, , 1–38 ()
© Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Query Reformulation for Dynamic Information Integration

YIGAL ARENS, CRAIG A. KNOBLOCK, AND WEI-MIN SHEN

{ARENS, KNOBLOCK, SHEN}@ISI.EDU

Information Sciences Institute and
Department of Computer Science
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292

Received June 15, 1995

Editor:

Abstract. The standard approach to integrating heterogeneous information sources is to build a global schema that relates all of the information in the different sources, and to pose queries directly against it. The problem is that schema integration is usually difficult, and as soon as any of the information sources change or a new source is added, the process may have to be repeated.

The SIMS system uses an alternative approach. A *domain model* of the application domain is created, establishing a fixed vocabulary for describing data sets in the domain. Using this language, each available information source is described. Queries to SIMS against the collection of available information sources are posed using terms from the domain model, and *reformulation operators* are employed to dynamically select an appropriate set of information sources and to determine how to integrate the available information to satisfy a query. This approach results in a system that is more flexible than existing ones, more easily scalable, and able to respond dynamically to newly available or unexpectedly missing information sources.

This paper describes the query reformulation process in SIMS and the operators used in it. We provide precise definitions of the reformulation operators and explain the rationale behind choosing the specific ones SIMS uses. We have demonstrated the feasibility and effectiveness of this approach by applying SIMS in the domains of transportation planning and medical trauma care.

Keywords: Information integration, multidatabase systems, query reformulation, heterogeneous databases.

1. Introduction

The overall goal of the SIMS project is to provide intelligent access to heterogeneous distributed information sources (databases, knowledge bases, flat files, and certain types of programs), while insulating human users and application programs from the need to be aware of the location of the sources, their query languages, organization, size, etc.

The standard approach to this problem has been to construct a global schema that relates all the information in the different sources and to have the user pose queries against this global schema or various views of it. The problem with this approach is that integrating the schemas is typically very difficult, and any changes to existing data sources or the addition of new ones requires a substantial, if not complete, repetition of the schema integration process. In addition, this standard approach is not suitable for including information sources that are not databases.

SIMS provides an alternative approach. A *domain model* of the application domain is created, using a knowledge representation language to establish a fixed vocabulary describing objects in the domain, their attributes, and the relationships among them. SIMS accepts queries in this high-level uniform language. It processes these queries in a manner hidden from the user, ultimately returning the requested data. Thus, the queries to SIMS need not contain information describing which sources are relevant to finding their answers or where they are located (although they may, if a user wishes to obtain data only from a specific source). Queries do not need to state how information obtained from different sources should be joined or otherwise combined or manipulated. It is the task of SIMS to determine how to efficiently and transparently retrieve and integrate the data necessary to answer a query.

The SIMS approach, where there is no fixed mapping from a query to the sources used to answer a query, has several important advantages. The approach is:

Flexible: The SIMS planner will consider alternative ways to retrieve the data requested by a query. If multiple databases contain the same data, or copies of portions of the data, SIMS will determine this in the course of its operation. It will then select the best source for retrieval of the data according to its criteria. If there is no direct source available for the requested information, the system will attempt to reformulate a query to use related classes of information that could provide the same data. The flexibility of considering alternative ways to retrieve a set of data forms the basis for SIMS' ability to dynamically recover from execution failures.

Scalable: New information sources are added to SIMS without regard to information sources that are already part of the system. A new source is modeled using terms from the shared domain model only. This simplifies the process of adding new information sources since the new source can be modeled independently of the other information sources.

Dynamic: Since the plan for retrieving requested data is produced at the time the query is submitted, the existing circumstances can be taken into account. SIMS can respond to information sources that are temporarily unavailable, or to recently added sources. SIMS can even replan if an information source is discovered to be unavailable during the process of executing a plan that was created under the assumption that the source was available.

There are four basic components to query-processing in SIMS [2]. These are:

- Query reformulation

  This component identifies the sources of information that are required in order to answer a query and determines how data from them must be combined to produce precisely what the user requested. This is done by reformulating the user's query expressed in domain terms into queries to specific information sources. This subtask of SIMS is the subject of this paper. An early version of the query reformulation process in SIMS was briefly described in [3]. This paper refines those early ideas, presents the detailed specification of the reformulation operators, and describes the search process.

- Query access planning

  The second component constructs a plan for retrieval of the information requested by the reformulated query (and hence, by the original query). The plan involves steps such as sending a specific query to some information source, moving data from one source to another, joining results from different information sources, and temporarily storing partial results. See [17], [18] for details.

- Semantic query-plan optimization

  The third component exploits learned knowledge about the contents of databases to perform semantic query optimization. We have extended semantic query optimization techniques to support multidatabase queries and have developed an approach to learn the rules for the optimization process. See [10], [11], [12], [13] for details.

- Execution

  Finally, the fourth component executes the optimized query plan. SIMS executes queries against the appropriate information sources (doing so in parallel when possible), transfers data, constructs a response to the user, and returns it. An execution failure will cause SIMS to replan part or all of a query. To support execution, SIMS makes use of *wrappers* that mediate between it and the information sources themselves. A wrapper will accept a query for an information source formulated in SIMS' query language, translate it into the appropriate query language for its information source, submit it, and forward the resulting data back to SIMS.

In this paper, we will focus on the first component and describe how query reformulation is used to identify relevant information sources, decide which data should be retrieved from them, and integrate it to satisfy the user's query.

SIMS relies on its model of an application domain and on models of the available information sources to reformulate a query expressed in domain-level terms into a query using only terms from the models of the information sources. Steps in the reformulation process consist of applications of any of several available *reformulation operators.* The application of each operator rewrites a number of clauses of the given query into a different, but semantically equivalent, set of clauses. Operators are repeatedly applied until the resulting query explicitly refers to information sources that contain (or can produce, in the case of programs) the needed information. Furthermore, the resulting query will make explicit how information from the various sources must be combined to result in an answer to the original query posed to SIMS.

This paper will present the details of query reformulation in SIMS. We start, in Section 2, with a description of the representation system used by SIMS to describe both the model of the application domain and models of the individual information sources, which are used in the reformulation process. Section 3 then describes the operators used to reformulate queries. Section 4 explains how the reformulation operators are applied and how they interact with query access planning. Section 5 presents experimental results. Section 6 discusses the limitations of the SIMS approach: which types of queries can be answered and which cannot. Section 7 describes related work and Section 8 summarizes our conclusions and directions for future work.

2. Modeling and Querying Information Sources

Before we can describe the query reformulation process, we must first provide some background on our approach to modeling a domain, modeling the contents of information sources, and querying these information sources. We describe each of these in turn.

2.1. The Domain Model

In order to combine information from heterogeneous information sources we need a shared ontology that can be used to describe the contents of sources available within some application domain. This is done using a domain model, which describes the classes of information in the domain and the relationships among the classes. This model is used to integrate the various information sources that are available and provide the terminology for accessing them. Queries are expressed in terms of the domain model, and all available information sources are defined in terms of this model.

Throughout this section we use an example from a transportation planning domain — planning the movement of personnel and materiel from one location to another using aircraft, ships, trucks, etc. The example has been simplified from the actual domain in order to provide a short, self-contained description of a model.

The domain model is described in the Loom language [23], which is a member of the KL-ONE family of knowledge representation systems [6]. The basic objects in Loom are *classes* (also called *concepts*), which define a set of related instances, and *roles*, which define the attributes of a class. The model is used to capture the following information about classes and roles:

- Definition of a class.
- Relationship to other classes (i.e., subclass, superclass, coverings).
- Definition of a role.

Figure 1 shows a small fragment of a domain model. Classes are indicated with circles, roles with thin arrows, and subclass relations with thick arrows. Roles are inherited down to subclasses. In Figure 1, there is a node in the model representing the class of ports and another node representing the class of airports. The thick arrow between these classes indicates that `Airport` is a subclass of `Port`. There are also roles on these classes, such as the role `country-code` specified between `Port` and `Country-Id` with a notation (not shown) indicating that each of the former has precisely one of the latter. Some of these roles are also marked as key roles, indicating that they uniquely identify the members of that class.

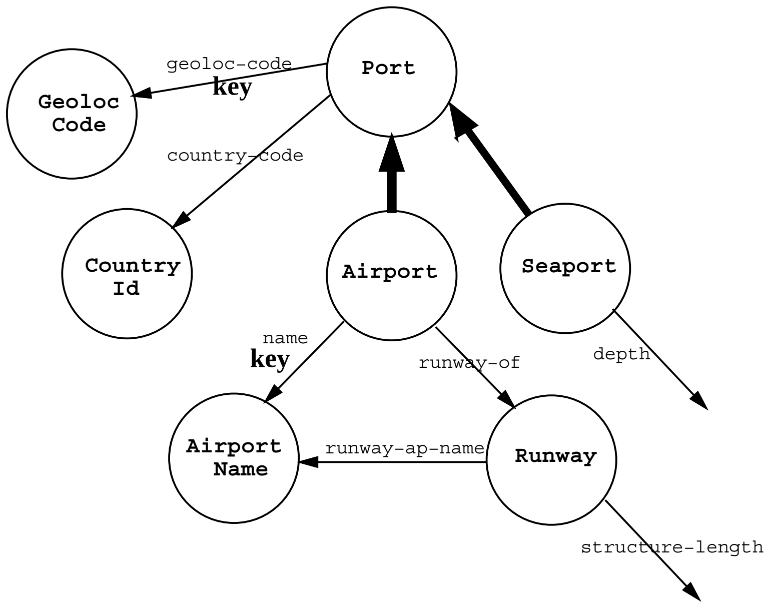

Figure 1. Domain Model Fragment

The Loom definition of the `Airport` concept is as follows:

```
(defconcept Airport
  :is-primitive
  (:and Port
        (:all name Airport-Name)
        (:all runway-of Runway)
        (:all altitude Number))
  :annotations
       ((key (name))))
```

This definition states that an `Airport` is a subclass of `Port` and, in addition to the inherited roles from `Port`, it has three additional roles: `name`, `runway-of`, and `altitude`. The roles provide the name, runways, and altitude for each airport. The Loom term "is-primitive" is used to indicate that this definition may not be complete and there may be additional distinguishing features that are not stated.

Each domain class typically has one or more roles defined as keys.[1] Each key can consist of a single role or a set of roles. The keys are used to uniquely identify instances of a class. For example, the role `name` is defined as a key for the class `Airport`. This is defined in the `annotations` of the `Airport` class. Key roles are critical in determining how information from different sources can be integrated. This integration process is described in detail in Section 3. In this case, the model also indicates that the `geoloc-code` uniquely identifies an airport.

The entities included in the domain model are not necessarily meant to correspond directly to objects described in any particular information source. The domain model is intended to be a description of the application domain from the point of view of someone who needs to perform real-world tasks in that domain and/or to obtain information about it.

For example, the class of high-altitude airports, which are airports with an altitude greater than 5,000 feet, might be particularly important for a given application, yet there may be no information source that contains only this class of airport. Nevertheless, we can define this class in terms of other classes for which information is available. The Loom definition of this concept would be:

```
(defconcept High-Altitude-Airport
   :is (:and Airport
             (> altitude 5000)))
```

Note that in this definition the concept is not marked as primitive, indicating that this is a complete definition of what it means to be a high-altitude airport. In Section 3.6 we will describe how the system exploits this definition in processing a query.

In addition to the subclass and superclass relationships, we can also define *coverings* of a class. A covering of a class is a set of subclasses whose union is equivalent to the original class. A class can have multiple coverings. For example, `Airport` and `Seaport` cover the `Port` class. This would be expressed in the definition of the `Port` class as follows:

```
(defconcept Port
    :is-primitive
       (:and Geographic-Location
             (:all primary-port-name String)
             (:all secondary-port-name String)
             (:all railroad-access String)
             (:all road-access String))
    :annotations
       ((key (geoloc-code))
        (covering Port (Airport Seaport))))
```

A role is typically defined simply by stating its domain and range. For example, the `name` role is defined with a domain of the class `Airport` and a range of the class `Airport-Name`.

```
(defrole[2] name
  :domain Airport
  :range Airport-Name)
```

A role can also be defined in terms of other roles. This is important because not all roles in the domain model have corresponding data in any information source. If there is no corresponding information for a role in any information source, the system can attempt to reformulate the query by substituting for the role an equivalent combination of other roles. For example, consider the role `runway-of`. This role is defined as follows:

```
(defrole runway-of
    :is (:satisfies (?a ?r)
          (for-some (?name)
              (:and (Airport ?a)
                    (name ?a ?name)
                    (Runway ?r)
                    (runway-ap-name ?r ?name)))))
```

This definition states that the `runway-of` role on `Airport` holds when there exists a runway whose `runway-ap-name` has the same value as the `name` of the airport. This knowledge will be of use in the course of processing queries (see Section 3.6).

2.2. Information Source Models

An information source is incorporated into SIMS by first modeling the contents of the information source and then relating the concepts and roles of the information source model to corresponding concepts and roles of the domain model. The model of an information source is quite simple and contains the classes of information available in the information source and the attributes of those classes.

Figure 2 provides an example illustrating the principles involved in representing an information source in SIMS. This figure shows how the contents of a relational database table are represented as an information source model. The table is represented by a class that stands for the collection of objects described by the rows of the table. In this case, for the `Airport` table in the `AFSC` database we will create the Loom class `AFSC:Airport` whose instances stand for the airports described in that table. For each column of the table there is a corresponding role on the class. In this case, `AFSC:Airport` has two roles corresponding to the two columns in the table. In general, we represent n-ary relations as a set of binary relations between the class and the individual columns of the relation.

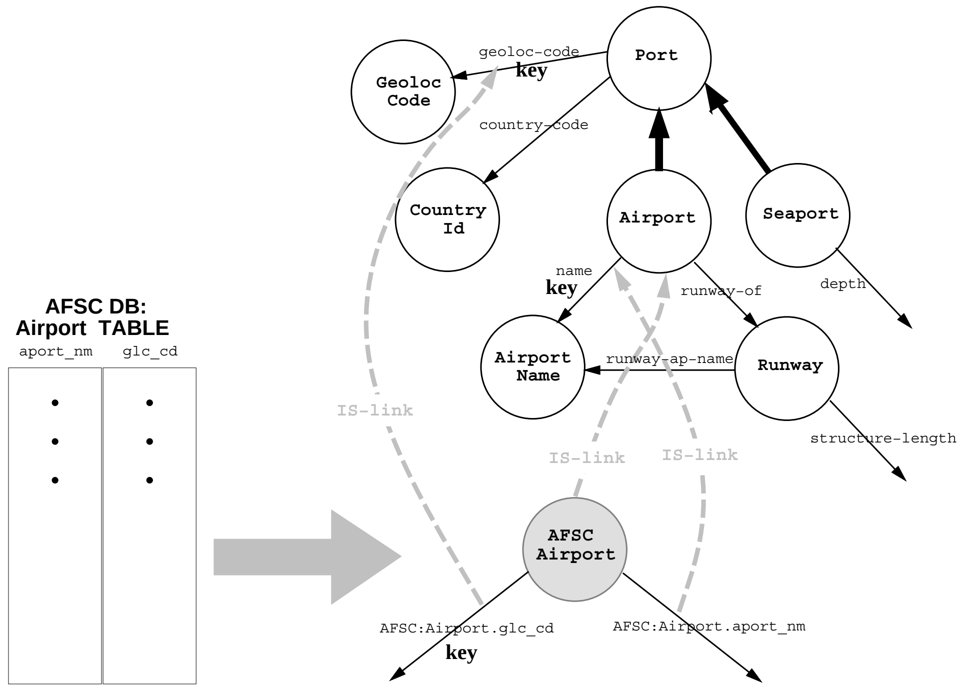

Figure 2. A Model of a Database Table Embedded in the Domain Model

An information source concept is defined similarly to a concept in the domain model. The class is marked as an *information source class* by an annotation that defines which source contains the data.

```
(defconcept AFSC:Airport
  :is-primitive
     (:and (:the AFSC:Airport.aport_nm String)
           (:the AFSC:Airport.glc_cd String))
  :annotations ((info-source AFSC)))
```

Each column in the table is represented in Loom as a role whose domain is the class corresponding to the table, and whose range corresponds to the class from which the values in the column are drawn. For example, the `aport_nm` column in the `Airport` table of the `AFSC` database is represented as the Loom role `AFSC:Airport.aport_nm`, as shown below.

```
(defrole AFSC:Airport.aport_nm
  :domain AFSC:Airport
  :range String)
```

Finally, each new concept and role must be related to the domain model. This is done by defining an information source link, `IS-link`, between the new concept and roles and the appropriate concept and roles in the domain model. The meaning of an `IS-link` between an information source class and a domain class is that the two classes contain exactly the same set of individuals, although the information source class might contain only a subset of the attributes for the class. The links between the roles indicate that those roles have the same meaning for the linked classes. Modeling an information source may require augmenting the domain model with additional classes in order to describe precisely the contents of a given information source. The advantage of this approach is that it provides the knowledge required to correctly integrate information in different sources.

An `IS-link` is the way SIMS makes explicit the semantics of the information in an information source. An information source may contain names, but the significance of those names is not self-evident. Are they indeed the names of the airports being described in each respective row of the table? Or are they the names of the closest alternative airports? Are they the names of the cities in which the airports are located? The possibilities are endless, and the schema alone is not sufficient to choose one. SIMS must know the precise relationship — and an `IS-link` to a previously defined domain model concept or role establishes it.

The `IS-link`s for the `AFSC:Airport` concept are defined as follows:

```
(def-is-link AFSC:Airport Airport
    ((AFSC:Airport.aport_nm name)
     (AFSC:Airport.glc_cd geoloc-code)))
```

This states that the information source concept `AFSC:Airport` is linked to the domain-level concept of `Airport`, the role `AFSC:Airport.aport_nm` is linked to `name`, and `AFSC:Airport.glc_cd` is linked to `geoloc-code`.

2.3. The Query Language

Domain model concepts and roles provide the vocabulary for the query language with which the user queries SIMS. To submit a query to SIMS, the user composes a Loom statement, using terms in the domain or information source models to describe the precise set of data that is of interest. If the user happens to be familiar with particular information sources and their representation, those concepts and roles may be used as well. But such knowledge is not required. SIMS is designed *precisely* to allow users to query without specific knowledge of the data's structure and distribution.

The query shown in Figure 3 requests the country codes for airports with runways longer than 10,000 feet. Line 1 in the query specifies that all possible values of the variable `?country-code` should be returned. Lines 2 through 6 state constraints that must be satisfied in retrieving the desired values. Line 2 states that the values of the variable `?aport` should be taken from the class `Airport`. Line 3 states that the role `country-code` must hold between the variables `?airport` and `?country-code`. And so on.

```
Line 1:   (retrieve (?country-code)
Line 2:      (:and (Airport ?aport)
Line 3:            (country-code ?aport ?country-code)
Line 4:            (runway-of ?aport ?rway)
Line 5:            (structure-length ?rway ?rlength)
Line 6:            (>= ?rlength 10000)))
```

Figure 3. Example Query

3. The Operators for Query Reformulation

This section defines the set of operators that can be used for query reformulation. Each operator defines a generic schema that describes the transformation of a clause or collection of clauses into another clause or collection of clauses. The applicability of these operators depends on the models, the given query, and the available information sources. This section describes the details of the individual operators and the next section describes how these operators are used to process a query.

3.1. The Motivation for the Choice of Operators

A query to SIMS is formulated using terminology from its domain model. We refer to one of these as a *Domain Query*. The first step in constructing a query plan involves the application of query-rewriting operators until a new, but semantically equivalent, query is obtained: one using *only* the terminology of information-source models. We refer to one of these as an *Information-Source Query*. The collection of query-rewriting operators available to SIMS is a major factor in determining which subsets of the data stored in the various information sources will be conveniently retrievable by a query to SIMS.

In designing SIMS, and specifically in defining its query-reformulation operators, we have attempted to follow these informal guidelines:

- **No Loss of Data:** SIMS should provide the ability to retrieve any data available from the individual sources. In other words, any data that can be retrieved by directly querying a particular information source using its own query language should be retrievable using SIMS.
- **No Loss of Expressive Power:** SIMS should support any query that commonly existing query languages support. If a distributed query language is available that can be used to access a database available in SIMS, the expressive power of SIMS should be at least as great as that language.
- **Natural Closure:** SIMS should be conveniently and naturally extensible. For any (Loom) concept/role arising naturally as a (well-formed Loom) combination of concepts/roles already in the domain model, it should be possible to add that concept/role to the model, and any domain query using it should result in retrieval of the expected data.

Desirable as these guidelines are, it may not be possible to follow them in all cases. For example, SIMS is bound by the expressive power of Loom. *Loss of Data* may occur if SIMS is dealing with a database supporting a structure and queries that cannot be naturally mapped to Loom – e.g., keyword-based queries against loosely structured text databases. However, we have selected reformulation operators and an approach to reformulation that will guarantee adherence to these guidelines at least for relational databases using SQL and distributed SQL, as well as for flat file databases using languages that have a weaker expressive power.

In the discussion that follows, we attempt to explain the reasons behind our choice of reformulation operators on the basis of the guidelines just presented.

3.2. The Minimal Model

In order to insure that there be no loss of data, the model used in SIMS must include concepts representing every simple collection of data in an information source which is retrievable using the query language of that information source. As pointed out above, this cannot always be guaranteed. However, in dealing with a relational database whose query language is SQL, we can rely on the fact that all of the SQL constructs can be expressed in Loom [31]. Thus, it is enough to make sure that the schema of the database and its information source model "mirror" each other; every database relation will have a corresponding information source model concept, and every attribute will have a corresponding role on that concept (Cf. Figure 2).

We define the following types of models:

The Minimal Model of an Information Source: A model that includes an information source-level model and enough of a domain-level model to exactly cover the information source model. There will be a one-to-one IS-link correspondence between the source- and domain-level classes and roles. This is the smallest model that provides sufficient domain-level entities to support reference to every existing information source concept and role.

The Minimal Model: The union of all minimal models for all the information sources available to the system. Informally, the minimal model is the smallest model that can describe the semantics of, and provide access to, the entire contents of the available information sources.

For example, the minimal model corresponding to the example information sources we have been using in this paper is pictured in Figure 4. There are two databases that are integrated in this model: AFSC and GEO. The first database has one table called `AFSC:Airport`, and the second database has two tables: `GEO:Runway` and `GEO:Port`. Each table consists of two columns. `AFSC:Airport` has `aport_nm` and `glc_cd`; `GEO:Runway` has `aport_nm` and `runway_length_ft`; and `GEO:Port` has `glc_cd` and `cy_cd`. In this minimal model, there are three domain concepts: `Airport` (corresponding to `AFSC:Airport`), `Runway` (corresponding to `GEO:Runway`), and `Port` (corresponding to `GEO:Port`). `Airport` has two roles: `name` (for `aport_nm`) and `geoloc-code` (for `glc_cd`). `Runway` has two roles: `runway-ap-name` (for `aport_nm`) and `structure-length` (for `runway_length_ft`). `Port` has `geoloc-code` (for `glc_cd`) and `country-code` (for `cy_cd`). Note that each role in this domain model is linked through IS-links to the representation of some column in some table in the databases (information sources). For example, `name` of `Airport` has a link from `aport_nm` of `AFSC:Airport`. In addition, every database table is linked to a single concept at the domain level. For example, `AFSC:Airport` is linked to `Airport`. These IS-links are shown by the dashed arrows. Note the differences between this model and the one shown in Figure 2: there is no relation between `Airport` and `Port` in this minimal model, and the column `AFSC:Airport.glc_cd` is linked to `geoloc-code` of `Airport`. Furthermore, the role `runway-of` is not in the minimal model, because there are no IS-links to it.

In the two subsections that follow, we present reformulation operators used by SIMS that enable information sources to be chosen for a query against the minimal model, thus attempting to ensure no loss of data. We will also show that the minimal models and these operators are sufficient to support all queries equivalent

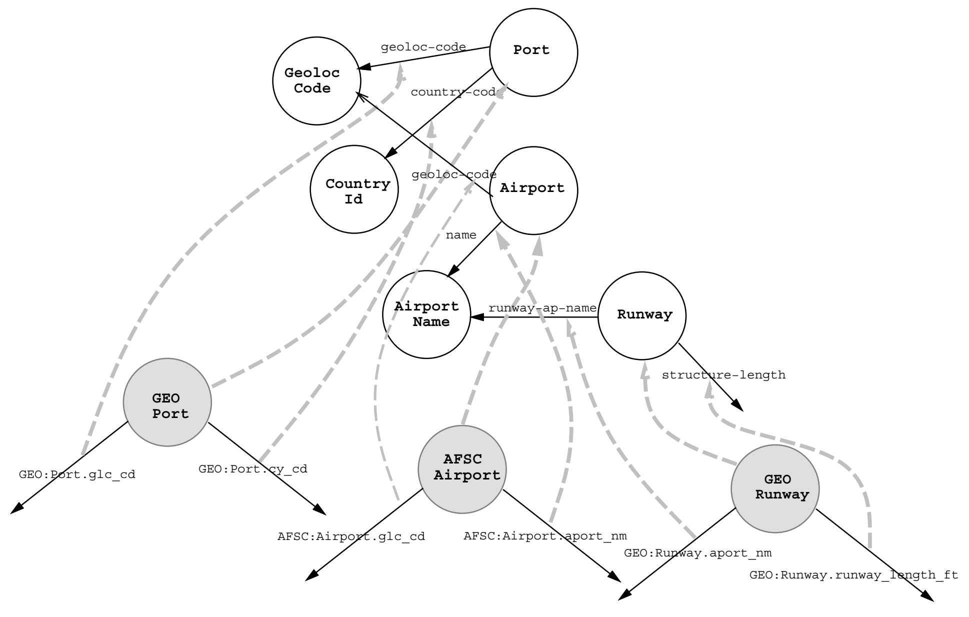

Figure 4. An Example of a Minimal Model

to those that can be posed in distributed SQL (assuming that the information sources being modeled are relational databases). This attempts to insure no loss of expressive power.

The functionality supported by the following two operators is roughly equivalent to that available in [21]. Levy *et al.* treat the problem as one of integrating materialized relations – each could represent an information source – into a view represented by a query. This work shows that the complexity of the problem is NP-complete (or worse in the case of a more expressive language).

3.3. The Choose-Source Operator

Let us consider the following distributed SQL query, Q1:

```
Q1: SELECT A.glc_cd, B.runway_length_ft
    FROM AFSC:Airport A, GEO:Runway B
    WHERE A.aport_nm = B.aport_nm
```

where AFSC and GEO are two different databases. To support such a query in SIMS, what is needed is a minimal model for these databases, containing domain-level concepts corresponding to the tables in them, and the ability to *choose* the corresponding tables and replace the domain concepts used in the query with suitable references to the corresponding tables.

The step of choosing the appropriate information source corresponding to a domain-level term used in a query is performed by an operator called `choose-source`.

For example, using the minimal model shown in Figure 4, the domain query that is equivalent to the query Q1 is:

```
(retrieve (?code ?length)
     (and (Airport ?c1)
          (geoloc-code ?c1 ?code)
          (name ?c1 ?v)
          (Runway ?c2)
          (structure-length ?c2 ?length)
          (runway-ap-name ?c2 ?v)))
```

We consider each SIMS query to be a composition of *clusters* of clauses. A cluster is the collection of clauses that refer to some variable and to any constraints upon it. A cluster is a *domain cluster* if at least some of its clauses are at the domain level. A cluster is an *information source cluster* if all its clauses are at the information source level. For example, there are two domain clusters in the above domain query. The first contains all the references to the variable `?c1`: its declaration as a member of the class `Airport` and the further specifications concerning the values of its roles `geoloc-code` and `name`. The second cluster contains all the references to `?c2`: its declaration as a member of the concept `Runway` and clauses concerning its roles `structure-length` and `runway-ap-name`.

The `choose-source` operator in this example rewrites all the domain clusters in the query as the corresponding information source clusters by following the IS-links that are associated with them. Specifically, the operator rewrites the above query as:

```
(retrieve (?code ?length)
     (and (AFSC:Airport ?c1)
          (AFSC:Airport.glc_cd ?c1 ?code)
          (AFSC:Airport.aport_nm ?c1 ?v)
          (GEO:Runway ?c2)
          (GEO:Runway.runway_length_ft ?c2 ?length)
          (GEO:Runway.aport_nm ?c2 ?v)))
```

When this query is sent to a wrapper for a relational database, it is translated into a query that is equivalent to the SQL Q1 listed at the beginning of this subsection. Note that the operator preserves the meaning of the domain query because it simply rewrites the domain concepts and roles using their corresponding, semantically equivalent, database tables and columns.

In the simple example above, each domain cluster is mapped through IS-links to a single information source cluster. In general, however, more than one information source may contain the requested data. In that case, one domain cluster may potentially map to several different information sources. In such a situation the operator generates a set of reformulated queries, each one corresponding to one of the possible mappings to information source clusters. For example, consider a case where, in addition to `AFSC:Airport`, there is another table in a database called `DB4` that contains the same information as `AFSC:Airport`. Let it be `DB4:Airport`. It, too, will be linked to the domain concept `Airport`. Then, the above domain query will be rewritten by the operator into two equivalent queries:

```
(retrieve (?code ?length)
     (and (AFSC:Airport ?c1)
          (AFSC:Airport.glc_cd ?c1 ?code)
          (AFSC:Airport.aport_nm ?c1 ?v)
          (GEO:Runway ?c2)
          (GEO:Runway.runway_length_ft ?c2 ?length)
          (GEO:Runway.aport_nm ?c2 ?v)))


(retrieve (?code ?length)
     (and (DB4:Airport ?c1)
          (DB4:Airport.glc_cd ?c1 ?code)
          (DB4:Airport.aport_nm ?c1 ?v)
          (GEO:Runway ?c2)
          (GEO:Runway.runway_length_ft ?c2 ?length)
          (GEO:Runway.aport_nm ?c2 ?v)))
```

The SIMS planner will consider both options and ultimately choose based on other information available, such as resource constraints, resource availability, and querying costs. For example, if the AFSC database is not available at the moment, SIMS will choose the second query to get the desired information from DB4.

The algorithm of the `choose-source` operator is as follows. It first checks if all the domain clusters in the query have IS-links to some information source. If so, it then replaces the clusters with the corresponding information source clauses. Using terminology standard in the AI Planning field, the algorithm can be defined as follows:

Operator		**Choose-Source**
Preconditions:	1.	For every domain cluster φ in the query, there is an information source Σ to which all clauses in φ have IS-links.
Actions:	1.	Replace every φ with its corresponding clauses using terms from Σ.

3.4. The Decompose Operator

The `choose-source` operator described in the previous section is designed to deal with situations in which an entire domain cluster can be mapped to a single information source through IS-links. But users will probably wish to write queries containing domain clusters that have roles intended for mapping to different information sources. That will enable users, for example, to request airports that satisfy a whole list of constraints, even if the attributes referred to in the constraints are not all present in a single database.

Queries concerning such a concept will have to be "decomposed" into several subqueries, each referring only to attributes present in a single information source – the original domain cluster will have to be decomposed into a set of new domain clusters. While this is done, additional clauses may have to be added to the clusters to make certain that the data retrieved from different information sources still refers to the same objects. This is accomplished by the `decompose` operator.

For example, suppose that the concept `Airport` in Figure 4 has another role called `main-runway-direction`, and it is IS-linked to a column called `runway_direction` of the table `DB4:Airport`. Assume that the `DB4:Airport` table has a key role called `apt_name` and it is IS-linked to the key role `name` of `Airport`. Then for a domain model query as follows:

```
(retrieve (?code ?dir)
     (and (Airport ?a)
          (geoloc-code ?a ?code)
          (main-runway-direction ?a ?dir)))
```

the `choose-source` operator is not directly applicable because there is no single information source to which the entire `Airport` cluster can be mapped. However, the `decompose` operator can rewrite the query as follows:

```
(retrieve (?code ?dir)
     (and (Airport ?a1)
          (geoloc-code ?a1 ?code)
          (name ?a1 ?n)
          (Airport ?a2)
          (main-runway-direction ?a2 ?dir)
          (name ?a2 ?n)))
```

Now the `choose-source` operator can apply and rewrite the new query into a query that can be executed over the two databases AFSC and DB4. Note that at the domain level, the two new clusters are joined on the key role `name`. But at the information source level, `?a1`'s `name` will be mapped onto `aport_name` of `AFSC:Airport`, while `?a2`'s `name` will be mapped onto `apt_name` of `DB4:Airport`.

The `decompose` operator acts as follows. It finds a cluster containing a concept C that needs to be decomposed, and a group β of roles in the cluster that have IS-links to an information source concept C_i. It then constructs a new cluster using a new concept clause (C $?c_i$) and the roles in β. This new cluster is joined with the rest of the original cluster through a key of C that has an IS-link to C_i:

Operator		**Decompose**
Preconditions:	1.	The query has an unmapped cluster φ that contains a concept clause $(C\ ?c)$.
	2.	The concept C and a group β of roles in φ have IS-links to an information source concept C_i.
	3.	C has a key role R, C_i has a key role R_i, and there is a IS-link between R and R_i.
Actions:	1.	Introduce a new concept clause $(C\ ?c_i)$.
	2.	Replace the concept variable $?c$ in the roles in β by $?c_i$.
	3.	Insert the roles $(R\ ?c\ ?v)$ and $(R_i\ ?c_i\ ?v)$ to join c and c_i.

The `decompose` operator allows the user more flexibility in writing a query. From the user's point of view, queries are more "concept-oriented". They can be written without concern for which information source each role will be retrieved from.

3.5. The Augmented Domain Model

In the last two sections we have been considering queries against a minimal domain model. The minimal model alone suffices to provide SIMS with the querying capability of distributed SQL. However, we wish to support the incorporation of any number of additional concepts and roles into the domain model. Enough new concepts and roles should be defined as are needed to support convenient formulation of queries that might be of use to a human or computer system performing some task in the domain. To support this *Natural Closure* of the model, however, more operators must be added to SIMS as well.

To illustrate one of the many ways in which new concepts may be defined in Loom to augment the minimal model, consider the augmented model in Figure 5. We define the new role `runway-of`, for example, as holding between an airport and a runway if and only if there exists some airport name that is shared by the airport (through `name`) and the runway (through `runway-ap-name`). Note that although the role `runway-of` has no IS-links, the two roles used to define it (`name` and `runway-ap-name`) have IS-links to some database attributes.

The following two subsections, Section 3.6 and Section 3.7, introduce the operators needed to handle newly defined Loom concepts and roles. They are grouped into two general types according to whether the newly defined term itself is explicitly used in the query, or whether the definition of the term is in the query. In both cases, the possibility of substituting the relevant clauses in the query must be inferred.

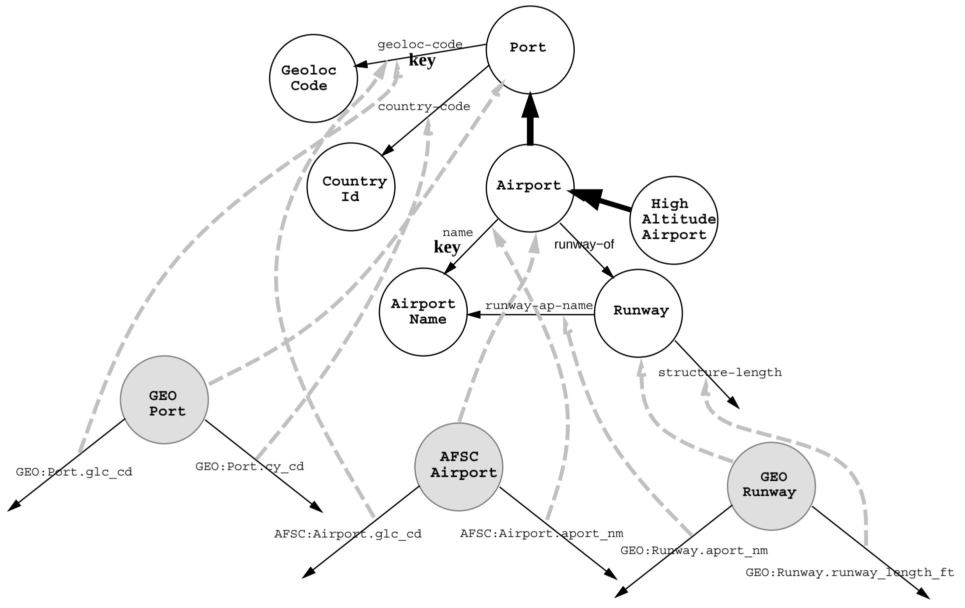

Figure 5. An Augmented Model

3.6. The Substitute Operators

There are two operators that use definitions in reformulating queries: the `substitute-by-definition` operator replaces roles or concepts in the query by their explicit definitions, and the `substitute-by-partition` operator replaces a concept in the query by its direct subconcepts in the hierarchy if they form a partition.

3.6.1. Substitute-by-Definition

A role or a concept can be defined explicitly in Loom using a set of clauses. For example, the role `runway-of` is defined as follows:

```
(defrole runway-of
    :is (:satisfies (?a ?r)
          (for-some (?name)
              (:and (Airport ?a)
                    (name ?a ?name)
                    (Runway ?r)
                    (runway-ap-name ?r ?name)))))
```

To illustrate how a role's definition is used, consider the following domain query:

```
(retrieve (?ap ?rw)
   (and (Airport ?ap)
        (runway-of ?ap ?rw)))
```

Based on the existence of the definition of `runway-of`, the `substitute-by-definition` operator will rewrite the query as:

```
(retrieve (?ap ?rw)
   (and (Airport ?ap)
        (name ?ap ?name1)
        (Runway ?rw)
        (runway-ap-name ?rw ?name1)))
```

In a similar fashion, the operator can also replace a concept by its definition, if a model-defined concept exists in the query. For example, suppose we define the new concept `High-Altitude-Airport` as follows:

```
(defconcept High-Altitude-Airport
   :is (:and Airport
             (> altitude 5000)))
```

In other words, an airport belongs to the class `High-Altitude-Airport` if it is located at an altitude above 5000 feet. Given the following domain query:

```
(retrieve (?name)
   (and (High-Altitude-Airport ?hap)
        (name ?hap ?name)
        ...))
```

the `substitute-by-definition` operator will rewrite it as:

```
(retrieve (?name)
   (and (Airport ?hap)
        (altitude ?hap ?altitude1)
        (> ?altitude1 5000)
        (name ?hap ?name)
        ...))
```

The complete specification of the operator is as follows:

Operator		**Substitute-by-Definition**
Preconditions:	1.	The query contains a clause X (a concept or a role) that has an explicit definition α.
Action:	1.	Replace X by α with appropriate variable substitutions.

3.6.2. Substitute-by-Partition

In addition to definitions of concepts and roles, concepts may also have covering definitions. For example, a concept C is equivalent to the union of all its direct subconcepts $\{C_1, ..., C_n\}$, if they form a *complete coverage*. The `substitute-by-partition` operator is designed to use this type of knowledge, and it replaces a concept in the query by its direct subconcepts in the hierarchy if they form a partition.

To illustrate this, consider the following domain query:

```
(retrieve (?name)
   (and (Port ?p)
        (name ?p ?name)
        ...))
```

Notice that the role `name` has no IS-links when associated with the concept `Port`, and neither of them is explicitly defined in the model. However, the `Port` concept has two subconcepts, `Airport` and `Seaport`, and in this case, they happen to form a partition of `Port` (see Section 2.1). So the operator replaces the appropriate clauses in the query by a union of two new subqueries:

```
(retrieve (?name)
 (union (and (Seaport ?p)
          (name ?p ?name)
          ...)
        (and (Airport ?p)
          (name ?p ?name)
          ...)))
```

Both of these subqueries will ultimately be reformulated further to obtain the `name` from the appropriate information sources.

The complete specification of this partition operator is as follows:

Operator		**Substitute-by-Partition**
Preconditions:	1.	The query contains a cluster $\varphi \equiv (C\ ?v) \wedge \alpha$, where α is the set of C's roles and constraints.
	2.	The concept C has a set of n direct subconcepts, $C_1, ..., C_n$, that form a partition of C.
Action:	1.	Replace the cluster φ by $\bigcup\{(C_i\ ?v) \wedge \alpha\}$, where $1 \leq i \leq n$.

3.7. The Infer-Equivalences Operators

There are two operators that rewrite clauses using concepts and role constraints to infer equivalence to an alternative query. The first, `generalize-with-join`, is used when it is possible to infer that a superconcept along with a set of constraints is equivalent to some concept used in the query. The second, **`specialize-using-constraints`**, is used when it is possible to infer that a concept along with constraints present in the query is equivalent to some subconcept. Both operators preserve the semantics of the original query.

3.7.1. Generalize-With-Join

The **`generalize-with-join`** operator handles cases where instances of a concept are identifiable among those of a superconcept by the fact the values of some of their roles are constrained in a particular way. The operator serves, in effect, to specify a *join* between the subconcept and the superconcept.

For example, consider the following query:

```
(retrieve (?cc ?name)
   (and (Airport ?aport)
        (country-code ?aport ?cc)
        (name ?aport ?name)))
```

Since the role of `country-code` has an IS-link only when it is associated with the `Port` concept and not the `Airport` concept (i.e., it corresponds to an attribute in a database of ports, not specifically airports), the `Airport` concept in the query must be generalized to `Port` and a join constraint added between these two concepts to preserve the semantics of the query.

The procedure for applying this operator is as follows. Given a domain concept C and a role on it R (e.g., `country-code` above), go up the concept hierarchy to find a superconcept of C, C', that has the desired role R with an IS-link. Introduce a new cluster of C' with the role R and additional constraints on a key of C and C' to join the two concepts (C inherits this key from C'). In our current example, the key they are joined over is `geoloc-code`. So the reformulated query is as follows:

```
(retrieve (?cc ?name)
   (and (Airport ?aport)
        (geoloc-code ?aport ?gc)
        (name ?aport ?name)
        (Port ?port)
        (country-code ?port ?cc)
        (geoloc-code ?port ?gc)))
```

Notice that even though a part of this new query requests the `country-code` for *all* ports (the concept `Port` also includes `Seaport`), the returned information will only be for airports because the constraints `(geoloc-code ?port ?gc)` and `(geoloc-code ?aport ?gc)` will filter out any port that is not an airport.

The algorithm for inferring a superconcept by introducing a join constraint as follows:

Operator		**Generalize-With-Join**
Preconditions:	1.	The query contains a cluster: $(C\ ?c) \wedge (R\ ?c\ ?v) \wedge \alpha$.
	2.	There exists a superconcept $C' \supset C$ that is in the domain of R and has a key role κ.
Actions:	1.	Add a new concept clause $(C'\ ?u)$.
	2.	Replace the role $(R\ ?c\ ?v)$ by $(R\ ?u\ ?v)$.
	3.	Add the join roles $(\kappa\ ?c\ ?w) \wedge (\kappa\ ?u\ ?w)$, where $?w$ is a new variable.

3.7.2. Specialize-Using-Constraints

In addition to introducing superconcepts constrained by joins, it may be possible to infer the possibility of replacing a concept used in a query with some subconcept of it, given constraints already present in the query. For example, in our domain model the concept `Port` has two subconcepts: `Airport` and `Seaport`. Suppose we are given the following domain query:

```
(retrieve (?n ?d)
   (and (Port ?p)
        (name ?p ?n)
        (depth ?p ?d)))
```

Notice that no information source contains depth information on ports. Nevertheless, since the `depth` role is defined only on one of the subconcepts of `Port` – `Seaport` – this query can be specialized as follows:

```
(retrieve (?n ?d)
   (and (Seaport ?p)
        (name ?p ?n)
        (depth ?p ?d)))
```

This specialization is safe to perform since the existing clauses `(Port ?p)` and `(depth ?p ?d)` already tell us that the query concerns only ports that have the depth role, i.e., seaports. In other words, there is a "semantic equivalence" between `(Port ?p)` with `(depth ?p ?d)` and the specialization `(Seaport ?p)`. The operator excludes all other specializations of the port concept, such as airport, because they do not possess the constraining role `(depth ?p ?d)`.

Sometimes the constraints on the concept in the query may satisfy the definition of a subconcept. For example, in the following query:

```
(retrieve (?name)
   (and (Airport ?p)
        (altitude ?p ?alt)
        (> ?alt 6000)
```

```
                (name ?p ?name)
                ...))
```

the set of clauses `((Airport ?p) (altitude ?p ?alt) (> ?alt 6000))` is subsumed by the definition of `High-Altitude-Airport` (Loom's subsumption algorithm is used by SIMS to support this type of reasoning). It is thus correct to replace the concept `Airport` by its subconcept `High-Altitude-Airport`:

```
(retrieve (?name)
   (and (High-Altitude-Airport ?p)
        (altitude ?p ?alt)
        (> ?alt 6000)
        (name ?p ?name)
        ...))
```

The effect of this operator is to replace a concept clause (possibly along with some constraints present in the query) with an "equivalent" subconcept. This equivalence may be implied by the mere presence of the roles used in the constraint, or it may be due to the specific nature of the constraints (e.g., subsumed by a concept definition). In either case, the reformulated query is equivalent to the original. The complete specification of this operator is as follows:

Operator		**Specialize-Using-Constraints**
Preconditions:	1.	The query contains a cluster $(C\ ?c) \wedge \alpha$, where α is a set of roles of C.
	2.	A role $(R\ ?c\ ?v)$ in α is defined only on a subconcept $C' \subset C$, **or** the cluster $(C\ ?c) \wedge \alpha$ is subsumed by a subconcept $C' \subset C$.
Action:	1.	Replace $(C\ ?c)$ by $(C'\ ?c)$, and remove those role clauses that are implied by the definition of C'.

4. Using the Query Reformulation Operators

In this section we describe how the reformulation operators are used to process a query. We first present our approach to dynamically selecting information sources to answer a given query, and then we present a complete reformulation of an example query.

4.1. The Reformulation Process

A domain query is expressed using the terms of the domain model, and the system must select an appropriate set of information sources to retrieve the data. This requires that the original query be converted into one or more queries that use only terms of the information-source models. Transformation of the domain-level query into a set of source-level queries is performed using the reformulation operators presented in the previous section. Each of these operators is used either to transform one set of domain-level terms into another set of domain-level terms, or to replace domain-level terms with information-source-level terms. All of the reformulation operators are semantics preserving, so the final query and every intermediate query will have the same semantics as the original query.

When applied to a given query, the reformulation operators produce a set of possible information-source queries. A number of possible reformulations may be applied to any given query. Since some reformulations will result in a better overall plan than others, the system must consider these alternative sequences of reformulations. At each point in the search process, the system selects the most promising intermediate query (described below) and applies the possible reformulation operators to it. This process is repeated until every domain-level term has been replaced by one or more information-source-level terms or the space of reformulations has been exhausted. In the latter case, SIMS determines that there is no way to obtain the requested information given the available information sources.

The space of possible reformulations may be very large. In order to constrain the search for suitable operators, the system considers reformulations only for domain-level terms that do not have a direct mapping to an information source. For example, a query about ports may be reformulated into one about airports if there is no database that contains information about ports. On the other hand, the system would not even attempt to reformulate that portion of the query if there were at least one database that contained all of the required information about ports.

In earlier versions of SIMS [3], the system would completely reformulate a domain query into an information-source query before generating the query access plan. In that case, the system would search for a reformulation of the query that required the smallest number of reformulation operators. The problem with this approach is that the shortest reformulation will not necessarily result in the lowest cost plan. In the current version of SIMS, reformulation and query access planning are tightly integrated. This means that the system reformulates queries and generates the query access plan all within the same search process [18].

The SIMS query planning process is described in [18] and will only be briefly reviewed here in order to describe how query reformulation fits within query planning. Traditional query processors determine the operations and the ordering on the operations for producing the requested set of data. In SIMS, the query processor determines the operations and orderings and also chooses the information sources to be used to answer a given query using the reformulation operators. In addition to the reformulation operators described previously, query processing also requires a set of data manipulation operators: `move`, for moving a set of data from one information source to another; `join`, for combining two sets of data into a combined set; `select`, for selecting a subset of the data; `assignment`, for deriving a new attribute; `union`, for merging sets of data; and `compute`, for performing additional processing in Loom that is not supported by either the remote information source or any of the other data manipulation operators (e.g., disjunction, group-by, aggregation, set-difference). Both the data manipulation operators and the query reformulation operators are specified in a general operator language and are used in a general-purpose planner called SAGE [18], which is built on the UCPOP [4] planning system.

The advantage of integrating the query planning and reformulation process is that the system can now generate estimates on the cost of processing partially constructed plans. This information is used within a branch-and-bound search to find the lowest cost plan to implement a query. The system uses statistics on the size of concepts and cardinality of attributes and assumes a uniform distribution of the data to estimate the amount of data that will be manipulated by each operation. These estimates can then be used to estimate the overall cost of each query plan, which is used to guide the search process. This approach is more efficient than other approaches to query processing that enumerate the set of possible query plans and then compare their costs [28].

4.2. An Example Query Plan

Suppose we are given the following query, which requests the names of the airports where a C5 aircraft can land in a given country.

```
(retrieve (?pname)
     (:and (Airport ?aport)
           (country-code ?aport "IT")
           (primary-port-name ?aport ?pname)
           (runway-of ?aport ?rway)
           (structure-length ?rway ?rlength)
           (Military-Transport-Aircraft ?acraft)
           (vehicle-type-name ?acraft "C-5")
           (wartime-min-runway-avg-landing ?acraft ?landlength)
           (>= ?rlength ?landlength)))
```

Figure 6 shows the complete query plan produced by the system for this query. This plan contains both data manipulation operations and reformulation operations, but only the former will actually be executed. The reformulation operations are just used in the process of producing the plan. In this case the system would retrieve data about the airports and runways from one information source and the information about the C-5 from another information source, and bring it all into the local system. It would then join this information over the runway length and landing length and move the result to the output, i.e., return it to the user.

The plan shown here is the final one selected for processing. A number of intermediate plans were considered during processing and this one was selected because it had the lowest estimated overall execution cost. In the remainder of this section we describe the details of the reformulation operators used to produce this query plan.

The query processor starts with the query posed by the user and transforms it into subqueries to individual information sources. The operators shown in Figure 6

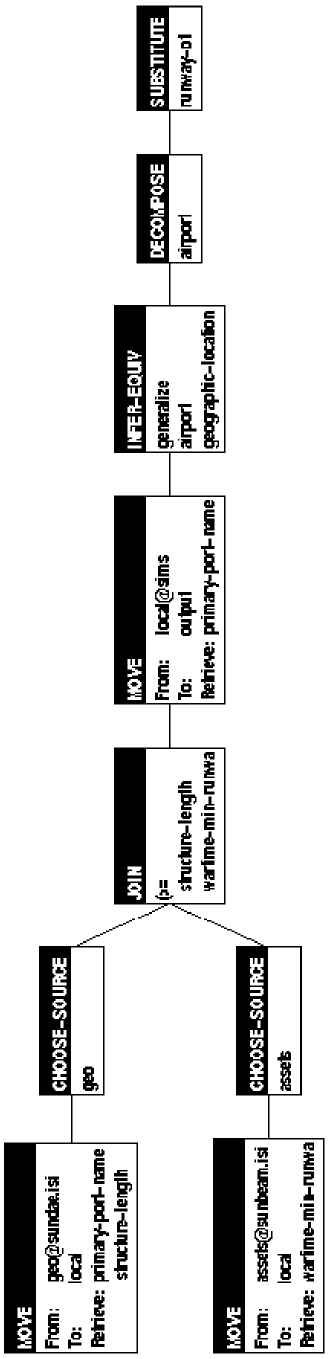

Figure 6. A Screen Shot of a Query Plan Produced by SIMS

are generated by working backward from the query, so we will discuss them from right to left. The plan is executed in the opposite order, starting with retrieving data from the information sources and ending with presenting the data to the user. The `substitute-by-definition` operator is considered first with the definition of `runway-of` since that relation is not linked to any information source. This action replaces `runway-of` with its definition, and the result is shown with the changes underlined:

```
(retrieve (?pname)
     (:and (Airport ?aport)
           (country-code ?aport "IT")
           (primary-port-name ?aport ?pname)
           (Runway ?rway)
           (runway-ap-name ?rway ?pname)
           (structure-length ?rway ?rlength)
           (Military-Transport-Aircraft ?acraft)
           (vehicle-type-name ?acraft "C-5")
           (wartime-min-runway-avg-landing ?acraft ?landlength)
           (>= ?rlength ?landlength)))
```

Next, the `decompose` operator is applied to the `Airport` cluster. Since only `primary-port-name` has an `IS-link`, the operator replaces this part of the domain cluster and introduces a join constraint using the key `geoloc-code`.

```
(retrieve (?pname)
     (:and (Airport ?aport)
           (country-code ?aport "IT")
           (geoloc-code ?aport ?key119)
           (Airport ?aport120)
           (primary-port-name ?aport120 ?pname)
           (geoloc-code ?aport120 ?key119)
           (Runway ?rway)
           (runway-ap-name ?rway ?pname)
           (structure-length ?rway ?rlength)
           (Military-Transport-Aircraft ?acraft)
           (vehicle-type-name ?acraft "C-5")
           (wartime-min-runway-avg-landing ?acraft ?landlength)
           (>= ?rlength ?landlength)))
```

The next step is to find a source for airports that contains country-code information. In this particular case, the `generalize-with-join` operator generalizes `Airport` to `Geographic-Location` and introduces a join constraint on the key of `Geographic-Location` and `Airport`. The key role used in this case is `geoloc-code`. The result is as follows:

```
(retrieve (?pname)
     (:and (Geographic-Location ?geographic-location#122)
           (geoloc-code ?geographic-location#122 ?key119)
           (country-code ?geographic-location#122 "IT")
           (Airport ?aport120)
           (primary-port-name ?aport120 ?pname)
           (geoloc-code ?aport120 ?key119)
           (Runway ?rway)
           (runway-ap-name ?rway ?pname)
           (structure-length ?rway ?rlength)
           (Military-Transport-Aircraft ?acraft)
           (vehicle-type-name ?acraft "C-5")
           (wartime-min-runway-avg-landing ?acraft ?landlength)
           (>= ?rlength ?landlength)))
```

Since all the clauses may directly into at least one information source, the system now considers the data-manipulation operators. In this case no single information source contains all of the required information, so the `move` operator is selected to move the data from the local system to the output. In order to get the information into the local system, the planner selects the `join` operator to combine the data over the runway length and landing length. This in turn requires retrieving two smaller sets of data, shown by the upper and lower branches in the plan.

The set of data to be retrieved in the upper branch of the plan corresponds to the information on the C-5 aircraft and is described by the following subquery:

```
(retrieve (?landlength)
     (:and (Military-Transport-Aircraft ?acraft)
           (vehicle-type-name ?acraft "C-5")
           (wartime-min-runway-avg-landing ?acraft ?landlength)))
```

The `choose-source` reformulation operator maps the subquery into the `ASSETS` database and replaces the domain-level terms with the corresponding terms used in that information source.

```
(retrieve (?landlength)
     (:and (ASSETS:Aircraft_Airfield_Chars ?acraft)
           (ASSETS:Aircraft_Airfield_Chars.ac_type_name ?acraft "C-5")
           (ASSETS:Aircraft_Airfield_Chars.wt_min_avg_land_dist_ft ?acraft
                                                          ?landlength)))
```

The `move` operator is inserted into the plan to indicate that this query must be sent to the `ASSETS` information source and the result brought into the local system for additional processing.

The lower branch of the query plan is handled similarly. This subquery concerns information on geographic locations, airports, and runways:

```
(retrieve (?pname ?rlength)
     (:and (Geographic-Location ?geographic-location#122)
           (geoloc-code ?geographic-location#122 ?key119)
           (country-code ?geographic-location#122 "IT")
           (Airport ?aport120)
           (primary-port-name ?aport120 ?pname)
           (geoloc-code ?aport120 ?key119)
           (Runway ?rway)
           (runway-ap-name ?rway ?pname)
           (structure-length ?rway ?rlength)))
```

The `choose-source` operator is used to select the `GEO` database for this data and translate the terms into those used in that information source.

```
(retrieve (?pname ?rlength)
     (:and (GEO:Geoloc ?geographic-location#122)
           (GEO:Geoloc.glc_cd ?geographic-location#122 ?key119)
           (GEO:Geoloc.cy_cd ?geographic-location#122 "IT")
           (GEO:Airports ?aport120)
           (GEO:Airports.aport_nm ?aport120 ?pname)
           (GEO:Airports.glc_cd ?aport120 ?key119)
           (GEO:Runways ?rway)
           (GEO:Runways.aport_nm ?rway ?pname)
           (GEO:Runways.runway_length_ft ?rway ?rlength)))
```

Finally, the `move` operator is inserted to execute this query against this information source and bring the results into the local system.

Notice that the query produced by the query processing is still expressed in Loom query syntax. However, each of these queries now corresponds to a single information source and is expressed in terms of the information source model. The problem of translating this query into a query appropriate for the given information source and executing them against the source is handled by a wrapper. In this case the queries must be translated into SQL. We use the Loom Interface Manager (LIM) [24], [25] to perform this translation and execute the queries against an Oracle database.

5. Experimental Results

The SIMS system has been tested and used in several real-world information integration applications, including trauma care information management and transportation information integration. In this section, we give a detailed description of the latter.

The transportation application involves eight relational databases (APPORTN, DEPLOY, FMLIB, UNITCHR, EQPMNT, ASSETS, GEO, and TPFDD) about airports, seaports, and geographical locations. These databases are replicated at ISI and a commercial site accessible over the Internet. The databases are quite large. For example, GEO has 23 tables, the number of columns per table ranges from 2 to 33, and the largest table has 52,687 rows. ASSETS has 20 tables, the number of columns per table ranges from 2 to 40, and the largest table has 4,490 rows.

To integrate these information sources, we have constructed a domain model that contains about 170 concepts. For testing purposes we selected a set of 21 queries; three of which are shown below:

```
List all airports in Italy where a C5 airplane can land:

(retrieve (?pname)
   (:and (Airport ?aport)
         (country-code ?aport "IT")
         (primary-port-name ?aport ?pname)
         (runway-of ?aport ?rway)
         (structure-length ?rway ?rlength)
         (Military-Transport-Aircraft ?acraft)
         (vehicle-type-name ?acraft "C-5")
         (wartime-min-runway-avg-landing ?acraft ?landlength)
         (>= ?rlength ?landlength)))

List all Italian airports with ramp space for 10 C-141B aircraft at peacetime:

(retrieve (?pname)
   (:and (Airport ?aport)
         (country-code ?aport "IT")
         (aircraft-parking-space ?aport ?appark)
         (primary-port-name ?aport ?pname)
         (Military-Transport-Aircraft ?acraft)
         (vehicle-type-name ?acraft "C-141B")
         (peacetime-ramp-space ?acraft ?acpark)
         (>= (/ (* 9 ?appark) ?acpark) 10)))

List wharves in Naples with container cranes and rail for half of
the wharf by pier name and berth ID

(retrieve  (?pname ?berthid)
   (:and (Seaport ?sport)
         (primary-port-name ?sport "Naples")
         (harbor-of ?sport ?harbor)
         (cargo-pier-of ?harbor ?pier)
         (pier-name ?pier ?pname)
         (cargo-wharf-of ?pier ?wharf)
         (wharf-berth-identifier ?wharf ?berthid)
         (wharf-length ?wharf ?wharflen)
```

```
        (container-crane-qty ?wharf ?cranenum)
        (> ?cranenum 0)
        (wharf-railway-capability ?wharf ?wharfrail)
        (>= ?wharfrail (/ ?wharflen 2))))
```

The average planning time on a Sparc 20 workstation for these 21 queries is 0.46 CPU seconds, with a standard deviation of 0.35. The average planning and execution time for these queries, when all databases are running at ISI, is 0.96 CPU seconds, with a standard deviation of 0.80. Note that the planning of complex queries can be performed in a fraction of a second, which shows that the reformulation process is efficient enough to be practical.

Note also that the merits of SIMS go beyond the efficiency of query execution. SIMS provides users with a simple and uniform way to gather information in a distributed environment. If all these queries were written manually for each individual database, the user would face a difficult task. For example, for the first example query above to be written manually, the following two separate SQL queries for GEO and ASSETS, respectively, must be written and executed:

```
For ASSETS database:

SELECT A.wt_min_avg_land_dist_ft
FROM aircraft_airfield_chars A
WHERE A.ac_type_name = "C-5"

For GEO database:

SELECT D.aport_nm, B.runway_length_ft
FROM runways B, geoloc C, airports D
WHERE C.cy_cd = 'IT' AND
      B.aport_nm = D.aport_nm AND
      C.gld_cd = D.glc_cd
```

after which the user must select those `D.aport_nm` whose values of `B.runway_length_ft` are greater than the values of `A.wt_min_avg_land_dist_ft`. This, of course, assumes that the user even knows where the data is located. Furthermore, SIMS is able to recover automatically from inaccessibility of data sources.

6. Limitations of the Approach and Implementation

We have identified several limitations of the reformulation process, and describe them below. The first two are difficulties stemming from the nature of the problem handled by SIMS, while the last is due to the current state of the SIMS implementation.

6.1. Object Identity

As with any system that obtains information from multiple sources, SIMS faces the problem of determining the identity of retrieved objects. To deal with this, the domain model in SIMS contains information about unique identifiers in the domain (such as a Social Security Number in the case of people), and it is required that each domain class include such a key. In addition, SIMS requires that for each pair of existing identifiers for a class of objects, there exist some information source (or sources) with the unique mapping between them.

For example, a SIMS model of the transportation planning domain might indicate that automobiles are uniquely identified either by Vehicle Identification Numbers or by license plate numbers combined with state of registration. Every information source that contains information about automobiles must then use (at least) one of these keys to identify the automobiles in it, and SIMS must have access to some information source (or combination of sources) that can be used to map one to the other. If we did not require this information, the responses to queries about automobiles would be unreliable. This is a problem that would be faced by *any* system attempting to answer such queries.

6.2. Value Mappings

Different information sources may contain data of the same semantic type, but using different units — such as temperature in degrees Celsius in one case, and Fahrenheit in another. SIMS modeling supports (and *requires*) the identification of the units for values contained in each information source. To be assured of correct responses to queries, a mapping must be available in some information source for every pair of units. This mapping may be in the form of a function, such as one converting between different temperature units, or in some databases or knowledge base; SIMS accepts all of these as information sources. It should be noted that the problem of obtaining such mappings is a complex one in many cases [30].

6.3. Non-Relational Databases

Although we believe that the SIMS approach is applicable to other information sources, in its current implementation SIMS deals only with Oracle relational databases, MUMPS databases (through an MSQL [15] wrapper), Loom knowledge bases, and programs that can be called within our computational environment to generate data. The extension of SIMS to OODBs and flat files would appear to be straightforward, and is planned in the near future.

7. Related Work

There are a variety of approaches to handling distributed, heterogeneous, and autonomous databases [27], [29]. Of these approaches, the tightly coupled federated systems are the most closely related to SIMS in that they attempt to support total integration of all information sources in the sense that SIMS provides. However, building a tightly coupled federated system requires constructing a global schema for the databases to be combined. A given query would then always be mapped to the same combination of queries to the underlying databases. An example of this is the work on Multibase [19], where the basic goals are the same as those of SIMS, but the mapping between the global schema and the local schemas is hard wired.

In contrast, the approach presented in this paper does not attempt to federate a fixed set of databases, but to provide a framework for integrating any number of information sources in a particular domain. SIMS provides a general domain model that can be used to describe all information in the domain. It adds a flexible query processing capability that dynamically reformulates queries to retrieve the requested data. In contrast to previous work, the domain model in SIMS is not specific to a particular group of information sources, nor is there necessarily a direct mapping from the concepts in the model to the objects in the information sources. The appropriate sources for the requested information are selected at run-time based on the specific information requested by the user, the data available in the various sources, and the current availability of these information sources. The combination of semantic modeling and dynamic reformulation of queries provides a much more flexible and easily extensible interface to a possibly changing collection of information sources.

The situation is similar with multidatabase systems such as Pegasus [1] and UniSQL [16]. Pegasus addresses the semantic heterogeneity problem primarily by requiring users/administrators to write specific programs that will reconcile semantic differences. Pegasus does have one kind of reformulation strategy that is similar to SIMS', in which a superclass can be specialized into a collection of its subclasses. For example, a query about *employees* may be mapped to queries about *engineers* and *programmers*, the two classes of employees that are present in two different databases in the system. However, Pegasus does not in general support dynamic mapping as SIMS does.

In UniSQL as well, fixed, unified views of the multiple databases are provided and queries are processed against them. Furthermore, UniSQL considers possible conflicts as arising among tables and/or among attributes – i.e., these are determined by database-to-database comparisons. In SIMS we attempt to address this n^2 problem by providing a single reference vocabulary (the domain model) with which to describe each new information source independently of the others.

Raschid *et al.* [26] provide an interesting alternative approach to interoperability among heterogeneous databases, specifically dealing with relational and object databases. Their system deals with mapping queries against one local schema to queries against the local schema of another database. Despite differences in focus, Raschid *et al.* will have to handle the same type of problems SIMS does. However, at this time their system still assumes a relatively straightforward correspondence between the data organization in the different databases, and handles only relatively simple queries.

The Carnot system [8], [14], similar to SIMS, integrates heterogeneous databases using a knowledge representation system (CYC [20]). Carnot uses a knowledge base to build a set of articulation axioms that describe how to map between the tables and columns in the databases and the concepts in the domain model. Although queries can be made against the domain concepts used in the axioms, as well as the integrated columns and tables [32], the knowledge base itself is no longer used in the processing. In contrast, the domain model in SIMS is an integral part of the system, allowing SIMS to combine information sources dynamically and in novel ways not anticipated at design time.

The Information Manifold [22], like SIMS, provides an approach to dynamically integrating information sources. The system also uses a knowledge representation system (Classic [5]) to integrate the various information sources. Their approach provides an interesting alternative to the one presented in this paper. Instead of using an object model (as in SIMS) for integrating the various information sources, they use an object-relational model, which is essentially the relational model with an object hierarchy on the relations. Their approach to integration is based on view integration [21], and it provides functionality that is equivalent to the operators for reformulating the minimal model in SIMS. However, without extending their modeling and/or processing, they cannot perform the other operators for traversing the augmented domain model (i.e., the substitution and infer-equivalence operators). Another important difference between the Information Manifold and SIMS is that the criterion used for selecting relevant sources in the Information Manifold is based simply on minimizing the number of different sources. In SIMS, source selection is integrated into query access planning, so SIMS will attempt to minimize the overall cost of a query and not just the number of sources accessed.

The *Model-assisted Global Querying System* (MGQS) [7] shares with SIMS the goal of providing users with access to multiple heterogeneous databases while insulating them from the details of the underlying sources of data. MGQS, like SIMS, uses a model (a *metadatabase* in their terminology) to describe the contents, organization, and data management facilities of the data sources they access. There are, however, significant differences between the two systems in the modeling methodology and the reformulation processes. The primary difference between the modeling approaches taken is that SIMS supports a richer language for representing relationships between the data contained in different information sources. For example, (assuming a relational model) one database may contain a *Patient* table with a *Diagnosis* attribute, and another database may have separate tables for patients with each differing diagnosis. The *Diagnosis* attribute is present, in principle, in the second database as well, but not explicitly. MGQS, however, relies on such explicit presence in order to perform joins between databases. The richer language in SIMS motivates the more powerful set of reformulation operators presented in this paper. To the extent that reformulations are performed in MGQS, they are predefined, determined by relationships expressed in the domain model. They are not dynamic in the sense of SIMS' reformulations.

The SIMS approach bears a superficial similarity to *equational rewriting*, e.g., [9]. However, our rewrite rules are not equations in the sense used by such systems. The SIMS equivalent of the right-hand side of an equation is not a *formula* in the representation language we are using – it is a search strategy. While it is possible in theory to enumerate all the equations one could get by applying the search strategy in all cases, that would be contrary to the point of our work and would defeat its purpose. It is precisely to avoid a priori specification of all possible mappings between query formulations that we developed the approach described here.

8. Discussion

In traditional approaches to integrating or federating databases, the user is required to build a global schema and provide specific mappings for each element in the schema to a corresponding information source. Using this approach, if any of the individual information sources change, the specific mappings must be updated and the global schema must be reworked to reflect this change. If a new information source is added, it may require significant effort to refine the global schema appropriately to accommodate the new source.

As described in this paper, SIMS does not use a global schema, but rather uses a domain or enterprise model to describe the information sources, and the integration is performed dynamically for each query. The model is similar to a global schema in that it provides the terminology for querying the information sources and is used to integrate the diverse terminology in these information sources. However, there is one very important difference: SIMS does not assume that the choice of information sources to answer a domain-level query is fixed. This means that a number of information sources can all be used to provide data about a specific class. This places a greater burden on the system in deciding how to process queries. However, this burden is outweighed by the benefit of much greater flexibility and extensibility.

Consider what is involved in adding a new information source to the system. If we have an existing system with the corresponding domain model and a number of information sources, adding a new information source within the existing domain model simply requires providing the model for that information source and relating that source model to the domain model. Since there is no requirement that there be a single, unique mapping from every element of the domain model to a source model, the amount of work required is proportional to the size of the information source to be integrated. In some sense this is the minimal amount of work possible, since there is no way to avoid modeling the contents of an information source if you want to provide access to the data. If the contents of the new information source go beyond what is modeled in the current domain model, the domain model must be extended to cover the new aspects of this information source. However, this work is also bounded by the size of the information source.

Similarly, changing an existing information source is also a fairly constrained process. Since each information source is related only to the domain model, any changes to the information source can be reflected by simply changing the model of the information source and the corresponding links to the domain model. If an information source is deleted entirely, only the model of the information source and its corresponding links must be removed. The rest of the system remains unchanged. Thus, changes to the model to reflect changes in the existing information sources are also bounded by the size of the changes.

The extensibility of SIMS is made possible by two factors. First, the modeling of each information source is independent of all of the other information sources. Second, the query reformulation process dynamically selects the appropriate information sources to handle a query. This makes it possible to add, change, and delete information sources in SIMS with relative ease. Also, as the number of information sources grows larger, the work required to add or change the information sources does not increase.

Although we have made much progress with our approach, many more research problems remain. For instance, we would like to improve our optimization process to consider the reliability and availability of information sources. Thus, when a domain query results in conflicting information, the user should be informed which is more reliable; when a domain query can only be answered partially due to some unavailable information sources, the user should be presented with the results along with information about the nature of the incompleteness. We believe that our model-based approach will give us a great deal of leverage in dealing with these problems.

Notes

1. There is no requirement that every class have a key, but the system would be unable to combine information from related classes unless they shared a key.
2. The construct in Loom is actually defrelation, but we wish to avoid use of the term relation in this context to avoid confusion with its use in database terminology.

References

1. Rafi Ahmed, Philippe De Smedt, Weimin Du, William Kent, Mohammad A. Ketabchi, Witold A. Litwin, Abbas Rafii, and Ming Chien Shan. The Pegasus heterogenous multidatabase system. *IEEE Computer*, pages 19–27, 1991.
2. Yigal Arens, Chin Chee, Chun-Nan Hsu, Hoh In, and Craig A. Knoblock. Query processing in an information mediator. In *Proceedings of the ARPA / Rome Laboratory Knowledge-Based Planning and Scheduling Initiative*, Tucson, AZ, 1994.
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*, 2(2):127–158, 1993.

4. Anthony Barrett, Keith Golden, Scott Penberthy, and Daniel Weld. UCPOP user's manual (version 2.0). Technical Report 93-09-06, Department of Computer Science and Engineering, University of Washington, 1993.

5. R. J. Brachman, A. Borgida, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Resnick. Living with CLASSIC: When and how to use a KL-ONE-like language. In John Sowa, editor, *Principles of Semantic Networks*, pages 401–456. Morgan Kaufmann, San Mateo, CA, 1991.

6. R.J. Brachman and J.G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

7. Waiman Cheung. *The Model-assisted Global Query System.* PhD thesis, Computer Science Department, Rensselaer Polytechnic Institute, Troy, NY, 1991.

8. Christine Collet, Michael N. Huhns, and Wei-Min Shen. Resource integration using a large knowledge base in Carnot. *IEEE Computer*, pages 55–62, December 1991.

9. Jean Gallier, Paliath Narendran, David Plaisted, Stan Raatz, and Wayne Synder. An algorithm for finding canonical sets of ground rewrite rules in polynomial time. *Journal of the ACM*, 40(1):1–16, 1993.

10. Chun-Nan Hsu and Craig A. Knoblock. Reformulating query plans for multidatabase systems. In *Proceedings of the Second International Conference on Information and Knowledge Management*, Washington, D.C., 1993. ACM.

11. Chun-Nan Hsu and Craig A. Knoblock. Rule induction for semantic query optimization. In *Proceedings of the Eleventh International Conference on Machine Learning*, New Brunswick, NJ, 1994.

12. Chun-Nan Hsu and Craig A. Knoblock. Estimating the robustness of discovered knowledge. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, Menlo Park, CA, 1995. AAAI Press.

13. Chun-Nan Hsu and Craig A. Knoblock. Using inductive learning to generate rules for semantic query optimization. In Gregory Piatetsky-Shapiro and Usama Fayyad, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 17. MIT Press, 1995.

14. M.N. Huhns, N. Jacobs, T. Ksiezyk, W.M. Shen, M.P. Singh, and P.E. Cannata. Integrating enterprise information models in Carnot. In *Proceedings of 1993 International Conference on Intelligent and Cooperative Information Systems*, Rotterdam, Holland, May 1993.

15. InterSystems, Cambridge, MA. *Open M/SQL Server User Guide*, RDBMS E.3 edition, 1993.

16. Won Kim and Jungyun Seo. Classifying schematic and data heterogeneity in multidatabase systems. *IEEE Computer*, pages 12–18, 1991.

17. Craig A. Knoblock. Generating parallel execution plans with a partial-order planner. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, Chicago, IL, 1994.

18. Craig A. Knoblock. Planning, executing, sensing, and replanning for information gathering. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Montreal, Canada, 1995.

19. Terry Landers and Ronni L. Rosenberg. An overview of Multibase. In H.J. Schneider, editor, *Distributed Data Bases.* North-Holland, 1982.

20. D. Lenat and R.V. Guha. *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project.* Addison-Wesley, Reading, MA, 1990.

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

22. Alon Y. Levy, Divesh Srivastava, and Thomas Kirk. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems*, 1995.

23. Robert MacGregor. The evolving technology of classification-based knowledge representation systems. In John Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. Morgan Kaufmann, 1990.

24. Donald P. McKay, Timothy W. Finin, and Anthony O'Hare. The intelligent database interface: Integrating AI and database systems. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, MA, 1990.

25. Jon A. Pastor, Donald P. McKay, and Timothy W. Finin. View-concepts: Knowledge-based access to databases. In *Proceedings of the First International Conference on Information and Knowledge Management*, pages 84–91, Baltimore, MD, 1992.
26. Louiqa Raschid, Yahui Chang, and Bonnie J. Dorr. Query transformation techniques for interoperable query processing in cooperative information systems. In *Proceedings of the 2nd International Conference on Cooperative Information Systems (CoopIS-94)*, Toronto, Canada, 1994.
27. M.P. Reddy, B.E. Prasad, and P.G. Reddy. Query processing in heterogeneous distributed database management systems. In Amar Gupta, editor, *Integration of Information Systems: Bridging Heterogeneous Databases*, pages 264–277. IEEE Press, NY, 1989.
28. P. Griffiths Selinger, M.M. Astrahan, D.D. Chamberlin, R.A. Lorie, and T.G. Price. Access path selection in a relational database management system. In *Artificial Intelligence and Databases*, pages 511–522. Morgan Kaufmann, Los Altos, CA, 1988.
29. Amit P. Sheth and James A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, 1990.
30. Michael Siegel. A metadata approach to resolving semantic conflicts. In *Proceedings of the 17th International Conference on Very Large Data Bases*, pages 133–145, Barcelona, Spain, 1991.
31. Sheila Tejada and Craig A. Knoblock. Mapping a relational query language into a knowledge representation query language. Technical report, Information Sciences Institute, University of Southern California, 1995.
32. D. Woelk, W.M. Shen, M.N. Huhns, and P. Cannata. Model driven enterprise information management in Carnot. In *Enterprise Integration Modeling: Proceedings of the First International Conference*, 1992.