name
FROM   Books, Publisher
WHERE  Books.medium = 'CD-ROM'
AND    Books.title CONTAINS 'multimedia'
AND    SaleDept.publisherID = Publisher.publisherID
AND    PublishDept.publisherID = Publisher.publisherID;
```

Obviously, these two subqueries are still incomplete with respect to relational systems because the relational SQL requires that (i)all relations involved in a query be declared in the FROM clause; and (ii)the join condition be explicitly expressed when multiple relations are involved in a query. Thus, the next step of the subquery translation is to complete the subquery expression and to refine the subqueries according to the constraint information available in the export schema of the source.

- **(2) Completion of the subquery expression** Since both the source DR2 and the source DR2 are relational data repositories, according to the relational SQL, all the relations involved in the query should be declared in the `FROM` clause, and the join conditions among the relations should be explicitly expressed in the `WHERE` clause. Based on the metadata maintained by the wrapper to DR2 and the wrapper to DR3, the join condition "`Books.bookID = Order.bookID`" should be added into the `WHERE` clause of the subquery `SubQ1`. The join conditions "`Books.deptID = SaleDept.deptID`" and "`Books.deptID = PublishDept.deptID`" should be added into the `WHERE` clause of the subquery `SubQ2`. Thus, the subquery `SubQ1` is further modified at the corresponding wrapper to DR2 as follows:

```
SubQ1:
TARGET DR2
SELECT B.description, O.price, C.name
FROM   Books B, Order O, BookClub C
WHERE  B.medium = 'CD-ROM'
AND    B.description CONTAINS 'multimedia'
AND    C.clubID = B.clubID
AND    B.bookID = O.bookID;
```

The subquery `SubQ2` is modified at the corresponding wrapper to DR3 by adding the additional join conditions to the `WHERE` clause and the additional relations to the `FROM` clause:

```
SubQ2:
TARGET DR3
SELECT Books.title, Books.price, Publisher.name
FROM   Books, Publisher, SaleDept, PublishDept
WHERE  Books.medium = 'CD-ROM'
AND    Books.title CONTAINS 'multimedia'
AND    SaleDept.publisherID = Publisher.publisherID
AND    PublishDept.publisherID = Publisher.publisherID
AND    Books.deptID = SaleDept.deptID
AND    Books.deptID = PublishDept.deptID;
```

- **(3) Subquery refinement**
  For example, suppose that the source DR2 has the following constraint "all the books in CD-ROM cost more than \$150". It is described as follows: "for any B in Books, if B.medium = 'CD-ROM' and for any O in Order, O.bookID = B.bookID, then O.price >= 150". Based on this constraint, the subquery SubQ1 can further be modified by incorporating this constraint into the query expression as follows.

```
SubQ1:
TARGET DR2
SELECT B.description, O.price, C.name
FROM   Books B, Order O, BookClub C
WHERE  B.medium = 'CD-ROM'
AND    B.description CONTAINS 'multimedia'
AND    C.clubID = B.clubID
AND    B.bookID = O.bookID AND O.price >= 150;
```

  Obviously this modified subquery has smaller search scope than the previous one.

### 4.5 Query Result Assembly

Once all the subqueries are processed and the results are returned to the wrappers (the DIOM local agents), the results must be assembled and attached with additional semantic information before presenting to the user in terms of the user's preferred terminology. Instead of enforcing the integration of structural heterogeneity and semantic heterogeneity in various information sources, we propose a semantic attachment approach to assemble the query results. The techniques we apply include source attachment, scale attachment, alternation attachment, hub-link attachment, and abscent attribute annotation. We are currently working on identifying the basic set of primitive operations for semantic attachment purpose and the desirable combinations. Due to the space limitation, we here omit the description and illurstration of these techniques.

## 5 Conclusion

We have described an adaptive approach to query mediation in a cooperative information system environment where information consumers and information producers are interconnected across organizational boundaries and customizable information gathering across heterogeneous information sources is critical. We demonstrated by examples how the adaptive query mediation framework is implemented through the DIOM IDL interface abstraction mechanisms, the flexibility of DIOM IQL expressions, and the systematic development of query routing, query decomposition, subquery translation and query result assembly. This proposed query mediation framework has a number of important features: (1) modularity and composability in terms of representing mediator's domain knowledge and information sources. (2) extensibility in terms of adding new information consumers and information sources. (3) transparency in terms of posing query requests. (4) scalability in terms of selecting the most appropriate information sources and reformulating to answer a query. (5) autonomy and robustness in terms of evolution of information sources and the requirement changes of information consumers. (6) efficiency in terms of minimizing the overall response time for a query. DIOM and Diorama are part of the Agile/Harvest project [4], adding heterogeneity to the scalability of Harvest architecture.

Comparing with traditional query processing work such as Pegasus, UniSQL/M, where query processing is concerned primarily with decomposition, translation, and optimization using the global database schema, our approach is more promising in a dynamicly evolving information system environment, since USECA properties require that we avoid depending on the existence of a global schema. Comparing with the recent information mediation projects, TSIMMIS and SIMS are more advanced in the system aspects of the project and have built prototypes; whereas DIOM/Diorama is conceptually more ambitious in terms of USECA properties and follows an incremental implementation plan. For example, information consumers may express queries even when they have no precise knowledge of which information sources are relevant. Mediated query processing currently proceeds in five steps: query routing, query decomposition, parallel access planning, query translation, and query result assembly. Explicit and dynamic query routing and result assembly allow the system to scale up to a large number of nodes. Thus, the same query from the information consumer will return increasingly richer answers as the system grows. Furthermore, the DIOM query mediation approach encourages the delay of resolution of heterogeneity problems to the query result assembly time. In contrast to the enforcement of an integrated and monolithic global schema, the heterogeneity problems are resolved by applying an explicit semantic attachment scheme to the assembled query results.

The development of DIOM and Diorama is just beginning. Our ongoing research includes efforts in several areas, for example, the formal development of a distributed object query algebra, including joins involving multiple information sources; the design of the algorithms for query decomposition and for generation of optimal parallel access plan which also takes into account the efficient processing of aggregate functions such as SUM, COUNT, MAX and MIN. We are also interested in caching/replication issues for performance and availability, and in introducing learning capabilities to improve the query processing efficiency. Last but not the least, systematic construction of wrappers for various types of data sources is a non-trivial task. We are currently investigating on the generic architecture for wrapper construction by building wrappers to UNIX files and ObjectStore databases, in addition to the wrappers to HTML files and ORACLE databases.

## Acknowledgement

The work was supported partially by NSERC grant OGP-0172859 and NSERC grant STR-0181014 for the first author, and by ARPA under contract DABT63-95-C-0010 and NSF under grant IRI-9510112 for the second author.

# References

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

[2] Y. Arens and et al. Retrieving and integrating data from multiple information sources. *International Journal of Intelligent and Cooperative Information Systems*, 2(2):127–158, 1993.

[3] Y. Arens and C. Knoblock. Planning and reformulating queries for semantically-modeled multidatabase systems. In *Proceedings of the first International Conference on Knowledge and Information Management*, 1992.

[4] C. Bowman, P. Danzig, D. Hardy, U. Manber, and M. F. Schwartz. The harvest information discovery and access system. In *Proceedings of the Second International World Wide Web Conference*, pages 763–771, Chicago, Illinois, October 1994.

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

[6] Y. Chang, L. Raschid, and B. Dorr. Transforming queries from a relational schema to an equivalent object schema: a prototype based on f-logic. In *Proceedings of the International Symposium on Methodologies in Information Systems (ISMIS)*, 1994.

[7] D. Florescu, L. Raschid, and P. Valduriez. Using heterogeneous equivalences for query rewriting in multidatabase systems. In *Proceedings of the International Conference on Cooperative Information Systems (CoopIS)*, 1995.

[8] H. Garcia-Molina and et al. The tsimmis approach to mediation: data models and languages (extended abstract). In *Technical Report, Stanford University*, 1994.

[9] C. Goh, S. Madnick, and M. Siegel. Context interchange: overcoming the challenges of large-scale interoperable database systems in a dynamic environment. In *Proceedings of International Conference on Information and Knowledge Management*, pages 337–346, 1994.

[10] W. Kim. Observations on the odmg-93 proposal. *ACM SIGMOD RECORD on Management of Data*, 23(1), March 1994.

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

[12] L. Liu. A recursive object algebra based on aggregation abstraction for complex objects. *Journal of Data and Knowledge Engineering*, 11(1):21–60, 1993.

[13] L. Liu and C. Pu. Customizable information gathering across heterogeneous information sources. Technical report, Department of Computer Science, University of Alberta, Dec. 1995.

[14] L. Liu and C. Pu. The distributed interoperable object model and its application to large-scale interoperable database systems. In *ACM International Conference on Information and Knowledge Management (CIKM'95)*, Baltimore, Maryland, USA, November 1995.

[15] L. Liu, C. 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.

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

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

[18] L. Qian and L. Raschid. Query interoperation among relational and object-oriented databases. In *Proceedings of the International Conference on Data Engineering*, TaiPei, 1995.

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

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

[21] A. Sheth and J. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Trans. Database Syst.*, Vol. 22, No.3 1990.

[22] M. Siegel and S. Madnick. Context interchange: sharing the meaning of data. In *ACM SIGMOD RECORD on Management of Data*, pages 77–78, 20, 4 (1991).

[23] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer Magazine*, March 1992.

[24] G. Wiederhold. I3 glossary. *Draft 7*, March 16 1995.