

Planning And Optimizing Semantic Information Requests Using Domain Modeling And Resource Characteristics (Technical Report)

Shuchi Patel and Amit Sheth
Large Scale Distributed Information Systems (LSDIS) Lab
Department of Computer Science, University of Georgia
Athens, GA 30602

<http://lsdis.cs.uga.edu>
Email: {shuchi, amit}@cs.uga.edu

Abstract

An information integration system that provides one-point access to data from a multitude of distributed, diverse and autonomous information sources (including web-based sources), needs to resolve the system, syntactic, structural and semantic heterogeneities between them. The next step would be to provide an environment that assists users in understanding and exploring domains of interest and relationships between them using the data from these sources.

We describe such an information integration system, InfoQuilt¹, that provides tools that help users analyze data retrieved from the sources, gain better understanding of the domains and different ways in which they are related to each other, explore possibilities of new relationships, and explore trends to support such hypothetical relationships, invalidate them or provide further insight into additional aspects about relationships that are already known to exist. InfoQuilt supports this by providing a framework to declaratively describe the domains of interest and their semantics, complex semantic relationships between them, sources available, their capabilities and limitations, and a powerful interface to specify information requests such that the system can “understand” them. We describe algorithms that use knowledge about domains, their relationships, and available information sources to efficiently create practical execution plans for information requests such that they can retrieve more comprehensive and precise results using the same set of available sources.

1. Introduction

The amount of “data” available has exploded immensely in the last few years. Enterprises double data every 90 days, but the internet is one of the main contributing factors for making it broadly accessible. In addition, we still have the traditional file systems, databases, *etc.* that are also potentially very useful sources of data. These can be viewed as autonomous repositories of data that were developed independently and hence may have different models for same domains. Formats of the data (especially on the web) also vary from the traditional text to audio, video, streaming audio/video and different formats within these categories as well. Most tools available to search information from web-based sources provide pure keyword based search that is not powerful enough to describe a user’s information need. For instance, how would you describe the following request to a search engine such that it knows precisely what you are asking for?

“Find information about all earthquakes that occurred after 1990 in India resulting in death of more than 1000 people.”

¹ One of the incarnations of the InfoQuilt system, as applied to the geographic information as part of the NSF Digital Library II initiative is the ADEPT-UGA system [Ade]

Also, the results have a low precision and recall. The condition of finding too much, and often irrelevant, data in response to any information need, whether using a search or browsing, is known as the problem of *information overload*.

Providing access to such diverse multiple sources via a common interface has been an interesting research problem called *information integration*. Sheth [She99] describes different types of heterogeneities that need to be dealt with and related issues for achieving interoperability. Use of metadata (data about data), especially domain specific metadata, to address this issue is interesting and has gone a long way. [SK98] presents several examples. However, merely providing a common interface for information integration only addresses certain types of heterogeneities between the sources. What makes it more interesting is the ability to semantically relate data to make sense out of it. Consider the following information request.

“Find all nuclear tests conducted by USSR or US after 1970 and find all earthquakes that could have occurred as a result of the tests.”

Apart from the constraint that the nuclear test should have been conducted by USSR or by US after 1970, the system needs to understand how a nuclear test can cause an earthquake. Most search engines today are not powerful enough to even understand the constraint “nuclear tests conducted by USSR or US after 1970”. Only structured databases can support it. Beyond that, understanding the relationship between a nuclear test and an earthquake requires understanding of semantics of the domains and the interaction between them. Structured databases that mainly focus on syntax and structure of data do not support this.

Very frequently, users make use of functions to post-process and analyze data (*e.g.* statistical analysis techniques). Of special interest are simulation programs that are used for forecasting, extrapolation, *etc.* Such functions and simulations can also be viewed as useful sources of information. Consider the following request.

“Forecast the urban growth pattern in the San Francisco bay area for the next 10 years in time steps of 1 year based on information about the roads, slopes and vegetation patterns in the area.”

Some of the important issues in an information integration system are:

- Modeling the domains of interest, including entities and relationships between them
- Resolving system, syntactic, structural and semantic heterogeneities between the sources [She99]
- A rich and powerful mechanism for specifying information requests (beyond traditional keyword queries or queries against structured database as in SQL) that can semantically describe the information need of user
- Scalability of the system is critical due to the potentially large number of data sources
- Efficient planning and optimization

InfoQuilt system aims at addressing several of these issues. Several other research efforts [AKS96, LRO96, MIKS00, CMH+94] have developed approaches to represent the semantics of domains and the characteristics of information sources to provide a common interface to query multiple distributed heterogeneous sources. In addition to these, InfoQuilt also provides frameworks to specify complex relationships between domains and complex information requests that capture the semantics of the user’s information need. Also, the vision of the InfoQuilt system is different from most other information integration systems. Our goal is to develop a system that provides an environment that allows users to analyze the data available from a multitude of diverse autonomous sources (including web-based sources), gain better understanding of the domains and their interactions as well as determine information leading to decision making via

analysis of potential semantic relationships between the data provided by different sources. This support for knowledge discovery is a novel feature of InfoQuilt.

This paper focuses on planning and optimization of information requests, which in the context of InfoQuilt, are known as Information Scapes or IScapes. The algorithms described in the paper have been implemented in InfoQuilt system and the examples presented have been tested. Given an IScape, the system needs to create an execution plan consisting of queries against relevant information sources and steps indicating how the information retrieved should be integrated. Creation of high-quality near-optimal plans is crucial to a system like this due to the following main reason. The sources available are highly diverse and web sources constitute a large portion of them. Being able to retrieve data from these sources over the network using wrappers is slow relative to a source that is a database. The query execution is therefore relatively slow. However, the use of certain semantic optimizations allows us to choose one execution plan over another one so that the IScape execution is faster. The following are key features of our algorithm:

- Efficient source selection – excluding sources that will not provide useful results or will provide redundant information
- Ability to use sources in conjunction to retrieve more complete information (e.g. missing attributes) from the same set of available resources
- Generation of *executable* plans which respect query capability limitations of resources
- Use of sources in conjunction to be able to use resources with query capability limitations
- Integration of information retrieved from the sources selected, including relationship evaluation and post-processing (e.g. evaluating functions and running simulations)

The remainder of the paper is organized as follows. Section 2 briefly describes how the system knowledge, referred to as its knowledge base, is represented in the system, points out features of the system that make efficient query planning and semantic optimization of the execution plans possible, describes how IScapes are represented in the system, and shows how InfoQuilt supports an environment for knowledge discovery. Section 3 describes planning of IScapes and optimization of the execution plans. Section 4 discusses the runtime architecture of the InfoQuilt system and how execution of an IScape proceeds. We compare our work with other related work in this area in section 5. Section 6 presents our conclusions and directions for future work.

2. Background

In a typical scenario in using the InfoQuilt system, an administrator would identify domains of interest and describe them to the system. Section 2.1 describes how a domain is represented in InfoQuilt. Next, he models complex relationships between the domains. Section 2.2 describes how these inter-ontological relationships are created. He then identifies useful data sources for those domains, creates their wrappers, also known as *Resource Agents*, and specifies the characteristics and limitations of each of them. Section 2.3 describes information source modeling. One of the distinguishing features of InfoQuilt system is its ability to use functions for mapping data values to resolve syntactic and semantic heterogeneities, evaluate complex criteria that cannot be expressed using only relational and logical operators, and post-process data available from the sources. We describe these functions in section 2.4. The knowledge about ontologies, relationships, resources and functions forms the *knowledge base* of the system. We provide a graphical toolkit called the *Knowledge Builder* that the administrator can use to easily create and update the knowledge base. Once the knowledge base is constructed, users can create IScapes. Section 2.5 describes how an information request can be described to the system as an IScape. Finally, Section 2.6 shows how InfoQuilt can help users discover knowledge. A detailed discussion on the knowledge base, Knowledge Builder, IScapes, and the support for learning can be found in [TPS01].

2.1 Domain Modeling

Modeling the domains of interest facilitates a better understanding of the domains and related concepts. One form of heterogeneity between different sources that the query answering system needs to resolve is the difference in the views of the domain adopted by them. InfoQuilt uses ontologies to support this. An *ontology* is meant to capture useful semantics of the domain such as the set of terms and concepts of interest, their meanings, relationships between them, and characteristics of the domain. The domain characteristics are described as a set of domain rules and functional dependencies and help in efficient source selection and semantic optimization of execution plans, as described in section 3.2. We will use the terms “ontology” and “domain” interchangeably in the rest of the paper unless explicitly specified.

Example 1: Consider the domain of nuclear tests. Some interesting attributes could be test site, explosive yield, seismic body wave magnitude, type of the test (underground, atmospheric, *etc.*), date the test was conducted, the name of the country that conducted it, and latitude and longitude of the test site. Also, suppose we knew that the seismic body wave magnitude of a nuclear test is in the range 0 to 10^2 . We can represent this as domain rules. Also, we know that given a specific test site, the latitude and longitude are the same. We can represent this as a functional dependency (FD). The ontology will therefore be:

```
NuclearTest( testSite, explosiveYield, waveMagnitude, testType,  
            eventDate, conductedBy, latitude, longitude,  
            waveMagnitude > 0, waveMagnitude < 10,  
            testSite -> latitude longitude );
```

We will use the above notation to represent ontologies throughout the paper.

2.2 Inter-ontological relationships

Real world entities are related to each other in various ways. These relationships could be simple ones like “*is-a*”, for example, a car “*is a*” vehicle, or they could be very complex, for example, a nuclear test “*causes*” an earthquake or an earthquake “*affects*” environment. A distinguishing feature of the InfoQuilt system is its ability to model simple as well as complex relationships between entities as a part of its knowledge base.

Example 2: Consider the relationship between a nuclear test and an earthquake. We use the `NuclearTest` ontology from example 1. Suppose we model earthquake as follows:

```
Earthquake( eventDate, description, region, magnitude, latitude,  
            longitude, numberOfDeaths, damagePhoto, magnitude > 02 );
```

We can say that some nuclear test could have “*caused*” an earthquake if the earthquake occurred “*some time after*” the nuclear test was conducted and “*in a nearby region*”. Here, “*some time after*” and “*in nearby region*” are specialized operators, possible due to the ability to support user-defined functions, as described in section 2.4. For now, assume that there are two functions called `dateDifference` and `distance` available to the system. The function `dateDifference` takes two dates as arguments and returns number of days from `date1` to `date2` and `distance` takes the

² The body wave magnitude does not have upper or lower bounds. However, all nuclear tests as well as earthquakes measured to date had a magnitude less than 10 and those with magnitudes less than 0 are very small. We use these bounds solely to demonstrate how the domain rules can be modeled and used.

latitudes and longitudes of two places and calculates the distance between them in miles. Given that we can use these functions, we can represent the relationship as follows:

```
NuclearTest Causes Earthquake
  <= dateDifference( NuclearTest.eventDate, Earthquake.eventDate ) < 30
    AND distance( NuclearTest.latitude, NuclearTest.longitude,
      Earthquake.latitude, Earthquake.longitude ) < 10000
```

The values 30 and 10000 are arbitrary. In fact, a user can try different values to analyze the data. This is a useful feature for supporting knowledge discovery, as discussed in section 2.6.

2.3 Information Source Modeling

The system can potentially have access to multiple diverse sources for each ontology, which requires it to resolve the heterogeneities between them to be able to integrate the data retrieved from them. This is done in part by creating an integrated view for each domain (ontology). The sources are then described to the system (source views) in terms of this integrated view. This approach, known as source-centric approach, has the advantage that the sources are completely independent of each other. Hence, they can be easily added, removed and their specifications could be updated without having to change the integrated views. Also, it allows us to model the domain independently of the source views. The other approach is the query-centric approach, where the integrated views are defined in terms of the source views. Adding, removing or updating sources to a system is difficult as it requires redefining the integrated views.

The information source descriptions used in InfoQuilt describe the characteristics, query capabilities and limitations of the sources. They help the planner to select the most appropriate sources for answering an IScape and perform certain optimizations on the execution plan, as described in section 3.2. They are described to the system by specifying the following:

- **Resource rules**

Resource or Data Characteristic (DC) rules, are similar to domain rules. They describe characteristics that always hold true for data retrieved from resource.

Example 3: Consider the ontology `Earthquake` used earlier. Consider a resource that lists only earthquakes that occurred in California from January 1, 1980. We can specify this by the rules:

```
region = "California"
eventDate >= "January 1, 1980"
```

- **Local Completeness**

These rules are similar to resource rules but are interpreted differently. They describe a subset of the domain for which the resource is known to have complete information. In other words, the resource has data for *each* entity in the subset of the domain described by local completeness (LC) rule.

Example 4: Consider the ontology `DirectFlight`, which models direct flights from one city to another within US. Suppose the system has access to the database at the Atlanta Airport. We know that this source will provide data on *every* flight departing from and arriving at Atlanta. We can describe this as the following local completeness rules:

```
toCity = "Atlanta"
fromCity = "Atlanta"
```

Here, `toCity` and `fromCity` represent the departure and destination cities respectively.

- **Binding patterns**

Often we come across web sites that allow the user to search their database via HTML forms using CGI scripts, servlets, etc. These scripts are generally programmed to not

answer queries such as “*get everything from the database*”. They require the user to provide values for certain parameters. They can therefore not be used when the query answering system is not able to provide values for these parameters. It is crucial for the system to be aware of these limitations of the query capability of the source. These are represented as binding patterns, which are sets of attributes for which the system should be able to provide values to be able to query the resource.

Example 5: Consider the `DirectFlight` ontology again. Suppose we use the AirTran Airway web site (www.airtran.com) as a source. The form on the web page requires the user to specify source, destination, dates of travel, *etc.* We represent this as the following binding pattern:

```
[toCity, toState, fromCity, fromState,  
  departureMonth, departureDay]
```

A web site could allow several optional combinations of attributes too. InfoQuilt therefore allows the user to specify multiple binding patterns for a resource. The planning algorithm selects one that is appropriate to answer the IScape.

2.4 Functions and simulations

As seen in example 2, functions are very useful. The example demonstrates use of two simple functions to create specialized operators used for defining an inter-ontological relationship between `NuclearTest` and `Earthquake`. Functions are very useful to define complex inter-ontological relationships that cannot be expressed by expressions involving only relational and logical operators.

Another use of functions is to post-process information retrieved from the sources. For example, use of various statistical analysis techniques to analyze data is very frequent. A type of post-processing operation of particular interest is simulation. For example, researchers in the field of Geographical Information Systems (GIS) use them to extrapolate patterns of urban growth, deforestation, global warming, *etc.* based on models. These are available as independent programs. They can be used over the data retrieved from available resources (assuming that the system has access to resources that provide data that the simulation can work with) and the result of the simulation could be presented to the user.

InfoQuilt views such functions and simulations as important sources of related information. It maintains information about them in a “*Function Store*”, a component of its knowledge base. The Function Store maintains a declarative description of the functions, which includes information about how the system can invoke them, what are the parameters needed and their formats, and the return value types. The administrator is responsible for providing an implementation for them.

2.5 Information Scapes (IScapes)

The ability to specify complex information needs is another distinguishing feature of the InfoQuilt system. Such information requests are referred to as *Information Scapes* or *IScapes*. Since they are specified in terms of the components of the knowledge base of the system, they are better able to model the semantics of a user request and the system is able to “understand” them.

Example 6: Consider the following IScape (described here as text).

“Find all nuclear tests conducted by USSR or US after 1970 and find any information about any earthquakes that could have potentially occurred due to these tests.”

In addition to the constraints that are obvious, the system needs to understand what the user means when he says “*earthquakes that could have potentially occurred due to these tests*”. This is where knowledge about the inter-ontological relationship “nuclear test causes earthquakes” available in the knowledge base of the system is useful.

In addition to modeling the semantics of the information request, IScape also abstracts the user from the characteristics, structure, access mechanisms, *etc.* of the actual data or information sources available to the system (which will eventually be used to answer the IScape), their heterogeneities, and the steps to integrate the data available from them, including any relationships and functions that may have to be evaluated or simulations that may have to be executed.

An IScape includes a set of ontologies, a set of inter-ontological relationships that indicate the particular inter-domain interactions of interest in the context of the information need, a constraint that describes the subset of information that the user is interested in (*e.g.* tests conducted after 1970), a runtime configurable constraint, how the results should be grouped, a constraint on the groups, if any (similar to the HAVING clause in SQL), and a projection list listing the information that needs to be returned as a part of the final result. The runtime configurable constraint in an IScape is similar to the other constraint except that it is created at the time of actually executing the IScape. This eliminates the need of creating new IScapes that are slightly different from existing ones. It is a useful functionality that helps support knowledge discovery, as described in section 2.6. We provide a graphical toolkit, known as *IScape Builder*, that allows users to create IScapes easily in a step-by-step manner, execute them and further analyze them. One such analysis tool provided is a chart creator. See [TPS01] for details.

2.6 Support For Knowledge Discovery

One of the most important goals of the InfoQuilt system is to support an environment to help users discover knowledge. This is one of its novel and distinguishing features. It is realized by providing an environment that allows users to access data available from a multitude of diverse, autonomous, and distributed resources (including web-based resources) and tools that help them to analyze the data, gain a better understanding of the domains and the different ways in which they are related to each other, explore possibilities of new relationships, and explore trends to support such potential relationships, invalidate them or provide further insight into additional aspects about relationships that are already known to exist. Availability of a large number of resources ranging from traditional databases to web sources for different domains enables a one-point access to a vast amount of information. The system, and not the user, is responsible for dealing with the heterogeneities between the sources and integrating the data. In addition, it provides a powerful means of expressing information requests (IScapes) that allow users to also request complex post-processing of the data using functions and simulations. It provides tools to mine and analyze the data retrieved, study new possible inter-domain interactions and try to prove or disprove them. A detailed discussion on the support for knowledge discovery appears in [TPS01]. The following example shows how a user can use InfoQuilt to study a potential relationship.

Example 7³: Several researchers in the past have expressed their concern over nuclear tests as one of the causes of earthquakes. This issue, although studied in the past, is still a hypothesis. We want to explore this potential relationship. We assume that the system has access to resources that

³ The information presented in this example is based on the findings of Gary T. Whiteford [Whi89]. It has been recognized as the most exhaustive study yet of the correlation between nuclear testing and earthquakes.

provide sufficient information for the analysis. If nuclear tests do cause earthquakes, then we should be able to see an increase in the number of earthquakes that have occurred after the nuclear testing started in 1950. We check the trend of the number of earthquakes that have occurred before and after 1950, considering only the earthquakes with intensity 5.8 or higher on Richter scale as some earthquakes below that intensity would have passed unrecorded in the earlier part of the century due to limitations of the measuring devices available then [Whi89].

IScape 1: *“Find the total number of earthquakes with a magnitude 5.8 or higher on the Richter scale per year starting from year 1900.”*

We then use the IScape Builder to create a chart, which reveals that there was a sudden increase in the number of earthquakes since 1950. We modify the IScape to quantify this rise approximately.

IScape 2: *“Find the average number of earthquakes with a magnitude 5.8 or higher on the Richter scale per year for the period 1900-1949 and for the period 1950-present.”*

We see that in the period 1900-1949, the average rate of earthquakes was 68 per year and that for 1950-present⁴ was 127 per year, that is it almost doubled [Whi89]. Next, we try to analyze the same data grouping the earthquakes by their magnitudes.

IScape 3: *“For each group of earthquakes with magnitudes in the ranges 5.8-6, 6-7, 7-8, 8-9, and magnitudes higher than 9 on the Richter scale per year starting from year 1900, find the average number of earthquakes.”*

The results show that the average number of earthquakes with magnitude greater than 7 have remained constant over the century (about 19) [Whi89]. So we can deduce that the earthquakes caused by nuclear tests are usually of magnitudes less than 7. We can then try to explore the data at a finer level of granularity by trying to look for specific instances of earthquakes that occurred within a certain period of time after a nuclear test was conducted in a near by region.

IScape 4: *“Find nuclear tests conducted after January 1, 1950 and find any earthquakes that occurred not later than a certain number of days after the test and such that its epicenter was located no farther than a certain distance from the test site.”*

Note the use of “not later than a certain number of days” and “no farther than a certain distance”. The time period and distance are defined as runtime configurable parameters in the IScape. The user can hence supply different values for them and execute the IScape repeatedly to analyze the data for different values. Also, note the use of functions as user-defined operators to calculate the difference in date (`dateDifference`) and the distance between the epicenter of the earthquake and the nuclear test site (`distance`).

3. Planning and Optimization

We now describe the creation of execution plans for IScapes and semantic optimizations possible due to the detailed knowledge base that InfoQuilt maintains. As we mentioned earlier, the information sources are described in terms of the ontologies defined in the knowledge base. Hence, information source descriptions are similar to views defined on them. The problem of

⁴ The period of 1950-present implies the period 1950-1989, since the data presented here was published by Gary T. Whiteford in 1989.

creating an execution plan is thus closely related to the problem of answering queries using views [YL87, LMSS95, CKPS95]. It was shown to be NP-complete in [LMSS95]. The algorithm we describe here uses domain and source characteristics and is therefore tractable and efficient.

3.1 Operator nodes used in the plan

Following are the different types of operator nodes used in the plans. We use the term *result set* to imply a set of results (both intermediate and final).

- **Resource Access Node**
This node represents access of a physical resource to retrieve data from it. As discussed in section 2.3, some resources have binding patterns associated with them. The node therefore also specifies values for the attributes in the binding pattern, if any. It also lists which attributes need to be retrieved from the resource.
- **Select Node**
This node is similar to the select relational operator in a traditional Relational DBMS (RDBMS). It takes a result set and evaluates a specified condition over it. It eliminates the results for which the condition evaluates to false.
- **Join Node**
The Join Node is also similar to the join relational operator in an RDBMS. It takes two result sets and joins them using the specified join condition.
- **Binding Pattern (BP) Supplier Node**
This node is used to supply values for a binding pattern on some resource. Values for a binding pattern may need to be supplied from another resource. This node is responsible for retrieving these values and supplying them to the resource that needs them.
- **Union Node**
This node is also similar to the union relational operator in a RDBMS. The difference, however, is that it is not a binary operator. It can compute the union of one or more result sets, eliminating duplicates.
- **Function Evaluator Node**
This node evaluates the specified functions for entities in a result set and adds the results as new columns to the result set⁵.
- **Relationship Evaluator Node**
This node evaluates complex relationships defined in the knowledge base of the system.
- **Cartesian Product Node**
This node is similar to the cartesian product relational operator in an RDBMS.
- **Group By Node**
This node groups the entities in a result set by the specified list of attributes and computes aggregates required by the IScape. The aggregations supported are COUNT, MIN, MAX, AVG and SUM.
- **Project Node**
This node is similar to the project relation operator. It projects a result set over a specified set of attributes.

3.2 Generation of execution plan for IScapes

We now describe how the execution plan for an IScape is generated. We use two IScapes and show how their execution plans are generated step-by-step as we explain the algorithm.

⁵ We use Java's reflection package to invoke functions, pass arguments and retrieve results back.

IScape 1

“Find all nuclear tests conducted by USSR after January 1, 1980 with seismic body wave magnitude > 5 and find all earthquakes that could have been caused due to these tests.”

The ontologies and relationships used are:

```
NuclearTest( testSite, explosiveYield, waveMagnitude, testType,
             eventDate, conductedBy, latitude, longitude,
             waveMagnitude > 0, waveMagnitude < 10,
             testSite -> latitude longitude );

Earthquake( eventDate, description, region, magnitude, latitude,
             longitude, numberOfDeaths, damagePhoto, magnitude > 0 );

NuclearTest Causes Earthquake
<= dateDifference( NuclearTest.eventDate, Earthquake.eventDate ) < 30
  AND distance( NuclearTest.latitude, NuclearTest.longitude,
               Earthquake.latitude, Earthquake.longitude ) < 10000
```

To represent information sources, we use a notation similar to the one we use for ontologies. To resolve the ambiguity between resource rules and local completeness rules, we prepend [dc] or [lc] to distinguish them. Following are the specifications of the information sources available to the system.

```
NuclearTestsDB( testSite, explosiveYield, waveMagnitude,
                testType, eventDate, conductedBy,
                [dc] waveMagnitude > 3, [dc] eventDate > "January 1, 1985" );

NuclearTestSites( testSite, latitude, longitude );

SignificantEarthquakesDB( eventDate, description, region, magnitude,
                           latitude, longitude, numberOfDeaths, damagePhoto,
                           [dc] eventDate > "January 1, 1970" );
```

NuclearTestsDB is a database of nuclear tests of seismic body wave magnitude greater than 3 conducted after January 1, 1985. NuclearTestSites contains exact locations of nuclear test sites in terms of their latitudes and longitudes. These two sources provide data for the ontology NuclearTest. A point to note here is that although intuitively NuclearTestSites does not provide any information about any nuclear tests conducted, it can still be considered to provide information for that ontology. The resource SignificantEarthquakesDB is a database of significant earthquakes around the world occurring after January 1, 1970.

IScape 2

“Find all direct flights from Atlanta, GA to Boston, MA for February 12, 2001 and show the weather in the destination city on that day.”

The ontologies used are:

```
DirectFlight( airlineCompany, airlineLogo, flightNo, aircraft, fromCity,
              fromState, toCity, toState, departureDate, fare, meals,
              departureTime, arrivalTime, airlineCompany -> airlineLogo );

DailyWeather( date, city, state, description, icon, hiTemp, loTemp );
```

There are no relationships defined between the two ontologies. The sources used are:

```
YahooTravel( airlineCompany, flightNo, aircraft, fromCity, fromState, toCity,
             toState, departureDate, meals, departureTime, arrivalTime,
             [fromCity, fromState, toCity, toState, departureDate] );

AirTranAirways( airlineCompany, flightNo, fromCity, fromState, toCity, toState,
                departureDate, fare, departureTime, arrivalTime,
                [dc] airlineCompany = "AirTran Airways",
                [lc] airlineCompany = "AirTran Airways",
                [fromCity, fromState, toCity, toState, departureDate]);

AirlineLogos( airlineCompany, airlineLogo );

WeatherChannel( date, city, state, description, icon, hiTemp, loTemp,
                [city, state] );
```

The first three resources provide information for `DirectFlight` and `WeatherChannel` provides information for `DailyWeather`. `YahooTravel` and `AirTranAirways` retrieve information from the Yahoo Travel (<http://travel.yahoo.com>) and AirTran Airways (<http://www.airtran.com>) web sites respectively. Resource `AirTranAirways` has a resource rule that the `airlineCompany` for all flights retrieved from `AirTranAirways` will be "AirTran Airways". Also, the resource is locally complete for *all* flights operated by AirTran Airways. `AirlineLogos` is some web site that keeps a listing of all airline companies operating in the US. In the context of the `DirectFlight` ontology, this source can provide two attributes, `airlineCompany` and `airlineLogo`.

We now describe the steps in generating execution plans for IScapes and show how they apply to the two IScape scenarios introduced above. In addition to the optimizations like pushing selects and projects down, etc. used in relational databases, the ability to capture the semantics of the domains and the characteristics of the resources allows us to perform several semantic optimizations. We also describe how these are possible using the two example IScapes.

Step 1 - Check Semantic Correctness

The first step is to check if the IScape is semantically correct by comparing the IScape constraint against the domain rules defined on the ontologies involved.

IScape 1: The constraint in the IScape is:

```
NuclearTest.waveMagnitude > 5 AND NuclearTest.conductedBy = "USSR" AND
NuclearTest.eventDate > "January 1, 1980" AND
dateDifference( NuclearTest.eventDate, Earthquake.eventDate ) < 30 AND
distance(NuclearTest.latitude, NuclearTest.longitude,
         Earthquake.latitude, Earthquake.longitude ) < 10000
```

As seen above, the system maps any relationships used in the IScape to corresponding constraints. Comparing the constraint with the domain rules defined on `NuclearTest` and `Earthquake`, we see that the IScape is semantically correct.

Assume that the IScape enquired about nuclear tests with a body wave magnitude > 12 instead of 5. Using the domain rule "waveMagnitude < 10", the planner can conclude that the IScape is semantically incorrect as the body wave magnitude of a nuclear test can never be higher than 10. The system stops processing the IScape and no execution plan is created. This semantic optimization is possible because the domain rules could capture the semantics about the body wave magnitude of a nuclear test. This particular aspect of our query planning and optimization has been supported by various systems that support semantic integrity constraints.

IScape 2: The constraint for this IScape is:

```
DirectFlight.fromCity = "Atlanta" AND DirectFlight.fromState = "GA" AND
DirectFlight.toCity = "Boston" AND DirectFlight.toState = "MA" AND
DirectFlight.departureDate = "February 12, 2001" AND
DailyWeather.city = DirectFlight.toCity AND
DailyWeather.state = DirectFlight.toState AND
DailyWeather.date = DirectFlight.departureDate
```

After comparing it with the domain rules of `DirectFlight` and `DailyWeather`, we conclude that this IScape is also semantically correct.

Step 2 - Source Selection

The next step is to select the sources that should be used to answer the IScape. This is the most important phase of planning in any information integration system. The IScape is specified in terms of the knowledge base components. It is the responsibility of the planner to decide which resources should actually be used to answer the IScape. It uses the domain rules and functional dependencies for ontologies along with the data characteristic and local completeness rules for the resources and their query capability limitations in terms of the binding patterns to do this.

The planner selects sources for the IScape by considering one ontology at a time and applying the following rules:

- **Locally Complete Sources**

First it makes use of the local completeness rules of the resources. If there exists a resource that is locally complete for some subset A of the domain of the ontology such that the part of the IScape's result that comes from that ontology is a subset of A, it can use only that resource. Other resources need not be used (unless the selected resource has some attributes missing or has a binding pattern) since the local completeness condition implies that using any additional sources will not provide any extra information.

A semantic optimization that is performed while using this rule is that the resource should satisfy the following criterion:

- The DC rules of the resource should not falsify the IScape constraint. In other words, the subset of the domain of the ontology available from the resource and the part of the IScape's result that comes from that ontology should not be disjoint sets. If so, we can again conclude that the IScape is semantically incorrect.

IScape 1: No resources in this example are locally complete. So, we cannot apply this rule.

IScape 2: The resource `AirTranAirways` is locally complete for flights with `airlineCompany = "AirTran Airways"`. However, the IScape we have is querying flights operated by all airline companies. So, we cannot apply this rule. But consider the following temporary modification to the IScape assuming the same ontologies and sources:

"Find all direct flights from Atlanta, GA to Boston, MA for February 12, 2001 operated by AirTran Airways and show the weather in the destination city on that day."

In this case, it can use the local completeness information about `AirTranAirways` and select only that resource. Other resources need not be used. (But as we will see later, we will still need to use the `AirlineLogos` resource since `AirTranAirways` does not provide the attribute `airlineLogo`).

- **Non Locally Complete Sources**

If a locally complete source could not be found, then the planner cannot be sure that all possible answers to the IScape can be found using the available sources. However, we would want to retrieve as much information as possible from them. It therefore considers *all* the resources that it can use. Again, it can make use of resource rules to prune sources that will not return any useful results. The following condition applies:

- The DC rules of a resource should not falsify the IScape's constraint. If they do, it will not return any useful results for this IScape as they will get filtered by the constraint. This condition performs semantic optimization.

IScape 1: For the ontology `NuclearTest`, there are two resources. It selects both and applies the condition to them. No resources are eliminated as their resource rules do not falsify the IScape's constraint. Similarly, `SignificantEarthquakesDB`, the only available resource for Earthquake ontology, is selected. The resource rule on it does not falsify the IScape's constraint.

Assume for a moment that instead of earthquakes that occurred *after* January 1, 1970, this resource provided information about earthquakes that occurred *before* January 1, 1970. In this case, the resource rule would falsify the IScape's constraint. So the planner will not find any resource to supply information about earthquakes. If such a situation occurs, it throws an exception indicating that it could not find sufficient resources to answer the IScape.

IScape 2: For the ontology `DirectFlight`, there are three resources. The planner selects all of them after applying the condition. Similarly, for the ontology `DailyWeather`, we select the resource `WeatherChannel`.

- **Binding Patterns**

For the resource(s) selected (by either of the first two rules), the planner needs to ensure that their query capability limitations (binding patterns) are respected. If a resource has binding pattern(s) associated with it, the plan needs to specify how the values for the binding pattern attributes will be supplied. There are three possible ways to do this.

- The values can be supplied from the IScape constraint directly.
- The values for binding pattern attributes can also come from attributes in other ontologies.
- If either of the above two cases are not possible, then we can use some other resource of the ontology as an associate resource with this one, just to supply values for its binding pattern attributes. The associate resource is selected as specified by the next rule. We will refer to the resource with the binding pattern as the primary resource.

IScape 1: None of the resources selected have binding patterns. Hence, this rule does not apply.

IScape 2: Some selected resources have binding patterns (BP) on them. Consider the resource `YahooTravel`. It has the binding pattern `[fromCity, fromState, toCity, toState, departureDate]`. The values for the attributes in this BP can all be supplied from the query constraint itself since the IScape supplies their values:

```
[fromCity: ("Atlanta"), fromState: ("GA"), toCity: ("Boston"),  
toState: ("MA"), departureDate: ("February 12, 2001")]
```

AirTranAirways has the same BP and values for them can be supplied in the same way as for YahooTravel. WeatherChannel has the BP [city, state]. One way to supply the values for this BP is by realizing that the IScape constraint can be rewritten as follows:

```
DirectFlight.fromCity = "Atlanta" AND DirectFlight.fromState = "GA" AND  
DirectFlight.toCity = "Boston" AND DirectFlight.toState = "MA" AND  
DirectFlight.departureDate = "February 12, 2001" AND  
DailyWeather.city = "Boston" AND DailyWeather.state = "MA" AND  
DailyWeather.date = "February 12, 2001"
```

The values can then be supplied from the constraint. However, to show how values can be supplied from attributes of other ontologies, we leave the constraint as before and see that the values can be supplied from attributes `toCity` and `toState` of `DirectFlight` ontology. At this stage, the planner just makes a note of this fact and when the resources for all the ontologies are selected, it updates the plan to associate the result set for that ontology with this resource so that binding pattern values can be supplied. Note that since there could be multiple sources selected for the other ontology, the plan uses the union of the result sets retrieved from all the resources (of that ontology) that do not themselves need BP values from other ontologies (to avoid deadlocks). If however, no such resource could be found for the other ontology, then this resource cannot be used.

- **Associate resource to supply values for binding patterns**

As mentioned in the previous rule, if values for all the binding pattern attributes on a resource cannot be provided from the query constraint or some attributes of other ontologies used in the IScape or the combination of the two, then an *associate resource* can be used to do so. The associate resource is queried to retrieve all possible values for a certain set of attributes, which can then be supplied as values for the binding pattern attributes on some other resource of the same ontology. The associate resource selected must satisfy the following criteria:

- It should supply at least all attributes in the binding pattern except those whose values can be supplied from query constraint or attributes of other ontologies.
- If the associate resource itself has a binding pattern, it should be supplied from the IScape constraint. We do not consider transitive use of another resource to supply for binding patterns on an associate resource. This is a performance tradeoff, as it will slow down the IScape processing considerably. This is one reason why our algorithm does not guarantee maximally contained plans [Dus97].
- If there are resource rules on the associate resource involving only the attributes in the first criteria, they should not falsify the IScape's constraint. If they do, then the results retrieved from the primary source using these values for binding pattern attributes will eventually get filtered because of the constraint. (A semantic optimization)

For both example IScapes, the binding patterns on all the resources can be supplied using the IScape's constraint or from attributes of another ontology. Hence this rule does not apply.

- **Associate resource to supply values for missing attributes**

The resources selected should be able to provide all the attributes that the IScape needs. However, it is very common to come across resources that do not. If a resource has one or more attributes missing, the planner can make use of the functional dependencies

defined for the ontology, involving all the missing attributes to see if it can couple it with some associate resource to retrieve values for those attributes for as many entities as possible. This is done by equating the values of attributes appearing on the LHS of the FD to join the data retrieved from the main resource with that from the associated resource. Without the knowledge about the FD, it would not have been possible to form a plan that can thus deduce more information about entities using multiple resources in conjunction. A point to note is that the attributes are equated using a default equality computing function defined for it in the function store or an exact match if no function was defined. The use of functions is necessary as it is highly unlikely that all sources will have exact same values for the attributes (syntactic heterogeneity) [Gun00]. For example, a nuclear test site available from one source could be “Nevada Test Site, Nevada, USA” and that from another source could be “Nevada Test Site, NV, USA”. The two are semantically equal but syntactically unequal.

However, if there is a resource that needs a binding pattern and also has some attributes missing, it is used only if values for the binding pattern attributes can be supplied from the query constraint. If not, then it would need two associate resources, one to supply the binding pattern values and the other to retrieve values for the missing attributes. This would make the processing a bit slower. Also, we use a functional dependency only if *all* the missing attributes appear in the right hand side (RHS). In general, we can make use of multiple functional dependencies each for a certain subset of missing attributes. But this would imply a possible use of multiple resources as associate resources. The functional dependency (FD) selected should satisfy the following criteria:

- All the missing attributes should be in the RHS of the FD
- All attributes in the LHS of the FD should be available from the primary resource.

The associate resource selected should satisfy the following constraints:

- It should provide all attributes in the LHS of the FD and all attributes missing from the primary resource. It does not need to provide *all* attributes in RHS of the FD.
- If the associate resource has a binding pattern, the IScape processing system should be able to supply values for its attributes using values for those attributes from the primary resource. So, effectively, the primary resource acts as an associate resource to supply values for the binding pattern without using an extra resource.
- Resource rules on the associate resource involving only attributes appearing in the LHS of the FD and missing attributes should not falsify the IScape constraint. If they do, it implies that the resource will not be able to return any useful results since they would get filtered by the constraint. (A semantic optimization)
- Resource rules on the primary resource involving only attributes appearing in the LHS of the FD should not falsify any resource rule on the associate resource. If they do, then it implies that the associate resource will not return any useful results since it does not have any data with the set of values for the LHS attributes (the attributes appearing in the join condition) available from the primary resource. (A semantic optimization.)

If a FD or a resource to associate with could not be found, then the resource with the missing attributes cannot be used.

IScape 1: The attributes that the IScape uses (including the attributes that we project on and those needed to evaluate relationships and constraints) are:

```
( NuclearTest.testSite, NuclearTest.explosiveYield,
  NuclearTest.waveMagnitude, NuclearTest.testType, NuclearTest.eventDate,
  NuclearTest.conductedBy, NuclearTest.latitude, NuclearTest.longitude,
  Earthquake.eventDate, Earthquake.description, Earthquake.region,
  Earthquake.magnitude, Earthquake.latitude, Earthquake.longitude,
  Earthquake.numberOfDeaths, Earthquake.damagePhoto )
```

NuclearTestsDB has two missing attributes – latitude and longitude. Applying the criteria listed above for selecting an FD, the planner concludes that the FD “testSite -> latitude longitude” can be used to retrieve their values. NuclearTestSites qualifies as an associate resource by the criteria above. The function testSiteEquals from the function store is used to equate the values of testSite from the two resources. Similarly, for NuclearTestSites, all attributes except testSite, latitude and longitude are missing. However, there is no FD that can be used to retrieve values for these missing attributes using an associate resource. So, we eliminate it as a primary source.

IScape 2: The attributes that this IScape uses are:

```
( DirectFlight.airlineCompany, DirectFlight.airlineLogo,
  DirectFlight.flightNo, DirectFlight.aircraft, DirectFlight.fromCity,
  DirectFlight.fromState, DirectFlight.toCity, DirectFlight.toState,
  DirectFlight.departureDate, DirectFlight.departureTime,
  DirectFlight.arrivalTime, DailyWeather.date, DailyWeather.city,
  DailyWeather.state, DailyWeather.description, DailyWeather.icon,
  DailyWeather.hiTemp, DailyWeather.loTemp )
```

YahooTravel has a missing attribute – airlineLogo. Similarly AirTranAirways also does not provide airlineLogo. Again, the FD airlineCompany -> airlineLogo defined on DirectFlight can be used. Applying the criteria for selecting an associate resource, AirlineLogos qualifies. The resource AirlineLogos itself as a primary resource again has missing attributes. However, the planner does not find any FD that it can use to retrieve values for them. Hence, it is eliminated.

A point to note is that the execution plan could use a source more than once. For example, a source could be used as a primary resource as well as an associate resource. However, it need not be accessed twice if it does not need any binding patterns. The Planning Agent is able to identify such resources and optimize the plan by creating a single Resource Access node that retrieves all the information (attributes) needed. Every node in the plan that needs information from this resource then points to this Resource Access node.

Figures 1 and 2 show the execution plans created for IScape 1 and IScape 2 respectively with numbers against the nodes and the links between the nodes indicating the step in which the node/link was created. They show how some sub-conditions can be pushed to resources and how some sub-conditions may even be eliminated using resource rules.

For IScape 1, the resource NuclearTestsDB is joined with NuclearTestSites to provide values for the missing attributes latitude and longitude. The Resource Access nodes show the resource names and projection lists. A point to note is that there is no check for the constraint NuclearTest.eventDate > “January 1, 1980”. This is because NuclearTestsDB provides data for nuclear tests that were conducted only after January 1, 1985.

For IScape 2, the resources YahooTravel and AirTranAirways are joined with AirlineLogos to provide values for the missing attribute airlineLogo. Note that the planner creates only one Resource Access node for AirlineLogos although it is used twice. Hence, the resource actually gets queries only once. The Resource Access nodes show the resource names, projection lists and

BPs, if any. Also note that since the IScape's constraint was entirely used as a binding pattern, there is no need to check for it again. Hence there is no select node.

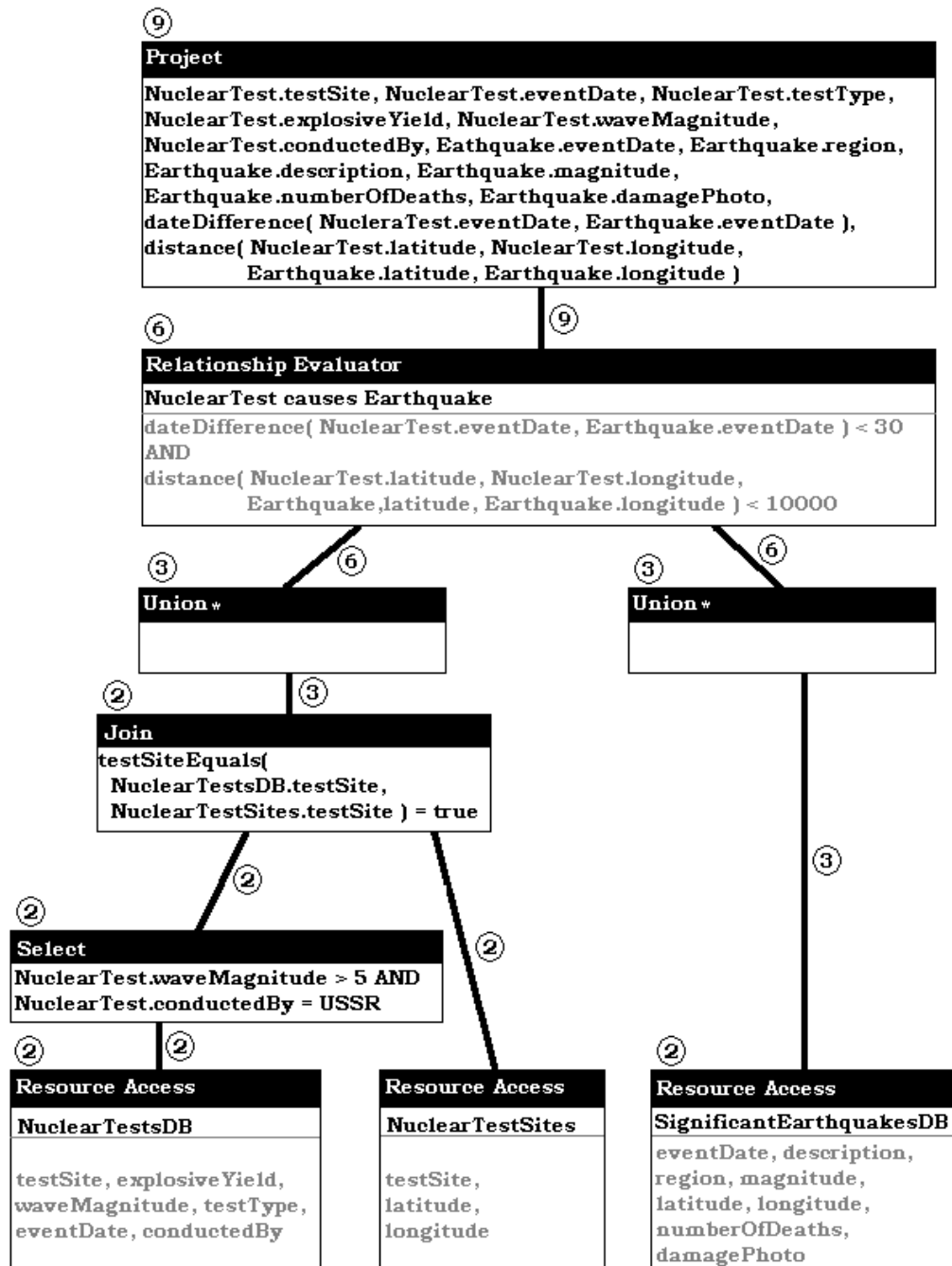


Figure 1: Execution plan for IScape 1

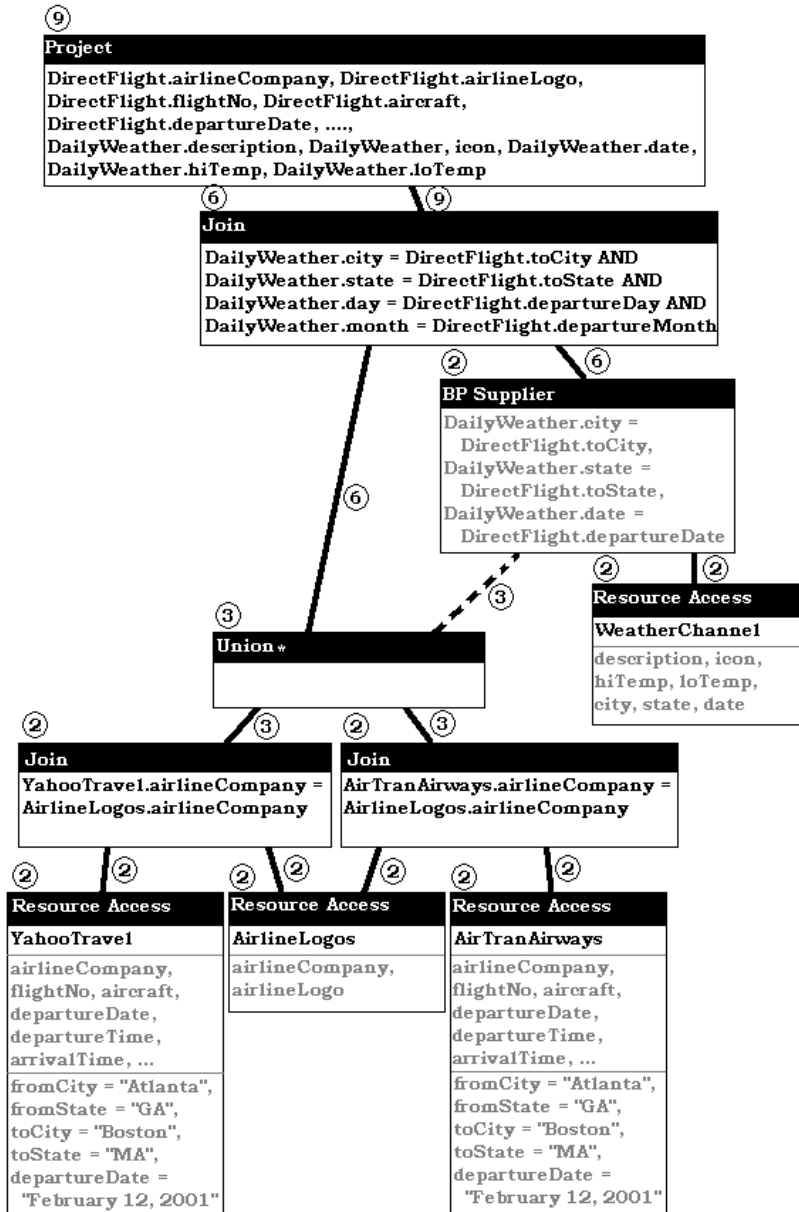


Figure 2: Execution plan for IScape 2

Step 3 - Integrate Data For Each Ontology

Once the sources for the ontologies are selected, the next step is to create the plan nodes that describe how the data retrieved from these sources is to be integrated, how the relationships and functions are evaluated, etc. The first step in doing this is to create unions of results retrieved from different resources selected for each ontology used in the IScape.

For this, the planner first adds a node to compute the union of results retrieved from only those resources that do not retrieve values for their BP from attributes of some other ontologies. Hence, if some resource uses values of some attribute(s) of this ontology for its BP, it can retrieve these values from the result set computed by the intermediate union. We use a * in the figures to denote a union node of this type and refer to it as *intermediate union*. It then adds another union node that computes the union of all the resources for the ontology. At all stages of execution plan

creation, if a node is found to be trivial, it is eliminated. For example, if the intermediate union node is found to have no children, it is eliminated. Similarly, if the final union node is found to have only one child, it is eliminated.

Step 4 – Supplying BP Values From Other Ontologies

After creating the unions, the planner goes through all the BPSupplier nodes created in the plan till now to check that if it needs values from attributes of other ontologies for some BP attributes, then there exists an intermediate union node for those ontologies. If it does not exist, the resource cannot be used since no resources supply the values it needs. If the intermediate union does exist, it updates the plan to point to the corresponding union nodes. This step was postponed until now since the planner needs to select resources for all ontologies and compute their unions first.

Note that no union node (other than the intermediate union to compute union of results retrieved from resources that do not use BP from other ontologies) was needed. It was eliminated since it would have been trivial. Also, note that there is no intermediate union for the ontology `DailyWeather` as the resource uses BP values from the `DirectFlight` ontology. The link from BPSupplier node to the Union node for `DirectFlight` indicated as a dashed line indicates that the values for the BP needed on the resource are retrieved from the Union node.

Step 5 - Functions and Constraints Involving Single Ontology

The conditions pushed towards the resources till now were those that did not involve any function operands and involved a single ontology. So the next step is to evaluate these functions and selections that involve only attributes or functions using attributes from that ontology. Both IScape examples do not have any such functions and conditions. Hence, the plans do not change.

Step 6 – Relationship Evaluations

The next step is to integrate the data from different ontologies by evaluating the relationships used in the IScape or creating joins if some ontology is not a part of any relationship.

Step 7 – Functions and Constraints Involving Multiple Ontologies

The functions that use attributes from more than one ontology as parameters are evaluated next and any constraints that involve multiple ontologies are then evaluated. Both example iscapecs do not have any such functions (`distance` and `dateDifference` got evaluated during relationship evaluation). Hence, the plan does not change.

Step 8 - Aggregation

The next step is to group the results by the specified attributes, compute the required aggregations and evaluate group constraints specified in the IScape, if any. Neither of the two example IScapecs we are using require grouping of the results.

Step 9 - Final Projection

The last step is to project the result on the operands (attributes, functions and aggregates) specified in the projection list of the IScape. This completes the generation of the execution plan for the IScape.

4. IScape execution

This section discusses how an IScape submitted to the InfoQuilt system is processed. We start by describing the runtime architecture.

4.1 Runtime Architecture

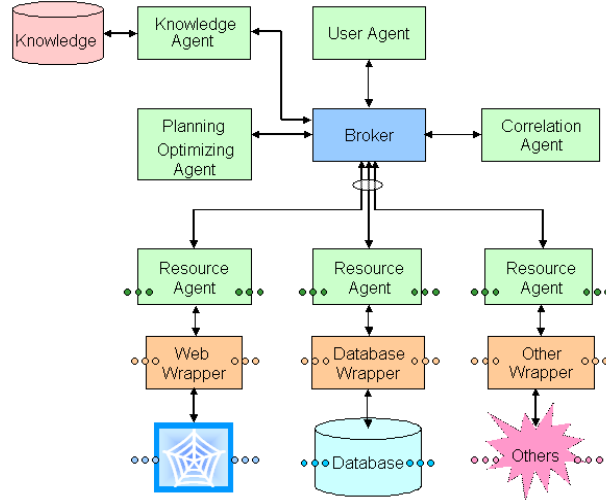


Figure 3: InfoQuilt Runtime Architecture

InfoQuilt uses an agent-based brokering architecture shown in Figure 3. Although the agent-based architecture itself is not unique (e.g., see [BBB97]), the capabilities of the agents to support complex relationships, multiple ontologies and IScapes differentiate the system from previous agent-based systems. This work is built upon the work done in [Sin00, Ber98, Par98].

The Knowledge Agent maintains knowledge about all the ontologies, relationships, resources as well as the functions in an integrated manner. The database “Knowledge” in the figure represents this knowledge base of the system. The User Agent acts as a programming interface to the system to execute IScapes. The Broker Agent is responsible for brokering requests from various agents in the system and co-ordination of IScape processing. The Planning Agent is responsible for the creation of high-quality optimal execution plans to execute the IScapes. The Correlation Agent executes the execution plan to retrieve data from the sources, correlate them and do any post-processing if required by the IScape. The Resource Agents provide a uniform interface to access all resources. There is one Resource Agent per resource accessible to the system. They address the issues related to access mechanisms, mapping data from the source views to global views of the system, *etc.*

The processing of an IScape goes through the following steps:

1. The User Agent sends the IScape to the Broker Agent for processing.
2. The Broker Agent sends the IScape to the Planning Agent.
3. The Planning Agent creates an execution plan for the IScape after enquiring about the specifications of various ontologies, relationships, *etc.* from the Knowledge Agent through the Broker Agent. (All interactions between the agents occur through the Broker). If the planning agent deduces that the IScape is semantically incorrect (see section 3.2), it informs the Broker Agent about it and aborts plan generation.
4. The Planning Agent returns the plan to the Broker Agent. If the plan generation was aborted, the Broker Agent responds back to the User Agent directly (skipping steps 5-7).
5. The Broker Agent then sends the plan to the Correlation Agent for processing.
6. The Correlation Agent starts executing the plan. The execution is completely multi-threaded. This allows the system to exploit the parallelism in the plan. We describe the execution in more detail in section 4.2.

7. The Correlation Agent returns the final result to the Broker Agent.
8. Finally, the Broker Agent forwards the result to the User Agent.

4.2 Multi-threaded execution of IScape by Correlation Agent

The IScape execution can exploit any parallelism in the plan by processing it in parallel. For example, the Correlation Agent can query several resources simultaneously. This can be very beneficial in an information integration system like InfoQuilt where a large number of the sources available to the system are web sources. Retrieving information from a web source over the network using some wrapper is relatively slow compared to a database. The Correlation Agent therefore employs completely multi-threaded execution of IScapes.

It starts by creating a *node processor* for each node in the plan. Each node processor is a separate thread. It thus creates a tree of node processor threads. Every node is responsible for starting the node processors from which it expects any inputs (result sets). It then waits for all its inputs and starts processing as soon as these inputs are available.

4.3 Function Evaluations (including simulations)

Evaluations of functions (including execution of simulations) is achieved by using Java's reflection package to pass arguments, invoke the function and retrieve the results back. The user is responsible for supplying the methods as static functions in Java classes.

However, simulations can be tricky as they are not always Java programs. They could be executable files, written in some other programming languages. Also, it is common to come across simulation programs that require that inputs be supplied to them as files in the file system. Considering the diversity that can be encountered, the framework provided may not be sufficient for all kinds of simulation programs available. This needs further investigation. However, a large part of these can be supported using the current framework. For example, we use Clarke's Urban Growth Model [Cla] to predict urban growth in San Francisco area. The program is available as an executable and needs an input configuration file from which it reads in its parameters. We added this simulation to the system by creating a static Java method in a class, that given the parameters, generates this configuration file, runs the program using JNI and retrieves its result.

4.4 Failure Handling in Correlation Agent

Failure handling is obviously crucial. Especially since the Correlation Agent uses a tree of node processor threads that wait on each other, it is possible that some node processor fails or is taking an unusually long time due to some problem. The threads that are waiting on it could keep waiting forever. To avoid this, the Correlation Agent uses time-outs. The threads wait only for a specified amount of time at the most. If all its inputs are not available by then, it assumes that one of those processors have failed and then continues processing assuming that the failed processor returned an empty result set. Any exception occurring during IScape processing is also handled by catching it and then continuing assuming an empty result set.

4.5 Execution monitoring

We provide a visual tool that helps monitoring the execution of an IScape. It is known as the IScape Processing Monitor (IPM). The agents involved in processing generate detailed log messages that are sent to the monitor along with any related data structures (execution plans and intermediate result sets). The data structures can be the execution plan sent by the Planning Agent or result sets sent by the Correlation Agent as the processing of the IScape progresses. The IPM displays the logs as color-coded entries in a table. It is very useful as analysis tool that helps us

analyze the performance of the system due to the availability of timestamps with the log entries. It is also useful as a high-level debugging tool. Any exception occurring in the system can be easily identified as the involved agent sends a detailed message about it to the monitor. However, the execution still proceeds as mentioned before assuming an empty result set. Figure 4 shows a screen shot of the IPM's log table.

The screenshot shows the IScape Processing Monitor window. At the top, there is a search bar with the query: "Find all nuclear tests conducted after January 1, 1985 with seismic body wave magnitude > 5 and find all earthquakes that could have been caused due to these tests". Below the search bar is a "Processing Log" section containing a table with the following data:

Message Id	Time Stamp	Message From	Brief Message
0	12:02:44.171	User Agent	Started processing
1	12:02:44.453	Broker Agent	Started processing
2	12:02:48.406	Planning Agent	Applied Domain Rules to IScape's constraint
3	12:02:48.500	Planning Agent	Checking IScape's constraint and relationshi...
4	12:02:48.593	Planning Agent	Selecting sources for Earthquake
5	12:02:48.687	Planning Agent	Selecting sources for NuclearTest
6	12:02:48.812	Planning Agent	Plan created by the planner. Returning it to Br...
7	12:02:49.125	Broker Agent	Received plan from planner
8	12:02:50.015	CorrelationAgent	Executing IScape
9	12:02:57.906	TestSitesDB Resource Agent	Queried TestSitesDB
10	12:02:58.093	SignificantEarthquakesDB Resource Agent	Queried SignificantEarthquakesDB
11	12:02:58.171	Correlation Agent	Computed Union
12	12:02:58.156	NuclearTestsDB Resource Agent	Queried NuclearTestsDB
13	12:02:58.515	Correlation Agent	Evaluated Join
14	12:02:58.687	CorrelationAgent	SelectNodeProcessor done processing
15	12:02:58.734	Correlation Agent	Computed Union
16	12:02:59.109	Correlation Agent	Evaluated functions on a relation.
17	12:02:59.187	CorrelationAgent	SelectNodeProcessor done processing
18	12:02:59.781	Correlation Agent	Evaluated Relationship on a set of relations
19	12:02:58.828	Correlation Agent	Evaluated projection on a relation
20	12:02:59.937	Broker Agent	Received IScape results from Correlation Ag...
21	12:03:00.078	User Agent	Returning final result to client

At the bottom of the window, there are two buttons: "Detailed Log Message" and "Show Associated Data".

Figure 4: IScape Processing Monitor (IPM)

5. Related Work

SIMS [AK97, AKS96], TSIMMIS [CMH+94], Information Manifold [LRO96], and OBSERVER [MIKS00] are some of the other efforts to integrate information from multiple heterogeneous sources. The goal of InfoQuilt is to provide an environment where users can query, analyze, reason about inter-domain relationships and analyze data available from multiple sources (including web-based sources). However, most other systems focus only on retrieving and integrating “data” from multiple sources and not on the “exploring, understanding, and knowledge discovery” aspects. The following are the features of InfoQuilt that are not supported by any other system:

- Ability to assist in learning about domains and complex inter-domain relationships
- Support for use of functions and simulations to post-process and thereby add value to data that is retrieved from the resources
- Support for complex relationships and constraints that cannot be expressed using relational and logical operators
- Powerful semantic query interface (IScapes)

The general approach of most of the systems is to model the domains and available information sources. They then use these models to translate a query specified by the user into an execution plan that specifies the actual available sources that will be used and how information retrieved from them will be integrated. We compare our vision and approach with other systems in this section.

SIMS [AK97, AKS96] adopts the approach of creating a model of the domain using a knowledge representation system establishing a fixed vocabulary and accepting queries specified in the same language. A major limitation of the system is that its mediator is specialized to a single application domain [AHK96]. InfoQuilt supports multiple ontologies that may be completely independent of each other. An application domain in SIMS models a single hierarchy of classes. It also does not support inter-ontological relationships and functions. They do not consider use of local completeness information about sources and support only one binding pattern per web resource.

OBSERVER [MIKS00] uses ontologies to describe information sources and inter-ontology relationships like synonyms, hyponyms and hypernyms across terms in different ontologies to be able to translate a query specified using some ontology that the user selected into another query that uses related ontologies describing relevant information. This approach of using relationships to achieve interoperability between the sources is interesting. However, it is limited to basic relationships. We use relationships to model complex real-world relationships that may exist between real-world entities that may belong to completely independent domains (ontologies). Also, our domain and source models are richer than the ontologies defined in OBSERVER.

TSIMMIS [CMH+94, GPQ+95] uses a mediator-based architecture [Wie92]. It uses Mediator Specification Language (MSL) to define mediators, encoding how the system should use the resources. The mediators are then generated automatically from these specifications. They therefore lack the descriptive view that an ontology can provide in our system. Also, since the MSL definitions need to be created manually, adding or removing information sources requires updating them after determining how the sources should be used to answer the queries and then recompiling them. It has a set of pre-defined query templates that it knows to answer. User queries are then answered by relating them to these templates. The query answering system (mediator) is thus query centric and can answer only a restricted set of queries. InfoQuilt has a dynamic planner that is not query-centric. It automatically considers newly added sources while planning IScapes.

Information Manifold [LRO96] uses an approach similar to ours in that the user creates a *world view*, a collection of virtual relations and classes. The world view however does not capture semantics of the domains as InfoQuilt can using domain rules and FDs. Information sources are described to the system as a query over the relations in the world view. The user queries are also specified over relations in this world view. The sources can be specified to be either a subset of the domain or equal to a subset of the domain [LSK95]. So a resource that is locally complete for a part of information that it provides cannot be modeled appropriately. This does not capture local completeness information about the sources precisely. IM uses capability records to capture query capability limitations of sources. These records specify, among others, a set of input parameters and the minimum and maximum number of inputs allowed. The system then arbitrarily selects a subset of the set of input parameters with at least the minimum number of allowed parameters in the set. The subset selected is arbitrary. Therefore, the capability records cannot precisely specify the binding patterns. This approach would not work if the source needs very specific combinations of attributes as input.

[Dus97] present a comprehensive theory on planning queries in information integration systems. They focus on creation of maximally-contained query plans. A maximally-contained query plan is defined as a plan that provides *all* the answers that are possible to obtain from the available sources, but the expression describing the plan may not be equivalent to the original query. They show that it is necessary to consider plans with recursion in order to generate maximally-contained plans. Considering that a large number of information sources would be web sources that are slow relative to a source that is a database, execution of plans with recursion could be very slow. A query plan that does not necessarily return *all* the possible results may still be useful. We adopt this approach.

6. Conclusion

We described query planning and optimization algorithms used in InfoQuilt, an information integration system that focuses on integrating information from a multitude of diverse distributed sources. We discussed how InfoQuilt goes beyond the traditional querying techniques to provide access to and integrate data in a more semantic manner with the goal of providing a querying, analyzing and knowledge discovery environment. The key features of the system are:

- ability to model inter-domain relationships
- ability to use value-adding functions and simulations
- powerful query interface to describe complex information needs (IScapes)
- environment for knowledge discovery

This is supported by use of domain rules, functional dependencies, resource rules, resource query capability limitations and their local completeness information. These enable the planner to apply several semantic optimizations while creating execution plans for IScapes. The planner is dynamic and flexible as it is not query-centric and allows dynamic addition/deletion of ontologies, relationships and resources. The contributions of this paper are practical algorithms to efficiently select sources most appropriate to answer an IScape by excluding sources with redundant or no useful information in the context of the IScape, use sources in conjunction to retrieve more comprehensive information using the same set of available resources, generate executable plans that respect query capability limitations of the resources, integrate information after it is retrieved including evaluating relationships and post-processing (evaluating functions and running simulations), and multi-threaded processing of the IScapes to exploit the parallelism in the plans.

Following are some of the planned future improvements. The Planning Agent could create backup plans that the Correlation Agent can switch to on failure. Simulations are currently supported using the framework used for functions. Simulation programs however are more complex and diverse in the kind of application (it could be an executable, a script, etc.), the method of accepting inputs, for example as files from local file systems, etc. The framework currently used may not suffice as it assumes that they are available as separate functions (through wrappers if needed). Several simulations however exist as programs written using languages supported by special software, for example, ArcInfo, and hence, may be difficult to add to the system. Different types of simulation programs therefore need to be explored more thoroughly to provide a framework better suited to support a large class of them, if not all.

7. References

- [Ade] Alexandria Digital Earth Prototype (<http://alexandria.ucsb.edu/>)
- [AHK96] Y. Arens, C. Hsu and C. A. Knoblock. Query processing in the SIMS information mediator. In Austin Tate, editor, *Advanced Planning Technology*. The AAAI Press, Menlo Park, CA, 1996.
- [AK97] J. Ambite, and C. Knoblock. Planning by Rewriting: Efficiently generating high-quality plans. *Proceedings of the 14th National Conference on Artificial Intelligence*, Providence, RI, 1997.
- [AKS96] Y. Arens, C. A. Knoblock, and W. Shen. Query reformulation for dynamic information integration. *Journal of Intelligent Information Systems*, Vol. 6, pp. 99-130, 1996. Look for a more recent SIMS paper
- [BBB97] R. J. Bayardo Jr., W. Bohrer, R. Brice, et al. InfoSleuth: Agent-Based Semantic Integration of Information in Open and Dynamic Environments. In *SIGMOD-97*, pp. 195-206, Tucson, AZ, USA, May 1997.

- [Ber98] C. Bertram. InfoQuilt: Semantic Correlation of Heterogeneous Distributed Assets. Masters Thesis, Computer Science Department, University of Georgia, 1998.
- [CKPS95] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, K. Shim. Optimizing queries with materialized views. Proceedings of international conference on Data Engineering, 1995.
- [Cla] Clarke's Urban Growth Model, Project Gigalopolos, Department of Geography, University of California, Santa Barbara.
<http://www.ncgia.ucsb.edu/projects/gig/ncgia.html>
- [CMH+94] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The TSIMMIS Project: Integration of Heterogeneous Information Sources. Proceedings of 10th Anniversary Meeting of the Information Processing Society of Japan, pp. 7-18, Tokyo, Japan, 1994.
- [Dus97] O. M. Duschka. Query Planning and Optimization in Information Integration. Ph. D. Thesis, Computer Science Department, Stanford University, 1997.
- [GPQ+95] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. The TSIMMIS Approach to Mediation: Data Models and Languages. In Proceedings of NGITS (Next Generation Information Technologies and Systems), 1995.
- [Gun00] M. Guntamadugu. MÉTIS: Automating Metabase Creation from Multiple Heterogeneous Sources. Masters Thesis, Computer Science Department, University of Georgia, 2000
- [KS00] V. Kashyap, A. Sheth. Information Brokering Across Heterogeneous Digital Data – A Metadata-based Approach. Kluwer Academic Publishers, 2000.
- [LMSS95] A. Y. Levy, A. O. Mendelzon, Y. Sagiv and D. Srivastava. Answering queries using views. Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of database systems, PODS-95, San Jose, California, May 1995.
- [LRO96] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. Proceedings of the 22nd International Conference on Very Large Databases VLDB-96, Bombay, India, September 1996.
- [LSK95] A.Y. Levy, D. Srivastava, and T. Kirk. Data Model and Query Evaluation in Global Information Systems. International Journal on Intelligent Information Systems, pp. 121-143, 1995.
- [MIKS00] E. Mena, A. Illarramendi, V. Kashyap, and A. P. Sheth. OBSERVER: An approach for query processing in global information systems based on interoperation across pre-existing ontologies. International Journal on Distributed and Parallel Databases, Vol. 8, No. 2, pp. 223-271, April 2000.
- [Par98] K. Parasuraman. A Multi-Agent System For Information Brokering In InfoQuilt. Masters Thesis, Computer Science Department, University of Georgia, 1998.
- [Sin00] D. Singh. An Agents Based Architecture for Query Planning and Cost modeling of Web Sources. Masters Thesis. Computer Science Department, University of Georgia, 2000.
- [She99] A. Sheth, Changing focus on Interoperability: From System, Syntax, Structure to Semantics. In M. Goodchild, M. Egenhofer, R. Fegeas, and C. Kottman, editors, Interoperating Geographic Information Systems, Kluwer Academic Publishers, 1999.
- [SK98] A. Sheth and W. Klas, Editors. Multimedia Data Management – Using Metadata to Integrate and Apply Digital Media. Mc Graw-Hill, 1998.

- [TPS01] S. Thacker, S. Patel, and A. Sheth. Knowledge Modeling for study of domains and inter-domain relationships – A Knowledge Discovery Paradigm. LSDIS Technical Report, University of Georgia, March 2001.
- [Whi89] G. T. Whiteford. Earthquakes and Nuclear Testing: Dangerous Patterns and Trends. In proceedings of the 2nd Annual Conference on the United Nations and World Peace, Seattle, Washington, April 1989.
- [Wie92] G. Wiederhold. Mediators in the architecture of future information systems. IEEE Computer, 25(3), pp. 38-49.
- [YL87] H. A. Yang, and P. A. Larson. Query transformation for PSJ queries. Proceedings of 13th International Conference on Very Large Databases VLDB-87, Brighton, England, 1987, pp. 245-254.