Query Processing in an Information Mediator*

Yigal Arens, Chin Chee, Chun-Nan Hsu,
Hoh In, and Craig A. Knoblock
Information Sciences Institute and
Department of Computer Science
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292, USA
{arens,chee,chunnan,hohin,knoblock}@isi.edu

Abstract

A critical problem in building an information mediator is how to translate a domain-level queries into an efficient query plan for accessing the required data. We have built a flexible and efficient information mediator, called SIMS. SIMS takes a domain-level query and dynamically selects the appropriate information sources based on their content and availability, generates a query access plan that specifies the operations and their order for processing the data, and then performs semantic query reformulation to minimize the overall execution time. This paper describes these three basic components of the query processing in SIMS.

1 Introduction

SIMS [Arens *et al.*, 1993] is an information retrieval system that provides an intelligent mediator between information sources and humans users or applications programs. Queries are expressed in a uniform language, independent of the distribution of information over sources, of the various query languages, the location of sources, etc. SIMS determines which data sources to use, how to obtain the desired information, how and where to temporarily store and manipulate data, and how to efficiently retrieve information.

The core part of the mediator is the ability to intelligently retrieve and process data. Information sources are constantly changing, new information becomes available, old information may be eliminated or temporarily unavailable, and so on. Thus, SIMS dynamically selects an appropriate set of information sources, generates a plan for processing the data, and then reformulates the plan to minimize the execution cost. The basic SIMS architecture is shown in Figure 1 and the query processing components are shown by the shaded rectangles.

This paper focuses on the query processing in SIMS. Before describing the query processing, Section 2 describes how the domain and information sources are modeled in SIMS. Section 3 describes the process of selecting the information sources. Section 4 present the approach to generating a query plan for execution. Section 5 describes the semantic query reformulation performed before execution. Section 6 reviews the most closely related work. Section 7 concludes with a summary of the contributions and directions for future work.

*The research reported here was supported in part by Rome Laboratory of the Air Force Systems Command and the Defense Advanced Research Projects Agency under contract no. F30602-91-C-0081, and in part by the National Science Foundation under grant number IRI-9313993. Views and conclusions contained in this report are those of the authors and should not be interpreted as representing the official opinion or policy of NSF, DARPA, RL, the U.S. Government, or any person or agency connected with them.

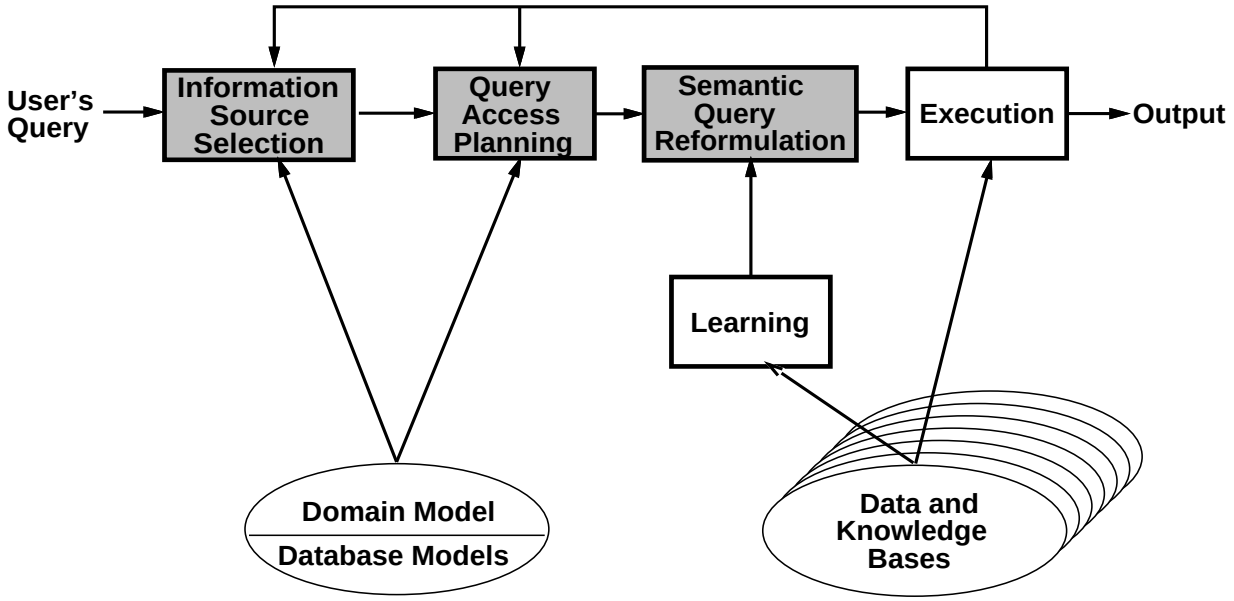

Figure 1: The SIMS Architecture

2 Representing the Knowledge of a Mediator

A mediator contains a model of its domain of expertise, which provides the terminology for interacting with the mediator, as well as models of all information sources that are available to it. Both the domain and information source models are expressed in the Loom language. The domain model provides class descriptions, subclass and superclass relationships, roles for each class, as well as other domain-specific information. The information source models describe both the contents of the information sources and the mapping between those models and the domain model.

2.1 Modeling the Domain

The SIMS mediator is specialized to a single "application domain" and provides access to the available information sources within that domain. The largest application domain that we have to date is a transportation planning domain, with information about the movement of personnel and materiel from one location to another using aircraft, ships, trucks, etc.

The application domain models are defined using a hierarchical terminological knowledge base (Loom) [MacGregor, 1988, MacGregor, 1990] with nodes representing each class of objects, and relations between nodes that define the relationships between the objects. For example, Figure 2 shows a fragment of the domain model in the transportation planning domain. In this figure, circles represent classes, the thick arrows represent subclass relationships, and the thin arrows represent relations between classes.

The classes defined in the domain model do not necessarily correspond directly to the objects described in any particular information source. The domain model is a high-level description of an application domain. Queries expressed in terms of this domain model are reformulated by SIMS into the appropriate subqueries to the various information sources. All of the information sources are defined using the terms in the domain model in order to make this reformulation possible. The reformulation of the domain level queries is described further in Section 3.

Queries to SIMS are expressed in terms of the general domain model, so there is no need to know or even be aware of the terms or language used in the underlying information sources. Instead the system translates the domain-level query into a set of queries to the underlying information sources. These information sources may be databases, knowledge bases, or other application programs.

Figure 3 illustrates a query expressed in the Loom language. This query requests all seaports and

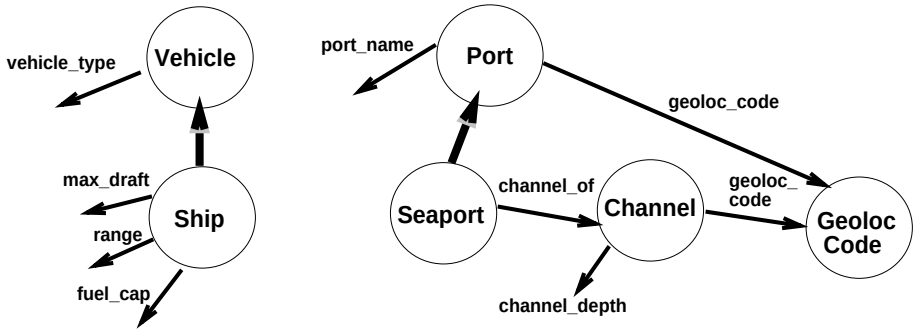

Figure 2: Fragment of the Domain Model

```
(retrieve (?port_name ?depth ?ship_type ?draft)
            (and  (seaport ?port)
                  (port_name ?port ?port_name)
                  (channel_of ?port ?channel)
                  (channel_depth ?channel ?depth)
                  (ship ?ship)
                  (vehicle_type ?ship ?ship_type)
                  (range ?ship ?range)
                  (> ?range 10000)
                  (max_draft ?ship ?draft)
                  (> ?depth ?draft)))
```

Figure 3: Example Loom query

corresponding ships with a range greater than 10,000 that can be accommodated within each port. The first argument to the `retrieve` expression is the parameter list, which specifies which parameters of the query to return. The second argument is a description of the information to be retrieved. This description is expressed as a conjunction of concept and relation expressions, where the concepts describe classes of information, and the relations describe constraints on these classes. The first subclause of the query is an example of a concept expression and specifies that the variable `?port` ranges over members of the class `seaport`. The second subclause is an example of a relation expression and states that the relation `port_name` holds between the value of `?port` and the variable `?port_name`. The query describes a class of seaports and a class of ships, and requests all seaport and ship pairs where the depth of the port exceeds the draft of the ship.

2.2 Modeling Information Sources

The critical part of the information source models is the description of the contents of the information sources. This consists of both a description of the objects contained in the information source as well as the relationship between these objects and the objects in the domain model. The mappings between the domain model and the information source model are used for transforming a domain-level query into a set of queries into actual information sources.

Figures 4 illustrates how an information source is modeled in Loom and how it is related to the domain model. In the picture, the shaded circle represents the Seaports table in the GEO database. All of the concepts and relations in the information source model are mapped to concepts and relations in the domain model. A mapping link between two concepts indicates that they represent the same class of information. Thus, if the user requests all seaports, that information can be retrieved from the GEO database, which has information about seaports.

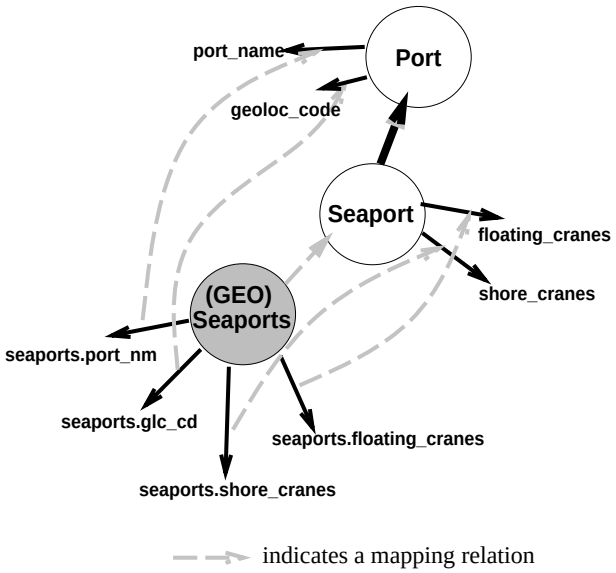

Figure 4: Relating an Information Source Model to a Domain Model

3 Information Source Selection

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 transforming a query expressed in terms of the concepts in the domain model into a query expressed in terms of the concepts in the information source models. If the user requests information about ports and there is an information source-level 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 do correspond to information sources.

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 Seaport, Channel and Ship have subconcepts shown by the shaded circles that correspond to concepts whose instances can be retrieved directly from some information source. Thus, the GEO information source contains information about both seaports and channels and the PORT information source contains information about only seaports. Thus, if the user asks for seaports, then it must be translated into one of the information source concepts — Seaports in GEO or Ports in PORT.

In order to select the information sources for answering a query, the system applies a set of reformulation operators to transform the domain-level concepts into concepts that can be retrieved directly from an information source. The system has a number of truth-preserving reformulation operations that can be used for this task. The operations include Select-Information-Source, Generalize-Concept, Specialize-Concept, and Decompose-Relation. These operations are described briefly below.

Select-Information-Source maps a domain-level concept directly to an information-source-level concept. In many cases this will simply be a direct mapping from a concept such as Seaport to a concept that corresponds to the seaports in some information source. There may be multiple information sources that contain the same information, in which case the domain-level query can be reformulated in terms of any one of the information source concepts. In general, the choice is made so as to minimize the overall cost of the retrieval. For example, the cost can be minimized by using as few information sources as possible.

Generalize-Concept uses knowledge about the relationship between a class and a superclass to reformulate a query in terms of the 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 retrieving extraneous data. For example, if a query requires some information about airports, but the

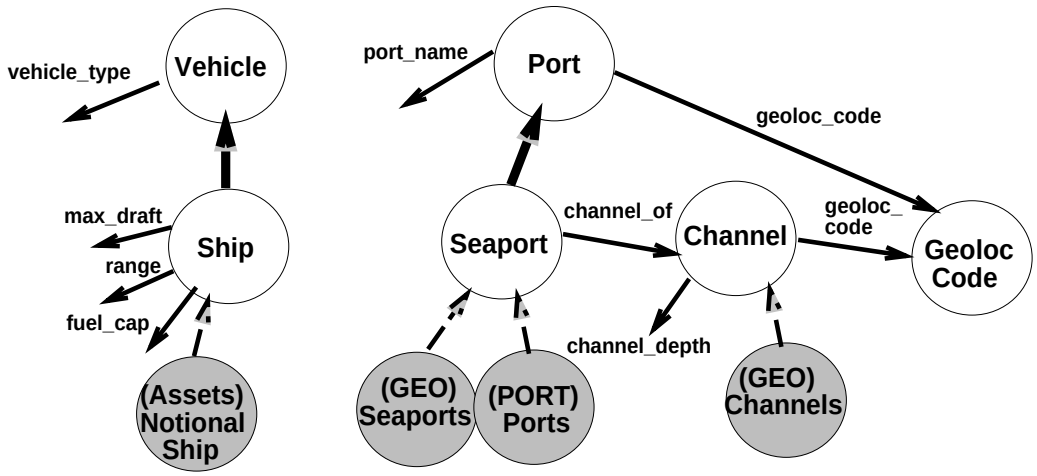

Figure 5: Fragment of Domain Model

information sources that correspond to the airport concept do not contain the requested information, then it may be possible to generalize airport to port and retrieve the information from some information source that contains port information. In order to ensure that no extraneous data is returned, the reformulation will include a join between airport and port.

Specialize-Concept replaces a concept with a more specific concept by checking the constraints on the query to see if there is an appropriate specialization of the requested concept that would satisfy it. For example, if a query requests all *ports* with an elevation greater than 1000 feet, it may be possible to reformulate this in terms of all *airports* with an elevation greater than 1000 feet since there are no seaports with an elevation this high. Even if there was an information source corresponding to the port concept, this may be a more efficient way to retrieve the data. Range information such as this is naturally represented and stored as part of the domain model.

Decompose-Relation replaces a relation defined between concepts in the domain model with equivalent terms that are available in the information source models. For example, `channel_of` is a property of the domain model, but it is not defined in any information source. Instead, it can be replaced by joining over a key, `geoloc-code`, that in this case happens to occur in both seaport and channel.

Reformulation is performed by treating the reformulation operators as a set of transformation operators and then using a planning system to search for a reformulation of the given query description. The planner searches for a way to map each of the concepts and relations into concepts and relations for which data is available.

For example, consider the query shown in Figure 3. There are two concept expressions – one about ships and the other about seaports. In the reformulation that is found, the first step translates the seaport expression into a information-source-level expression. Unfortunately, none of the information sources contain information that corresponds to `channel_of`. Thus, the system must reformulate `channel_of`, using the decompose operator. This expresses the fact that `channel_of` is equivalent to performing a join over the keys for the seaport and channel concepts. The resulting reformulation is shown in Figure 6.

The next step reformulates the seaport portion of the query into a corresponding information source query. This can be done using the select-information-source operator, which selects between the GEO and PORT databases. In this case GEO is selected because the information on channels is only available in the GEO database. The resulting query is shown in Figure 7.

The channel and ship portions of the query are then similarly reformulated. The final query, which is the result of reformulating the entire query is shown in Figure 8.

```
(retrieve (?port_name ?depth ?ship_type ?draft)
            (:and (seaport ?port)
                  (port_name ?port ?port_name)
                  (geoloc_code ?port ?geocode)
                  (channel ?channel)
                  (geoloc_code ?channel ?geocode)
                  (channel_depth ?channel ?depth)
                  (ship ?ship)
                  (vehicle_type ?ship ?ship_type)
                  (range ?ship ?range)
                  (> ?range 10000)
                  (max_draft ?ship ?draft)
                  (> ?depth ?draft)))
```

Figure 6: Result of Applying the Decompose Operator to Eliminate `channel_of`

```
(retrieve (?port_name ?depth ?ship_type ?draft)
            (:and (seaports ?port)
                  (seaports.port_nm ?port ?port_name)
                  (seaports.glc_cd ?port ?geocode)
                  (channel ?channel)
                  (geoloc_code ?channel ?geocode)
                  (channel_depth ?channel ?depth)
                  (ship ?ship)
                  (vehicle_type ?ship ?ship_type)
                  (range ?ship ?range)
                  (> ?range 10000)
                  (max_draft ?ship ?draft)
                  (> ?depth ?draft)))
```

Figure 7: Result of Applying the Select-Information-Source Operator on Seaport

```
(retrieve (?port_name ?depth ?ship_type ?draft)
            (:and (seaports ?port)
                  (seaports.port_nm ?port ?port_name)
                  (seaports.glc_cd ?port ?glc_cd)
                  (channels ?channel)
                  (channels.glc_cd ?channel ?glc_cd)
                  (channels.ch_depth_ft ?channel ?depth)
                  (notional_ship ?ship)
                  (notional_ship.sht_nm ?ship ?ship_type)
                  (notional_ship.range ?ship ?range)
                  (> ?range 10000)
                  (notional_ship.max_draft ?ship ?draft)
                  (< ?draft ?depth))))
```

Figure 8: Result of Selecting Information Sources for Channels and Ships

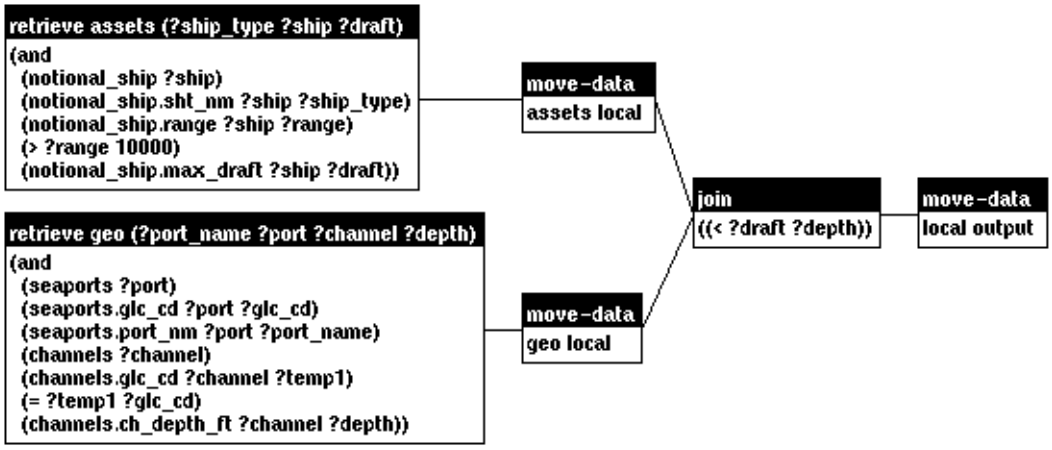

Figure 9: Parallel Query Access Plan

4 Query Access Planning

Once the system has reformulated the query so that it uses only terms from the information source models, the next step is to generate a query plan for retrieving and processing the data. The query plan specifies the precise operations that need to be performed, as well as the order in which they are to be performed.

There may be a significant difference in efficiency between different plans for a query. Therefore, the planner searches for a plan that can be implemented as efficiently as possible. To do this the planner must take into account the cost of accessing the different information sources, the cost of retrieving intermediate results, and the cost of combining these intermediate results to produce the final results. In addition, since the information sources are distributed over different machines or even different sites, the planner takes advantage of potential parallelism and generates subqueries that can be issued concurrently.

There are three basic operators that are used to plan out the processing of a query:

- Move-Data – Moves a set of data from one information source to another information source.
- Join – Combines two sets of data into a combined set of data using the given join relations.
- Retrieve-Data – Specifies the data that is to be retrieved from a particular information source.

Each of these operations manipulates one or more sets of data, where the data is specified in the same terms that are used for communicating with SIMS. This simplifies the input/output since there is no conversion between languages.

The planner is implemented in a version of UCPOP [Penberthy and Weld, 1992, Barrett *et al.*, 1993] that has been modified to generate parallel execution plans [Knoblock, 1994]. The system searches through the space of possible plans using a best-first search until a complete plan is found.

The plan generated for the example query in Figure 8 is shown in Figure 9. In this example, the system partitions the given query such that the ship information is retrieved in a single query to the ASSETS database and the seaport and channel information is retrieved in a single query to the GEO database. All of the information is brought into the local system (Loom) where the draft of the ships can be compared against the depth of the seaports. Once the final set of data has been generated, it is output for the user.

The system attempts to minimize the overall execution time by searching for a query that can be implemented as efficiently as possible. It does this by using a simple estimation function to calculate the expected cost of the various operations and then selecting a plan that has the lowest overall parallel execution cost. In the example, the system performs the join between the seaport and channel tables in the remote database since this will be cheaper than moving the tables into the local system. If the system could perform all of the work in one remote system, then it would completely bypass the local system and return the data directly

to the user. Once an execution plan has been produced, it is sent to the reformulation system for global optimization, as described in the next section.

5 Semantic Query Reformulation

This section first describes how the system reformulates a single subquery and then describes how the system reformulates the entire query plan.

5.1 Subquery Reformulation

The goal of the semantic query reformulation is to use reformulation to search for the least expensive query in the space of semantically equivalent queries. The reformulation from one query to another is done through logical inference using *database abstractions*, the abstracted knowledge of the contents of relevant databases. See [Hsu and Knoblock, 1993a] for an explanation of how rules like these are automatically learned. 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.

Consider the example shown in Figure 10. The input query retrieves ship types whose ranges are greater than 10,000 miles. This query could be expensive to evaluate because there is no index placed on the `range` attribute. The system must scan the entire database table `notional_ship` and check the values of `range` to retrieve the answer.

Input Query:

```
(db-retrieve (?sht-type ?ship ?draft)
  (:and (notional_ship ?ship)
        (notional_ship.sht_nm ?ship ?ship-type)
        (notional_ship.max_draft ?ship ?draft)
        (notional_ship.range ?ship ?range)
        (> ?range 10000)))
```

Figure 10: Example Subquery

A set of applicable rules for this query is shown in Figure 11. These rules would either be learned by the system or provided as semantic integrity constraints. Rule `R1` states that for all ships with maximum drafts greater than 10 feet, their range is greater than 12,000 miles. Rule `R2` states that all ships with range greater than 10,000 miles have fuel capacities greater than 5,000 gallons. The last rule `R3` simply states that the drafts of ships are greater than 12 feet when their fuel capacity is more than 4,500 gallons.

Based on these rules, the system infers a set of additional constraints and merges them with the input query. The resulting query is the first query shown in Figure 12. This query is semantically equivalent to the input query but is not necessary more efficient. The set of constraints in this resulting query is called the `inferred set`. The system will then select a subset of constraints in the `inferred set` to complete the reformulation. The selection is based on two criteria: reducing the total evaluation cost, and retaining the semantic equivalence. Detailed description of the algorithm is in [Hsu and Knoblock, 1993b]. In this example, the input query is reformulated into a new query where the constraint on the attribute `range` is replaced with a constraint on the attribute `max_draft`, on which there is a secondary index in the database. The reformulated query can therefore be evaluated more efficiently.

The reformulation is not limited to removing constraints. There are cases when the system can reformulate a query by adding new constraints or proving that the query is unsatisfiable. The `inferred set` turns out to be useful information for extending the algorithm to reformulate an entire query plan. Previous work only reformulates single database queries. In addition, our algorithm is polynomial in terms of the number of database abstraction rules and the syntactic length of the input query [Hsu and Knoblock, 1993b]. A large number of rules may slow down the reformulation. In this case, we can adopt sophisticated indexing and hashing techniques in rule matching, or constrain the size of the database abstractions by removing database abstractions with low utility.

Database Abstractions:

```
R1: (:if (:and (notional_ship ?ship)
               (notional_ship.max_draft ?ship ?draft)
               (notional_ship.range ?ship ?range)
               (> ?draft 10))
      (:then (> ?range 12000)))

R2: (:if (:and (notional_ship ?ship)
               (notional_ship.range ?ship ?range)
               (notional_ship.fuel_cap ?ship ?fuel_cap)
               (> ?range 10000))
      (:then (> ?fuel_cap 5000)))

R3: (:if (:and (notional_ship ?ship)
               (notional_ship.max_draft ?ship ?draft)
               (notional_ship.fuel_cap ?ship ?fuel_cap)
               (> ?fuel_cap 4500))
      (:then (> ?draft 12)))
```

Figure 11: Applicable Rules in the Database Abstractions

Query with inferred set:

```
(db-retrieve (?ship-type ?ship ?draft)
  (:and (notional_ship ?ship)
        (notional_ship.sht_nm ?ship ?ship-type)
        (notional_ship.max_draft ?ship ?draft)
        (notional_ship.range ?ship ?range)
        (notional_ship.fuel_cap ?ship ?fuel_cap)
        (> ?range 10000)
        (> ?fuel_cap 5000)
        (> ?draft 12)))
```

Reformulated Query:

```
(db-retrieve (?sht-type ?ship ?draft)
  (:and (notional_ship ?ship)
        (notional_ship.sht_nm ?ship ?ship-type)
        (notional_ship.max_draft ?ship ?draft)
        (> ?draft 12)))
```

Figure 12: Reformulated Query

5.2 Query Plan Reformulation

We can reformulate each 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 9, the constraint on the final subqueries involves the variables `?draft` and `?depth` that are bound in the preceding subqueries. If we can reformulate these preceding subqueries so that they retrieve only the data instances possibly satisfying the constraint `(< ?draft ?depth)` 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. We developed a query plan reformulation algorithm which achieves this by updating the database abstractions and rearranging constraints. We explain the algorithm using the query plan in Figure 9.

The algorithm first reformulates each subquery in the partial order (i.e., the data flow order) specified in the plan. The two subqueries to databases are reformulated first. The database abstractions are updated and saved in `Inferred-Set`, which is returned from the subquery reformulation to propagate the constraints to later subqueries. For example, when reformulating the subquery on `notional_ship`, `(> ?draft 12)` is inferred and saved in the inferred set. In addition, the constraint `(> ?range 10000)` in the original subquery

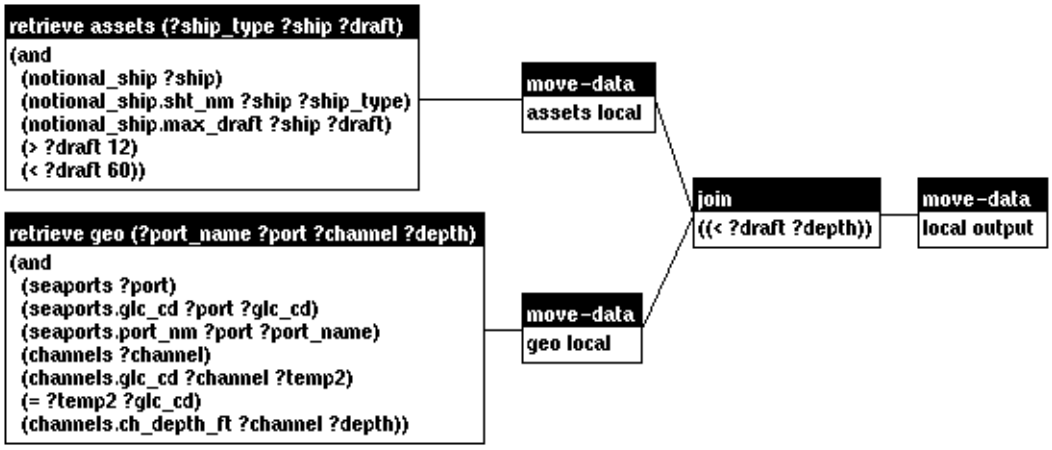

Figure 13: Reformulated Query Access Plan

is propagated along the data flow path to its succeeding subquery. Similarly, the system can infer the range of `?depth` in this manner. In this case, the range of `?depth` is `41` $\leq$ `?depth` $\leq$ `60`.

Now that the updated ranges for `?draft` and `?depth` are available, the subquery reformulation algorithm can infer from the constraint `(< ?draft ?depth)` a new constraints `(< ?draft 60)` and add it to the subquery for join. However, this constraint should be placed on the remote subquery instead of the local Loom query because it only depends on the data in the remote database. In this case, when updating the query plan with the reformulated subquery, the algorithm locates where the constrained variable of each new constraint is bound, and inserts the new constraint in the corresponding subqueries. In our example, the variable is bound by `(max_draft ?ship ?draft)` in the subquery on `notional_ship` in Figure 9. The algorithm will insert the new constraint on `?draft` in that subquery.

The semantics of the modified subqueries, such as the subquery on `notional_ship` in this example, are changed because of the newly inserted constraints. However, the semantics of the overall query plan remain the same. After all the subqueries in the plan have been reformulated, the system reformulates these modified subqueries again to improve their efficiency. In our example, the subquery reformulation algorithm is applied again to the `notional_ship` subquery. This time, no reformulation is found to be appropriate. The final reformulated query plan is returned and shown in Figure 13.

This query plan is more efficient and returns the same answer as the original one. In our example, the subquery to `notional_ship` is more efficient because the constraint on the attribute `range` is replaced with another constraint that can be evaluated more efficiently. The intermediate data are reduced because of the new constraint on the attribute `?draft`. The logical rationale of this new constraint is derived from the constraints in the other two subqueries: `(> ?range 10000)` and `(< ?draft ?depth)`, and the rules in the database abstractions. The entire algorithm for query plan reformulation is still polynomial. Our experiments shows that the overheads of reformulation is very small compared to the overall query processing cost. On a set of 32 example queries, the query reformulation yielded significant performance improvements with an average reduction in execution time of 43%.

6 Related Work

In the database community, there are a variety of approaches to handling distributed, heterogeneous, and autonomous databases [Reddy *et al.*, 1989, Sheth and Larson, 1990]. Of these approaches, the tightly-coupled federated systems (e.g., Multibase [Landers and Rosenberg, 1982]) are the most closely related to SIMS in that they attempt to support total integration of all information sources in the sense that SIMS provides. However, building tightly-coupled federated system requires constructing a global schema for the databases to be combined and then hard-wiring the mapping between the global schema and the local schemas.

The information source selection in SIMS is used instead of the standard schema integration used in database systems [Batini and Lenzerini, 1986]. Our approach requires constructing a general domain model that encompasses the relevant parts of the database schemas. Then each of the database models is related to this general domain model. The integration problem is shifted from how to build a single integrated model to how to map between the domain and the information source models. After defining this mapping, the remaining integration process is performed automatically by the reformulation system using operators that transform the high-level query into an information source-level query. An important advantage of this approach is that it is easier and simpler to map one model to another model than it is to provide a completely integrated view of a number of models with different structures, types, and dependencies.

The query planning in SIMS is similar to the query access planning performed in many database systems. The primary difference between the planning in SIMS and query access planning in other systems is that the planning in SIMS is performed by an AI planner, which provides a level of flexibility that goes well beyond what standard database systems can provide. Currently, it allows the system to construct a plan that takes into account the availability of the various databases. In addition, we are in the process of integrating the planning with the execution system, which will allow the system to dynamically replan parts of a query that fail while continuing to execute the other subqueries of the overall plan.

The semantic reformulation approach to query optimization was first developed by King [King, 1981] and has since been extended in a number of systems [Adam *et al.*, 1993, Shenoy and Ozsoyoglu, 1989, Shekhar *et al.*, 1988, Siegel, 1988]. Our approach to this problem differs from other related work in 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*. Then the system selects the most efficient set of the constraints from the inferred set to form the reformulated subqueries.

The Carnot project [Collet *et al.*, 1991] is similar to SIMS in that it uses a knowledge base to integrate a variety of information sources. Carnot 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 [Lenat and Guha, 1990] 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.

7 Conclusion

SIMS provides a flexible system for processing queries to multiple information sources. In this paper we described the process of transforming a domain-level query into an efficient and executable query plan. This process consists of three steps. First, the information sources are selected in the process of transforming the domain-level query into an information source-level query. Second, the system generates a query plan, which specifies all the operations to be performed on the data as well as the order of these operations. Third, the system performs global optimization through the use of database abstractions to reformulate the query plan.

Future work will focus on extending the selection, planning, and reformulation capabilities described in this paper. An important issue that we have not yet addressed is how to handle the various forms of incompleteness and inconsistency that will inevitably arise from using autonomous information sources. We plan to address these issues by exploiting available domain knowledge and employing more sophisticated planning and reasoning capabilities to both detect and recover from these problems.

References

[Adam *et al.*, 1993] N.R. Adam, A. Gangopadhyay, and J. Geller. Design and implementation of a knowledge-based query processor. *International Journal of Intelligent and Cooperative Information Systems*, 2(2):107–125, 1993.

[Arens *et al.*, 1993] Yigal Arens, Chin Y. Chee, Chun-Nan Hsu, and Craig A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal on Intelligent and Cooperative Information Systems*, 2(2):127–158, 1993.

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

[Batini and Lenzerini, 1986] Carlo Batini and Maurizio Lenzerini. A comparative analysis of methodologies for database schema integration. *ACM Computer Survey*, 18(4):323–364, 1986.

[Collet *et al.*, 1991] 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.

[Hsu and Knoblock, 1993a] Chun-Nan Hsu and Craig A. Knoblock. Learning database abstractions for query reformulation. In *Proceedings of the AAAI Workshop on Knowledge Discovery in Databases*, 1993.

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

[King, 1981] Jonathan Jay King. *Query Optimization by Semantic Reasoning.* Ph.D. Thesis, Stanford University, Department of Computer Science, 1981.

[Knoblock, 1994] Craig A. Knoblock. Generating parallel execution plans with a partial-order planner. Information Sciences Institute,University of Southern California, 1994.

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

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

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

[MacGregor, 1990] 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.

[Penberthy and Weld, 1992] J. Scott Penberthy and Daniel S. Weld. Ucpop: A sound, complete, partial order planner for adl. In *Proceedings of KR-92*, pages 189–197, 1992.

[Reddy *et al.*, 1989] 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.

[Shekhar *et al.*, 1988] Shashi Shekhar, Jaideep Srivastava, and Soumitra Dutta. A formal model of trade-off between optimization and execution costs in semantic query optimization. In *Proceedings of the 14th VLDB Conference*, Los Angeles, CA, 1988.

[Shenoy and Ozsoyoglu, 1989] S.T. Shenoy and Z.M. Ozsoyoglu. Design and implementation of a semantic query optimizer. *IEEE Trans. Knowledge and Data Engineering*, I(3):344–361, 1989.

[Sheth and Larson, 1990] Amit P. Sheth and James A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, 1990.

[Siegel, 1988] 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.