To appear in the *Proceedings of the Second International Conference on Cooperative Information Systems*, University of Toronto Press, Toronto, Ontario, Canada, 1994

Cooperating Agents for Information Retrieval*
Research

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

Abstract

With the vast number of information resources available today, a critical problem is how to locate, retrieve and process information. It would be impractical to build a single unified system that combines all of these information resources. A more promising approach is to build specialized information retrieval agents that provide access to a subset of the information resources and can send requests to other information retrieval agents when appropriate. In this paper we present the architecture of the individual information retrieval agents and describe how this architecture supports a network of cooperating information agents. We describe how these information agents represent their knowledge, communicate with other agents, dynamically construct information retrieval plans, and learn about other agents to improve efficiency. We have already built a small network of agents that have these capabilities and provide access to information for transportation planning.

Introduction

With the expanding amount of information available, the problem of how to combine distributed, heterogeneous information sources becomes more and more critical. The available information sources include traditional databases, flat files, knowledge bases, programs, etc. Traditional approaches to building distributed or federated systems do not scale well to the large, diverse, and growing number of information sources. Recent Internet systems such as Mosaic, WAIS, and Gopher allow users to search through large numbers of information sources, but provide very limited capabilities for locating, combining, processing, and organizing information.

A promising approach to this problem is to provide access to the large number of information sources by organizing them into a network of *information agents* [Papazoglou *et al.*, 1992]. The goal of each agent is to provide information and expertise on a specific topic by drawing on relevant information from other information agents. Similar to the way current information sources are independently constructed, these information agents can be developed and maintained separately, drawing on the other available information agents and providing a new information source that others can then build upon. Each information agent is another information source, but provides a desperately needed abstraction of the many information sources available. Similarly, an information source, such as a database or program, can be turned into a simple information agent by building the appropriate interface code around the information source. Given this simple mapping between information agent and information source, we will use these terms interchangeable throughout the rest of this paper.

Figure 1 shows an example network of information retrieval agents. This network includes agents such as a USC computer science technical report agent, which is an agent that only provides information about and access to the department technical reports. This agent is in turn used to construct both a computer science technical report agent that spans multiple universities as well as USC technical report agent that spans departments within the university. These agents could in turn be used to construct other agents, and so on. An important feature of this organization is that the individual agents can be independently built and maintained. This makes it possible to scale the architecture to large numbers of information sources.

To build a network of specialized information agents, we need an architecture for a single agent that can be instantiated to provide multiple agents. In previous work we developed an information server, called SIMS [Arens *et al.*, 1993], which

*The research reported here was supported in part by Rome Laboratory of the Air Force Systems Command and the Advanced Research Projects Agency under contract no. F30602-91-C-0081, and in part by the National Science Foundation under grant number IRI-9313993. The 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 RL, ARPA, NSF, the U.S. Government, or any person or agency connected with them.

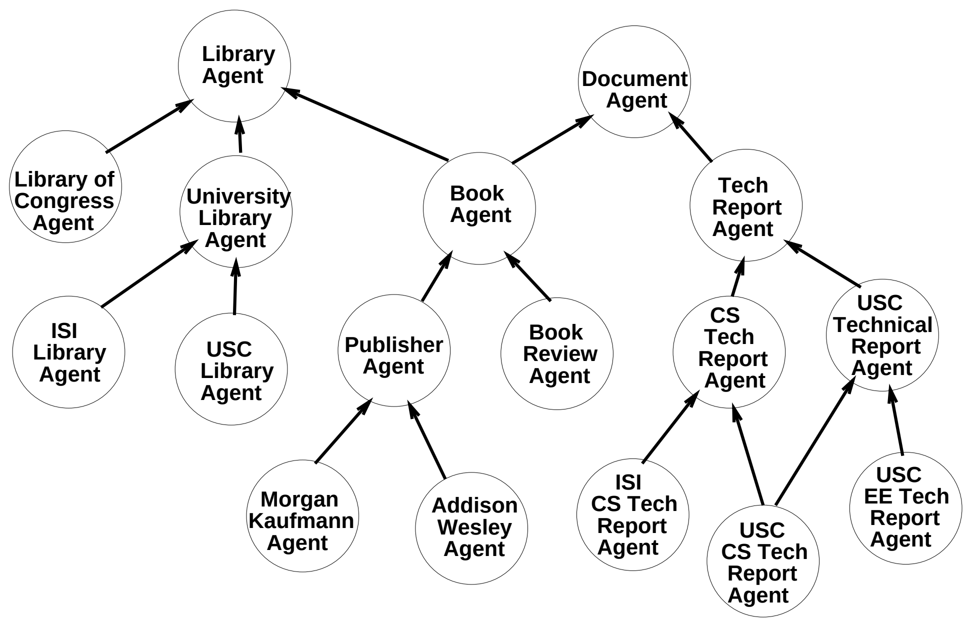

Figure 1: Network of Information Retrieval Agents

provides access to heterogeneous data and knowledge bases. In addition, Pastor et al. [1992] developed a system called the Loom Interface Manager (LIM), which we use to build agents for accessing individual relational databases. Using SIMS and LIM, we have built a small network of information retrieval agents that interact over the Internet. We are now in the process of building a more ambitious network of interacting agents.

The SIMS architecture is shown in Figure 2. Each SIMS agent contains a detailed model of its domain of expertise and models of the information sources that are available to it. Given an information request, an agent selects an appropriate set of information sources, generates a plan to retrieve and process the data, uses knowledge about the information sources to reformulate the plan for efficiency, and then executes it.

This paper presents the design of an individual SIMS agent and discusses the issues that arise in using this design to build a network of cooperating information retrieval agents. First, we describe how the knowledge of an agent is represented. Second, we describe how the agents exchange queries and data with one another. Third, we describe how information requests are flexibly and efficiently processed. Fourth, we describe how the system learns about the other agents in order to improve performance over time. Fifth, we identify how the different features of this design support flexible, efficient, and modular agents. Sixth, we describe the closely related work in this area. Finally, we conclude with a discussion of the current status and the work that remains to be done.

Representing an Agent's Knowledge

Each agent contains a model of its domain of expertise and models of the other agents and information sources that can provide relevant information. We will refer to these two types of models as the domain model and information source models. The domain model provides descriptions of the classes of objects in the domain, relationships between these classes (e.g.., subclass and superclass), relations on each class, and other domain-specific information. The information source models describe both the contents of the information sources and the relationship between those models and the domain model. Both the domain and information source models are expressed in the Loom knowledge representation language [MacGregor, 1990]. Loom is an AI knowledge representation system based on KL-ONE [Brachman and Schmolze, 1985]. Loom provides a language for representing hierarchies of classes and relations, as well as efficient mechanisms for classifying instances of classes and reasoning about descriptions of object classes.

The domain and information source models constitute the general knowledge of an agent and are used to determine how to process an information request. The domain model of an agent defines its area of expertise and the terminology for interacting with that agent. The information source models describe the resources that are available to an agent to answer information requests. These models do not need to contain a complete description of another agent or information source, but rather only the portions of those agents or information sources that are directly relevant. Specializing the

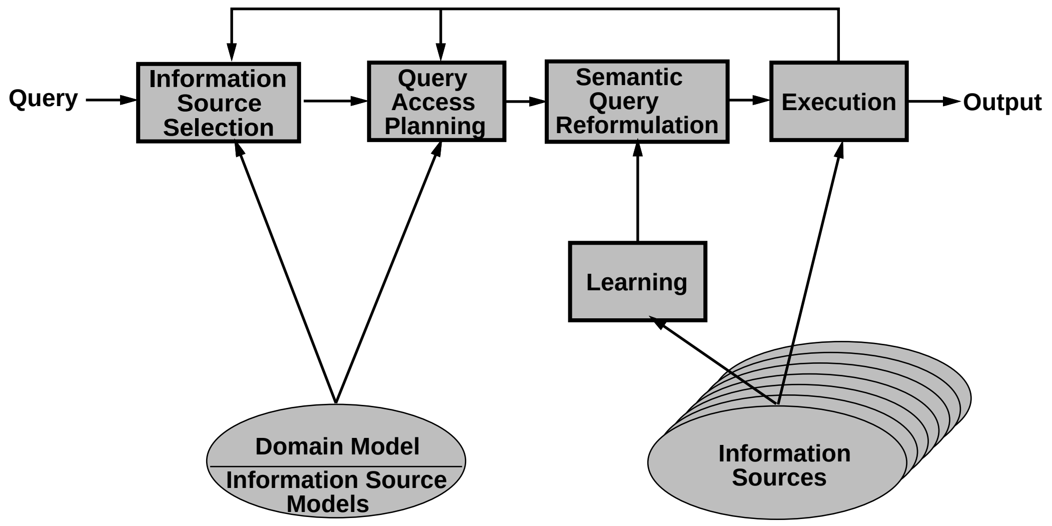
Query
Information Source Selection
Query Access Planning
Semantic Query Reformulation
Execution
Output
Learning
Domain Model
Information Source Models
Information Sources

Figure 2: The Architecture of an Agent

agents for specific areas provides a modular organization of the vast number of information sources and provides a clear delineation of the types of queries each agent can handle. In complex domains, the domain can be broken down into meaningful subparts and an information agent can be built for each subpart.

Domain Models

Each information agent 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, which involves information about the movement of personnel and materiel from one location to another using aircraft, ships, trucks, etc.

As described above, the application domain models are defined in the Loom knowledge representation system. This provides a semantic description of the objects and relations in a domain, which is used extensively for processing queries. Figure 3 shows a fragment of the domain model in the transportation planning domain. In this figure, the nodes represent classes of objects, 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 intended to be a description of the application domain from the point of view of users or other information agents that may need to obtain information about the application domain. The terms in the domain model provide the *language* to define the contents of an information source to the agent.

Modeling Information Sources

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

Figure 4 illustrates how an information source is modeled in Loom and how it is related to the domain model. 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 agent, which has information about seaports.

Communication

Queries to an information agent are expressed in the Loom query language. These queries are composed of terms in a general domain model, so there is no need for other agents or a user to know or even be aware of the terms used in the underlying information sources. Given a query, an information agent identifies the appropriate information sources and issues queries to those sources to obtain the requisite data for answering the query. To do this, an information agent translates the domain-level query into a set of queries to more specialized information agents using the terms appropriate to each of those agents.

The queries to other agents are also expressed in the Loom query language. In order to make

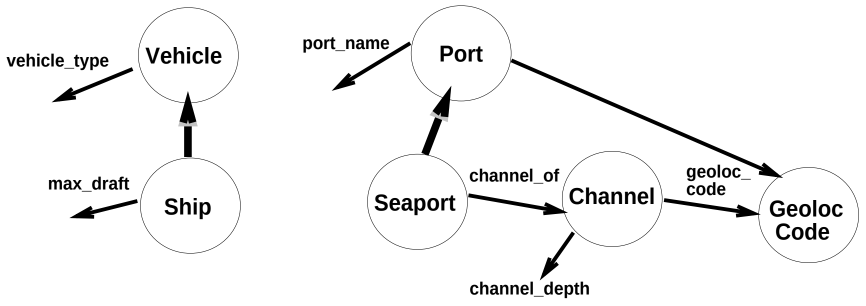

Figure 3: Fragment of the Domain Model

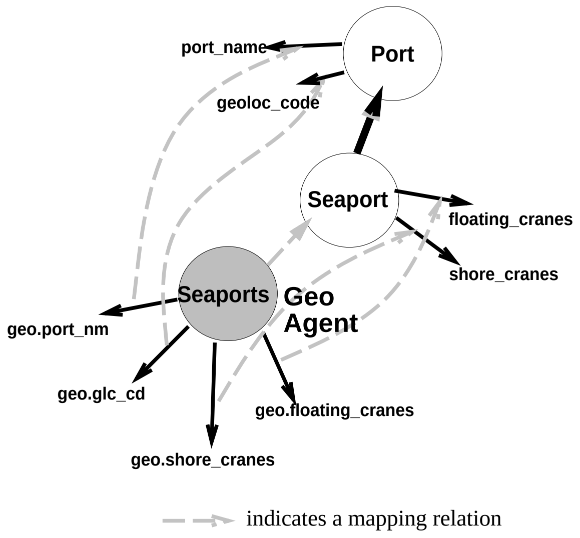

Figure 4: Relating an Information Source Model to a Domain Model

an existing database or other application program available to the network of agents requires building a *wrapper* around the existing system to turn it into an agent with access to that information source. Note that only one such wrapper would need to be built for any given type of information source (e.g., relational database, object-oriented database, flat file, etc). The advantage of this approach is that it greatly simplifies the individual agents since they only needs to handle one underlying language. This makes it possible scale the network into many agents with access to many different types of information sources.

Figure 5 illustrates a query expressed in the Loom language. This query requests all seaports and the corresponding ships 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 the classes of information, and the relations describe the constraints on these classes. The first clause of the query is an example of a concept expression and specifies that the variable `?port` describes a member of the class `seaport`. The second clause 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`. This query requests all seaport and ship pairs where the depth of the port exceeds the draft of the ship.

```
(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)
          (max_draft ?ship ?draft)
          (> ?depth ?draft)))
```

Figure 5: Example Loom Query

In addition to sending queries to other agents, the agents also need the capability to send back objects in response to their queries. Communication between information agents is done using the Knowledge Query Manipulation Language (KQML)[Finin *et al.*, 1992]. KQML is an agent communication language that handles the interface protocols for transmitting queries, returning the appropriate information, and building the appropriate internal structures. We currently use this language to send queries between a SIMS agent and the LIM agents, which provide access to relational databases [Pastor *et al.*, 1992].

Processing an Information Request

The core contribution of an information agent 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, an agent needs the capabilities to *dynamically* select an appropriate set of information sources, construct a plan for retriev-

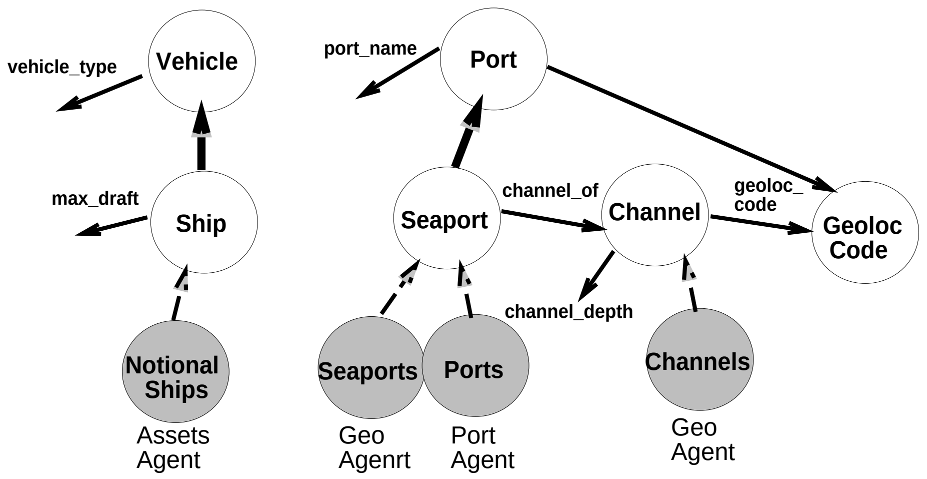

Figure 6: Fragment of the Domain and Information Source Models

ing and processing the information, and optimize this plan to ensure that the data is retrieved efficiently. This section describes each of these processing steps in turn.

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 mapping from the concepts in the domain model to the concepts in the information source models. If the user requests information about ports and there is a single information agent that provides access to information on ports, then the mapping is straightforward. However, in some cases there may be several agents that provide access to the same information and in other cases no single agent can provide the required information and it will need to be drawn from several different agents. The process of selecting the information sources is performed by reformulating the terms in the original query into the terms that correspond to the available information sources.

Consider the fragment of the knowledge base shown in Figure 6, which covers the knowledge relevant to the example query in Figure 5. The concepts Seaport, Channel and Ship have subconcepts, shown by the shaded circles, that correspond to concepts whose instances can be retrieved from some information agent. Thus, the GEO AGENT contains information about both seaports and channels, and the PORT AGENT contains information about only seaports. Thus, if the user asks for seaports, then the query must be translated into one of the information source concepts — seaports from the GEO AGENT or ports from the PORT AGENT.

In order to select the information sources for answering a query, an agent 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, Partition-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 number of different information sources used to answer a query.

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 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.

Partition-Concept 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, if a query requests information about ports and there are no information source that cover ports, it may be possible to reformulate the query into a set of subqueries that cover ports. If ports are covered by seaports, airports, and rail ports, then the original query can be replaced by queries on each of these subconcepts.

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 mapping from each of the concepts and relations in the query into concepts and relations for which data is available.

For example, consider the query shown in Figure 5. There are two concept expressions – one about ships and the other about seaports. In the first step, the system attempts to translate 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 7.

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 information agents. In this case GEO is selected because the information on channels is only available in the GEO agent. 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 7: 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 ?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

Query Access Planning

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

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.

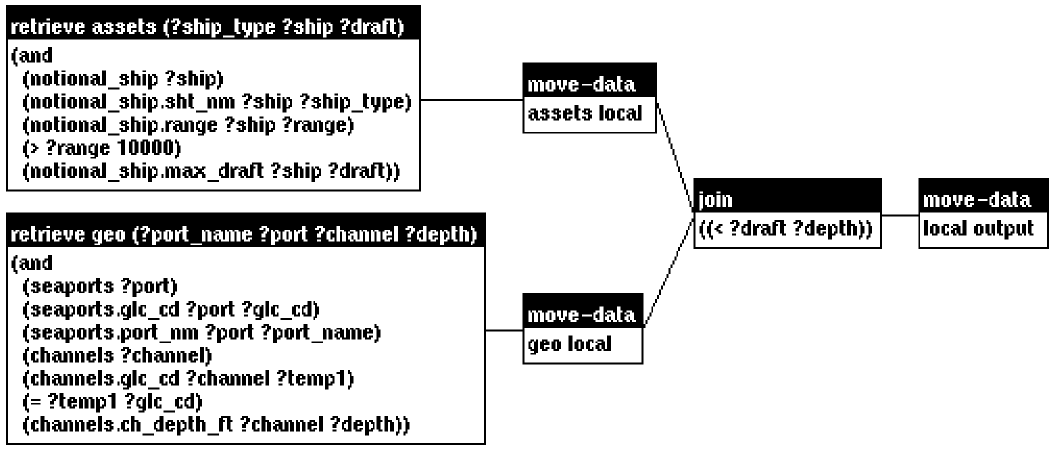
retrieve assets (?ship_type ?ship ?draft)
(and
(notional_ship ?ship)
(notional_ship.sht_nm ?ship ?ship_type)
(notional_ship.range ?ship ?range)
(> ?range 10000)
(notional_ship.max_draft ?ship ?draft))
retrieve geo (?port_name ?port ?channel ?depth)
(and
(seaports ?port)
(seaports.glc_cd ?port ?glc_cd)
(seaports.port_nm ?port ?port_name)
(channels ?channel)
(channels.glc_cd ?channel ?temp1)
(= ?temp1 ?glc_cd)
(channels.ch_depth_ft ?channel ?depth))
move-data
assets local
move-data
geo local
join
((< ?draft ?depth))
move-data
local output

Figure 9: Parallel Query Access Plan

There are five general operators that are used to plan out the processing of a query:

- Move – 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 – Specifies the data that is to be retrieved from a particular information source.
- Select – Selects a subset of the data using the given constraints.
- Assign – Constructs a new term in the data from some combination of the existing data.

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 [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 agent and the seaport and channel information is retrieved in a single query to the GEO agent. 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 returned by the agent.

The planner 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 agent leaves the join between the seaports and channels to be performed by the remote GEO agent since this will be cheaper than moving the information into the local system. If the system could perform all of the work in one remote system, then it would completely bypass the local agent and return the data directly to the agent that requested the information. Once an execution plan has been produced, it is sent to the reformulation system for global optimization, as described in the next section.

Semantic Query 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 *information-source abstractions*, the abstracted knowledge of the contents of relevant information sources. See [Hsu and Knoblock, 1993a] for an explanation of how rules like these are automatically learned. The information-source abstractions describe the information in terms of a set of closed formulas of first-order logic. These formulas describe an information source in the sense that they are true with regard to all instances in the information source.

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 all of the instances of `notional_ship` and check the values of the `range` to retrieve the answer.

A set of applicable rules for this query is shown

Input Query:
```
(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

in Figure 11. These rules would either be learned by the system or provided as semantic integrity constraints about an information source. 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.

Information-Source 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 Information-Source Abstractions

Based on these rules, the reformulation component 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`, which turns out to be cheap to access because of the way the information is indexed. The reformulated query can therefore be evaluated more efficiently.

Query with inferred set:
```
(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:
```
(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

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 information source abstraction rules and the syntactic length of the input query. 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 information-source abstractions by removing information-source abstractions with low utility.

We can reformulate each subquery in the query plan with the subquery reformulation algorithm and improve their efficiency. However, the most expensive aspect of queries to multiple information sources 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 information-source 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 information sources are reformulated first. The information-source 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 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` ≤ 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 the join operation. 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 information source. 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 reformulated subquery of the final query plan is shown in Figure 13.

The resulting 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

Reformulated Subquery:
```
(retrieve (?sht-type ?ship ?draft)
  (:and
    (notional_ship ?ship)
    (notional_ship.sht_nm ?ship ?ship-type)
    (notional_ship.max_draft ?ship ?draft)
    (> ?draft 12)
    (< ?draft 60)))
```

Figure 13: Reformulated Query

the constraints in the other two subqueries: `(> ?range 10000)` and `(< ?draft ?depth)`, and the rules in the information-source abstractions. The entire algorithm for query plan reformulation is still polynomial. Our experiments shows that the overhead 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%.

Learning

An intelligent agent for information retrieval should be able to improve its performance over time. To achieve this goal, the information agents currently support two forms of learning. First, they have the capability to cache frequently retrieved or difficult to retrieve information. Second, for those cases where caching is not appropriate, an agent can learn about the contents of the information sources in order to minimize the costs of retrieval. Since information retrieval agents serve as information sources for other agents, both caching and learning can be applied to information agents as well as data and knowledge bases. This section describes these two forms of learning.

Caching Retrieved Data

Data that is required frequently or is very expensive to retrieve can be cached in the local agent and then retrieved more efficiently. An elegant feature of using Loom to model the domain is that cached information can easily represented and stored in Loom. The data is currently brought into the local agent for processing, so caching is simply a matter of retaining the data and recording what data has been retrieved.

To cache retrieved data into the local agent requires formulating a description of the data so it can be used to answer future queries. This can be extracted from the initial query, which is already expressed in the form of a domain-level description of the desired data. 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 system has defined a new class and stored the data under this class, the cached information becomes a new information source for the agent. The reformulation operations, which map a domain query into a set of information source queries, will automatically consider this new information source. Since the system takes the retrieval costs into account in selecting the information sources, it will naturally gravitate towards using cached information where appropriate. In those cases where the cached data does not capture all of the required information, it may still be cheaper to retrieve everything from the remote site. However, in those cases where the cached information can be used to avoid an external query, the use of the stored information can provide significant efficiency gains.

The use of caching raises a number of important questions, such as which information should be cached and how the cached information is kept up-to-date. We are exploring caching schemes where, rather than caching the answer to a specific query, general classes of frequently used information are stored. This is especially useful in the Internet environment where a single query can be very expensive and the same set of data is often used to answer multiple queries. To avoid problems of information becoming out of date, we have focused on caching relatively static information.

Learning about the Contents of Information Sources

The agent's goal is to provide efficient access to a set of information sources. Since accessing and processing information can be very costly, the system strives for the best performance that can be provided with the resources available. This means that when it is not processing queries, it gathers information to aid in future retrieval requests. The information agents improve performance by learning about the contents of the information sources [Hsu and Knoblock, 1993a].

The learning is triggered when an agent detects an excessively expensive query. In this way, the agent will incrementally gather a set of rules to reformulate expensive queries. The learning subsystem uses induction on the contents of the information sources to construct a less expensive specification of the original query. This new query is then compared with the original to generate a set of rules that describe the relationships between the two equivalent queries. The learned rules are integrated into the agent's domain model and then used for semantic query reformulation.

Advantages of the Architecture

Now that we have described the basic architecture, this section first reviews the critical features of this architecture and then describes the advantages provided by these features.

The critical features of this architecture that support multiple cooperating agents are:

1. A uniform query language that is used as the interface for the user as well as the interface between agents.
2. A unified model of the domain and separate models of the contents of the information sources.
3. The dynamic selection of an appropriate set of information sources.
4. The generation of parallel query access plans.
5. The use of semantic knowledge to optimize the query plans.
6. A learning system that improves the performance of an agent by caching frequently used information and learning about the contents of the information sources.

First, the uniform query language and separate models provide a **modular** architecture for multiple information agents. An information agent for one domain can serve as an information source to other information agents. This is can done seamlessly since the interface to every information source is exactly the same – it takes a query in a uniform language (i.e., Loom) as input and returns the data requested by the query. The domain model provides a uniform language for queries about information in any of the sources to which an agent has access. The contents of each agent is represented as a separate information source and is mapped to the domain model of an agent. Each information agent can export some or all of its domain model, which can be incorporated into another information agent's model. This exported model forms the shared terminology between agents.

Second, the separate domain and information source models and the dynamic information source selection make the overall architecture easily **extensible**. Adding a new information source simply requires building a model of the information source that describes the contents of the information source as well as how it relates to the domain model. It does not require integrating the new information source model with the other information source models since the mapping between domain and information source models is not fixed. Similarly, changes to the contents of information sources require only changing the model of the specific information source. Since the selection of the information sources is performed dynamically, when an information request is received, the agent will select the most appropriate information source that is currently available.

Third, the separate domain and information source models and the dynamic information source selection also make the agents very **flexible**. The agents can choose the appropriate information sources based on what they contain, how quickly they can answer a given query, and what resources are currently available. If a particular information source or network goes down or if the data is available elsewhere, the system will retrieve the data from sources that are currently available. An agent can take into consideration the rest of the processing of a query, so that the system can take advantage of those cases where retrieving the data from one source is much cheaper than another source because the remote system can do more of the processing. This flexibility also makes it possible to cache and reuse information without extra work or overhead.

Fourth, building parallel query access plans, using semantic knowledge to optimize the plans, caching retrieved data, and learning about information sources provide **efficient** access to large numbers of information sources. The planner generates plans that minimize the overall execution time by maximizing the parallelism in the plan to take advantage of the fact that separate information sources can be accessed in parallel. The semantic query reformulation provides a global optimization step that minimizes the amount of intermediate data that must be processed. The ability to cache retrieved data allows an agent to store frequently used or expensive-to-retrieve information in order to provide the requested information more efficiently. And the ability to learn about the contents of the information sources allows the agent to exploit time when it would not otherwise be used to improve its performance on future queries.

Related Work

A great deal of work has been done on building agents for various kinds of tasks. This work is quite diverse and has focused on a variety of issues. First, there has been work on multi-agent planning and distributed problem solving, which is described in [Bond and Gasser, 1988]. The body of this work deals with the issues of coordination, synchronization, and control of multiple autonomous agents. Second, a large body of work has focused on defining models of beliefs, intentions, capabilities, needs, etc., of an agent. [Shoham, 1993] provides a nice example of this work and a brief overview of the related work on this topic. Third, there is more closely related work on developing agents for information gathering.

The problem of information gathering is also quite broad and the related work has focused on various issues. Kahn and Cerf [1988] proposed an architecture for a set of information-management agents, called Knowbots. The various agents are hard-coded to perform particular tasks. Etzioni et al. [1992, 1994] have built agents for the Unix domain that can perform a variety of Unix tasks. This work has focused extensively on reasoning and planning with incomplete information, which arises in many of these tasks. Levy el al. [1994] are also working on building agents for retrieving information from the Internet. The focus of their work has been on developing a formal framework for selecting a minimal set of sites to answer a query.

In contrast to much of this previous work, the focus of our work is on flexible and efficient retrieval of information from heterogeneous information sources. Since most of these other systems have in-memory databases, they assume that the cost of a database retrieval is small or negligible. One of the critical problems when dealing with large databases is how to issue the appropriate queries to efficiently access the desired information. We are focusing on the problems of how to organize, manipulate, and learn about large quantities of data.

Research in databases has also focused on building integrated or federated systems that combine information sources [Landers and Rosenberg, 1982, Sheth and Larson, 1990]. The approach taken in these systems is to first define a global schema, which integrates the information available in the different information sources. However, this approach is unlikely to scale to the large number of evolving information sources (e.g., the Internet) since building an integrated schema is labor intensive and difficult to maintain, modify, and extend.

The Carnot project [Collet *et al.*, 1991] also integrates heterogeneous databases using a knowledge representation system. Carnot uses a knowledge base to build a set of articulation axioms that describe how to map between SQL queries and domain concepts. After the axioms are built the domain model is no longer used or needed. In contrast, the domain model of one of our agents is an integral part of the system, and allows an agent to both combine information stored in the knowledge base and to reformulate queries.

Conclusion

This paper described the SIMS architecture for intelligent information retrieval agents. This particular architecture has a number of important features: (1) modularity in terms of representing an information agent and information sources, (2) extensibility in terms of adding new information agents and information sources, (3) flexibility in terms of selecting the most appropriate information sources to answer a query, and (4) efficiency in terms of minimizing the overall execution time for a given query.

To date, we have built information agents that plan and learn in the transportation planning domain. These agents contain a detailed model of this domain and extract information from a set of nine relational databases. The agents select appropriate information sources, generate parallel plans, execute the queries in parallel, and learn about the information sources.

Future work will focus on extending the planning and learning 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. Our plan is 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.

Acknowledgments

We gratefully acknowledge the contributions of Chin Chee and Hoh In in building the various components in SIMS. We also thank Don Mckay, Jon Pastor, and Robin McEntire at Unisys for setting up the LIM agents and providing us with an implementation of KQML.

References

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

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

[Bond and Gasser, 1988] Bond, Alan H. and Gasser, Les 1988. *Readings in Distributed Artificial Intelligence.* Morgan Kaufmann, San Mateo.

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

[Collet *et al.*, 1991] Collet, Christine; Huhns, Michael N.; and Shen, Wei-Min 1991. Resource integration using a large knowledge base in carnot. *IEEE Computer* 55–62.

[Etzioni *et al.*, 1992] Etzioni, Oren; Hanks, Steve; Weld, Daniel; Draper, Denise; Lesh, Neal; and Williamson, Mike 1992. An approach to planning with incomplete information. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, Cambridge, MA.

[Etzioni *et al.*, 1994] Etzioni, Oren; Golden, Keith; and Weld, Dan 1994. Tractable closed-world reasoning with updates. In *Fourth International Conference on Principles of Knowledge Representation and Reasoning*, Bonn, Germany.

[Finin *et al.*, 1992] Finin, Tim; Fritzson, Rich; and McKay, Don 1992. A language and protocol to support intelligent agent interoperability. In *Proceedings of the CE and CALS*, Washington, D.C.

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

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

[Kahn and Cerf, 1988] Kahn, Robert E. and Cerf, Vinton G. 1988. An open architecture for a digital library system and a plan for its development. Technical report, Corporation for National Research Initiatives.

[Knoblock, 1994] Knoblock, Craig A. 1994. Generating parallel execution plans with a partial-order planner. In *Artificial Intelligence Planning Systems: Proceedings of the Second International Conference (AIPS94)*, Chicago, IL.

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

[Levy *et al.*, 1994] Levy, Alon Y.; Sagiv, Yehoshua; and Srivastava, Divesh 1994. Torwards efficient information gathering agents. In *Proceedings of the AAAI Spring Symposium Series on Software Agents*, Palo Alto, CA.

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

[Papazoglou *et al.*, 1992] Papazoglou, Mike P.; Laufmann, Steven C.; and Sellis, Timos K. 1992. An organizational framework for cooperating intelligent information systems. *International Journal of Intelligent and Cooperative Information Systems* 1(1):169–202.

[Pastor *et al.*, 1992] Pastor, Jon A.; McKay, Donald P.; and Finin, Timothy W. 1992. View-concepts: Knowledge-based access to databases. In *Proceedings of the First International Conference on Information and Knowledge Management*, Baltimore, MD. 84–91.

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

[Shoham, 1993] Shoham, Yoav 1993. Agent-oriented programming. *Artificial Intelligence* 60(1):51–92.