Published in the *Proceedings of the First International Conference on Information and Knowledge Management*, Baltimore, MD, 1992

Planning and Reformulating Queries for Semantically-Modeled Multidatabase Systems*

Yigal Arens and Craig A. Knoblock
USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
(310) 822-1511
ARENS@ISI.EDU, KNOBLOCK@ISI.EDU

Abstract

With vast amounts of information available from various sources, integrating data from multiple databases is an important problem. The SIMS project attacks this problem using a variety of Artificial Intelligence techniques, including planning, knowledge representation, problem reformulation, and learning. To integrate multiple databases, the user provides a semantic model of the application domain and then uses this model to describe the contents of the available databases. Given a query, the system uses a planner to decide which databases must be queried and in what order the queries should be executed. This paper focuses on the *query planning* problem — the selection of appropriate data sources and ordering the accesses to them, and on the *reformulation* of queries — the use of knowledge both about the domain and the databases to modify queries to make the retrieval plans for them more efficient.

1 Introduction

Most tasks performed by users of complex information systems involve interaction with multiple databases. Examples can be found in the areas of analysis (both of intelligence data and logistics forecasting) and in resource planning and briefing applications. Retrieval of desired information dispersed in multiple databases 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 databases, 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 databases, it is difficult to find individuals who possess the required knowledge, and automation becomes a necessity. There has been some work on this problem in the database community [6, 7]. Our work differs in that a complete semantic model of the application domain is created and used in order to provide a collection of terms with which to describe the contents of (i.e., to create a semantic model of) available databases. In contrast to previous work, the model is not specific to a particular set of databases and there is not necessarily a direct mapping from the concepts in the model to the objects in the database. This approach supports a much more flexible and easily extensible interface to a collection of databases.

The SIMS[1] project applies a variety of techniques and systems from Artificial Intelligence to build an intelligent interface to databases. SIMS builds on the following ideas:

Knowledge Representation/Modeling, which is used to describe the domain about which information is stored in the databases, as well the structure and contents of the databases themselves. The domain model is a declarative description of the objects and activities possible in the application domain as seen by a typical user. The model of each database indicates the data-

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

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 databases.

Planning/Search, which is then used to construct a sequence of queries to individual databases that will satisfy the user's query. SIMS identifies databases that contain relevant data and determines what additional data may be necessary in order to access them.

Reformulation/Learning. Alternative databases and queries to retrieve the desired information are considered. The detailed semantics provided by the application domain model are used to guide the search for more efficient query formulations. Knowledge about the contents of the databases is also learned from the databases and then used to reformulate the queries.

A query is provided to SIMS in the form of a description of a semantic class of objects about which information is desired. This description is made up of statements in the Loom knowledge representation language (Section 2.1). SIMS proceeds to reformulate this query as a collection of more elementary statements that refer to knowledge stored in individual databases. These are passed on to the LIM system (Section 2.2) which does the final translation into database queries in the appropriate language(s).

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 people and equipment from one geographic location to another using available transportation facilities and vehicles. An earlier prototype was applied to information needed for daily Naval briefings given in Hawaii about the status of the Pacific Fleet [1]. Several databases with information about ships, ports, locations, activities, etc., are accessible to the system. The ideas and techniques for planning the access to databases (Section 4) have already been implemented and the reformulation of queries (Section 5) is currently under development.

The remainder of this paper is structured as follows. The next section describes the technological infrastructure used in SIMS. Section 3 presents an example problem solved by SIMS. Section 4 describes our treatment of the query planning and subquery formation problems. Section 5 discusses the reformulation of queries to improve efficiency. We conclude with a brief summary and directions for future work.

2 Technological Infrastructure

2.1 Loom

Loom serves as the knowledge representation system SIMS uses to describe the domain model and the contents of the databases. 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 may be considered a development of the KL-ONE [2] family.

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.

This model-driven programming system supports frame-based structured inheritance with formal semantics. Its formal semantics allow automatic classification of new descriptions, increasing its utility in the construction of large declarative knowledge bases. Loom emphasizes usability through its functionality, efficiency, and its associated tools. For a detailed description of Loom see [10, 11].

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 `:and` specify, respectively, that the variable `?port` describes a member of the model class `Port`, that the relation `Name` holds between the value of `?port` and the string `SAN-DIEGO`, and that the relation `Channel_Depth` holds between the value of `?port` and the value of the variable `?depth`. The semantic query specifies that the value of the variable `?depth` be returned. This query does not necessarily correspond to a single database query, since there may not exist one database that contains information about ports, port names, and port depths.

2.2 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* of the knowledge base — specific objects

```
(db-retrieve (?depth)
  (:and (Port ?port)
        (Name ?port "SAN-DIEGO")
        (Channel_Depth ?port ?depth)))
```

Figure 1: Example SIMS/Loom Query

whose identity and relationship to other objects and classes is known. 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) [12] 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.

2.3 Prodigy

Prodigy [4, 13] is a means-ends analysis planner complete with six modules for learning search-control rules, abstractions, operator models, and more. SIMS uses Prodigy for planning the accesses to the individual databases in order to satisfy a query.

Prodigy is a given a problem space definition and a problem and is asked to find a solution to the problem. A *problem space* is defined by the legal operators and states. *Operators* are composed of a set of conditions, called *preconditions*, that must be true in order to apply an operator and a set of *effects* that describe the changes to the state that result from applying an operator. *States* are composed of a set of conditions that describe the relevant features of a model of the world. A problem consists of an *initial state*, which describes the initial configuration of the world, and a *goal*, which describes the desired configuration. To solve a problem, the planner must find a sequence of operators that transform the initial state into a state that satisfies the goal.

Prodigy has been linked to Loom, so that it can use the Loom domain model as its model of the world. The system is given a query as the goal and a set of operators that define the actions that can be used to achieve the query. Using this set of operators, Prodigy searches for a plan that will achieve the query.

3 Overview of SIMS

The problem addressed in SIMS is, given a semantic query, how is that query decomposed into subqueries that can be mapped directly to individual databases. The initial query is expressed as a query to the Loom knowledge representation system. This query must be re-expressed as a partially ordered set of Loom subqueries, each of which will be mapped directly to a query to a particular database by LIM.

An initial Loom query of the kind SIMS handles is shown in Figure 2. For example, the first constraint, `(Ship ?ship)`, is a concept expression that constrains the variable `?ship` to the set of ship records in the knowledge base. The Loom class `Ship` need not necessarily correspond to the contents of a specific field in some database. If it does not, the planner may have to find some combination of subqueries that will obtain all necessary records. This is discussed further below. The second constraint is a relation on the ID code of ships, that further restricts the set of ships to those with an ID code of 2401. The entire query requests the name, depth, width, and description of all ports that have sufficiently deep channels to accommodate ships with an identification code of "2401" and have a mobilization condition of "10C".

If the information about ships and ports was stored

```
(db-retrieve (?port-name ?depth ?width ?desc)
  (:and (Ship ?ship)
        (Id_Code ?ship "2401")
        (Mob_Condition ?ship "10C")
        (Min_Draft ?ship ?draft)
        (Port ?port)
        (Channel_Depth ?port ?depth)
        (Channel_Width ?port ?width)
        (Description ?port ?desc)
        (< ?draft ?depth)
        (Name ?port ?port-name)))
```

Figure 2: Example SIMS Semantic Query

directly in the Loom knowledge base, then Loom could have simply been used itself to answer this query. However, we are using Loom to semantically model a domain about which data is stored in multiple databases, so the information required to answer this query must be retrieved from the appropriate databases, with the help of LIM. Thus, if the information about ships and ports were all stored in one database, this query could be passed directly to LIM. But that is also not the case here.

Data pertaining to this query is spread over two databases — one containing information about ships and the other containing information about ports. The system is given the query shown in Figure 2 and it must formulate a set of subqueries that can be executed directly by either LIM or Loom to derive the desired result. Since LIM provides a transparent interface to a single database, we can use it 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 process of formulating these queries is the topic of the next section, Section 4.

The first step in processing a semantic query is to produce a **grounded plan** to implement the query. 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 databases, i.e., all data a step requests must be contained in a single database. Any data-manipulation steps of the plan are performed using the Loom reasoning facilities. The grounded plan produced takes the form of a graph 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 databases, 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 database 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 database, or to internal manipulation by SIMS of data so acquired.

After a plan for the query has been obtained, the system estimates the processing time of the individual subqueries based on whether steps involve data-retrieval or manipulation and the amount of data involved. The system then attempts to reformulate or eliminate those subqueries that are particularly costly. The time estimate is used to determine the amount of time that can be spent on attempts to reformulate the plan. This **reformulation** process is described in Section 5.

4 Query Planning

There are a number of steps required to plan the accesses to the individual databases. The planning process described here selects the databases to be used in answering the query, finds a legal ordering of the database accesses and comparisons by analyzing the dependency structure of the constraints, and generalizes the plan to remove any unnecessary ordering constraints in order to maximize the potential parallelism in the plan. This complete database access plan is then converted back into a partially ordered set of grounded Loom subqueries that can be handed to LIM or executed directly. The first subsection below describes the selection and ordering, and the second subsection describes how the plan is converted into the appropriate subqueries.

4.1 Database Selection and Ordering

Given a semantic query, there is not necessarily a one-to-one mapping between the KB concepts in it and the information as organized in the databases. A concept mentioned in the query may be retrievable from several possible databases, or may not be directly retrievable from any database. For example, the KB may have a concept of *ship*, while the databases may organize ships by country of registry. There is thus no single database to which one could direct a query concerning *ship*s. However, the concept hierarchy can often be used to enable retrieval of this information by checking for information about either the superconcepts or the subconcepts of the desired concept. In our example, SIMS may determine — by analyzing the query — that the ships of interest are naval ships, and ask only about those.

In the case of using a superconcept, the corresponding database, if such exists, may provide too much information and in the case of using subconcepts, a com-

plete covering of the original concept must be used in order to provide the desired information. In some cases, obtaining the information may require combining data from several databases.

We are dealing with large database systems, so there can be a huge difference in efficiency between different possible implementations of a query. We would therefore like to find queries 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 also like to take advantage of potential parallelism and generate subqueries that can be issued concurrently.

Consider the fragment of the knowledge base shown in Figure 3, which covers some of the knowledge relevant to the example query in Figure 2. The circles in the figure denote concepts in the knowledge base, the upward arrows indicate *is-a* links, and the downward arrows indicate relations to other concepts. For example, the *ship* concept has two subconcepts, *commercial ship* and *naval ship*, and is the domain of a relation which indicates its *draft*. The shaded circles correspond to concepts whose instances can be retrieved directly from some database. Thus, the CHSTR database contains information about all ships and the AFSC database contains information about ports and naval ships. The selection of which databases are used to provide the information for each concept is done in the course of planning the query.

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 bindings for variables and which steps are serving as filters. If one step depends on information produced in another step, then it must be ordered after that second one.

The Prodigy system, described in Section 2.3 is used to select the databases, determine the subqueries, and order them. The problem is cast as a set of Prodigy operators for both selecting and ordering the database accesses. The original semantic query constitutes the *goal* that is to be achieved by the planner. The planner starts out with information about the databases that are available and the KB classes that "correspond" to data in them.

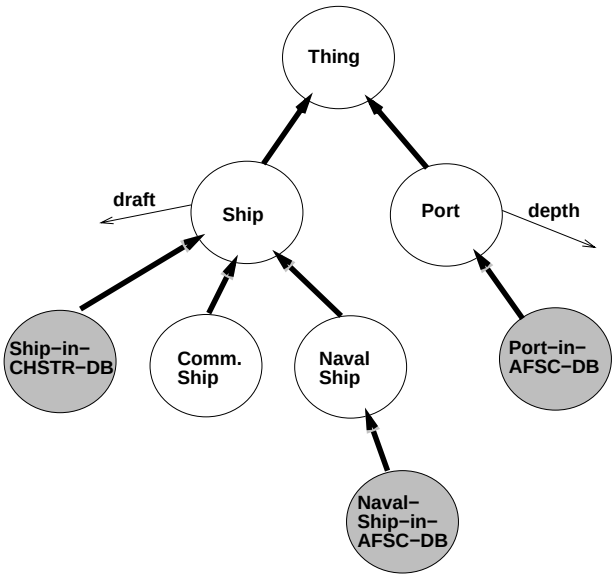

Figure 3: Fragment of the Knowledge-Base Model

The example query described in the last section is mapped by Prodigy into the goal for the planner shown in Figure 4. Each subclause of the query is annotated with additional information indicating whether it is a concept, relation, or comparison subclause. Also, a final goal is inserted, which specifies that all the pending queries must be completed.

```
(and (concpt ship ?ship)
     (relatn id_code ?ship 2401)
     (relatn mob_condition ?ship 10c)
     (relatn min_draft ?ship ?draft)
     (concpt port ?port)
     (relatn channel_depth ?port ?depth)
     (relatn channel_width ?port ?width)
     (relatn description ?port ?desc)
     (comparison < ?draft ?depth)
     (relatn name ?port ?port-name)
     (forall (<x>) (database <x>)
                   (closed-db <x>)))
```

Figure 4: Goal Statement for the Planner

The set of operators used by the planner is shown in Figure 5. The first three operators, `retrieve-concept`, `specialize-concept`, and `generalize-concept`, select the databases 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.

Operator	Purpose
Retrieve-Concept	Retrieves information from a particular database.
Specialize-Concept	Replaces a concept with an appropriate set of subconcepts.
Generalize-Concept	Replaces a concept with a superconcept.
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 5: Operators for Planning a Query

As an illustration, the `retrieve-concept` operator is shown in Figure 6. 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 this concept is directly available from some database and that this database has been opened. If the first precondition does not hold, then the system can consider a more specialized or more generalized concept in order to find one or more concepts about which information can be obtained directly from a known database. If the database has not been opened for retrieval, then the planner would create the subgoal of doing so and eventually 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.

```
(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 6: Operator for Retrieving a Concept from a Database

The system generates a plan to achieve the goal in Figure 4 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. Plan steps will specialize references to high-level Loom concepts like *ship* into references to classes like *chstr_ship_record_s* which are associated directly with some database.

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 the operator's preconditions in the database access plan explicitly state the conditions on which that operator depends. We use the algorithm of Veloso [16] to convert the totally ordered plan into a partially ordered plan from the definition of the operators. This algorithm is polynomial in the length of the plan. The resulting partially ordered plan is shown in Figure 7. Note that the Loom classes now used are only those which correspond directly to some database.

4.2 Subquery Formation

The second step in the query planning process is to formulate the 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 8. It shows that to implement the original query, three operations are necessary. The first two

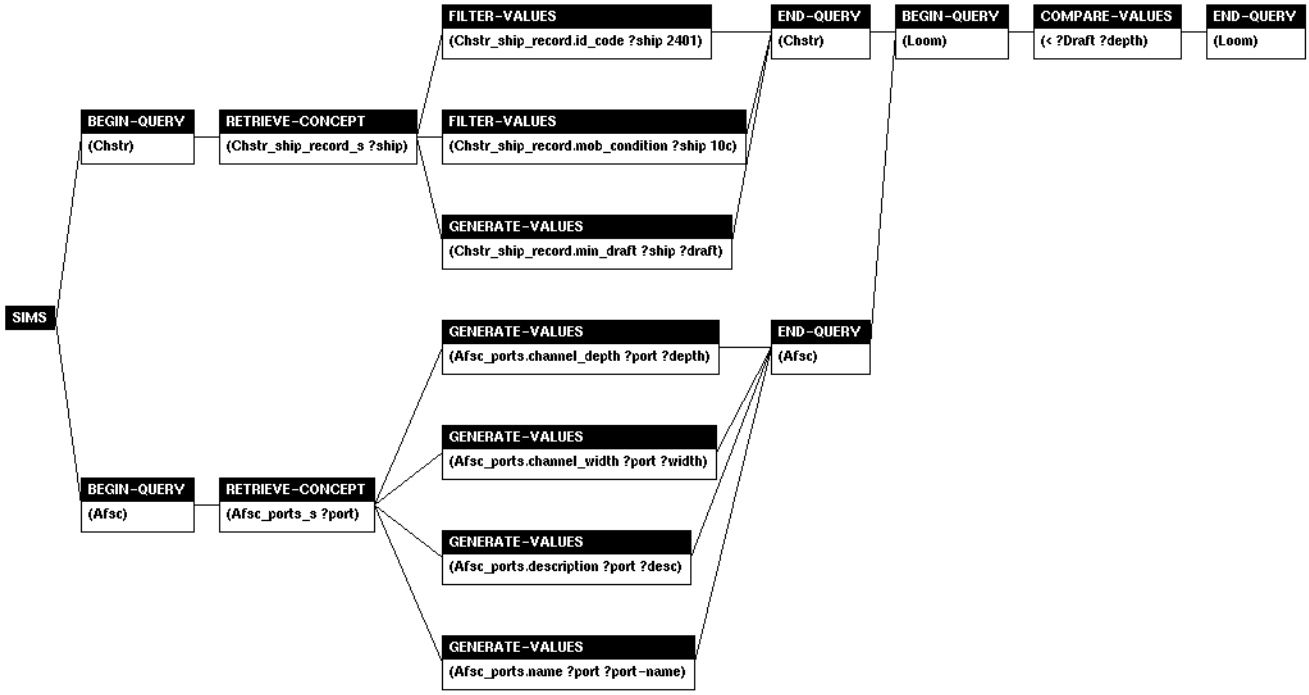

Figure 7: Preliminary SIMS Plan for Example Query

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.

It should be noted that this plan itself can be viewed as a (reformulated) query. Its component subqueries are still in the Loom query notation. It is only LIM that transforms these subqueries into true database queries.

5 Query Reformulation

Constructing a plan for retrieving information is only part of the problem. An important consideration in mapping the initial semantic 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 the total cost. Database management systems often perform syntactic query reformulation [8]. We leave that task to the respective DBMS then, and focus instead on more global semantic query reformulation [5, 9]. 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 8. The final step in this query, comparing the depth and the draft, could be quite costly since the comparison is done between potentially large numbers of data items and is performed in Loom, which is considerably slower than 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 information 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 two important respects. First, we do not rely on semantic integrity constraints to perform the reformulation process. Instead we use a richer collection of domain knowledge combined with knowledge compiled from the databases. Second, the reformulation process is integrated with the planning and focuses on costly aspects of a query, avoiding wasteful effort which could conceivably take longer than executing a flawed plan. This is necessary in our case, since the richness of the domain

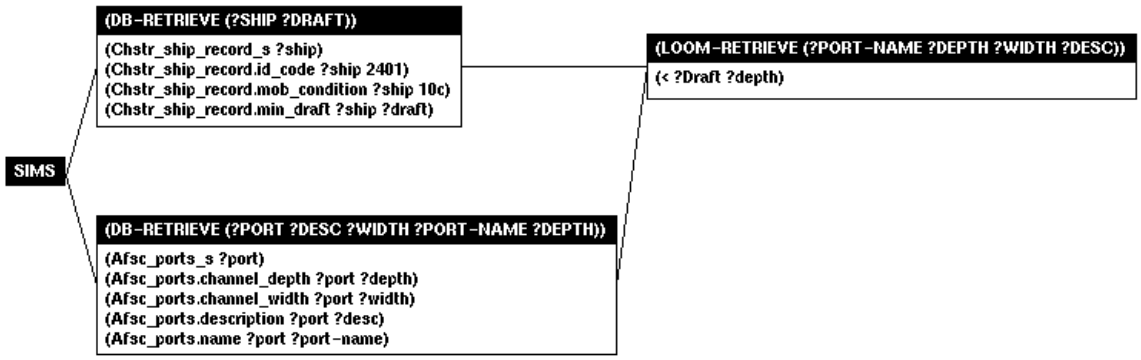

Figure 8: Final SIMS Plan for Example Query

model provides an almost endless search space for potential reformulations.

Below we describe how the system decides when to reformulate a query, which knowledge is used for performing the reformulation, and the actual reformulation techniques. Unlike the work in Section 4, the work described below has not yet been fully implemented.

5.1 When to Reformulate

We wish to reformulate queries in order to reduce their overall cost. In order to decide when to reformulate and how much time to spend, SIMS must be able to estimate the cost in terms of time and space of executing the initial query produced by the system as described above. Estimates are obtained from information stored in the KB concerning the size of databases, the type of DBMS used by them, and by directly querying databases that are able to provide time estimates. If these costs are relatively small, then the query can simply be executed as is. If it is above a particular threshold, SIMS will examine the separate subqueries to identify those whose modification can most reduce the cost of the plan. It then focuses its effort on reformulating those subqueries. The total effort to be expended by the system on reformulation is set not to exceed some fraction of the total estimated cost. This prevents SIMS from spending more time reformulating a query than it would spend executing the original plan.

Consider the example query described in the last section. As we pointed out earlier, the last step of comparing the depths and drafts may be particularly expensive. Thus, the system would search for reformulations that would reduce the cost of this specific step.

5.2 Knowledge for Reformulation

Most work on semantic query reformulation uses semantic integrity constraints to reformulate queries. The value of the standard approach is limited since the semantic integrity constraints available in a typical system are quite restricted in their expressive power compared to what is possible using a full-fledged knowledge representation of the complete domain, like that provided by Loom. SIMS has a much richer set of knowledge available for performing reformulation. For example, using Loom, SIMS can represent constraints regarding knowledge stored in more than one database.

5.2.1 Domain Knowledge

A central source of reformulation knowledge is the semantic model of the domain. This knowledge consists of the relationships among concepts, the relations between concepts and subconcepts, as well as more specific knowledge about the relationship between particular objects. For example, Figure 3 showed a fragment of the knowledge base where the ship concept has two subconcepts — naval ship and commercial ship. If the system also contains the fact that a ship with an ID code of 2401 is a naval ship, then this domain knowledge could be used to transform the query into one in which all of the information can be obtained with a single query to the AFSC database. That database would then contain all necessary data about both the ships we are concerned with and the ports. The comparison previously done in Loom could then be done in the AFSC database as well.

5.2.2 Knowledge Compiled from Databases

Instead of limiting the system to knowledge that must hold for the entire domain, we can use a compilation process that extracts knowledge from the individual databases and stores it in the knowledge base [3, 14, 15]. The compilation of knowledge about a database is driven by the need for particular types of information. Thus, when an expensive query is given and the semantic query reformulator cannot find a reformulation of it, the system makes a note of that along with the aspect of the query that made it expensive. The knowledge compiler can exploit time when the system and databases are not in use, to search for knowledge that could have been used to reformulate the expensive queries. If any relevant knowledge is found, the system records it for future use. Note that the system does not simply cache data from databases, but compiles more abstract knowledge about the data. Since the compiled knowledge can be affected by changes in the database, SIMS must maintain dependency information and update the compiled knowledge regularly.

For example, one particular type of knowledge that could reduce the cost of retrieving and comparing the depth and draft is the ranges for each of these attributes. Ports with a depth less than the minimum draft for ships of the type we are interested need not be retrieved. Let us assume that in our case the draft of the ships in the CHSTR database ranges from 25 to 50 feet and the depth of the ports in the AFSC database ranges from 20 to 40 feet. Such information about the ranges could be extracted from the databases and stored in the knowledge base to help with future queries. We explain below how such information may be used.

The one existing system that does provide a more general approach to learning for reformulation was developed by Siegel [15]. However, the particular learning mechanisms are quite limited and what the system learns is guided by a set of heuristics instead of being driven by the need to reformulate specific queries.

5.3 Reformulation Processes

Using available knowledge sources, reformulation involves modifications to the parallelized subqueries shown in Figure 8. The subqueries can be modified by adding constraints, deleting unnecessary constraints, and replacing constraints with different ones. We currently require that the entire reformulated query plan be semantically equivalent to the original.

Returning to our example, we would like to reduce the cost of retrieving the data into Loom and performing the comparison. One natural way to do this is to reduce the amount of information that is compared. This can be done by adding constraints to the two subqueries. If these constraints are necessarily entailed by the information in the knowledge base, the semantics of the overall query will not change. In this particular case, the maximum draft of any ship in the relevant class is known to be 50, while the maximum draft of the relevant ports is only 40, so we can add a constraint to the subquery to extract only ships with a maximum draft of 40. Any ship with a draft greater than 40 could never be less than the depth of any port, so this restriction will not change the final result. Similarly, the subquery for ports can be similarly augmented by noticing that the minimum draft of any ship is 25, while the minimum port depth is 20, so only ports with depth greater than 25 need to be considered.

This type of reformulation can potentially provide tremendous reductions in execution cost. Exactly how much reduction is obtained depends on the databases, knowledge and queries. We are in the process of implementing and testing the described approach.

6 Conclusions

This paper described the initial work in SIMS on query processing for multidatabase systems. The system described here addresses the important problem of how to efficiently integrate information from multiple databases. The approach described integrates this information in the context of a knowledge representation system (Loom) and builds on the Loom Interface Manager (LIM), which provides access to the individual databases. This allows SIMS to focus specifically on the issue of how to map a global query to the separate subqueries to individual databases. This paper address both the problem of how to plan out these queries, and of how to reformulate queries in order to implement them more efficiently.

The query planning is done using Prodigy, which provides a flexible and easily extensible system for selecting the databases, and partitioning and ordering the queries. The plans produces by Prodigy are then parallelized, to take advantage of databases that can be accessed simultaneously. Parallelization is done using the dependency structure of the plan produced by Prodigy. The result is passed on to the reformulation component.

The reformulation component searches for modifica-

tions that can be made to the query plan in order to improve its efficiency. The reformulation process is driven by estimates of the costs of executing the various parts of a query. The amount of time spent in the reformulation process is determined by the estimated cost of executing the entire plan. Some of the knowledge used to perform the reformulation is obtained from the domain model, and some may be learned by the system. Learning, too, is driven by the planning of past queries. The system will thus tailor itself to the types of queries that are frequently asked.

There are a variety of issues that this paper does not address. One important issue is how to integrate information when there are multiple sources for the same knowledge. To address this problem, we plan to build on the work of Dayal [6]. Dayal defines a language with which to express the integration of various sources of data. We believe this language can be incorporated into the existing system by defining additional operators for reasoning about the language. As in Dayal's work, the user will have to specify the particular method of integration for the data, and then the system will construct a plan to retrieve the data according to this method.

Another important issue that was just briefly mentioned in this paper is compiling knowledge from databases to be used for reformulating future queries. We plan to rely heavily on the ability to learn about the actual contents of each database. This will provide a much more flexible reformulation system.

Acknowledgments

Chin Y. Chee, another member of the SIMS project, is responsible for much of the programming that has gone into SIMS, as well as for the grapher used to display the partially ordered database access plans. Thanks to Manuela Veloso for providing us with her code for generating partial orders. Thanks also to the LIM project for providing us with one of the databases used for our work, as well as the query used in the example.

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

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

[4] 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.

[5] 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.

[6] Umeshwar Dayal. Query processing in a multidatabase system. In *Query Processing in Database Systems*, pages 81–108. Springer Verlag, New York, 1985.

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

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

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

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

[11] 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.

[12] 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.

[13] 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.

[14] G. Piatetsky-Shapiro. *Knowledge Discovery in Databases*. MIT Press, Cambridge, MA, 1991.

[15] 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.

[16] 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.