To appear in the *International Journal on Intelligent and Cooperative Information Systems.*

RETRIEVING AND INTEGRATING DATA FROM MULTIPLE INFORMATION SOURCES*

YIGAL ARENS
CHIN Y. CHEE
CHUN-NAN HSU
CRAIG A. KNOBLOCK

Information Sciences Institute, University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292, U.S.A.
{ARENS, CHEE, CHUNNAN, KNOBLOCK}@ISI.EDU

ABSTRACT

With the current explosion of data, retrieving and integrating information from various sources is a critical problem. Work in multidatabase systems has begun to address this problem, but it has primarily focused on methods for communicating between databases and requires significant effort for each new database added to the system. This paper describes a more general approach that exploits a semantic model of a problem domain to integrate the information from various information sources. The information sources handled include both databases and knowledge bases, and other information sources (e.g., programs) could potentially be incorporated into the system. This paper describes how both the domain and the information sources are modeled, shows how a query at the domain level is mapped into a set of queries to individual information sources, and presents algorithms for automatically improving the efficiency of queries using knowledge about both the domain and the information sources. This work is implemented in a system called SIMS and has been tested in a transportation planning domain using nine Oracle databases and a Loom knowledge base.

Keywords: Information server, multidatabases, planning, query reformulation, knowledge representation, SIMS

1 Introduction and Related Work

Most tasks performed by users of complex information systems involve interaction with multiple information sources.[1] Examples can be found in the areas of analysis

*The research reported here was supported by Rome Laboratory of the Air Force Systems Command and the Defense Advanced Research Projects Agency under contract no. F30602-91-C-0081. Views and conclusions contained in this report are the authors' and should not be interpreted as representing the official opinion or policy of DARPA, RL, the U.S. Government, or any person or agency connected with them.

[1] By the term *information source* we refer to any system from which information can be obtained. SIMS currently deals with Oracle databases and Loom knowledge bases.

(e.g., of intelligence data or logistics forecasting) and in resource planning and briefing applications. Retrieval of desired information dispersed among multiple sources requires general familiarity with their contents and structure, with their query languages, with their location on existing networks, and more. The user must break down a given retrieval task into a sequence of actual queries to information sources, and must handle the temporary storing and possible transformation of intermediate results — all this while satisfying constraints on reliability of the results and the cost of the retrieval process. With a large number of information sources, it is difficult to find individuals who possess the required knowledge, and automation becomes a necessity.

SIMS[2] accepts queries in the form of a description of a class of objects about which information is desired. This description is composed of statements in the Loom knowledge representation language (Section 1.1.1). The user is not presumed to know how information is distributed over the data- and knowledge bases to which SIMS has access — but he/she *is* assumed to be familiar with the application domain, and to use standard terminology to compose the Loom query. The interface enables the user to inspect the domain model as an aid to composing queries. SIMS proceeds to reformulate the user's query as a collection of more elementary statements that refer to data stored in available information sources. SIMS then creates a plan for retrieving the desired information, establishing the order and content of the various plan steps/subqueries. Using knowledge about the contents and structure of information sources, SIMS reformulates the plan to minimize its expected execution time. The resulting plan is then executed by performing local data manipulation and/or passing subqueries to the LIM system (Section 1.1.2), which generates the final translation into database queries in the appropriate language(s). A graphical user interface enables the user to inspect the plan in its various stages and to supervise its execution.

The SIMS project applies a variety of techniques and systems from Artificial Intelligence to build an intelligent interface to information sources. SIMS builds on the following ideas:

Knowledge Representation/Modeling, which is used to describe the domain about which information is stored in the information sources, as well the structure and contents of the information sources themselves. The domain model is a declarative description of the objects and activities possible in the application domain as viewed by a typical user. The model of each information source indicates the data-model used, query language, network location, size estimates, update frequency, etc., and describes the contents of its fields in terms of the domain model. The user formulates queries using terms from the application domain, without needing to know anything about specific information sources. SIMS' models of different information sources are completely independent, greatly easing the process of incorporating new information sources into the system.

Planning/Search, which is used to construct a sequence of queries to individual information sources that will satisfy the user's query. A planner is used in an initial reformulation step that selects the information sources to be used in answering a query. It is also used to order the queries to the individual information sources,

[2] **S**ervices and **I**nformation **M**anagement for decision **S**ystems.

select the location for processing the intermediate data, and determine which queries can be executed in parallel.

Reformulation/Learning. SIMS considers alternative information sources and queries to them to retrieve the desired information. This search for more efficient query formulations is guided by the detailed semantics provided by the application domain model. Additional knowledge about the contents of the information sources may be learned from the databases and used to reformulate the queries.

An initial prototype incorporating many features of the SIMS approach has been built and applied to the domain of transportation planning — organizing the movement of personnel and materiel from one geographic location to another using available transportation facilities and vehicles [2]. An earlier prototype was applied to information needed for daily Naval briefings given in Hawaii about the status of the Pacific Fleet [1]. The system currently has access to nine Oracle databases and a Loom knowledge base with information about ships, ports, locations, relevant activities, etc. SIMS is controlled via a graphical user interface. It is written in Common Lisp and uses CLIM for its graphics.

There has been some work on the problem of accessing information distributed over multiple sources both in the AI-oriented database community and in the more traditional database community. Work in heterogeneous distributed databases includes the MULTIBASE, MERMAID, NDMS, IISS, IMDAS, ADDS, PRECI* and MRDSM systems. A survey and comparison of these can be found in [24]. Of these systems, only the first four attempt to support total integration of all information sources in the sense that SIMS provides. SIMS is distinguished from work in this community in that a complete semantic model of the application domain is created in a state-of-the-art knowledge representation language with powerful reasoning facilities. The model provides a collection of terms with which to describe the contents of (i.e., to create semantic models of) available information sources — and these include knowledge bases in addition to databases. Furthermore, a sophisticated planning mechanism is used at run-time in order to determine the potentially very complex relationship between the collection of information requested by the user and the data available from the various sources. In contrast to previous work, the domain model in SIMS is neither 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. Our approach thus provides a much more flexible and easily extensible interface to a possibly changing collection of information sources.

The AI-oriented database community has done work on various aspects of using a knowledge base to integrate a variety of information sources. The Carnot project [9] integrates heterogeneous databases using a set of articulation axioms that describe how to map between SQL queries and domain concepts. Carnot uses the Cyc knowledge base [16] to build the articulation axioms, but after the axioms are built the domain model is no longer used or needed. In contrast, the domain model in SIMS is an integral part of the system, and allows SIMS to both combine information stored in the knowledge base and to reformulate queries. Illarramendi et al. [3, 13] present an approach to automatically integrating knowledge-base models from individual relational database schemas. In SIMS, the integration of the database models is not automated, although the translation of the individual database schemas into knowledge-base models is automated by the LIM system, which is used by SIMS. Elements of the approach described in that work can be applied to further automating the process of database modeling in SIMS. Finally, Papazoglou et al. [22] present a framework for intelligent information systems where, like SIMS, an explicit knowledge model is an integral part of an intelligent information agent.

Some additional related research has been performed by those working on semantic and object-oriented data models, e.g., [8, 12, 26]. Since they are interested in constructing a single DBMS, however, they take an almost diametrically opposed view of the problem from that of SIMS. While SIMS attempts to preserve its independence from the data models of the constituent data- and knowledge-bases, using a planner to bridge this gap at query time, they attempt to *closely integrate* the given data model into their DBMS.

The remainder of this paper is structured as follows. The rest of this section is devoted to overviews, first of the technological infrastructure used by SIMS, and then of the operation of the SIMS system itself. Section 2 follows with a description of the modeling that provides SIMS with the knowledge needed to plan data retrieval. A full description of SIMS' planning and reformulation components is provided in Sections 3, 4, and 5. SIMS' user-interface is described in Section 6. The paper ends with a brief summary and directions for future work, Section 7.

1.1 Technological Infrastructure

This subsection is provided for readers who may not be familiar with the systems underlying SIMS. A general understanding of Loom, LIM, and planners like Prodigy is assumed in the rest of this paper.

1.1.1 Loom

Loom serves as the knowledge representation system SIMS uses to describe the domain model and the contents of the information sources, as well as serving as an information source in its own right. It provides both a language and an environment for constructing intelligent applications. Loom combines features of both frame-based and semantic network languages, and provides some reasoning facilities. As a knowledge representation language it is a descendent of the KL-ONE [4] system.

The heart of Loom is a powerful knowledge representation system, which is used to provide deductive support for the declarative portion of the Loom language. Declarative knowledge in Loom consists of definitions, rules, facts, and default rules. A deductive engine called a *classifier* utilizes forward-chaining, semantic unification and object-oriented truth maintenance technologies in order to compile the declarative knowledge into a network designed to efficiently support on-line deductive query processing. For a detailed description of Loom see [17, 18].

To illustrate both Loom and the form of SIMS' queries, consider Figure 1, which contains a simple semantic query to SIMS. This query requests the value of the depth of the San Diego port. The three subclauses of the query specify, respectively, that the variable `?port` describes a member of the model class `port`, that the relation `port.name` holds between the value of `?port` and the string `SAN-DIEGO`, and that the relation `port.depth` holds between the value of `?port` and the value of the

```
(db-retrieve (?depth)
  (:and (port ?port)
        (port.name ?port "SAN-DIEGO")
        (port.depth ?port ?depth)))
```

Figure 1: Example SIMS/Loom Query

variable `?depth`. The semantic query specifies that the value of the variable `?depth` be returned. A query to SIMS need not necessarily correspond to a single database query, since there may not exist one database that contains all the information requested.

1.1.2 LIM

In Loom the members of a class (e.g., the possible values of the variable `?port` in the expression in Figure 1) are *instances* in the knowledge base. In the case of large-sized realistic domains it is preferable not to define all objects of the domain as knowledge base instances. Instead, databases provide more efficient structures for organizing large numbers of such objects, and DBMSs are more efficient than AI languages for manipulating them.

The Loom Interface Module (LIM) [19] is being developed by researchers at Paramax Systems Corp. to mediate between Loom and databases. LIM reads an external database's schema and uses it to build a Loom representation of the database. The Loom user can then treat classes whose instances are stored in a database as though they contained "real" Loom instances. Given a Loom query for information in that class, LIM automatically generates a query in the appropriate database query language to the database that contains the information, and returns the results as though they were Loom instances. However, LIM focuses primarily on the issues involved in mapping a semantic query to a single database. After SIMS has planned a query and formed subqueries, each grounded in a single database, it hands the subqueries to LIM for the actual data retrieval. SIMS handles direct queries to the Loom knowledge base on its own.

1.1.3 Prodigy

The two problems of selecting information sources and ordering queries can be easily cast as planning problems. SIMS uses Prodigy [6, 21], a means-ends analysis planner, to solve both these problems. Prodigy has an expressive operator and control language and has been linked to Loom, so that it can use the Loom domain model as its model of the world. SIMS formulates the selection of information sources and the ordering of queries as planning problems and hands them off to Prodigy.

A problem is specified in Prodigy by giving the system a set of operators that define the legal operations on a problem and an initial state description that defines the current state of the world. The system is then given a goal, which in this case is the query to be answered, and Prodigy generates a sequence of operators that transforms the initial state into a state in which the goal is satisfied.

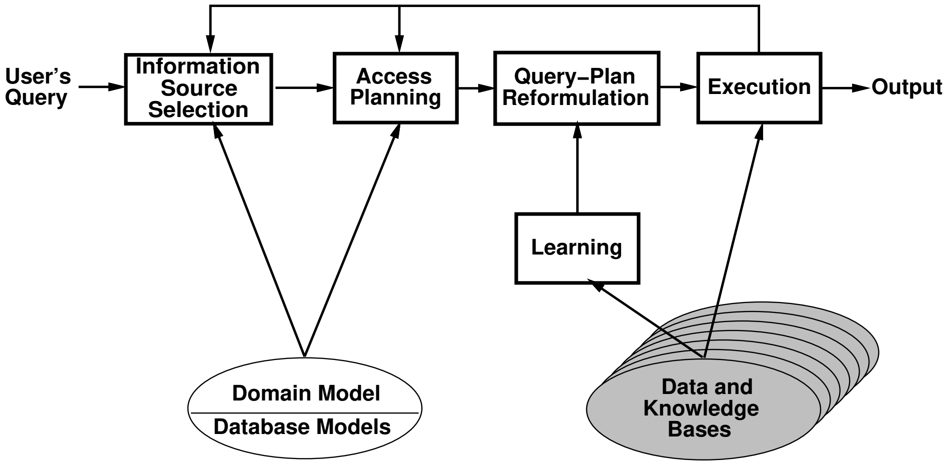

Figure 2: SIMS Overview Diagram.

Prodigy is used for solving the planning problems in SIMS for two main reasons. First, it provides an expressive language for both defining the problem and constructing a set of rules to control the search. Second, it provides a natural framework for planning the operations and monitoring the execution of those operations. In the case of failures, the failure points are easily identified and the system can return to the planner to select an alternative plan for retrieving the data.

1.2 Overview of SIMS

SIMS addresses several problems that arise when one tries to provide a user familiar only with the general domain with access to a system composed of numerous separate data- and knowledge-bases.

Specifically, SIMS deals with the following:

- Determining which information sources contain the data relevant to the knowledge-base classes used in formulating a given query.
- For those classes mentioned in the query which appear to have no matching information source, determining if any knowledge encoded in the domain model (such as relationship to other classes) permits reformulation in a way that will enable suitable information sources to be identified.
- Creating a plan, a sequence of subqueries and other forms of data-manipulation that when executed will yield the desired information.
- Using knowledge about databases to optimize the plan.
- In general, providing a uniform way to describe information sources to the system, so that data in them is accessible.

```
(retrieve (?name)
      (:and (rail_port ?port)
            (port.refrig ?port ?refrig)
            (> ?refrig 0)
            (port.geocode ?port ?geocode)
            (port.name ?port ?name)
            (geoloc ?geoloc)
            (geoloc.country_name ?geoloc "Germany")
            (geoloc.geocode ?geoloc ?geocode)))
```

Figure 3: Example SIMS Query.

A visual representation of the components of SIMS is provided in Figure 2.[3]

An initial Loom query of the kind SIMS handles is shown in Figure 3. The first clause, `(rail_port ?port)`, is a concept expression that constrains the variable `?port` to a set of port objects in the knowledge base. The Loom class `rail_port` (standing for sea ports with rail facilities) need not necessarily correspond to the contents of a specific field in some single information source. If it does not, the planner will have to find some combination of subqueries that will obtain all necessary objects. This case is discussed further later. The second clause is a relation expression that states that the `port.refrig` relation holds between fillers of the variables `?refrig` and `?port`. This clause will bring about the retrieval of possible fillers of `?refrig` — refrigeration facilities in a relevant port. The third clause is a constraint: a ">" relation on the number of refrigeration facilities, requiring it to be a positive integer. The entire query requests the names of all ports with rail facilities and refrigeration facilities whose geographic code designation indicates that they are in Germany.

A fragment of the model describing some of the hierarchy of concepts relevant to this query is presented in Figure 5. In this figure, the circles denote concepts in the knowledge base, the upward arrows indicate *is-a* links, and the other arrows indicate relations between concepts. So, for example, the Port concept has two subconcepts, Sea_Port and Air_Port, and Sea_Port has a subconcept Rail_Port, seaports with a railway capability. Shaded concepts represent those that can be retrieved directly from some database.

If the information about rail ports and geographic locations were stored directly in the Loom knowledge base, then Loom could be used directly to answer this query. But, as the figure indicates, that is note the case. SIMS uses Loom to semantically model a domain about which data is stored in multiple information sources, and the information required to answer this query will be retrieved from the appropriate sources, with the help of LIM where necessary. Thus, if all the referenced information were stored in one database, this query could be passed directly to LIM as is. But that is not the case either.

Data pertaining to this query is spread over two databases — one containing information about ports and the other containing information about geographic locations. The system is handed the query shown in Figure 3 and it must first de-

[3] Work on the links back from the Execution component to Information Source Selection and Access Planning will not be discussed in this paper.

termine which information sources to access. Then it formulates a set of subqueries that can be executed directly by either LIM or Loom to derive the desired result. SIMS can use LIM to return intermediate results, which can then be processed further in Loom. As we will see, the execution of the example query will require three subqueries. One to each of the databases and one to combine the intermediate results obtained from them. The processes described in overview here are discussed in more depth in the remaining sections of the paper.

The very first step in processing a query is to determine where the requested data resides. For instance, inspecting the model fragment in Figure 5 reveals that `rail_port` does not have a directly corresponding database (a shaded concept). However, the model relation `port.rail` can be used to distinguish it from other ports. Specifically, it can identify the desired ports from among those in `sea_port`, which *does* have a corresponding database. This and other reformulations of this nature are described further in Section 3.

The next step in processing the query is to produce a *plan* to implement the required retrieval. By this we mean that SIMS must produce a plan consisting of data-retrieval and data-manipulation specifications, with an associated partial ordering of the specified actions. The data-retrieval steps of the plan must be grounded in specific information sources, i.e., all data one step requests must be contained in a single information source. Any data-manipulation steps of the plan are performed using the Loom reasoning facilities. The plan produced takes the form of a lattice of plan steps.

The steps in a plan are partially ordered based on the structure of the query. This ordering is determined by the fact that some steps make use of data that is obtained by other steps, and thus must logically be considered after them. For example, a plan step may compare two items of data according to some measure. If the data are obtained from two different information sources, then the comparison must come later than the retrievals of the data items.

Next, the plan produced as above is inspected and, when appropriate, data-retrieval steps that are grounded in the same information source are grouped — eventually their execution will result in a single query. We therefore call this process *subquery formation*. The result of this grouping process is a new graph in which each node ultimately corresponds either to a query to some information source, or to internal manipulation by SIMS of data so acquired. The processes involved in subquery formation is described in Section 4.

After a plan for the query has been obtained, the system reformulates the query plan into a less expensive yet semantically equivalent plan. The reformulation is based on logical inference from content knowledge about each of the queried databases. The cost reduction from the reformulated plan is due to the reduction in the amount of the intermediate data and the refinement of each subquery. This *reformulation* process is described in Section 5.

First, however, we discuss our approach to modeling.

2 Domain and Information Source Models

SIMS must reason about data and other knowledge stored in a variety of locations and formats. It is imperative that SIMS have available detailed descriptions of the various information sources to which it has access. This is not merely an artifact of the SIMS approach — no system can retrieve requested information if it does not have knowledge about where the information in question may be stored and how to go about accessing it.

In SIMS a *model* of each information sources is created to describe it to the system. In addition, a *domain model* is constructed to describe objects and actions that are of significance in the performance of tasks in the application domain.[4] The domain model's collection of terms forms the "vocabulary" used to characterize the contents of an information sources.

It is important to note that the models of different information sources are independent of each other. This greatly simplifies the task of modeling, and at the same time enables new components to be added to SIMS without the need for any recompilation process. The planner simply makes use of the new information as appropriate.

2.1 Modeling Information Sources

For each information source, SIMS' model must include every fact that can influence decisions concerning when and whether to utilize it.

- In order to decide whether a query to LIM is necessary or whether processing can be performed locally, the model specifies if the source is a database or a Loom knowledge base (the two types of information sources currently supported);
- In order to decide whether to expend effort reformulating plans and whether to be concerned with the cost of transmitting intermediate data, the model describes the size of databases and tables, and their location;
- In order to help further with decisions concerning reformulation, the model defines key columns in the database, if such exist; and, finally,
- In order to enable SIMS to determine in which information source desired information resides, the model describes the *content* of the information source.

In fact, most of the modeling effort done for SIMS goes to describing the content of databases. These models are used by both LIM and SIMS, for their own respective purposes (cf. [19] for LIM's work on database modeling). Simply put, the model of a database must describe precisely what type of information is stored in it. To do so we choose a key column (or columns) in each table and create a Loom class corresponding to it — the class from which items in that column are drawn. Every other column in the table is viewed as corresponding to a Loom relation —

[4] In fact, all the knowledge described here is stored by SIMS in a single model defined in a uniform way. It is thus only for purposes of exposition that we describe different parts of the model as "separate" models.

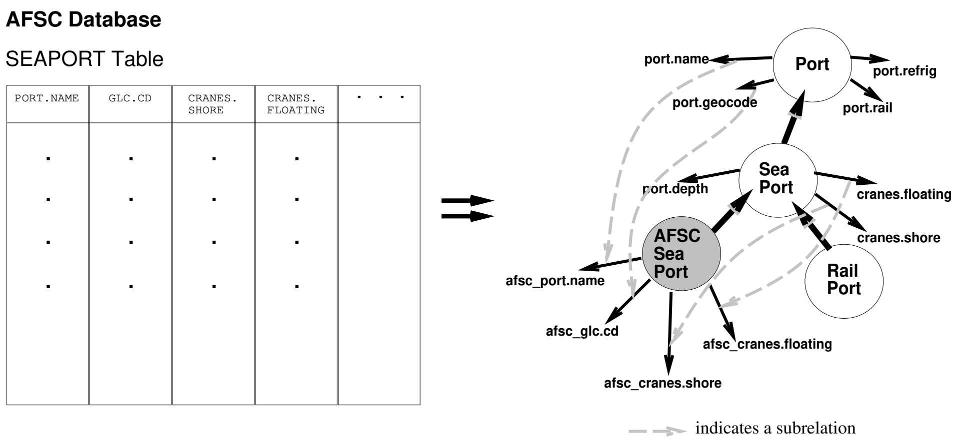

Figure 4: A Model of a Database Table Embedded in the Domain Model.

one describing the relationship between the key item and the one in that column. Figures 4 provides a simple illustration of content modeling.

2.2 Modeling the Domain

SIMS deals with a single "application domain", i.e., with organizing the retrieval of information relevant to some coherent collection of tasks. Currently, the application domain we have selected is the *military transportation planning* domain — tasks involving the movement of personnel and materiel from one location to another using aircraft, ships, trucks, etc.

SIMS' model of the application domain includes a hierarchical terminological knowledge base with nodes representing all objects, actions, and states possible in the domain. In addition, it includes indications of all relationships possible between nodes in the model. For example, there is a node in the model representing the class of ports and a node representing the class of geographic location codes. There is a relation specified between `ports` and `geoloc codes` with a notation indicating that each of the former has precisely one of the latter.

The Loom knowledge representation language is used to describe SIMS' domain model. Statements in Loom are used to express more elaborate relationships among model entities, such as that rail-ports are sea-ports which have a rail terminal as well (cf. Figure 5).[5]

The entities included in the domain model are not meant to correspond to any objects described in any particular database. 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. However, the domain model is used, effectively, as the *language* with which to describe the contents of a database to SIMS. This is done by including relations —

[5]We have chosen simple examples for use in this paper. Loom supports far more powerful statements. For a full description see [17, 18].

hierarchical (*is-a*) or others — to precisely describe every aspect of the contents of the database in terms of the domain model (cf. Section 2.1). In order to submit a query to SIMS, the user composes a Loom statement, using terms and relations in the domain model to describe the precise class of objects that are of interest. If the user happens to be familiar with particular databases and their representation, those concepts and relations may be used as well. But such knowledge is not required. SIMS is designed *precisely* to allow users to query it without such specific knowledge of the data's structure and distribution.

The task of accurately relating a database (and other information source) model must be engaged in for every database and knowledge base that SIMS is to be capable of utilizing. SIMS includes a graphical interface that simplifies this process (Section 6).

The modeling work that is a prerequisite for SIMS to be able to access information sources is a substantial effort, the importance of which cannot be overemphasized. The extent to which SIMS can find information and the accuracy of its retrievals are completely dependent on it. The scalability of the modeling process in SIMS is discussed next.

2.3 Scalability and Expandability

SIMS' dependence on models of the domain and the information sources it utilizes requires that the question of its scalability be addressed. Separate issues arise when considering the application domain model and the information resource models.

2.3.1 Expanding the Application Domain Model

A considerable effort must be expended to model the application domain before any use of SIMS is possible. Although this task's extent should not be minimized, it is a relatively tractable one no different than that engaged in in many other areas of artificial intelligence. In fact, it has more clearly defined limits, since full utility is possible from the moment that enough of the model has been built to cover data objects described in desired databases. Any model building beyond that point only increases the expressivity of the query language and adds to the user's convenience, but it still provides access to the same data.

It is reasonable to anticipate that the domain model will have to be incrementally enlarged to accommodate new data sources as they are added to the system. However, since SIMS is designed to handle one domain at a time, it can safely be assumed that this modeling effort will gradually reach closure.

2.3.2 Adding Information Source Models

Additional modeling will have to be engaged in for every new information source added to SIMS. While this need will remain constant as the system grows, the SIMS approach greatly limits the required effort compared to what it potentially might be. Obviously, no approach to this problem can avoid modeling information sources, since without a complete description of the content of a database or knowledge base it is simply impossible to intelligently decide whether or not to attempt to retrieve desired information from it. However, SIMS allows one to model a new information

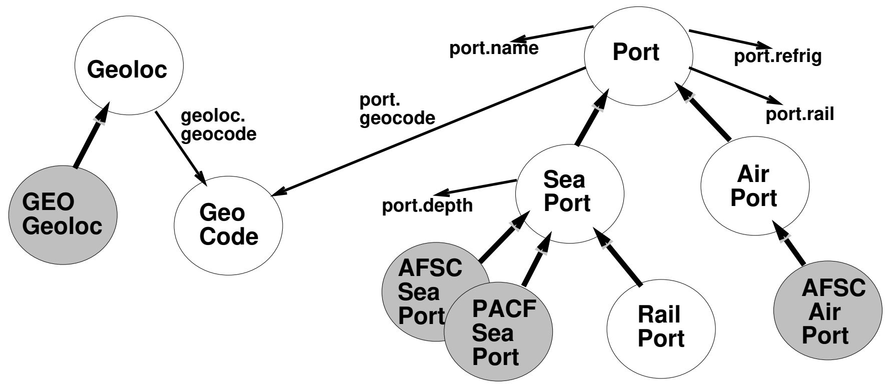

Figure 5: Fragment of Domain Model

source independently of any that are already incorporated into the system. There is no need to try to anticipate interactions or overlaps between different information sources, to decide how joins over databases will be performed, etc., since all such decisions are made at run time by the SIMS planner.

To further simplify any modeling that does have to be performed, the SIMS project includes an ongoing effort to develop modeling aids, among them a graphical Loom knowledge base builder (see Section 6).

3 Selecting Information Sources

The first step in answering a query expressed in the terms of the domain model is to select the appropriate information sources. This is done by mapping from the concepts in the domain model to the concepts in the database models that correspond directly to database information. If the user requests information about ports and there is a database concept that contains ports, then the mapping is straightforward. However, in many cases there will not be a direct mapping. Instead, the original domain-model query must be reformulated in terms of concepts that correspond to database concepts.

Consider the fragment of the knowledge base shown in Figure 5, which covers the knowledge relevant to the example query in Figure 3. The concepts Sea_Port, Air_Port, and Geoloc have subconcepts shown in by the shaded circles that correspond to concepts whose instances can be retrieved directly from some database. Thus, the AFSC database contains information about both seaports and airports and the PACF database contains information about only seaports. Thus, if the user asks for seaports, then it must be translated into one of the database concepts — AFSC_Sea_Port or PACF_Sea_Port. If the user asks for rail-ports, then it must first be translated into a request for sea_ports by augmenting the original query with a constraint that each port must have a railroad capability.

In addition to retrieving data from the databases, data can also be stored in and retrieved from the Loom knowledge base. This knowledge base is simply treated as another information source. However, the Loom KB has the added advantage that information from database queries can be cached in it and the model can be updated to indicate what information has been stored in Loom.

In this section, we describe the set of problem reformulation operations[6] that are implemented in SIMS and the reformulation process used to transform a user's query into one that can be used to retrieve data. We also describe how this reformulation mechanism supports the catching and retrieval of data in Loom.

3.1 Reformulation Operations

In order to select the information sources for answering a query, SIMS applies a set of reformulation operators to transform the domain-level concepts into concepts that can be retrieved directly from databases. The system uses four operators: Select-Database, Generalize-Concept, Specialize-Concept, and Partition-Concept. These reformulation operators are described next.

3.1.1 Select Database

The Select-Database reformulation operator maps a domain-level concept directly to a database-level concept. In many cases this will simply be a direct mapping from a concept such as Sea_Port to a concept that corresponds to the seaports in a particular database. There may be multiple databases that contain the same information, in which case the domain-level concept can be reformulated into any one of the database concepts. In Figure 5, Sea_Port can be transformed into either AFSC_Sea_Port or PACF_Sea_Port. The following example shows how a simple query would be reformulated using AFSC_Sea_Port. In general, the choice is made so as to minimize the number of queries to different databases.[7]

Input Query:
```
(retrieve (?name)
     (:and (sea_port ?port)
           (port.name ?port ?name)))
```

Reformulated Query
```
(retrieve (?name)
     (:and (afsc_sea_port ?port)
           (afsc_port.name ?port ?name)))
```

3.1.2 Generalize Concept

The Generalize-Concept operator uses knowledge about the relationship between a class and a superclass to reformulate a requested concept in terms of a more general concept. In order to preserve the semantics of the original request, one or more additional constraints may need to be added to the query in order to avoid

[6] These are to be distinguished from query-plan reformulation operations, which are described in Section 5.

[7] Currently we assume the databases contain consistent information, so the choice of databases only effects the efficiency of the query and not the accuracy.

retrieving extraneous data. For example, a request for rail ports can be replaced with a request for seaports with the additional constraint that the seaports have a rail capability (i.e. (port.rail ?port "Y")). This is illustrated in the following example.

Input Query:
```
(retrieve (?name)
     (:and (rail_port ?port)
           (port.name ?port ?name)))
```

Reformulated Query
```
(retrieve (?name)
     (:and (sea_port ?port)
           (port.name ?port ?name)
           (port.rail ?port "Y")))
```

3.1.3 Specialize Concept

The Specialize-Concept reformulation operator attempts to replace a given concept with a more specific concept. This is done by checking the constraints on the query to see if there is an appropriate specialization of the requested concept that would satisfy it. Identifying a specialization of a concept is implemented by building a set of Loom expressions representing each concept and then using the Loom classifier to find any specializations of the concept expression.

For example, consider the hierarchy fragment shown in Figure 5 again. Given the query shown below, which requests the ports with a depth greater than 25, the Loom classifier uses the fact that only seaports have a relation that corresponds to port.depth. Therefore, only seaports could possible satisfy the query, and in the original request ports can be replaced with seaports. There are several databases that correspond to seaports, so the requested information can now be retrieved.

Input Query:
```
(retrieve (?name)
     (:and (port ?port)
           (port.name ?port ?name)
           (port.depth ?port ?depth)
           (> ?depth 25)))
```

Reformulated Query
```
(retrieve (?name)
     (:and (sea_port ?port)
           (port.name ?port ?name)
           (port.depth ?port ?depth)
           (> ?depth 25)))
```

3.1.4 Partition Concept

The Partition-Concept operator uses knowledge about set coverings (a set of concepts that include all of the instances of another concept) to specialize a concept. This information is used to replace a requested concept with a set of concepts that cover it. For example, given the knowledge that the Port is covered by Sea_Port and Air_Port, a request for ports can be satisfied by retrieving and combining these two subconcepts. This is illustrated in the example below.

Input Query:

```
(retrieve (?name)
     (:and (port ?port)
           (port.name ?port ?name)))
```

Reformulated Query

```
(retrieve (?name)
     (:or (:and (sea_port ?port)
                (port.name ?port ?name))
          (:and (air_port ?port)
                (port.name ?port ?name))))
```

3.2 The Reformulation Process

Reformulation is performed by treating the reformulation operators as a set of planning operators and then using a planning system to search for a reformulation of the given set of concepts. The initial clauses of the query are divided into references to individual concepts and their associated constraints. The planner then searches for a way to map each of these concepts with their associated constraints into database concepts.

For example, consider the query shown below. It is first decomposed into two separate expressions – one about ports and the other about geolocs. Then the reformulation operators are used to find mappings to database concepts. Any remaining clauses (e.g., comparisons across concepts) are dealt with when a plan for accessing the data is generated.

```
(retrieve (?name)
      (:and (rail_port ?port)
            (port.refrig ?port ?refrig)
            (> ?refrig 0)
            (port.geocode ?port ?geocode)
            (port.name ?port ?name)
            (geoloc ?geoloc)
            (geoloc.country_name ?geoloc "Germany")
            (geoloc.geocode ?geoloc ?geocode)))
```

Using the reformulation operators described previously, the planner determines that the Geoloc concept expression can be mapped directly to a database and the Rail_Port concept expression needs to be reformulated. It can be reformulated into a Sea_Port concept expression, as described in Section 3.1.2, by adding a constraint. The resulting plan for reformulating the initial query is shown below.

```
generalize-concept rail_port (:and sea_port (filled-by port.rail "Y"))
select-database sea_port afsc_sea_port
select-database geoloc geo_geoloc
```

The final step is to take this plan and execute it. This is a straightforward process of applying the transformations in the query plan in the order listed. The resulting query is as follows.

```
(retrieve (?name)
      (:and (afsc_sea_port ?port)
            (afsc_port.rail ?port "Y")
            (afsc_port.refrig ?port ?refrig)
            (> ?refrig 0)
            (afsc_port.geocode ?port ?geocode)
            (afsc_port.name ?port ?name)
            (geo_geoloc ?geoloc)
            (geo_geoloc.country_name ?geoloc "Germany")
            (geo_geoloc.geocode ?geoloc ?geocode)))
```

3.3 Caching Retrieved Data

Data that is required frequently or is very expensive to retrieve can be cached in the Loom knowledge base and retrieved directly from Loom. An elegant feature of using Loom to model the domain is that caching the data fits nicely into this framework. The data is currently brought into Loom to perform the local processing, so caching is simply a matter of retaining the data and recording what data has been retrieved.

To cache retrieved data into Loom requires formulating a description of the data. This can be extracted from the initial query since queries are expressed in Loom in the first place. The description defines a new subconcept and it is placed in the appropriate place in the concept hierarchy. The data then become instances of this concept and can be accessed by retrieving all the instances of it.

Once the data is stored, it can be retrieved using the specialization operator that was described above. When the user poses the same query, the system can reformulate that query into the newly stored one and when the stored query is used, the cached data is retrieved directly from Loom.

4 Access Planning

The planning process described in this section finds an ordering of the database accesses and data comparisons by analyzing the dependency structure of the constraints on the query. It then generalizes the plan to remove any unnecessary ordering constraints in order to maximize the plan's potential parallelism. The complete database access plan is converted back into a partially ordered set of grounded subqueries that can be handed to LIM or executed directly in Loom. The first

```
(and (concept afsc_sea_port ?port)
     (relation port.refrig ?port ?refrig)
     (relation port.geocode ?port ?geocode)
     (relation port.name ?port ?name)))
     (comparison > ?refrig 0)
     (concept geoloc ?geoloc)
     (relation geoloc.country_name ?geoloc "Germany")
     (relation geoloc.geocode ?geoloc ?geocode))
```

Figure 6: Goal Statement for the Planner

subsection below describes how the initial access plan is generated, and the second subsection describes how the plan is converted into the appropriate subqueries.

4.1 Generating an Access Plan

Since some of the databases are quite large, there can be a significant difference in efficiency between different possible plans for a query. Therefore, we would like to find subqueries that can be implemented as efficiently as possible. To do this the planner must take into account the cost of accessing the different databases, the cost of retrieving intermediate results, and the cost of combining these intermediate results to produce the final results. In addition, since the databases are distributed over different machines or even different sites, we would like to take advantage of potential parallelism and generate subqueries that can be issued concurrently.

A central task of the planner is to determine the ordering of the various accesses to databases. In the course of executing this task it also selects the databases from which to extract information. The ordering is determined by analyzing which steps in the plan for the query are generating values for variables and which steps are filtering the possible values. If one step depends on information produced in another step, then they must be done in the correct order. The Prodigy system, described in Section 1.1.3 is used to form the subqueries and order them. The problem is cast as a set of Prodigy operators, where the original semantic query constitutes the *goal* that is to be achieved by the planner.

In a straightforward process, the reformulated example query described in the last section is mapped by Prodigy into the goal for the planner shown in Figure 6. (Note that the language being used is no longer Loom.) Each subclause of the query is annotated with additional information indicating whether it is a concept, relation, or comparison subclause.

The set of operators used by the planner is shown in Figure 7. The first operator, **`retrieve-concept`** simply maps a concept to the database used to retrieve the desired information. The next three operators, **`generate-values`**, **`filter-values`**, and **`compare-values`**, determine the constraints on the order of the accesses to the individual databases. The remaining operators, **`begin-query`** and **`end-query`**, delimit the operations performed on an individual database.

As an illustration, the **`retrieve-concept`** operator is shown in Figure 8. This operator specifies a set of preconditions that must be true in order to apply the operator. In this case the preconditions are that information about the concept is

Operator Purpose

`Retrieve-Concept` Retrieves information from a particular database.

`Generate-Values` Uses a given relation to generate values for a given variable.

`Filter-Values` Uses a given relation to filter values for a given variable.

`Compare-Values` Performs a comparison between two sets of values.

`Begin-Query` Indicates the beginning of a query to one of the databases.

`End-Query` Indicates the end of a query.

Figure 7: Operators for Planning a Query

```
(retrieve-concept
   (params (<pred> <object> <db>))
   (preconds (and (database-concpt <pred> <db>)
                  (open-db <db>)))
   (effects ((add (concpt <pred> <object>))
             (add (available <object> <db>)))))
```

Figure 8: Operator for Retrieving a Concept from a Database

directly available from some database and that this database has been opened. If the database has not been opened for retrieval, then the planner would create the subgoal of doing so and insert a `begin-query` operation. The `retrieve-concept` operator has two effects. The first specifies that the information for this concept is now available, and the second specifies in which database the information is available.

The system generates a plan to achieve the goal in Figure 6 by selecting operators to achieve each of the goal conditions. If the preconditions of a selected operator do not hold, then the system must recursively achieve each of the preconditions. Once the system has achieved all of the goal conditions, it will have a plan for retrieving the information to satisfy the initial query. The resulting plan specifies which databases are to be used to satisfy the query as well as any constraints on the order in which the information is retrieved.

Prodigy initially produces a totally ordered plan for retrieving information. This plan is then converted into a partially ordered set of plan steps free of unnecessary ordering constraints. Each of an operator's preconditions in the database access plan explicitly states the conditions on which that operator depends. We use the algorithm of Veloso [27] to convert the totally ordered plan into a partially ordered plan from the definitions of the operators. This algorithm is polynomial in the length of the plan. The resulting partially ordered plan is shown in Figure 9.

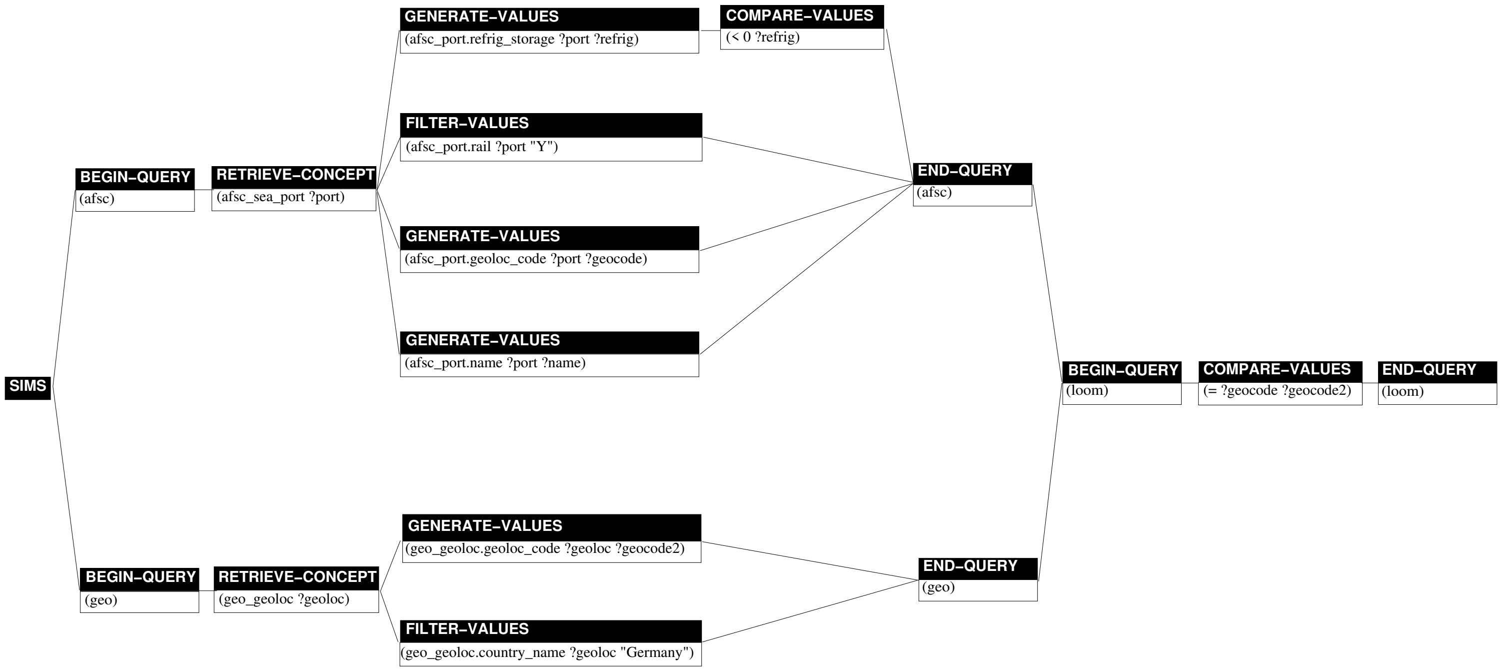

Figure 9: Preliminary SIMS Plan for Example Query

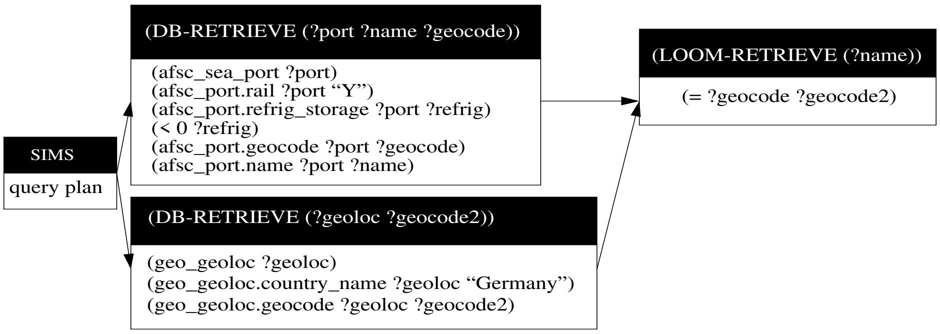

Figure 10: Final SIMS Plan for Example Query

4.2 Subquery Formation

The second step in the query planning process is to formulate the actual subqueries which will be passed on to LIM and eventually translated into database queries. Since LIM takes care of such details, we do not need to worry about the access languages of the individual databases, their locations, etc. Instead, we only need to formulate Loom queries that refer to information in one database. LIM and the DBMSs for the individual databases are responsible for selecting the appropriate access paths and locally optimizing the query within that database (we discuss global optimization in the next section).

The subqueries are formed by grouping together steps of the original plan. This is a relatively straightforward process that is aided by the presence of `begin-query/end-query` steps in the plan graph. The grouping is done by combining nodes in the plan partial order, to produce a final partial order on the subqueries. The subqueries for the example problem are shown in Figure 10. It shows that to implement the original query, three operations are necessary. The first two are accesses to separate databases that can be done in parallel. The third operation is a comparison in Loom on the results from these two subqueries. This last step cannot begin until the other two are complete.

5 Query-Plan Reformulation

Constructing a plan for retrieving information is only part of the problem. An important consideration in mapping the initial query into a set of subqueries is the total time that it will take to execute all of the subqueries. One approach to reducing this cost is to search for reformulations of the query access plan that reduce it. Database management systems (DBMSs) often perform syntactic query reformulation [14]. We leave that task to the respective DBMS then, and focus instead on more global semantic query reformulation [7, 15]. The idea is to transform the query resulting from the planning process into a semantically equivalent one that can be executed more efficiently.

Consider the planned query illustrated in Figure 10. The final step in this query, comparing two geographic location codes `?geocode` and `?geocode2`, could be quite costly since the cost of comparison is proportional to the square of the potentially large number of intermediate data items. Moreover, the comparison is performed in Loom, which is not as highly optimized as state-of-the-art DBMSs. There are a variety of ways in which this query could be reformulated to reduce or eliminate the cost of this last step. For example, knowledge about the contents in the databases could be used to augment the earlier subqueries, so that less intermediate information would be generated. Or, knowledge about the domain could be used to transform a subquery into an equivalent one that can be more efficiently executed.

Our approach to this problem differs from other related work on semantic query reformulation in an important respect that we do not rely on explicit heuristics of the database implementation to guide search for reformulations in the combinatorially large space of the potential reformulated subqueries. Instead, our algorithm considers all possible reformulations by firing all applicable rules and collecting candidate constraints in an *inferred set*. And then we select the most efficient set of the constraints from the inferred set to form the reformulated subqueries. This algorithm is not only more flexible and efficient, but the results of the rule firing turn out to be the useful information for extending subquery reformulation to the reformulation of the entire query plan. Most of other related work only reformulates single database queries.

Below we describe the principle behind the semantic reformulation, what knowledge is used for performing the reformulation, and the reformulation algorithms for subqueries and query plans.

5.1 Reformulation of Subqueries

The subquery reformulation problem is analogous to the problem of semantic query optimization for single database queries in previous work. The goal of query reformulation is to use reformulation to search for the least expensive query from the space of semantically equivalent queries to the original one. Two queries are defined to be *semantically equivalent*[25] if they return identical answers given the same contents of the database. The alternative definition of semantic equivalence[15] requires that the queries return identical answers given **any** contents of the database, but this definition would limit us to using only semantic integrity constraints which are often not available. The use of the less restrictive definition of semantic integrity requires that the system updates the learned knowledge as the databases change.

The reformulation from one query to another is by logical inference using *database abstractions*, the abstracted knowledge of the contents of relevant databases. The database abstractions describe the databases in terms of the set of closed formulas of first-order logic. These formulas describe the database in the sense that they are true with regard to all instances in the database. We define two classes of formulas: *range information*, which are propositions that assert the value range of database attributes; and *rules*, which are implications with an arbitrary number of range propositions on the antecedent side and one range proposition on the consequent side. Figure 11 shows a small set of the database abstractions. In all formulas the variables are implicitly universally quantified.

Range Information:

```
1:(geo_geoloc.country_name∈("France" "Taiwan" "Japan" "Italy" "Germany"))
2:(afsc_port.geocode ∈ ("BSRL" "HNTS" "FGTW" "VXTY" "WPKZ" "XJCS"))
3:(0 ≤ afsc_port.refrig_storage ≤ 1000)
```

Rules:

```
1:(geo_geoloc.country_name = "Germany") ⟹ (geo_geoloc.country_code = "FRG")
2:(geo_geoloc.country_code = "FRG") ⟹ (geo_geoloc.country_name = "Germany")
3:(geo_geoloc.country_code = "FRG")⟹(47.15 ≤ geo_geoloc.latitude ≤ 54.74)
4:(afsc_port.rail = "Y" ) ⟹ (afsc_port.geocode ∈ ("BSRL" "HNTS" "FGTW"))
5:(6.42 ≤ geo_geoloc.longitude ≤ 15.00) ∧
  (47.15 ≤ geo_geoloc.latitude ≤ 54.74)
  ⟹ (geo_geoloc.country_code = "FRG")
```

Figure 11: Example of Database Abstractions

SUBQ1:

```
(retrieve (?geoloc ?geocode2)
  (:and (geo_geoloc?geoloc)
        (geo_geoloc.geocode ?geoloc ?geocode2)
        (geo_geoloc.country_name ?geoloc "Germany")))
```

SUBQ2:

```
(retrieve (?geoloc ?geocode2)
  (:and (geo_geoloc?geoloc)
        (geo_geoloc.geocode ?geoloc ?geocode2)
        (geo_geoloc.country_code ?geoloc "FRG")))
```

SUBQ3:

```
(retrieve (?geoloc ?geocode2)
  (:and (geo_geoloc?geoloc)
        (geo_geoloc.geocode ?geoloc ?geocode2)
        (geo_geoloc.country_code ?geoloc "FRG")
        (geo_geoloc.latitude ?geoloc ?latitude)
        (?latitude >= 47.15) (?latitude <= 54.74)))
```

Figure 12: Equivalent Subqueries

The first two rules in Figure 11 state that for all instances, the value of its attribute country name is `"Germany"` if and only if the value of its attribute country code is `"FRG"`. With these two rules, we can reformulate the subquery `SUBQ1` in Figure 12 to the equivalent subquery `SUBQ2` by replacing the constraint on `geo_geoloc.country_name` with the constraint on `geo_geoloc.country_code`. We can inversely reformulate `SUBQ2` to `SUBQ1` with the same rules. Given a subquery Q, let $C_1, \ldots, C_k$ be the set of range and interaction constraints in Q, the following *reformulation operators* return a semantically equivalent query:

- **Range Refinement:** A range-information proposition states that the values of an attribute A are within some range R_d. If a range constraint of A in Q constrains the values of A in some range R_i, then we can refine this range constraint by replacing the constraining range R_i with $R_i \cap R_d$.

```
SUBQ-REFORMULATION(Subquery, DB-Knowledge, Cost-Model)
 1.refine range constraints, if Subquery refuted, return  Nil;
 2.for all applicable rules A  →  B in DB-Knowledge:
    if Subquery refuted, return  NIL;
    else add B to Inferred-Set, add (B,A) to Dependency-List;
 3.for all B in Inferred-Set in the order of their cost:
    if B is not indexed and ∃ (B,A) in Dependency-List
       delete B from Subquery, delete (B,A) from Dependency-List;
       replace all (C,B) in dependency list with (C,A);
 4.return (reformulated Subquery, Inferred-Set)
END.
```

Figure 13: Subquery Reformulation Algorithm

- **Constraint Addition:** Given a rule $A \rightarrow B$, if a subset of constraints in Q implies A, then we can add constraint B to Q.
- **Constraint Deletion:** Given a rule $A \rightarrow B$, if a subset of constraints in Q implies A and B implies C_i, then we can delete C_i from Q.
- **Subquery Refutation:** Given a rule $A \rightarrow B$, if a subset of constraints in Q implies A, and in the query there exists a range constraint C_i such that B implies $\neg C_i$, then we can assert that Q will return `NIL`.

Replacing constraints is treated as a combination of addition and deletion. Note that these reformulation operators do not always lead to more efficient versions of the subquery. Knowledge about the access cost of attributes is required to guide the search. For example, suppose the only database index is placed on the attribute `geo_geoloc.country_name`. In that case reformulating SUBQ2 to SUBQ1 will reduce the cost from $O(n)$ to $O(k)$, where n is the size of the database and k is the amount of data retrieved. However, if either `geo_geoloc.country_name` and `geo_geoloc.country_code` are not indexed, then we will prefer the lower cost short string attribute `geo_geoloc.country_code`. In this case, reformulating SUBQ1 to SUBQ2 becomes more reasonable. Figure 13 shows our subquery reformulation algorithm. We explain the algorithm below by showing how `SUBQ-REFORMULATION` reformulates the subquery SUBQ1, the lower query in the query plan in Figure 10.

There are three input arguments to the algorithm: the subquery to be reformulated, the database abstractions, and the cost model. The first step in the algorithm is to refine the range constraints. The only range constraint in SUBQ1 is on `geo_geoloc.country_name`, and its constrained value `Germany` is within the range of possible values (see the first formula of range information), so this constraint remains unchanged.

The second step is to match all applicable rules from the set of database abstractions using the reformulation operators defined above. The first rule in Figure 11 is matched and fired for SUBQ1 and we get an additional constraint `(geo_geoloc.country_code ?geoloc "FRG")`, which is added to the `Inferred-Set`. Then the second and third rules are matched because of the additional constraint on country code. The constraints `geo_geoloc.latitude` and `geo_geoloc.country_name` are added to the `Inferred-Set`.

The third step is to select the constraints in the `Inferred-Set` to delete from the subquery. The selection is based on the constraint's relative estimated execution cost which is computed by the type of the constraints (range constraint, or interaction constraint), the type of the attribute's values (integer, string, and their length), and whether they are indexed. The attribute `geo_geoloc.country_name` is deleted because its long string type is the most expensive. The next most expensive constraint is the one on attribute `geo_geoloc.country_code`. However, it should be preserved because the cause of its deletability (i.e., the constraint on `geo_geoloc.country_name`) was just deleted. Finally, the constraint on `geo_geoloc.latitude` is kept because it is an indexed attribute that will improve the efficiency of the subquery. The algorithm returns the reformulated subquery SUBQ3 as shown in Figure 12, as well as the `Inferred-Set`, which will be used for reformulating the succeeding subqueries in the query plan.

The worst case complexity of `SUBQ-REFORMULATION` is $O(R^2MN)$, where M is the maximum length of the antecedent of the rules, N is the greatest number of constraints in the partially reformulated query, that is, the number of original constraints plus the number of added constraints before final selection, R is the size of `DB-Knowledge`. In the average case, the complexity is much smaller than this worst case estimation. Because $R^2 \gg MN$, R is the dominating factor in the complexity and should be kept within a manageable size. This complexity analysis assumes that the system matches database abstractions by linear search. Therefore, a very large set of database abstractions could make the reformulation costly. To avoid this problem, we plan to adopt a more sophisticated rule match algorithm, such as the RETE algorithm[10], that will improve the algorithm's efficiency.

5.2 Reformulation of Query Plans

We can reformulate every subquery in the query plan with the subquery reformulation algorithm and improve their efficiency. However, the most expensive aspect of the multidatabase query is often processing intermediate data. In the example query plan in Figure 10, the constraint on the final subqueries involves the variables `?geocode` and `?geocode2` that are bound in the preceding subqueries. If we can reformulate these preceding subqueries so that they retrieve only those data instances possibly satisfying the constraint `(= ?geocode ?geocode2)` in the final subquery, the intermediate data will be reduced. This requires the query plan reformulation algorithm to be able to propagate the constraints along the data flow paths in the query plan. The query plan reformulation algorithm defined in Figure 14 achieves this by updating the database abstractions and rearranging constraints. We explain the algorithm below using the query plan in Figure 10.

The algorithm takes three input arguments: the query plan, the database abstractions, and the cost model. This algorithm reformulates each subquery in the partial order (i.e., the data flow order) specified in the plan using `SUBQ-REFORMULATION`. In addition, the database abstractions are updated with the `Inferred-Set` returned from `SUBQ-REFORMULATION` to propagate the constraints to later subqueries. In this example, the second formula of the initial range information is replaced by `(afsc_port.geocode` $\in$ `("BSRL" "HNTS" "FGTW"))`, the consequent condition of the fourth rule. The algorithm uses this updated range information to

```
QPLAN-REFORMULATION(Plan, DB-Knowledge, Cost-Model)
 1.KB ← DB-Knowledge;
 2.for all subqueries S in the order specified in Plan:
    (S',Inferred-Set) ← SUBQ-REFORMULATION(S,KB,Cost-Model);
    if S' refuted, return  Nil;
    else update KB with Inferred-Set; update Plan with S';
 3.for all subqueries S whose semantics are changed:
    SUBQ-REFORMULATION(S, DB-Knowledge, Cost-Model);
 4.return reformulated Plan
END.
```

Figure 14: Query Plan Reformulation Algorithm

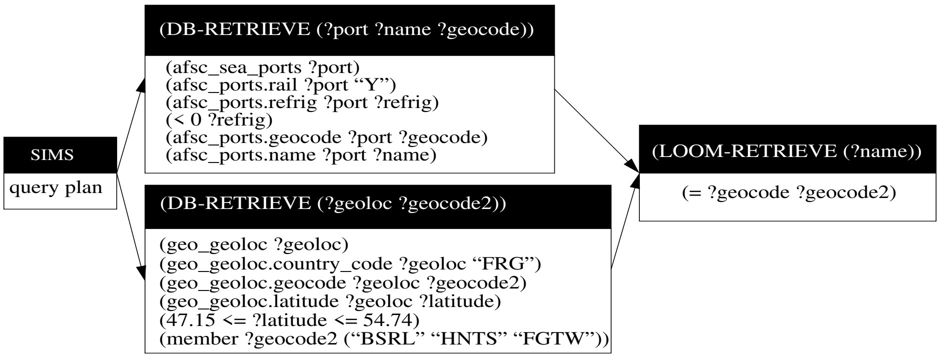

Figure 15: Reformulated SIMS Plan for Example Query

reformulate the final subquery and reduces the possible values from six to three. In addition, the constraint `(afsc_port.rail ?port "Y")` in the upper subquery is propagated along the data flow path to its succeeding subquery.

Now that the updated range information for `?geocode` is available, the subquery reformulation algorithm can infer from the constraint `(= ?geocode ?geocode2)` a new constraint `(member ?geocode2 ("BSRL" "HNTS" "FGTW"))`. In our example, the variable is bound by `(geo_geoloc.geocode ?geoloc ?geocode2)` in the lower subquery in Figure 10. The algorithm will insert the new constraint on `?geocode2` in that subquery. In this way, the constraints `(afsc_port.rail ?port "Y")` and `(= ?geocode ?geocode2)` are propagated back along the data flow path to the lower subquery. This process of new constraint insertion is referred to as *constraint rearrangement*. The final reformulated query plan is shown in Figure 15.

This query plan is more efficient than and returns the same answer as the original one. In our example, the lower subquery is more efficient because of the new constraint on the indexed attribute `geo_geoloc.latitude` (by `SUBQ-REFORMULATION`). The intermediate data items are reduced because of the new constraint on the attribute `geo_geoloc.geocode`. The logical rationale of this new constraint is derived from the constraints in the other two subqueries: `(afsc_port.rail ?port "Y")` and `(= ?geocode ?geocode2)`, and the fourth rule in the database abstractions.

query	1	2	3	4	5	6	7	8	9	10
planning time (sec)	0.5	0.3	0.6	2.1	1.1	0.7	0.7	0.5	0.5	0.8
reformulation time	0.1	0.1	0.0	0.5	0.1	0.0	0.0	0.1	0.1	0.3
rules fired (times)	37	18	11	126	63	8	17	15	19	71
query exec. time w/oR[a]	0.3	8.2	0.6	12.3	11.3	2.0	251.0	401.8	255.8	258.8
query exec. time w/R[b]	0.3	1.5	0.0	11.3	11.1	0.0	0.3	207.5	102.9	195.2
total elapsed time w/oR	0.8	8.5	1.2	14.4	12.4	2.7	251.7	402.3	256.3	259.6
total elapsed time w/R	0.9	1.9	0.6	13.9	12.3	0.7	1.0	208.1	103.5	196.3
intermediate data w/oR	-	-	-	145	41	1	810	956	808	810
intermediate data w/R	-	-	-	145	35	0	28	233	320	607

[a] w/oR = Without reformulation.
[b] w/R = With reformulation.

Table 1: Experimental Results

The complexity of **QPLAN-REFORMULATION** is $O(SR^2MN)$, where S is the number of subqueries in the query plan, and R^2MN is the cost of **SUBQ-REFORMULATION**. In actual queries, S is relatively small, so the dominating factor is still the cost of the subquery reformulation R^2MN, in which the size of the database abstractions R is the most important factor, as shown in section 5.1. Thus, with a manageable size of the database abstractions, our algorithms are efficient enough to be neglected in the total cost of the multidatabase retrieval.

The earliest work in query reformulation was referred to as *semantic query optimization* and was applied to the single database query processing domain in a system called QUIST[15]. In contrast with *syntactic query optimization*, which has been widely studied in the database community, QUIST uses the rules of semantic integrity constraint of the database as background knowledge to reformulate the given query. However, QUIST and the following work[25, 7] use heuristics to select the reformulation operators and rules to reformulate the query in a hill-climbing manner. Our reformulation algorithm does not require heuristic control and is thus more flexible. Moreover, our algorithm utilizes the database abstractions to the greatest possible extent, while hill-climbing only searches for the local optimum.

5.3 Experimental Results of Reformulation

Table 1 provides statistical data concerning the preliminary experimental results of the query plan reformulation algorithm. In this experiment, the SIMS system is connected with two remote Oracle databases. One of the databases consists of 16 tables, 56,078 instances, the other database consists of 14 tables, 5,728 instances. The queries used were selected from the set of SQL queries constructed by the original users of the databases. The first three queries are single database queries, while the others are multidatabase queries. This initial results indicate that our algorithm can reduce the total cost of the retrieval substantially. In most multidatabase queries, the amount of intermediate data is reduced significantly. The overheads of reformulation is included in the total execution time and is relatively small compared to the cost of executing most queries.

The system used 267 database abstraction rules in this experiment. These rules were prepared by compiling the databases. The compiling procedure summarizes the range of each relation of the database by extracting the minimum and maximum

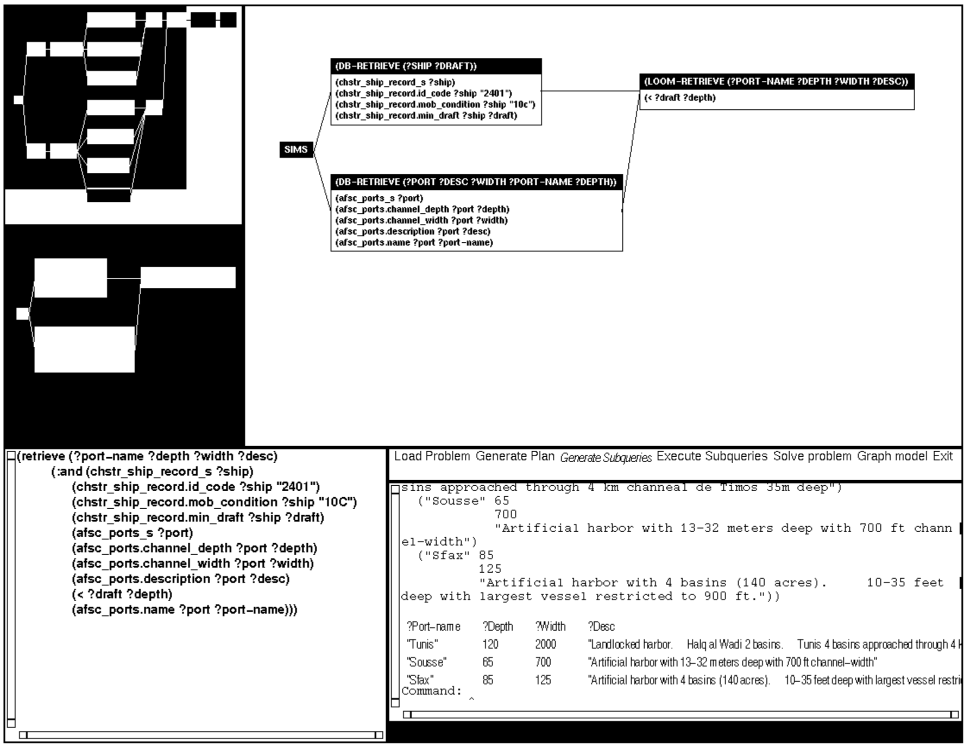

Figure 16: Sample SIMS Interface Screen

values for numerical relations, or enumerating the possible values for string type relations. If the number of possible values exceeds a threshold, this range information is discarded. The rules were prepared by a semi-automatic learning algorithm similar to the KID3[23]. This algorithm takes a user input condition A, and learns a set of rules of the form $A \rightarrow B$ from the database. The algorithm retrieves the data that satisfy the condition A, then compiles the data for the conclusions B.

We are now developing an automatic learning algorithm that is driven by example queries. We plan to use inductive learning[5, 11, 20] to identify the costly aspects of the example subqueries, propose candidate rules to learn, and then refine the candidate rules to the desired operationality. Previous work that automatically derives the content knowledge is proposed by Siegel[25]. Our approach differs from theirs in that it is driven by the need for reformulation in the example instead of by a fixed set of heuristics, and it is flexible with regard to various database implementations. This is necessary in our case, since the databases integrated by SIMS are usually heterogeneous.

6 The SIMS Interface

Our intention is that the user view SIMS as a black box that allows the user to query multiple sources of information as if they were one single source. Given this scenario, an important issue is the ease with which the user can pose queries to the system and receive an answer. Since the exact terms used in a model by the developer will often differ from those which a user may be familiar with, it is very important that the developer's model be accessible to the user. We have thus stressed the importance of providing an easy to use interface for posing queries, one that allows the user convenient access to the model and help in construction of the query.

At the same time it is important that the model be constructed accurately by the developer. The model defines the application domain ontology, for both the user and SIMS. Not only will the model builder need to be able to build a good model of the domain, but he needs to be able to connect terms in the domain model with the corresponding terms in the database model. In order to build a model containing hundreds, possibly thousands of concepts, it is essential that the model builder have tools to view the models. The model builder also needs tools to help connect models fragments.

A common need for both users and model builders is a good way of viewing the model. Given that SIMS models are Loom models, a subsumption-based hierarchy of concepts, the logical visual representation to use is a graph. But a subsumption hierarchy only shows part of the definition of a model, the *is-a* relations, to show how a subconcept differs from its superconcept, it is usually necessary to show its roles. Hence our graph shows not only the concepts but the roles of concepts and their ranges.

SIMS does not dictate a single mode of interaction. We believe that the full range of underlying user interface management modalities should be made available to the user. Commands can be issued by mouse gestures applied to the desired objects, through a menu, or by keyboard commands. The user interface management system used by SIMS is CLIM 1.1, which is a high level presentation-based user interface system.

6.1 The Query Interface

SIMS is accessed by the user through the query interface. Central to the ability to pose a query is knowledge of the terms in which the domain is defined.

The SIMS query interface will provide the user aid in the following manner:

- A forms based query input facility.
- Access to the models via a graph of the domain and database models.
- The ability to specify terms of the query by clicking on nodes in the graph.
- Intelligent defaulting — automatic filling in of appropriate variable names for a sub-query.

6.2 The Model Building Interface

The domain model defines the ontology of the domain, i.e., all the terms and relations that one can use to query the various information sources. It also defines the expressiveness of the domain, as well as how powerfully SIMS can be in reformulating the queries. A domain model for a realistic application can easily contain hundreds of concepts and relations, and depending on the complexity of the application, can get out of hand very fast, especially if created using a text editor. At the same time, many concepts are likely to be very similar, being no more than slightly modified copies of already existing subconcepts of some concepts and hence tedious to enter. To ease the model building process, we provide the following tools:

- Two editors:
  - a form-based editor that is knowledgeable about the syntax of Loom terms and allowable inputs.
  - a text based editor for direct manual entry/modification of definitions.
- Interactive gesture-based editing, nodes can be modified, added or deleted by clicking the mouse on the relevant node.
- Graph navigation aids — panning, node hiding/unhiding and node centering.

7 Conclusions

This paper describes a system for efficiently accessing and integrating information from multiple information sources (e.g., databases and knowledge bases). The various information sources are integrated using the Loom knowledge representation language. The system requires a model of the application domain and a model of the contents of each of the information sources. Then, given a query, the system generates and executes a plan for accessing the appropriate information sources. Before executing a query, the system first reformulates the individual subqueries to minimize the cost and the amount of intermediate data that is processed. Then the subqueries are executed, exploiting any parallelism in the plan.

SIMS currently integrates information from data stored in nine Oracle databases and information stored in a Loom knowledge base. The system uses the Loom Interface Manager (LIM) to retrieve data from the Oracle databases and then processes all the data in Loom. The plan for selecting and accessing the various information sources is generated using the Prodigy planning system. The resulting plan is reformulated using a set of special purpose algorithms for semantic query optimization over multiple database queries.

Acknowledgments

Thanks to Manuela Veloso for providing us with her code for generating partial orders. Thanks also to the LIM project for providing us with the databases and the database models used for our work, as well many sample queries.

References

[1] Yigal Arens. Services and information management for decision support. In *AISIG-90: Proceedings of the Annual AI Systems in Government Conference*, George Washington University, Washington, DC, 1990.

[2] Yigal Arens and Craig A. Knoblock. Planning and reformulating queries for semantically-modeled multidabase systems. In *Proceedings of the First International Conference on Information and Knowledge Management*, Baltimore, MD, 1992.

[3] J.M. Blanco, A. Illarramendi, and A. Goñi. Using a terminological system to integrate relational databases. Facultad de Informatica, Universidad del País Vasco, Apdo 649, San Sebastián, Spain, 1992.

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

[5] Yandong Cai, Nick Cercone, and Jiawei Han. Learning in relational databases: An attribute-oriented approach. *Computational Intelligence*, 7(3):119–132, 1991.

[6] Jaime G. Carbonell, Craig A. Knoblock, and Steven Minton. PRODIGY: An integrated architecture for planning and learning. In Kurt VanLehn, editor, *Architectures for Intelligence*, pages 241–278. Lawrence Erlbaum, Hillsdale, NJ, 1991.

[7] Upen S. Chakravarthy, John Grant, and Jack Minker. Logic-based approach to semantic query optimization. *ACM Transactions on Database Systems*, 15(2):162–207, 1990.

[8] Arvola Chan, Sy Danberg, Stephen Fox, Wen-Te K. Lin, Anil Nori, and Daniel Ries. Storage and access sturctures to support a semantic data model. In *Proceedings of the 8th International Conference on Very Large Data Bases*, pages 122–130, Very Large Database Endowment, Saratoga, CA, 1982.

[9] 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.

[10] C.L. Forgy. RETE: A fast algorithm for the many pattern/many object pattern matching problem. *Artificial Intelligence*, pages 17–37, 1982.

[11] David Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, 36:177–221, 1988.

[12] R. Hull and R. King. Semantic database modeling: Survey, applications, and research issues. *ACM Computing Surveys*, 19(3):201–260, 1987.

[13] A. Illarramendi, J.M. Blanco, and A. Goñi. One step to integrate data and knowledge bases. Facultad de Informatica, Universidad del País Vasco, Apdo 649, San Sebastián, Spain, 1992.

[14] M. Jarke and J. Koch. Query optimization in database systems. *ACM Computer Surveys*, 16:111–152, 1984.

[15] Jonathan Jay King. *Query Optimization by Semantic Reasoning*. PhD thesis, Stanford University, Department of Computer Science, 1981.

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

[17] R. MacGregor. A deductive pattern matcher. In *Proceedings of AAAI-88, The National Conference on Artificial Intelligence*, St. Paul, MN, 1988.

[18] R. 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.

[19] Donald P. McKay, Timothy W. Finin, and Anthony O'Hare. The intelligent database interface: Integrating AI and database systems. In *AAAI-90: Proceedings of The Eighth National Conference on Artificial Intelligence*, 1990.

[20] Ryszard S. Michalski. A theory and methodology of inductive learning. In *Machine Learning: An Artificial Intelligence Approach*, volume I, pages 83–134. Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1983.

[21] Steven Minton, Jaime G. Carbonell, Craig A. Knoblock, Daniel R. Kuokka, Oren Etzioni, and Yolanda Gil. Explanation-based learning: A problem solving perspective. *Artificial Intelligence*, 40(1-3):63–118, 1989.

[22] Mike P. Papazoglou. An organizational framework for cooperating intelligent information systems. *International Journal of Intelligent and Cooperative Information Systems*, 1(1):169–202, 1992.

[23] G. Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. In G. Piatetsky-Shapiro, editor, *Knowledge Discovery in Databases*, pages 229–248. MIT Press, 1991.

[24] 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.

[25] Michael D. Siegel. Automatic rule derivation for semantic query optimization. In Larry Kerschberg, editor, *Proceedings of the Second International Conference on Expert Database Systems*, pages 371–385. George Mason Foundation, Fairfax, VA, 1988.

[26] Shalom Tsur and Carlo Zaniolo. An implementation of gem – supporting a semantic data model on a relational back-end. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pages 286–295, ACM, New York, 1984.

[27] Manuela M. Veloso. Nonlinear problem solving using intelligent casual-commitment. Technical Report CMU-CS-89-210, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1989.