Query Processing in the Information Manifold

Alon Y. Levy
AT&T Bell Laboratories
levy@research.att.com

Anand Rajaraman*
Stanford University
anand@cs.stanford.edu

Joann J. Ordille
AT&T Bell Laboratories
joann@research.att.com

This manuscript includes an appendix, which has been added just for completeness purposes.

Abstract

The World Wide Web (WWW) provides access to a multitude of information sources. A growing number of these sources are structured, and provide a query interface, and even more sources behave as structured sources if we use an appropriate interface program (e.g., for parsing structured text files). Unfortunately, even though these sources, when put together, have the potential of answering complex user queries, the current interaction with them is much like browsing. A user must consider the list of sources available, decide which ones to access, and then interact with each one individually. Furthermore, combining information from different sources can only be done manually.

This paper describes the Information Manifold, a system that provides a uniform query interface to large collections of structured information sources. A user poses queries to the Information Manifold using a uniform schema, and the system creates a plan for answering the query based on descriptions of the contents and capabilities of the available information sources. The contributions of this paper include (1) a language for describing information sources that enables expressing fine grained distinctions between their contents and describing their capabilities, (2) efficient algorithms for creating query plans based on these descriptions, and (3) a description of the implemented Information Manifold system that currently answers queries using over 50 WWW information sources.

*Part of this work was done while the author was visiting AT&T Bell Laboratories.

1 Introduction

The World Wide Web (WWW) provides access to a multitude of information sources. A growing number of these sources are structured, and provide a query interface (e.g., HTML forms). Even more sources behave as structured sources if we use an appropriate interface program (e.g., for parsing structured text files). Unlike unstructured sources that can only be viewed and browsed by the user, structured sources can be used to answer specific user queries. In fact, several Internet providers see the growing market potential of services that access rich collections of structured information sources. Currently, the interaction with such a service (e.g., the French Téletel system) is much like browsing. The user must consider the list of sources available, decide which ones to access, and then interact with each one individually, using their specific schema and query interface. Furthermore, there are few tools to aid the user in combining information from multiple structured sources. The large number of information sources makes the process of searching for information unwieldy even if the list of sources is available, not to speak of the user having to *find* the relevant sources on the WWW.

A system that provides a database interface to the multitude of information sources can address these problems. In such a system, the user expresses *what* he or she wants and the system answers the query using the available sources. In particular, the system frees the user of having to find the relevant information sources and interact with each one separately, and it combines information from multiple sources to answer user queries. There are several challenges in providing such an system. First, as in federated and multidatabase systems, the information sources are autonomous and are therefore heterogeneous, using different data models, schemas and representations for objects. Second, there are many information sources with closely related data (e.g., many publication repositories, market databases, name servers). We need to be able to model the finer-grained differences between contents of sources, so we can determine efficiently which sources are relevant to a given query. Third, sources tend to contain incomplete information (e.g., few publication repositories can claim to contain *all* publications of a certain class), and therefore multiple sources need to be queried for the same information. Finally, information sources differ not only in their contents, but also in their *capabilities.* For example, one source may require that certain arguments be bound in queries, while another source may have the ability to perform limited forms of processing (e.g., selections on certain attributes), which we would like to exploit, in order to minimize the amount of data that needs to be transmitted over the network, and the amount of processing that has to be done.

The Information Manifold system, developed at AT&T Bell Laboratories, provides a uniform query interface to large collections of structured information sources, based on the architecture shown in Figure 1. A user of the Information Manifold is presented with one schema, the *world-view,* which is a set of classes and relations about the domain of the information sources. Queries are formulated in terms of the world-view relations. The system contains *descriptions* of the available information sources, in which the contents and the capabilities of the sources are described. The declarative language for describing the sources enables expressing fine-grained distinctions between the contents of different sources. Given a query, the system uses the descriptions to determine which information sources are *relevant*, and generates a plan for answering the query that satisfies and exploits capabilities of the sources. The actual interaction with the sources is done through *interface programs* that consider the details of interacting with each source.

This paper describes how information sources are modeled in the Information Manifold and the novel algorithms the system uses for creating plans for answering queries. Specifically, the contributions of the paper are the following:

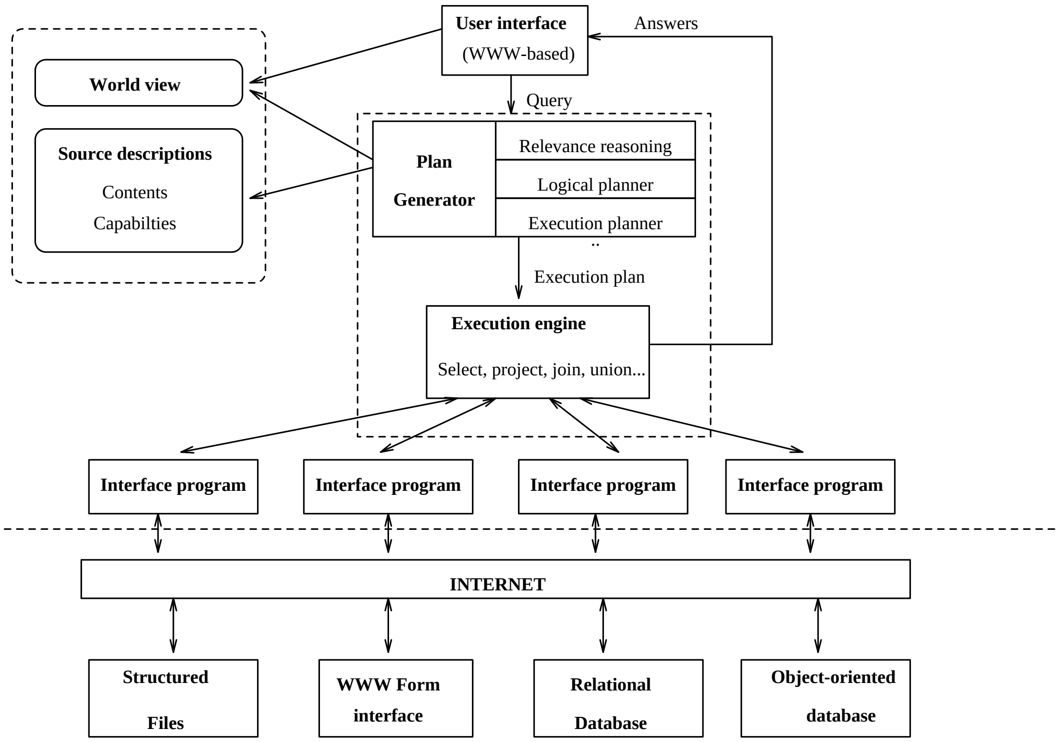


Figure 1: Architecture of the Information Manifold

1. A language for describing information sources. There are two novel aspects to this language. First, the contents of information sources are described by *queries* over the world-view. Consequently, it is easy to model a large number of sources and to express fine-grained distinctions between the contents of the information sources. Second, the language enables describing the capabilities of the sources, (e.g., their binding requirements and their local processing capabilities) which is crucial in order to interact with many existing sources. The language was designed to describe information sources commonly found, and has proven itself in practice.

2. Efficient algorithms for creating query plans. The algorithms have two components. The first component determines which information sources are relevant to a given query. The second determines whether the relevant information sources can be combined in a way that satisfies the capabilities of the sources and exploits those capabilities as much as possible.

3. Description of the implemented Information Manifold system. The Information Manifold uses the language and algorithms described to provide a uniform query interface to over 50 structured WWW information sources. Importantly, our architecture and tools make it very easy to make new information sources available to the system.

Several other systems (e.g.,TSIMMIS [7], HERMES [28], Nomenclator [22, 23]) have the goal of providing access to multiple information sources using the notion of a *mediator* [30]. However, in these systems mediators are built either for specific information sources or for specific query forms. In contrast, the Information Manifold can be viewed as a source independent and query independent mediator. The input to Information Manifold is a set of descriptions that relate the contents of the source to the world-view relations. Given a query, the Information Manifold will consider the descriptions and the query, and will create a plan for answering the query using the sources.

Consequently, it is easy to add information sources to the system because they can be described independently of the queries that will subsequently be asked of them. As in HERMES [28] and TSIMMIS [7], we separate between two aspects of the integration problem, the logical integration of sources and the interaction details with each source (that is handled by the interface programs).

There are many issues to be addressed in providing a uniform interface to multiple information sources, many of them not addressed in this paper. In particular, it should be emphasized that the goal of Information Manifold is to provide only a *query* interface, and not update or transaction facilities. As a consequence, we do not address some of the issues addressed in multidatabase systems such as consistency and transaction processing. Another set of issues that is not addressed concerns how descriptions of information sources are obtained[1] and where they are stored, and issues concerning the design of interface programs. Finally, issues of security and payment for information are also beyond the scope of this paper.

The paper is organized as follows. Section 2 illustrates the challenges involved in building a uniform query interface to multiple information sources and our contributions through an example. Section 3 briefly describes the data model used in Information Manifold, and Section 4 details our language for describing contents and capabilities of information sources. Section 5 describes the query processing algorithms in Information Manifold. Section 6 discusses the implemented Information Manifold system. Section 7 discusses related work, and Section 8 ends with concluding remarks. Appendix A has a representative list of information sources accessed by the Information Manifold.

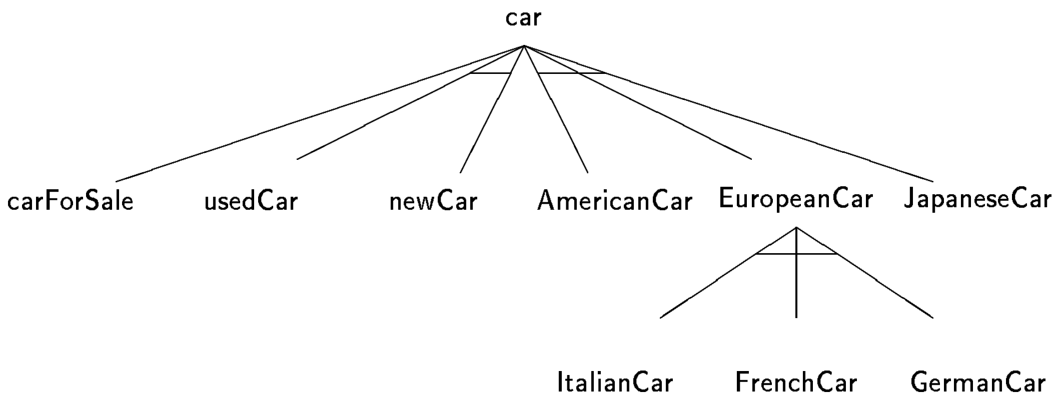
2 Illustrative Example

In this section we illustrate the problems involved in providing a query interface to many information sources and the solutions we propose. Our illustration is an example that we use throughout the paper.

Example 2.1: Consider an information broker for purchasing cars. The broker provides access to several databases describing cars for sale and to several servers providing online quotes for insurance premiums. The prospective buyer is provided with a schema identifying various classes of cars (see Figure 2), and a relation quote, that for a given insurance agent, year, model, mileage and accident history of a car, (i.e., the number of accidents in which it has been involved) and the age and city of a driver, provides the annual insurance premium. The class of cars also has the attributes mileage, year, price, ownerNumber, model, history and makeCountry.

The relation quote and the car classes are only virtual, i.e., they do not actually contain any tuples or objects. All the actual data is stored in external information sources. The Information Manifold contains descriptions of these information sources, as seen in Figure 3. A description consists of specifying the *contents* of the information source and its *capabilities*. The contents are described by a query, i.e., a view definition. In the example, information sources 1 and 2 contain instances of used cars and some information about them. Source 1 (as described by the definition of view v_1) contains only used cars manufactured in 1985 or later, while source 2 has only European cars. The capability descriptions of the sources tell us which queries they can answer on their contents. That is, they tell us which inputs *must* be supplied (a subset of the equality inputs), and which inputs can be further supplied to constrain the query to the source (a subset of the equality and selection inputs). Source 1 can be given either the model of the car or the country

[1]See [15] for a first large-scale experiment in obtaining descriptions of information sources that provide name services.



Disjoint classes: {AmericanCar, JapaneseCar, EuropeanCar}
{ItalianCar, FrenchCar, GermanCar}

Figure 2: Class hierarchy of of cars. Note that a car can be an instance of two or more non-disjoint classes (e.g., newCar and FrenchCar). The class ItalianCar is associated with the class description EuropeanCar $\sqcap$ (fills makeCountry Italy).

of its manufacturer as an input, and can also perform selections on the manufacturing year and mileage. However, the source must be given at least one of the equality inputs (i.e., a binding for either the model or country), and it can take a maximum of 4 inputs (either equality or selection inputs). Source 2 requires either the manufacturing year or the country of the manufacturer as input, and it can perform selections only on the mileage. Both sources can output any of the roles of the car, except that source 1 cannot output the accident history of the car. Note that neither information source guarantees that it has *all* the cars that satisfy the constraints in its contents description. Therefore, not finding a specific European car in source 2 does not mean it cannot be found in source 1.

Sources 3 and 4 give insurance quotes, one for the Allstate and the other for AAA, both only for the New York City area. Source 3 considers the accident history of the car as a factor in the premium, and therefore requires it as input, while source 4 ignores this information in its quotes. Otherwise, both sources require identical information in order to return a quote. The system obtains information from the sources 1–4 via *interface programs*. For example, the interface program may supply input to an HTML form interface and obtain the answer, or, alternatively, obtain data from a structured text file (e.g., using techniques as in [1, 27]).

Ordinarily, a person looking for a car from these sources would have to access each of the car sources individually, retrieve all the cars that may be relevant. Then, the user would have to take the data and enter it into one or more of the insurance agents to get premium quote. This is a tedious task, and the user is most likely to first make a final decision about which car to buy, and then search for insurance. This may not be optimal because insurance premiums vary between models and other attributes of cars, and therefore a car that seemed like a good buy may seem less so after the insurance premium is considered, or vice versa. Providing a uniform query interface to the multiple information sources of the two kinds provides two advantages. First, the user need not consider every car database (or even be aware of all existing databases), and second, the user can explicitly ask for the information from the car databases to be combined with the insurance information. For example, the user may pose the following query asking for Japanese cars (i.e., the

Source 1:
Used cars for sale in New York City after 1985.
Contents:
$\mathsf{v}_1(X) :- \mathsf{carForSale}(X)\ \&\ \mathsf{usedCar}(X)\ \&\ \mathsf{year}(X) \geq 1985.$
Capabilities:
equality inputs: $\{\mathsf{model}(X), \mathsf{makeCountry}(X)\}$
selection inputs: $\{\mathsf{year}(X), \mathsf{mileage}(X)\}$
minimum number of inputs: 1
maximum number of inputs: 4
outputs: $\{\mathsf{year}(X), \mathsf{model}(X), \mathsf{mileage}(X), \mathsf{price}(X), \mathsf{makeCountry}(X), \mathsf{ownerContact}(X)\}$.

Source 2:
European cars for sale in New York City

$\mathsf{v}_2(X) :- \mathsf{carForSale}(X)\ \&\ \mathsf{EuropeanCar}(X).$

$\{\mathsf{year}(X), \mathsf{makeCountry}(X)\}$
$\{\mathsf{mileage}(X)\}$
1
3
$\{\mathsf{year}(X), \mathsf{model}(X), \mathsf{mileage}(X), \mathsf{history}(X), \mathsf{price}(X), \mathsf{ownerContact}(X)\}$

Source 3:
Allstate insurance agency.
Contents:
$\mathsf{v}_3(Agent, Year, Model, Mileage, Hist, Age, City, Premium) :- \mathsf{quote}(Agent, Year, Model, Mileage, Hist, Age, City, Premium)\ \&\ Agent = Allstate\ \&\ City = NYC$
Capabilities:
equality
inputs: $\{Year, Model, Mileage, Hist, Age, City\}$
minimum number of inputs: 6
maximum number of inputs: 6
outputs: $\{Premium\}$

Source 4:
AAA insurance agency.

$\mathsf{v}_4(Agent, Year, Model, Mileage, Age, City, Premium) :- \mathsf{quote}(Agent, Year, Model, Mileage, Hist, Age, City, Premium)\ \&\ Agent = AAA\ \&\ City = NYC$

$\{Year, Model, Mileage, Age, City\}$
5
5
$\{Premium\}$

Figure 3: Example information sources

role values of a car) whose mileage is less than 30,000, and their annual insurance premium:

$$\mathsf{q1}(Car, Premium) :- \mathsf{JapaneseCar}(Car)\ \&\ \mathsf{mileage}(Car) < 30000\ \&\ \mathsf{year}(Car) \geq 1990\ \&\ \mathsf{quote}(Agent, \mathsf{year}(Car), \mathsf{model}(Car), \mathsf{mileage}(Car), \mathsf{history}(Car), Age, City, Premium)\ \&\ Age = 25\ \&\ City = NYC.$$

To answer this query, our query planning algorithm will first have to decide which information sources are relevant. Source 1 is relevant because the classes of used cars and of Japanese cars are not disjoint, and the manufacturing years given in source (≥ 1985) overlap with the years requested in the query (≥ 1990). Source 2 is irrelevant to the query because the class of European cars is disjoint from the class of Japanese cars. Sources 3 and 4 are relevant to the query because they provide quotes for the NYC area. Therefore, to obtain answers, source 1 can be combined with information from sources 3 and 4.

In order to create a plan for answering the query, we need to ensure that the capabilities of sources 1, 3 and 4 enable such a combination. When we consider combining information from sources 1 and 3, we see that source 3 requires the accident history of the car, but source 1 does not supply it. Therefore, source 1 can only be combined with source 4. Finally, we would like to exploit as much as possible the ability of the sources to perform selections. In our case, we can push the selection on the year and the mileage into source 1 and, reduce the number of irrelevant cars we obtain from the source. However, had we been using source 2, we would have to do the selection on year after retrieving the data from the source. □

The main costs involved in answering queries in this setting are contacting information sources and transmitting information to and from them. Therefore, the key challenges we need to address are (1) determining which information sources are relevant to a query by using the semantic descriptions of their contents, and (2) creating plans that adhere to the requirements of the sources and exploit their local processing power as much as possible.

3 The Data Model

The Information Manifold provides a uniform interface to a set of information sources. The user poses queries on one set of virtual relations and classes, called the *world-view.* It should be emphasized that the world-view is not a set of relations that attempts to capture all possible domains, but only of the domains of the information sources accessed by the system. Furthermore, the world-view should remain relatively stable when we add more information sources in these domains.

The system contains descriptions of information sources, that specify the precise relationship between contents of the sources and the relations in the world-view. Before we explain how we describe information sources, we introduce the data model underlying the world-view. The main motivation behind the data model is to be able to state complex constraints on the contents of sources, and therefore make fine-grained distinctions between them. The data model includes relations and class hierarchies. Class hierarchies are used both for modeling the declarative aspects of object oriented sources, and for expressing constraints on contents of sources.

Formally, the world-view includes relations of any arity and the standard *interpreted* relations for order (i.e., $\leq, <, =, \neq$). Unary relations in the domain model are called *classes,* and binary relations are called *roles.* If r is a role and a is a constant, the phrase *the* r *fillers of* a refers to the set of constants X for which $\mathsf{r}(a, X)$ holds. More generally, the fillers of the *role-chain* $\mathsf{r}_1 \cdot \mathsf{r}_2 \cdots \mathsf{r}_n$ on a constant a are all the values Y for which the following conjunction holds:

$$\mathsf{r}_n(a, X_1) \,\&\, \mathsf{r}_{n-1}(X_1, X_2) \,\&\, \ldots \,\&\, \mathsf{r}_1(X_{n-1}, Y),$$

i.e., all the values obtained by following the path expression $\mathsf{r}_1, \ldots, \mathsf{r}_n$ beginning from a. A subset of the binary relations (e.g., mileage, year in Example 2.1) are identified as *attributes*, meaning that any individual can have at most one r filler. When all the roles in a role-chain are attributes, we will abuse the notation slightly and use $\bar{\mathsf{r}}(a)$ to denote the single value of the r filler of a (e.g., $\mathsf{mileage}(Car)$). We use sans serif font for all relations.

Class descriptions are expressions that use *primitive* classes and a set of *description constructors* (described below) to define sets of objects in the domain. For example, suppose the fillers of the role owner of a car are all the people who owned the car. The following is a class description defining the class of Japanese cars, for which all owners are American and that had at most two owners:

$$\mathsf{JapaneseCar} \sqcap (\mathsf{all\ owner\ AmericanPerson}) \sqcap (\leq\ 2\ \mathsf{owner}).$$

Formally, class descriptions are built as follows: (A denotes a primitive class, C and D denote class descriptions, r denotes a role, and $\bar{\mathsf{r}}$ denotes a role chain).

A:	Satisfied exactly by objects of class A.
$C \sqcap D$:	Satisfied by objects satisfying both C and D.
(all $\bar{r}$ C):	Satisfied by objects for which all $\bar{r}$ fillers satisfy C.
(fills $\bar{r}$ a):	Satisfied by objects that have a as one of their $\bar{r}$ fillers.
($\geq$ n r):	Satisfied by objects that have at least n r fillers.
($\leq$ n r):	Satisfied by objects that have at most n r fillers.

The purpose of class descriptions is to be able to express complex constraints on the contents of information sources. The added expressive power enables us to better ***distinguish*** between sources, and decide which are and which are not relevant to a given query. We also use class descriptions to describe relationships between classes in the world-view (in addition to the explicit subclass relationships that are given). We do so by associating class descriptions with some classes in the world-view. Associating a class in the world-view with a description defines the class *intentionally* to contain the objects that satisfy the description. Note that if a class E is associated with a description D, then D cannot mention E, and recursively, neither can the definitions of the classes mentioned in D, etc.

In order for class descriptions to be useful, we need to be able to answer two kinds of questions about them. First, we need to be able to determine whether one description ***subsumes*** another: a description C subsumes a description D if an object that satisfies the constraints of D also necessarily satisfies the constraints of C. Second, we need to be able to determine whether two descriptions are *disjoint*, i.e., it is not possible that an object satisfying D will also satisfy C. The important property of our class description language is that these two kinds of queries are decidable in time that is polynomial in the size of the descriptions (because it is a variant of the language used in the CLASSIC system [4].)

Example 3.1: The description D_1 = ItalianCar $\sqcap$ (fills year 1995) is subsumed by the description D_2 = EuropeanCar $\sqcap$ (fills year 1995). This is because ItalianCar is equivalent to EuropeanCar $\sqcap$ (fills makeCountry *Italy*), and is therefore a subclass of EuropeanCar. The constraints on the attribute year are identical in D_1 and D_2, and therefore, D_1 is subsumed by D_2. The description JapaneseCar is disjoint from ItalianCar $\sqcap$ (fills year 1995) because Japanese and Italian cars are disjoint sets. $\Box$

Using class descriptions and interpreted relations, there are several ways of expressing constraints in the data model. Formally, a *constraint formula* is a conjunction of atoms of the following forms: (X denotes a variable, and τ, τ_1, and τ_2 denote variables or constants).

1. $\tau_1 \, \theta \, \tau_2$, $\theta \in \{\leq, <, =, \neq\}$,
2. $C(\tau)$, where C is a class description.

An *atom* is either a literal of a relation in the world-view (i.e., of the form owner(X, Y)), a description atom (of the form $D(X)$, where D is a description), or an atom of one of the interpreted predicates (e.g., $X \leq 5$). In our discussion, a *query* is a conjunction of atoms (as in Example 2.1).

3.1 Objects and Values in the Data Model

One of the most critical problems when combining information from multiple sources is the different treatment of objects and values in various sources. First, some sources (e.g., OODB's) are object based while others (e.g., relational DB's) are value based. Second, the same value may be represented differently in each source. For example, one source may have an id associated with every person, to which their name, social security number and address are attached. Another source may represent these three values as attributes of one relation. One source may represent a person's name by first name followed by last name, while another may have the last name first, or only the first initial of the first name.

Combining information from multiple sources requires that we be able to resolve these different representations, and decide when two information sources are referring to the same object or value. In general, it may not always be possible to do so, because there may not be unique mappings between representations in different sources, and ultimately we may need to resort to approximate matches.

Our data model contains *host values,* which are used to model various kinds of values and *logical id's,* that are used to model objects. Host values are either primitive (e.g., integers, strings) or complex, i.e., records whose fields are other host values. The fields of complex host values are modeled by roles in the world-view. As an example, an instance of class person is a logical id. The filler of the role name of an instance of person is an instance of class personName, whose instances are complex host values. The roles of personName are firstName, middleInitial and lastName, each of which is a simple host value of type string.

In some cases we will be able to compare host values *exactly*, and in others we will have to resort to approximate matching. Exact comparison between primitive host values is straightforward, and approximate matching is done by substring matching or by using heuristics that depend on the particular value (e.g., determining that 'Univ. of Toronto' is the same as 'The University of Toronto'). Two complex host values are exactly equal if each of their corresponding role values are exactly equal. In addition to the problem of equality between the role fillers, an information source may not always be able to provide all the roles (e.g., some source may not give the middle initial of a person). Here too we use heuristics that depend on the value being compared. In addition, complex host values always have a *catch-all* role, whose value is one string that captures all the other roles. For example, personName has the role fullName, that is a string including all the components of the name. The filler of this role is used for approximate comparisons if we are not able to parse enough of the other role fillers from the information source. Our approach to comparing host values is similar to the *correspondence functions* used in the Remote-Exchange system [9].

Comparing logical id's is more subtle. Some sources do not actually have objects corresponding to logical id's in the world-view, and only have the fillers of the roles of the object. Even if id's actually exist in some sources, they are not meaningful outside a specific source, and can therefore not be directly compared to each other. Deciding that two objects represented by logical id's (e.g., instances of person) in two different sources are the same can only be done by comparing the values of their role fillers. Therefore, we associate with every class in our domain model a set of *keys.* A key is a set of role-chains. If two objects agree on the values for the fillers of the role-chains of a key, we will conclude that they are the same. For example, a key on an instance of person can be social security number, or the combination of name and home phone number. When a logical id needs to be displayed as part of an answer to a query (as in our car example), we display the known values of its roles.

3.2 Main Aspects of the Data Model

To summarize, several aspects of our data model should be emphasized. This data model has the ability to express complex relationships between collections of data in the information sources. It uses an expressive language for describing constraints about classes and the other interpreted relations. It should be noted that although we support the additional expressive power, reasoning about these complex relationships is still efficient. In particular, our class description language is based on that used in the CLASSIC system, which was designed to provide an expressive language in which polynomial-time reasoning is possible. Furthermore, reasoning is only done on the descriptions of sources and not on the actual data. Therefore, the number of times we invoke these reasoning procedures depends on the number of sources and not on the sizes of the information sources.

The second point to emphasize about the data model is that we do not *store* data in it, and the manipulations we perform on data are on relatively small amounts of data and are done in main memory. Therefore, in designing our data model we are not concerned with issues that arise when data needs to be stored and manipulated efficiently in secondary storage.

Clearly an important element in the ability to combine information from multiple sources is the choice of comparison functions we use between objects and values. Our treatment of objects and values provides a framework in which such comparison functions can be expressed and tested. A detailed description and analysis of these comparison functions is beyond the scope of this paper.

Finally, it should be noted that we do not claim that our data model *integrates* the relational and object oriented data model. It simply provides the expressive power necessary to *describe* sources using those data models, so that we can query them.

4 Describing Information Sources

A key component of the Information Manifold architecture is the description of information sources. The description of a source tells the system how the contents of the source are related to the virtual relations in the world-view, and how they can be accessed, thereby enabling the system to locate data for the virtual relations. An important distinction that is made in our source description language is the distinction between the *contents* of the information source (i.e., data of which relations are stored in the source and the constraints satisfied by that data), and the *capabilities* of the source (i.e., which operations the query engine of the information source can perform). The Information Manifold uses these descriptions in order to generate plans for answering a given query.

4.1 Contents of Information Sources

The contents of an information source are described by a set of views definitions. Each view definition is a conjunctive query. The meaning of a view v in the contents description is that the information source contains tuples that satisfy the definition of v, and these can be obtained by the interface program to the source. Note that this does not mean that the information source contains *all* the tuples that satisfy v.[2]

In the simple case, the view definition may say that the view contains tuples of one of the world-view relations or a selection on such a relation. However, the ability to describe the contents by views enables the Information Manifold to model information sources that contain data that doesn't fit exactly into the schema of the world view, but instead may be the result of joins, selections and projections on world-view relations. Note that this is a key difference from the way external information sources are integrated in multidatabase systems (e.g., Pegasus [2]), where each external source corresponds to a distinct set of classes in the schema. The content descriptions of information sources may also have multiple view definitions, and in particular, this allows us to

[2] In the system we attach a number between 0 and 1 that describes the probability of finding a tuple that satisfies v in the information source. Currently we only use these numbers to order the possible query execution plans. The issues of obtaining such numbers and using them in more sophisticated ways is a subject of current research.

express that the source contains tuples satisfying a query involving unions, or tuples of different relations.

Example 4.1: In the description of the contents of source 1 (Figure 3), the view v_1 states that the source contains data about cars that are in the intersection of the classes carForSale and usedCar, and that furthermore satisfy the constraints on the year and location. Source 3 contains a projection and selection on a relation quote in the world-view, ignoring the accident history of the car. The main point to note is that the use of the class and order constraints makes it possible to describe the contents of the sources precisely, even though these precise classes do not exist in the world-view. □

4.2 Capabilities of Information Sources

The content descriptions of the information sources only tell us *what* is in the source, but do not tell us which queries the source can answer about the contents. For example, the source may only allow queries in which some of the arguments are bound. In Example 2.1 the car databases required that at least one of the attributes of the car be given. This may be because the size of the answers may otherwise be too big, or because the server does not want to give away its entire database for security reasons. On the other hand, sources may have 'positive' capabilities, for example, the ability to perform some relational operations locally (e.g., equality and range selections). Our car servers were able to perform selections on the mileage and year of the car. Applying a selection locally at the source can significantly reduce the amount of information that needs to be transmitted over the network, and the amount of processing done.

In this paper we consider the two kinds of capabilities encountered most often in practice, which are the ability of sources to apply a (perhaps limited) number of selections, and the variable binding requirements of information sources.[3] Sources with capabilities to perform arbitrary relational operators are considered in [16].

Formally, for every view definition v in the content description of a source we add a capability description of the form $(S_{in}, min, max, S_{out}, S_{sel})$, where S_{in}, S_{out} and S_{sel} are sets containing variables in the head of v or role-chains on those variables, and min and max are integers. Every variable in the head of v must appear in either S_{in} or S_{out} (either the variable itself or a role-chain on it). The meaning of the capability description is the following. In order to obtain a tuple of v from the information source, the interface program must be given a binding for at least min elements of S_{in}. If we provide the values $a_1, \ldots, a_n$ for the elements $\alpha_1, \ldots, \alpha_n$ in S_{in}, we will obtain all the tuples in the information source that satisfy $\alpha_1 = a_1 \& \ldots \& \alpha_n = a_n$. Note that if α_i is a role-chain on a variable in the head of v, then we obtain the tuples for which at least *one* of the fillers of the role-chain is a_i.

The elements in S_{out} determine which head variables of v or role-chains on them can be returned from the information source. Therefore, providing the interface program with the values $a_1, \ldots, a_n$ for the elements $\alpha_1, \ldots, \alpha_n$ in S_{in}, and asking for the values of $\beta_1, \ldots, \beta_l$ in S_{out} will produce the tuples $(Y_1, \ldots, Y_l)$ that satisfy the following conjunction:

$$\mathsf{v}(X_1, \ldots, X_m) \;\&\; \alpha_1 = a_1 \;\&\; \ldots \;\&\; \alpha_n = a_n \;\&\; \beta_1 = Y_1 \;\&\; \ldots \;\&\; \beta_l = Y_l.$$

[3] It should be noted that using the content descriptions of the sources we can, to a limited extent, model the capability of a source to perform *specific* join and projection operations. These capabilities can be described by stating that the source contains tuples of views resulting from these operations. However, this is only a limited form of specifying capabilities because we need to specify specific select-project-join queries that the source can answer, rather than specifying the relational operations separately from the relations to which they can be applied.

Finally, the set S_{sel} denotes on which variables of v (or role-chains on these variables) the information source can perform selections.[4] The number of inputs given to the source from $S_{in} \cup S_{sel}$ cannot exceed max. We assume conjunctive selections that use the interpreted order relations ($\{\leq, <, \neq, =\}$). Note that the elements of S_{in} can be viewed also arguments on which the source can perform equality selections.

Example 4.2: In Example 2.1, source 1 requires that either the country of the manufacturer of the car or its model be given in the query (or both can be given). Sources 3 and 4 require that all the data about the car and the driver be given in order to produce the annual premium, with the exception that source 4 does not require the accident history of the car. In addition, source 1 is able to perform selections on the mileage of the car and the year it was manufactured. □

4.3 Other Elements of Source Descriptions

In addition to content and capability descriptions, the descriptions of information sources contain additional information that is needed for the query processor, such as the name of the interface program to the source, the location (i.e., URL) of the source and locations of replicas of the source. The interface programs do the actual interaction with the information sources. For example, the interface program may interact with the source by supplying input to an HTML form interface and obtaining the answer, or, alternatively, it may obtain data from a structured text file. The interface program translates between representations of objects in the source and their representation as complex host constants in the world-view.

5 Generating Query Plans

We now describe the novel algorithms used in the Information Manifold to generate query plans. A query plan is a sequence of accesses to information sources interspersed with local processing operations. A query plan must combine information from various sources in a way that guarantees semantically correct answers. It must adhere to the capabilities of the information sources, and should try to use their local processing capabilities as much as possible.

Informally, given a query, we proceed in two steps. In the first, we consider the content descriptions of the information sources, and find which combinations of sources yield semantically correct solutions, which we call *logical solutions.* In the second step, we consider each logical solution, and check whether the accesses to the information sources can be ordered in a way that would satisfy their capability requirements, and exploit their local processing power. The results of the second step are *executable solutions*, which are passed to the query executor.

In order to evaluate the completeness properties of our query answering algorithms, we need to define clearly what we mean by *the solution* to a given query in this setting. Unlike in traditional database setting, the set of answers to a query needs to be defined with respect to the information sources available and their capabilities. We define formally the set of logical solutions, which are all the possible ways of combining information from sources to obtain correct answers, and the subset of the logical solutions that are executable solutions, and show that our algorithms indeed generate all executable solutions.

In our reasoning about relevance of sources, we augment the descriptions of the contents of the sources to reflect which roles can be obtained from the source (as specified in the capability

[4] In the implementation we also specify which kinds of selections can be performed on each argument.

descriptions). Formally, if the contents of a source are described by the following view definition:

$$\mathsf{v}(\bar{X}) :- g_1(\bar{Y}_1) \;\&\; \ldots \;\&\; g_n(\bar{Y}_n)$$

and the associated capability description is $(S_{in}, min, max, S_{out}, S_{sel})$, where $S_{in} \cup S_{out} \cup S_{sel} = \{\alpha_1, \ldots, \alpha_r\}$, the augmented view definition of v is:

$$\mathsf{v}(\bar{X}, U_1, \ldots, U_r) :- g_1(\bar{Y}_1) \;\&\; \ldots \;\&\; g_n(\bar{Y}_n) \;\&\; \alpha_1 = U_1 \;\&\; \ldots \;\&\; \alpha_r = U_r.$$

Hereafter whenever we refer to the view definition v, we will always mean the corresponding augmented view definition.

Example 5.1: The augmented view definition corresponding to v_1 from Figure 3 is:

$$\begin{aligned} \mathsf{v}_1(X, U_1, U_2, U_3, U_4, U_5, U_6) \quad :- \quad & \mathsf{carForSale}(X) \;\&\; \mathsf{usedCar}(X) \;\&\; \\ & \mathsf{year}(X) \geq 1985 \;\&\; \mathsf{model}(X) = U_1 \;\&\; \\ & \mathsf{makeCountry}(X) = U_2 \;\&\; \mathsf{mileage}(X) = U_3 \;\&\; \\ & \mathsf{year}(X) = U_4 \;\&\; \mathsf{price}(X) = U_5 \;\&\; \\ & \mathsf{ownerContact}(X) = U_6. \end{aligned}$$

□

5.1 Generating Logical Solutions

A query Q is posed to the Information Manifold using the relations in the world-view. The contents of the information sources are described as views over the world-view relations. Finding a logical solution, i.e., a combination of information sources that produce answers to the query, means finding a query Q' that uses only the views in the content descriptions (and interpreted relations), such that every answer to Q' is an answer to Q. Described in this way, the problem of finding a logical solution is closely related to the problem of answering queries using views [14, 26, 32, 6, 25].

The problem of answering queries using views is the following. Given a query Q using the relations $\mathsf{e}_1, \ldots, \mathsf{e}_m$, and a set of view definitions $\mathsf{v}_1, \ldots, \mathsf{v}_n$, over the same relations, find a query Q' that uses $\mathsf{v}_1, \ldots, \mathsf{v}_n$, such that Q is equivalent to Q'. The queries Q and Q' are equivalent if for any instantiation of the relations $\mathsf{e}_1, \ldots, \mathsf{e}_m$, the answers produced by Q and Q' are identical.

The problem we need to solve here is different in two ways. First, we do not require that Q' be equivalent to Q, but only that it is *contained* in Q, i.e., that for any instantiation of the relations $\mathsf{e}_1, \ldots, \mathsf{e}_m$, the answers produced by the logical solution Q' are a *subset* of those produced for Q. To see why, consider an example in which we are looking for Japanese cars, and have access to a database of Toyotas for sale. The second difference is that since information sources usually provide only a subset of the views by which they are described, we need all possible Q' that can generate answers for Q. For example, if we are looking for used cars, and we have two information sources that say that they give Japanese cars, we need to consider *both* of them, since neither of them contains *all* Japanese cars.

We can therefore define our problem as finding all the logical solutions to a query Q:

Definition 5.2: *Given a conjunctive query Q, a logical solution to Q is a non-empty conjunctive query Q'[5] that uses only views in the contents descriptions and interpreted relations, and such that Q' is contained in Q.* □

[5] An empty query is a query that produces the empty set of tuples for *any* database.

Note that if the query Q can be answered using a disjunction of conjunctive queries, then each one of them will be a logical solution. The problem of answering queries using views was shown to be NP-complete in [14], even for conjunctive queries without constraints. The algorithm suggested there is not very practical since it requires guessing a Q', and then checking whether it is a correct solution. We now describe an algorithm that exploits the characteristics of the domain to be efficient in practice, and considers the subtleties arising from the existence of class constraints, which are not considered in previous work.

Our algorithm has two steps. In the first, we compute, separately for each subgoal in the query, the relevant information sources from which tuples of that subgoal can be obtained. In the second step, we consider all possible combinations for information sources for each subgoal in the query, and check for each one whether it is a correct logical solution. Formally, suppose our query Q is

$$Q : \mathsf{q}(\bar{X}) \quad :- \quad \mathsf{g}_1(\bar{X}_1) \,\&\, \ldots \,\&\, \mathsf{g}_m(\bar{X}_m) \,\&\, C_q.$$

where C_q is the conjunction of the subgoals of interpreted predicates in the query. The set of constraint subgoals of the query, that includes both the interpreted subgoals and the class description subgoals, is denoted by $C(Q)$. The first step of the algorithm computes for every non-interpreted subgoal g_i, a set of pairs of the form (I, v), where I is an information source, and v is one of the views in the content description of I. The head of v is unified with the variables of g_i. The key behind the algorithm is to check that the constraints in Q do not contradict the constraints in v after the unification with g_i. The details are shown in Figure 4.

procedure generate-relevant-sources($\mathcal{I}$,Q)
/* $\mathcal{I}$ is a set of information sources, and Q is a conjunctive query of the form */
$Q : \mathsf{q}(\bar{X}) : -\mathsf{g}_1(\bar{X}_1) \,\&\, \ldots \,\&\, \mathsf{g}_m(\bar{X}_m) \,\&\, C_q$. */

for every non-interpreted subgoal $\mathsf{g}_i(\bar{X}_i)$, $1 \le i \le m$ **do:**
    $relevantSourceViews_i = \emptyset$
    for every non-interpreted subgoal $\mathsf{u}(\bar{Y})$ in the body of a view v in the content description of $I \in \mathcal{I}$ **do:**
        if $\mathsf{g}_i = \mathsf{u}$ **or** g_i and u are non-disjoint class descriptions **then:**
            Let ψ be the mapping defined on the variables of v as follows:
                if Y is the j'th variable in $\bar{Y}$ and is in the head of v
                    then $\psi(Y) = X_j$, where X_j is the j'th variable in $\bar{X}_i$.
                    else $\psi(Y)$ is a new variable that does not appear in Q or v.
            Let $C(Q)$ and $C(\mathsf{v})$ be the conjunction of constraint subgoals in Q and $\psi(\mathsf{v})$, respectively.
            if $C(Q)$ and $C(\mathsf{v})$ are non-disjoint constraints **then** add $(I, \psi(\mathsf{v}))$ to $relevantSourceViews_i$.
return $relevantSourceViews_1, \ldots, relevantSourceViews_m$.
end generate-relevant-sources.

Figure 4: Algorithm for determining which source views are relevant to a subgoal in the query. Disjointness of constraints is determined by combining algorithms for disjointness of order constraints [29] and of class constraints [4, 17]. Note that when we consider a subgoal g_i in the query with a view v, we try to unify g_i with every subgoal of v, because each one may result in a different unification and therefore a different set of constraints. In our example, we consider the subgoal $\mathsf{JapaneseCar}(Car)$ with both $\mathsf{carForSale}(X)$ and with $\mathsf{usedCar}(X)$ in v_1, but they both result in the same unification.

Example 5.3: In our example query,

$$\begin{aligned}\mathsf{q1}(Car, Premium) :- & \mathsf{JapaneseCar}(Car) \,\&\, \mathsf{mileage}(Car) < 30000 \,\&\, \mathsf{year}(Car) \geq 1990 \,\&\\ & \mathsf{quote}(Agent, \mathsf{year}(Car), \mathsf{model}(Car), \mathsf{mileage}(Car), \mathsf{history}(Car),\\ & \quad Age, City, Premium) \,\&\, Age = 25 \,\&\, City = NYC.\end{aligned}$$

we consider the subgoal JapaneseCar(Car) and the view v_1 in source 1. We unify JapaneseCar(Car) with carForSale(X), resulting in the unification $\{X \rightarrow Car\}$, and check whether the constraints imposed in the definition of v_1 are consistent with the constraints imposed in the query. Specifically, we consider whether the following constraint is satisfiable:

$$\begin{aligned}&\mathsf{JapaneseCar}(Car) \,\&\, \mathsf{mileage}(Car) < 30000 \,\&\, \mathsf{year}(Car) \geq 1990 \,\&\\ &\mathsf{carForSale}(Car) \,\&\, \mathsf{usedCar}(Car) \,\&\, \mathsf{year}(Car) \geq 1985.\end{aligned}$$

Since the classes carForSale and usedCar are not disjoint with JapaneseCar, and the years of manufacturing overlap with those in the query, we deem $\mathsf{v}_1(Car)$ as one of the relevant source views. We also unify JapaneseCar(Car) with usedSale(X), but since that results in the same unification as before, we don't explore that possibility further. Source 2 is deemed irrelevant because European cars and Japanese cars are disjoint classes. We then consider the quote subgoal and find that sources 3 and 4 are relevant because their location matches the query. Therefore, the output of procedure **generate-relevant-sources** is

$$\begin{aligned}\{(&Source1, \mathsf{v}_1(Car, \mathsf{model}(Car), \mathsf{makeCountry}(Car), \mathsf{mileage}(Car), \mathsf{year}(Car),\\ &\mathsf{price}(Car), \mathsf{OwnerContact}(Car)))\}\end{aligned}$$

for the first subgoal, and

$$\begin{aligned}\{&(Source3, \mathsf{v}_3(Agent, \mathsf{year}(Car), \mathsf{model}(Car), \mathsf{mileage}(Car), \mathsf{history}(Car), 25, NYC, Premium)),\\ &(Source4, \mathsf{v}_4(Agent, \mathsf{year}(Car), \mathsf{model}(Car), \mathsf{mileage}(Car), 25, NYC, Premium))\}\end{aligned}$$

for the second subgoal. Note that the unification of the query subgoal with v_4 does not consider the history of the car in the query, as opposed to v_3. □

In the second step of the algorithm we consider every conjunctive query Q' of the form

$$Q' : q'(\bar{X}) \quad :- \quad \mathsf{v}_1(\bar{Y}_1) \,\&\, \ldots \,\&\, \mathsf{v}_n(\bar{Y}_n) \& C_q$$

where v_i is the view in one of the pairs computed for the subgoal g_i. Any minimal subset of Q' that is contained in Q (or that can be made to be contained by adding interpreted subgoals) is added to the list of logical solutions. The containment test we use is an extension of a containment algorithm for conjunctive queries with built-in predicates [18] that considers the semantics of the class descriptions. Although containment is known to be intractable in general, its intractability is in the size of the query (which tend to be small), and only occurs when queries have multiple occurrences of the same relations, which does not happen frequently in practice.

Example 5.4: In our example, we would consider combining sources 1 and 3, and obtain the following:

$$\begin{aligned}\mathsf{q}_1'(Car, Premium) :- & \mathsf{v}_1(Car, \mathsf{model}(Car), \mathsf{makeCountry}(Car), \mathsf{mileage}(Car), \mathsf{year}(Car),\\ & \quad \mathsf{price}(Car), \mathsf{OwnerContact}(Car)) \,\&\\ & \mathsf{v}_3(Agent, \mathsf{year}(Car), \mathsf{model}(Car), \mathsf{mileage}(Car), 25, NYC, Premium) \,\&\\ & \mathsf{mileage}(Car) < 30,000 \,\&\, \mathsf{year}(Car) \geq 1990.\end{aligned}$$

To test whether this query is contained in Q, we expand it with the definitions of the views, to obtain the following equivalent form, on which we check containment. Recall that we are using the abbreviated notation for attribute values, i.e., $\mathsf{model}(Car)$ will actually be replaced by a variable Y, and the subgoal $\mathsf{model}(Car, Y)$ would be in the query for the containment check.[6]

$$\begin{aligned}\mathsf{q}''(Car, Premium) :- & \mathsf{carForSale}(Car) \,\&\, \mathsf{usedCar}(Car) \,\&\, \mathsf{year}(Car) \geq 1985 \,\&\\ & \mathsf{quote}(Agent, \mathsf{year}(Car), \mathsf{model}(Car), \mathsf{mileage}(Car), \mathsf{history}(Car), 25, NYC, Premium) \,\&\\ & Agent = Allstate \,\&\mathsf{mileage}(Car) < 30,000 \,\&\, \mathsf{year}(Car) \geq 1990.\end{aligned}$$

To see that q'' is contained in Q we need to check that every subgoal in Q will be entailed by some subgoal in q''. It is easy to see that this holds for all subgoals except for the subgoal $\mathsf{JapaneseCar}(Car)$. In order to see that we can also enforce $\mathsf{JapaneseCar}(Car)$, our containment algorithm will note that this subgoal can be rewritten as $\mathsf{car}(Car) \,\&\, \mathsf{makeCountry}(Car, Japan)$. The first conjunct is entailed by $\mathsf{carForSale}(Car)$, and the second conjunct can be enforced by adding the subgoal $\mathsf{makeCountry}(Car) = Japan$ to q'_1. Hence the algorithm will decide that the following is a logical solution:

$$\begin{aligned}\mathsf{q}'_1(Car, Premium) :- & \mathsf{v}_1(Car, \mathsf{model}(Car), \mathsf{makeCountry}(Car), \mathsf{mileage}(Car), \mathsf{year}(Car),\\ & \quad \mathsf{price}(Car), \mathsf{OwnerContact}(Car)) \,\&\\ & \mathsf{v}_3(Agent, \mathsf{year}(Car), \mathsf{model}(Car), \mathsf{mileage}(Car), 25, NYC, Premium) \,\&\\ & \mathsf{mileage}(Car) < 30,000 \,\&\, \mathsf{year}(Car) \geq 1990 \,\&\, \mathsf{makeCountry}(Car) = Japan.\end{aligned}$$

The same will be done when trying to combine sources 1 and 4. □

It follows from [14] that if the view definitions in the content descriptions of the sources and the query do not contain non-primitive class descriptions, our algorithm produces *all* the needed logical solutions. The subtle point to note is that even though there may be logical solutions that are not produced by the algorithm, it is guaranteed that these additional solutions will not yield any answers to the query that are not yielded by the logical solutions that are found. In the presence of arbitrary class descriptions, we preprocess the query to obtain an equivalent query of polynomial size, such that our algorithm will find all the needed logical solutions. Because of space limitations, we do not go into these details here.

In practice, our algorithm has several advantages over the non deterministic algorithm described in [14]. Whereas that algorithm provides no guidance as to which possible solutions to apply the containment check (as in the second step of our algorithm), the first step of our algorithm, in which we consider the sources relevant to each subgoal in isolation, is a very effective filter on candidate logical solutions. Therefore, even though in the second step we perform a containment check on every element in the cross product of sources relevant to every subgoal in the query, the filter of the first step reduces considerably the number of elements in the cross product. This reduction is especially effective when there are few interactions between subgoals in the query (i.e., the relevance of a source depends mostly on the subgoal it is being matched with and on the constraints in the query, and not the other subgoals of the query). Additionally, the order in which the possible logical solutions are considered in the second step of the algorithm makes it possible to reuse some of the effort involved in the containment tests among different solutions, and to produce plans that exploit common subexpressions.

[6]Note that for clarity, in q'' we have omitted some of the subgoals that were added to v_1 during the view augmentation, since they are not needed for the containment test.

5.2 Computing Executable Solutions

A logical solution guarantees that the answers obtained by the solution will be semantically correct. However, it does not consider the capabilities of the information sources to answer the queries. We now describe an algorithm that checks whether the subgoals of a logical solution can be ordered in a way that satisfies the capability requirements of the information sources. Informally, in order to be executable, an ordering of the subgoals needs to satisfy two conditions: (1) the input binding requirements of the information sources must be satisfied, i.e., enough bindings must be available either from the input to the query or from the output of previous subgoals, and (2) we must ensure that if a variable occurs in two subgoals, we will be able to compare its values from the different sources, either directly or through keys.

Formally, an executable solution is an ordering $\mathsf{v}_1(\bar{X}_1), \ldots, \mathsf{v}_n(\bar{X}_n)$ on the view literals of the logical solution. An *argument* is either a variable in the logical solution or a role-chain on one of the variables. We denote the capability description of v_i by $(S^i_{in}, S^i_{out}, min^i, max^i, S^i_{sel})$. For every subgoal the plan will assign triples $(\mathsf{v}^i_{in}, \mathsf{v}^i_{out}, \mathsf{v}^i_{sel})$. The set v^i_{in} is the set of arguments that will be passed to the ith information source, v^i_{out} is the set of arguments that will be obtained from it, and v^i_{sel} is the set of arguments on which it will perform selections. We denote by Q_{in} the set of attributes available explicitly in the query and by Q_{out} the set of arguments required in the answer. An executable solution can be defined as follows.

Definition 5.5: *An ordering $\mathsf{v}_1(\bar{X}_1), \ldots, \mathsf{v}_n(\bar{X}_n)$ and assignments $(\mathsf{v}^i_{in}, \mathsf{v}^i_{out}, \mathsf{v}^i_{sel})$ for $1 \leq i \leq n$ is said to be an executable plan if the following conditions hold for every i, $1 \leq i \leq n$:*

1. $\mathsf{v}^i_{in} \subseteq S^i_{in}$, *and* $\mathsf{v}^i_{out} \subseteq S^i_{out}$, *and and* $\mathsf{v}^i_{sel} \subseteq S^i_{sel} \cup S^i_{in}$,
2. $\mathsf{v}^i_{in} \subseteq Q_{in} \cup \mathsf{v}^1_{out} \cup \ldots \cup \mathsf{v}^{i-1}_{out}$, *and*
3. *The cardinality of* $\mathsf{v}^i_{in} \cup \mathsf{v}^i_{sel}$ *is at most* max^i, *and the cardinality of* v^i_{in} *is at least* min^i,
4. $Q_{out} \subseteq Q_{in} \cup \mathsf{v}^1_{out} \cup \ldots \cup \mathsf{v}^n_{out}$, *and*
5. *If a variable X appears in two subgoals v_i and v_j then the two occurrences are* connected. *The connection relation is the transitive closure of the* directly-connected *relation, defined as follows. An occurrence of X in v_i and in v_j are said to be directly connected if $\mathsf{v}^i_{in} \cup \mathsf{v}^i_{out}$ contains a key on X that is also in $\mathsf{v}^j_{in} \cup \mathsf{v}^j_{out}$.*

□

The first four conditions guarantee that the capability restrictions of the sources are satisfied, i.e., that the minimum number of inputs are supplied to the sources, and that the inputs are available from previous subgoals. The last condition guarantees that values for different occurrences of variables can be compared properly.

Figure 5 describes an algorithm that given a logical solution, finds an ordering on its subgoals that is executable if there is one, and produces the plan. The algorithm proceeds by maintaining a list of available arguments (using procedure **propagate-available-arguments**), and at every point adds to the ordering any subgoal whose input requirements are satisfied. Finally, the algorithm checks that variable occurrences are connected and pushes as many selections as possible to the sources. The procedure **propagate-available-arguments** determines which arguments are *implicitly* available. For example, if $\mathsf{makeCountry}(X,Y)$ is a subgoal in the query, and $\mathsf{makeCountry}$ is an attribute (i.e., can only have one value), then if the argument $\mathsf{continent} \cdot \mathsf{makeCountry}(X)$ is available, we can also conclude that continent(Y) is available. In words, if we know that Y is the country of make of Toyota, and the continent of make of Toyota is Asia, we can conclude that the continent of Y is Asia.

Example 5.6: Consider the logical solution q'_1 in Example 5.4. The arguments available in the query are the makeCountry of the car (this is derived from the class constraint JapaneseCar), and the age and city of the driver. Therefore, the subgoal of v_1 is put first, since one of its inputs (makeCountry(Car)=$Japan$) is available. As a result of querying source 1, we obtain the values for year, model and mileage. However, since source 3 needs also the accident history of the car, we cannot satisfy the input requirements of the v_3 subgoal, and therefore this logical solution is not executable. In contrast, when we consider source 4, we can satisfy its requirements because the source does not require the accident history. Since source 1 can perform selections on year and mileage, our plan will push these selections to the source. Finally, since all the values passed between source 1 and 4 are primitive host objects (i.e., strings and integers), all variables are connected. □

The following theorem shows that our algorithm will find an executable ordering whenever possible, and will do so in polynomial time.

Theorem 5.7: *Let Q' be a logical solution. If there is an executable ordering of the subgoals of Q', then procedure* **create-executable-plan** *will find it. The running time of the procedure is polynomial in the size of Q'.* □

Proof Sketch: There are two observations underlying the proof. The first is that if there is an ordering of the subgoals of Q', and a subgoal v_i appears in the i'th position, then its input requirements will still be satisfied even if other subgoals are pushed in front of it. This property is because the set of available arguments increases monotonically. The second observation is that the connectivity of multiple occurrences of variables is independent of the ordering of the subgoals.

Given these observations, suppose there is an executable ordering $v_1, \ldots, v_n$ of the subgoals of Q'. The subgoal v_1 will be added at some point to the ordering by the algorithm because its input requirements are already satisfied by the explicit bindings in the query. Inductively, if the subgoals $v_1, \ldots, v_{i-1}$ are added at some point to the ordering, then the algorithm will ultimately add v_i to the ordering, since its input requirements are satisfied after $v_1, \ldots, v_{i-1}$ have been added. Therefore, since all subgoals will be added at some point, some ordering will be found. □

Algorithm **create-executable-plan** generalizes an algorithm by Morris [21] for ordering subgoals in the presence of binding constraints in two ways. First, in our case, the values passed to the subgoals are not simply values of variables, but possibly fillers of role-chains on these variables, and there are requirements on which sets of role-chains are sufficient. Second, we need to ensure that multiple occurrences of values of the same variable from different sources can be compared via keys, which is a problem that does not arise in Morris' context.

Our language for describing information sources allows only one capability description for every view definition in the contents description of a source. This restriction essentially means that the arguments that can be obtained from the source do not depend on which arguments were chosen in order to satisfy its capability requirements (note that we are referring to the names of the arguments, not their values!). In practice, we have found our language to be sufficient to describe the sources we encountered. Conceivably, there may be situations in which our language will not suffice, and the output set depends on which set of input attributes we used. The following theorem shows that in such a case, the problem of determining whether there exists an executable ordering for a logical solution is intractable, and therefore the choice we made in our language also has important computational advantages. The proof of the theorem is given in Appendix B.

procedure propagate-available-arguments(Q, *bindings*)
/* Q is a conjunctive query and *bindings* are a set of arguments in Q. */
/* $\bar{r}_1$ and $\bar{r}_2$ denote (perhaps empty) role-chains. */
until no more arguments are added to *bindings* **do:**
  if $\bar{r}_1(X) \in bindings$ **and** X is a filler on the chain $\bar{r}_2(Y)$ in Q, **then** add $\bar{r}_1\bar{r}_2(Y)$ to *bindings*.
  if $\bar{r}_1\bar{r}_2(Y) \in bindings$ **and** X is a filler on the chain $\bar{r}_2(Y)$ in Q, **then**
    if the constraint $\{Z = \bar{r}_2(Y) \& X = \bar{r}_2(Y) \& X \neq Z\}$ is not satisfiable, **then** add $\bar{r}_1(X)$ to *bindings*.
return *bindings*.
end propagate-available-arguments.

procedure create-executable-plan(Q')
/* Q' is a logical solution whose non-interpreted subgoals are $u_1, \ldots, u_n$. */
/* We assume all bindings in Q' are given explicitly using the = relation as a conjunct. */
QueryBindings = The set of variables in Q' bound by values in the query.
QuerySelections = The set of variables in Q' for which the query contains a selection.
$BindAvail_0$ = **propagate-available-arguments**(Q', *QueryBindings*).
for $i = 1, \ldots, n$
  The i'th subgoal in the ordering, v_i, is any subgoal u_j of Q' that was not chosen earlier and at least u^j_{min} of the arguments in u^j_{in} are in $BindAvail_{i-1}$.
  if there is no such subgoal, **return** `non-executable`, **else**
    $BindAvail_i$ = **propagate-available-arguments**(Q', $BindAvail_{i-1} \cup u^j_{out}$).
    v^i_{in} = A minimal set of arguments in $BindAvail_{i-1}$ that satisfied the input requirement.
    v^i_{out} = All the arguments in u^j_{out}.
end for
if $Q_{out} \not\subseteq BindAvail_n$ **return** `non-executable`.
Remove any element from v^i_{out} that is not needed for connectivity of variables in the query.
if there are two occurrences of a variable $X \in Q'$ that are not connected, **return** `non-executable`.
for $i = 1, \ldots n$
  v^i_{sel} is the maximal number of arguments from $QuerySelections \cup BindAvail_{i-1}$ such that the cardinality of $v^i_{in} \cup v^i_{sel}$ does not exceed the input capacity of its source.
return $(v_1, v^1_{in}, v^1_{out}, v^1_{sel}), \ldots, (v_n, v^n_{in}, v^n_{out}, v^n_{sel})$.
end create-executable-plan.

Figure 5: Computing executable solutions. The procedure **propagate-available-arguments** determines which arguments are available at every point in the plan.

Theorem 5.8: *If every view definition in the content descriptions of information sources could have more than one capability description of the form $(S_{in}, min, max, S_{out}, S_{sel})$, then the problem of determining whether a logical solution has an executable ordering is NP-complete.* □

6 The Information Manifold System

The Information Manifold system uses the techniques described in the previous sections to provide a uniform query interface to structured information sources on the World Wide Web and internal sources at AT&T Bell Laboratories. This section briefly describes some of the unique aspects of the implementation of the system.

Users interact with the Information Manifold through a web based interface. The interface enables users to browse the categories of information available (i.e., the world-view), and to see which information sources are available. Users can formulate queries either using templates that are available for classes in the world-view, or by combining such templates into an arbitrary conjunctive query.

When a query is posed, the system uses the descriptions of information sources, as explained in the previous section, in order to decide which sources are relevant, and to compute the various ways in which answers to the query can be found (i.e., the executable solutions). An important aspect of the system is that it provides a *stream* of answers to the user, and therefore tries to minimize the time taken to begin and sustain the stream, as opposed to minimizing the time taken to provide *all* the answers to the query. Minimizing the time to the early tuples is important because the user is likely to find a satisfactory answer before all answers are exhausted. It does so by generating the executable solutions in the order of their likelihood to find answers to the query, and executing them as soon as they are available. The likelihood of the executable solutions is based on estimates on how likely a source is to contain tuples of a certain view (these estimates are given with the descriptions). For example, when looking for books, the Library of Congress is very likely to have a given book in comparison with other bibliography servers. The plan executor also tries to access information sources in parallel, whenever the plan allows for parallelism.

The system currently provides access to over 50 information sources in various domains, including name servers, publication databases, market databases and entertainment sources (e.g., movie and video databases, CD stores). A sample list of the information sources available is shown in Appendix A. Our language for describing information sources has proved to be useful in practice by enabling us to quickly and accurately model a large number of sources, while leaving the world-view relatively stable. We have developed a set of tools which enable us to speed up the process of generating interface programs to information sources on the WWW. In particular, many sources on the WWW have a form-based interface. We have developed a tool in which we simply determine the correspondence between the variables used in the form and the world-view, and then specify a grammar describing the format of the answers obtained from the source, and the bulk of the interface program is generated automatically. Several of the interface programs use an outerjoin-based technique [27] to convert hierarchically structured documents into relations.

7 Related Work

Our approach to integrating multiple information sources can be seen as treating the sources as a multidatabase or as a federated database (e.g., [2, 9, 11, 20]). In fact, the content descriptions of information sources can be viewed as a generalization of *exported schemas* in multidatabases. However, in multidatabases the correspondence between the contents of the individual databases and the global schema is more direct. As one example, in the Pegasus system [2] every external database is modeled as a class in the class hierarchy, which is disjoint from other classes. It is then possible to define superclasses that represent *unions* of databases. In the Information Manifold the contents of an information source can be defined as an arbitrary conjunctive query on classes and relations in the world-view. Therefore we do not have to create a class for every source, and we are able to make more fine-grained distinctions between the contents of sources. This difference has proven very important in practice in order to quickly add information sources. In Pegasus, determining the set of databases to access in order to answer a query is simple, because it is immediate from the query. One of the key difficulties in the Information Manifold is determining which information sources are relevant to a query. Finally, source capabilities are not considered in the multidatabase literature. The upshot of these differences is that in the Information Manifold the user can more easily specify *what* he or she wants, rather than having to formulate a query in a way that would guarantee that all the relevant information sources are accessed. On the other hand, it should be noted that since our goal is only to provide a query interface to the information sources, we are not concerned with issues of consistency and updating the information sources as in multidatabases.

In [19] a language for describing information sources that was less expressive than the one we describe here was proposed. The language did not consider the capability descriptions, and the algorithms described for finding relevant information sources did not deal with the case where source descriptions are given as views on the world-view relations. Consequently, only a limited range of information sources could be incorporated.

Several systems (e.g., TSIMMIS [7, 24], SIMS [3], HERMES [28], CARNOT [31],Nomenclator [22, 23]) for integrating multiple information sources are being built on the notion of a *mediator* [30]. The key aspect distinguishing Information Manifold from the other systems is its generality, i.e., that it provides a source independent, query independent mediator. Instead of being tailored to specific information sources and specific queries on these information sources, the input to Information Manifold is a set of descriptions of the contents and capabilities of the sources. Given a query, the Information Manifold will consider the descriptions and the query, and will create a plan for answering the query using the sources. Consequently, we do not have to build a new mediator for different queries or information sources. For example, the Nomenclator system incorporates multiple CCSO, X.500 and relational name servers. Source descriptions are given as equality selections on a *single* relation, and queries can only reference one relation.

The SIMS system [3] also describes information sources via class descriptions. The descriptions in the Information Manifold are richer than those in SIMS because they allow relations of arbitrary arity, and in particular allow us to express the fact that an information source contains a conjunctive view over world-view relations (either classes, roles or relations of higher arity). SIMS does not consider capability descriptions of the sources. SIMS, as well as the Internet Softbot [8] use Artificial Intelligence planning techniques for determining the relevant information sources and creating a query plan. These approaches do not provide the guarantees of ours, that is that we find *all* and *only* the relevant sources. Furthermore, it is not clear how their techniques will scale up to large numbers of information sources. In contrast, most of the algorithms we use are efficient, and the sources of complexity (e.g., containment checks) are well understood and are limited in practice. The advantage of general purpose planning techniques is the flexibility in dealing with unexpected problems during query evaluation [12].

Finally, there are several indexing systems for finding information on the World Wide Web (e.g., Harvest [5], Gloss [10], Yahoo, Lycos). However, all these systems are based on keyword searches on contents or annotations of documents, and are only able to retrieve documents that match these keywords. They cannot answer semantically meaningful queries that require considering the contents of the sources. The W3QS [13] is a system for specifying high-level queries over unstructured information sources. This system enables the user to specify in the query patterns of nodes on the web and properties of such nodes (that can be checked using existing Unix utilities). This is a very useful tool that enables a lot of otherwise manually done search to be done by a search engine, but it does not make use of contents of structured sources, and combine information from multiple sources.

8 Conclusion

We described the Information Manifold, a novel system that provides a uniform query interface to distributed structured information sources. The Information Manifold frees the user from having to interact with each information source separately, and to combine information from multiple sources. The techniques underlying the Information Manifold are applicable to sources on the WWW as well as other collections of information sources such as company-wide databases. The Information Manifold provides access to both sources on the WWW and to internal company databases that are available on line at AT&T Bell Laboratories. A key aspect of our system is a language for describing the contents and capabilities of the available information sources. This language enables expressing fine-grained distinctions between the contents of different information sources, thereby enabling us to prune the sources that are irrelevant to a given query. A novel aspect of our language is that it describes the capabilities of information sources in addition to their contents, which is crucial in order to interact with remote sources. Our contributions include practical algorithms for deciding which information sources are relevant to a query, and how to combine them in a way that adheres to the capabilities of the sources and exploits them when possible. Our language and algorithms have been useful in practice, allowing us to describe many existing information encountered. The end result is the first system that provides a database-like interface to wide variety of structured information sources on the WWW.

The world-view with which the user interacts is designed with respect to the set of information sources to which we expect to provide access. An important issue is designing tools to obtain descriptions of information sources easily. In [15] we describe one such fielded tool designed to obtain descriptions of name server sources. The tool is based on asking the administrators of the name servers to annotate example entries from their databases so that we can infer the content descriptions of the source from the annotations.

There are several important areas of research we are currently pursuing. First, we are considering how to extend our language for describing sources so that we will be able to infer that a source is relevant to a query with some degree of likelihood. For example, if we are searching for papers on database systems, and have access to a repository of papers on operating systems, we cannot completely ignore the repository, because we cannot state that these two fields are disjoint. However, we would like to access this repository only after we have accessed all other repositories that are closer to database systems. The second extension we are considering is extending our modeling language and algorithms to be able to deal partially with unstructured information sources. Doing so requires that we extend our language for describing capabilities of sources, which can be viewed as being extremely limited in the case of unstructured sources.

Acknowledgements

The authors thank Marie-Christine Rousset, Avi Silberschatz and Jeff Ullman for comments on earlier versions of this paper.

Appendix A

The following are some of the information sources available to the Information Manifold.

Source	**URL**
Glimpse CS Bibliography Server	http://glimpse.cs.arizona.edu:1994/bib/
AT&T 800 Directory	http://www.tollfree.att.net/cgi-bin/taos_mf.pl?unix
NYNEX Yellow pages	http://www.niyp.com/enterbusiness.html
Library of Congress Book Database	http://lcweb.loc.gov/z3950/mums.html
Powells Bookstore Database	http://www.technical.powells.portland.or.us/search.technical.html
Classifieds 2000: Cars	http://www.classifieds2000.com/cgi-cls/cls.exe?B1

San Jose Mercury Car Classifieds	http://www.sjmercury.com/class/trans/today
M.I.T. Stock Database	http://www.ai.mit.edu/stocks/
APL Stock Quote Server	http://qs.secap1.com/cgi-bin/qs/
The Internet Movie Database	http://www.msstate.edu/Movies/
Joan Ellis Movie Reviews	http://movie.infocom.net/
Telerama Movie Review Database	http://www.cinema.pgh.pa.us/movie/reviews
University of Wisconsin white pages	telnet to wisc.edu, port 105.

Appendix B

Proof of Theorem 5.8: The problem is in NP because we can simply guess an ordering, and check whether it satisfies the requirements of the information sources in polynomial time.

We show the NP-hardness by reducing the satisfiability problem of 3CNF formulas to our problem. Let Δ be a set of 3CNF formulas, with the propositional variables $p_1, \ldots, p_n$ and clauses $c_1, \ldots, c_m$. We construct a logical solution with $n + m$ subgoals. Each subgoal uses a different information source, and all subgoals have exactly one variable, X. For each of the first n subgoals, we have the following capability description: $(\emptyset, 0, 0, \{p_i^1(X)\}, \emptyset)$, $(\emptyset, 0, 0, \{p_i^2(X)\}, \emptyset)$ that is, there are no requirements on the input or selection arguments, but there are two possible outputs, either the p_i^1 or p_i^2 argument. Intuitively, the choice of output determines the truth value of the proposition p_i. The next m subgoals are one for each clause. Suppose the i'th clause is $\{L_1, L_2, L_3\}$, where each L_i is a literal. The $(n+i)$'th subgoal has one capability description. The input set S_{in} contains three elements: for $1 \leq j \leq 3$ it contains $p_i^1(X)$ if $L_i = p_i$ and $p_i^2(X)$ if $L_i = \neg p_i$. The minimum number of inputs is 1 and the maximal number is 3. There are no output or selection arguments.

Clearly, if Δ is satisfiable, there is a way to satisfy the capability requirements in the query. We choose the capabilities of the first n subgoals to produce exactly the satisfying assignment to the variables of Δ (i.e., $p_i^1(X)$ if p_i is assigned $True$, and $p_i^2(X)$ otherwise). The next m subgoals are satisfied, since each of the clauses in Δ are satisfied by the truth assignment. Similarly, if there is a way to satisfy the requirements of the information sources then we can build a satisfying assignment in a straightforward fashion. Finally, our reduction is polynomial because the query we produced is polynomial in the size of Δ. $\Box$

References

[1] Serge Abiteboul, Sophie Cluet, and Tova Milo. Querying and updating the file. In *Proceedings of the 19th VLDB Conference, Dublin, Ireland*, 1993.

[2] Rafi Ahmed, Phillippe De Smedt adn Weimin Du, William Kent, Mohammad A. Ketabchi, Witold A. Litwin, Abbas Rafii, and Ming-Chien Shan. The pegasus heterogeneous multi-database system. *IEEE Computer*, pages 19–26, December 1991.

[3] Yigal Arens, Chin Y. Chee, Chun-Nan Hsu, and Craig A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal on Intelligent and Cooperative Information Systems*, 1994.

[4] Alex Borgida, Ronald Brachman, Deborah McGuinness, and Lori Resnick. CLASSIC: A structural data model for objects. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pages 59–67, 1989.

[5] C. Mic Bowman, Peter B. Danzig, Udi Manber, and Michael F. Schwartz. Scalable internet resource discovery: Research problems and approaches. *CACM*, 37(8):98–107, August 1994.

[6] Surajit Chaudhuri, Ravi Krishnamurthy, Spyros Potamianos, and Kyuseok Shim. Optimizing queries with materialized views. In *Proceedings of International Conference on Data Engineering*, 1995.

[7] Sudarshan Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey Ullman, and Jennifer Widom. The TSIMMIS project: Integration of heterogenous information sources. In proceedings of IPSJ, Tokyo, Japan, October 1994.

[8] Oren Etzioni and Dan Weld. A softbot-based interface to the internet. *CACM*, 37(7):72–76, 1994.

[9] Douglas Fang, Joachim Hammer, and Dennis McLeod. The identification and resolution of semantic heterogeneity in multidatabase systems. In *Multidatabase Systems: An Advanced Solution for Global Information Sharing*, pages 52–60. IEEE Computer Society Press, Los Alamitos, California, 1994.

[10] Luis Gravano, Hector Garcia-Molina, and Anthony Tomasic. The effectiveness of gloss for the text database discovery problem. In *Proceedings of SIGMOD-94*, pages 126–137, 1994.

[11] A. R. Hurson, M. W. Bright, and S. H. Pakzad, editors. *Multidatabase Systems: An Advanced Solution for Global Information Sharing.* IEEE Computer Society Press, Los Alamitos, California, 1994.

[12] Craig A. Knoblock. Planning executing, sensing and replanning for information gathering. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 1995.

[13] David Konopnicki and Oded Shmueli. W3QS: A query system for the WWW. In *Proceedings of the 21st VLDB Conference, Zurich, Switzerland*, 1995.

[14] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, San Jose, CA*, 1995.

[15] Alon Y. Levy and Joann J. Ordille. Integrating internet information sources. In *Working Notes of the AAAI Fall Symposium on AI Applications in Knowledge Navigation*, 1994.

[16] Alon Y. Levy, Anand Rajaraman, and Jeffrey D. Ullman. Answering queries using limited external processors. AT&T Technical Memorandum, 1995.

[17] Alon Y. Levy and Marie-Christine Rousset. CARIN: a representation language integrating rules and description logics. In *Proceedings of the International Workshop on Description Logics, Roma, Italy*, 1995.

[18] Alon Y. Levy and Yehoshua Sagiv. Queries independent of updates. In *Proceedings of the 19th VLDB Conference, Dublin, Ireland*, pages 171–181, 1993.

[19] Alon Y. Levy, Divesh Srivastava, and Thomas Kirk. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems, Special Issue on Networked Information Discovery and Retrieval*, 5 (2), September 1995.

[20] Witold Litwin, Leo Mark, and Nick Roussopoulos. Interoperability of multiple autonomous databases. *ACM Computing Surveys*, 22 (3):267–293, 1990.

[21] K. A. Morris. An algorithm for ordering subgoals in NAIL! In *Proceedings of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 82–88, 1988.

[22] Joann J. Ordille. *Descriptive Name Services for Large Internets.* PhD thesis, University of Wisconsin, Madison, WI, 1993.

[23] Joann J. Ordille and Barton P. Miller. Distributed active catalogs and meta-data caching in descriptive name services. In *Proceedings of the 13th International IEEE Conference on Distributed Computing Systems, Pittsburgh, USA*, pages 120–129, 1993.

[24] Yannis Papakonstantinou, Ashish Gupta, Hector Garcia-Molina, and Jeffrey Ullman. A query translation scheme for rapid implementation of wrappers. In *In proceedings of the Conference on Deductive and Object Oriented Databases, DOOD-95, To appear*, 1995.

[25] Xiaolei Qian. Query folding. Technical Report SRI-CSL-95-09, Stanford Research Institute, Menlo Park, California, 1995.

[26] Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman. Answering queries using templates with binding patterns. In *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, San Jose, CA*, 1995.

[27] Anand Rajaraman and Jeffrey D. Ullman. Integrating information by outerjoins and full disjunctions. Submitted for publication, 1995. Available by anonymous ftp from db.stanford.edu as the file pub/rajaraman/1995/outerjoin.ps, 1995.

[28] V.S. Subrahmanian, S. Adali, A. Brink, R. Emery, J. Lu, A. Rajput, T. Rogers, R. Ross, and C. Ward. HERMES: A heterogeneous reasoning and mediator system. Technical report, University of Maryland, 1995.

[29] Jeffrey D. Ullman. *Principles of Database and Knowledge-base Systems, Volumes I, II.* Computer Science Press, Rockville MD, 1989.

[30] Gio Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, pages 38–49, 1992.

[31] Darrel Woelk, Paul Attie, Phil Cannata, Greg Meredith, Amit Seth, Munindar Sing, and Christine Tomlinson. Task scheduling using intertask dependencies in Carnot. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 491–494, 1993.

[32] H. Z. Yang and P. A. Larson. Query transformation for PSJ-queries. In *Proceedings of the 13th International VLDB Conference*, pages 245–254, 1987.