1 Agents for Information Gathering

Craig A. Knoblock and José Luis Ambite
Information Sciences Institute and Department of Computer Science
University of Southern California

Abstract

With the vast number of information resources available today, a critical problem is how to locate, retrieve and process information. It is impractical to build a single unified system that combines all of these information resources. A more modular approach is to build specialized information agents where each agent provides access to a subset of these resources and can serve as an information source to other agents. In this paper we present the architecture of the individual information 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 their accuracy and efficiency. We have already built a small network of agents that have these capabilities and provide access to information for logistics planning.

1.1 Introduction

With the growing number of information sources 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 World Wide Web browsers 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. To build such a network, we need an architecture for a single agent that can be instantiated to provide multiple agents. We will base our design on our previous work on SIMS [Arens *et al.*, 1993, Knoblock *et al.*, 1994, Arens *et al.*, 1995], an information mediator that provides access to heterogeneous data and knowledge bases.

This chapter focuses on the design of an individual (SIMS) agent and discusses the issues that arise in using this design to build a network of information gathering agents. In Section 1.2, we present an approach to organizing a group of information agents.

Then, in Sections 1.3 to 1.6, we present the design of the individual agents. Section 1.3 describes how the knowledge of an agent is represented as a set of interrelated models. Section 1.4 describes how the agents exchange queries and data with one another. Section 1.5 describes how information requests are flexibly and efficiently processed. Section 1.6 describes how an agent learns about the other agents in order to improve its accuracy and performance over time. Section 1.7 describes closely related work in this area. Section 1.8 concludes with a discussion of the current status and the work that remains to be done.

1.2 Agent Organization

In order to effectively use the many heterogeneous information sources available in large computer networks, such as the Internet, we need some form of organization. The concept of an agent that provides expertise on a specific topic, by drawing on relevant information from a variety of sources, offers the basic building block. We believe that agents will be developed to serve the information needs of users in particular domains. More complex agents that deal with wider and/or deeper areas of knowledge will appear in an evolutionary fashion, driven by the market forces of applications that can benefit from using them.

Similar to the way current information sources are independently constructed, information agents can be developed and maintained separately. They will draw on other information agents and data repositories to provide a new information source that others can build upon in turn. Each information agent is another information source, but provides an abstraction of the many information sources available. An existing database or program can be turned into a simple information agent by building the appropriate interface code, called a *wrapper*, that will allow it to conform to the conventions of the organization. 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 need to handle one underlying language. This makes it possible scale the network into many agents with access to many different types of information sources.

Some agents will answer queries addressed to them, but will not actively originate requests for information to others; we will refer to these as data repositories. Usually, these agents will correspond to databases, which are systems specially designed to store a large amount of information but in which the expressive power of their data description languages and their reasoning abilities have been traded off for efficiency. In the rest of the chapter, we will use the term data repository when we want to emphasize

such behaviour, otherwise we will use the terms information agent or information source (interchangeably).

Figure 1.1 shows an example network of information agents that will be used throughout the chapter to explain different parts of the system. The application domain of interest is Logistics Planning. In order to perform its task, this agent needs to obtain information on different topics, such as transportation capabilities, weather conditions and geographic data. The other agents also integrate a number of sources of information that are relevant to their domain of expertise. For example, the `Sea_Agent` combines assets data from the `Naval_Agent` (such as ships from different fleets), harbor data from the `Harbor_Agent` and the `Port_Agent` (such as storage space or cranes in harbors, channels, etc, information that has been obtained, in turn, from repositories of different geographical areas). These four agents (circled by the dotted line in the figure) will be examined in greater detail in the following sections.

There are several points to note about this network that relate to the autonomy of the agents. First, each agent may choose to integrate only those parts of the ontologies of its information sources necessary for the task that it is designed for. For example, the `Transportation_Agent` might have a fairly complete integration of the `Sea, Land and Air agents`, while the `Logistics_Planning_Agent` might only draw on some parts of the knowledge of the `Weather` and `Geographic agents`. Second, we may need to build new agents if we cannot find an existing one that contains all the information needed. For example, if the `Geographic_Agent` did not include some particular geopolitical facts required by the `Logistics_Planning_Agent`, the latter could access directly the `Geopolitical_Information_Agent`. However, if much of the information was not represented, an alternative geographic agent would need to be constructed (and linked). Third, the network forms a directed acyclic graph, not a tree, because a particular agent may provide information to several others that focus on different aspects of its expertise (like the `Port_Agent`, that is accessed by the `Geopolitical, Air and Sea agents`). Nevertheless, cycles should be avoided, otherwise a query may loop endlessly without finding some agent that can actually answer it. In summary, in spite of the complexity introduced by respecting the autonomy of the agents in the organization, the fact that individual agents can be independently built and maintained makes the system flexible enough to scale to large numbers of information sources and adaptable to the needs of new applications.

In order 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, which provides access to heterogeneous data and knowledge bases [Arens *et al.*, 1993]. We use the Loom Interface Manager (LIM) [Pastor *et al.*, 1992] as a wrapper for relational databases. Using SIMS and LIM, we built

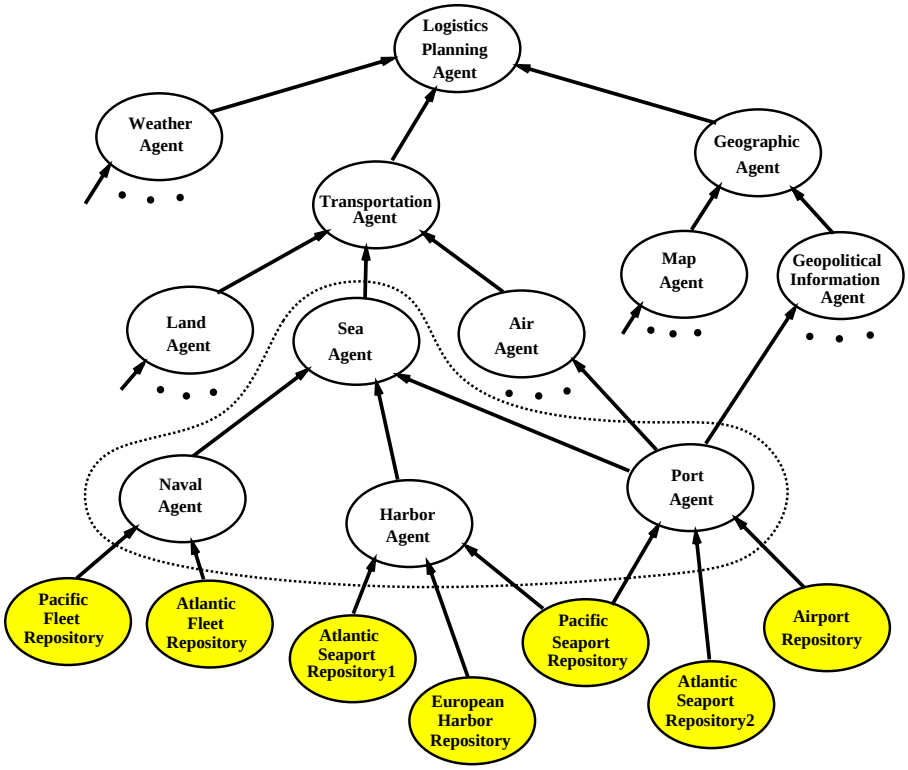

Figure 1.1
Network of Information Gathering Agents

a small network of information gathering agents that interact over the Internet. 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 executes the plan. An agent can also learn about other agents to improve both their efficiency and accuracy. The following sections describe the knowledge representation, communication, query processing, and learning capabilities of the individual agents.

1.3 The Knowledge of an Agent

Each agent contains a model of its domain of expertise and models of the other agents that can provide relevant information. We will refer to these two types of models as the domain model and information source models. These models constitute the general knowledge of an agent and are used to determine how to process an information request.

The domain model is an *ontology* that represents the domain of interest of the agent and establishes the terminology for interacting with the agent. The information-source models describe both the contents of the information sources and their relationship to the domain model. These models do not need to contain a complete description of the other agents, but rather only those portions that are directly relevant. They constitute the resources that are available to an agent to answer information requests when they cannot be handled locally.

Both the domain and information-source models are expressed in the Loom knowledge representation language [MacGregor, 1990]. Loom is an AI knowledge representation system of the KL-ONE family[1] [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.

1.3.1 The Domain Model of an Agent

Each information agent is specialized to a single *application domain* and provides access to the available information sources within that domain. 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 domain model of an agent defines its area of expertise and the terminology for communicating with it. That is, it provides an ontology to describe the application domain. This ontology consists of descriptions of the classes of objects in the domain, relationships between these classes (including subsumption), and other domain-specific information. These classes and relationships do not necessarily correspond directly to the objects described in any particular information source. The model provides a semantic description of the domain, which is used extensively for processing queries.

The largest application domain that we have to date is a logistics planning domain, which involves information about the movement of personnel and materiel from one location to another using aircraft, ships, trucks, etc. Currently, we are building another network of agents for a trauma care domain.

[1] These type of languages are also known as description logics, terminological logics or concept languages.

Figure 1.2 shows a fragment of the domain model of the `Sea_Agent` that belongs to the organization of Figure 1.1. The nodes represent concepts (i.e., classes of objects), the thick arrows represent subsumption (i.e., subclass relationships), and the thin arrows represent concept roles (i.e., relationships between classes). Some concepts that specify the range of roles have been left out of the figure for clarity. Some are simple types, such as strings or numbers (such as `ship-name`), while others are defined concepts (such as `geoloc-code`).

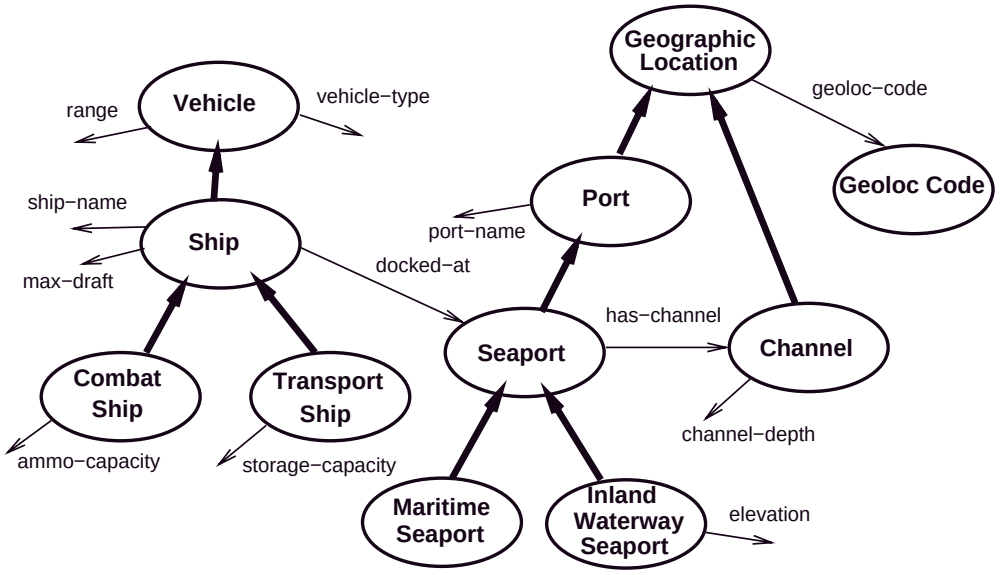

Figure 1.2
Fragment of the Domain Model of the `Sea Agent`

1.3.2 Modeling other Agents

An agent will have models of several other agents that provide useful information for its domain of expertise. Each information-source model has two main parts. First, there is the description of the contents of the information source. This comprises the concepts of interest available from that information source in terms of the ontology of that information source. The terms in the ontology provide the language that the information source understands (and that will be used to communicate with it, as described in Sections 1.4 and 1.5). Second, the relationship between these information source concepts and the concepts in the domain model needs to be stated. These mappings are used for

transforming a domain model query into a set of queries to the appropriate information sources.

Figure 1.3 illustrates how an information source is modeled in Loom and how it is related to the domain model. All of the concepts and roles in the information-source model are mapped to concepts and roles in the domain model. A mapping link between two concepts or roles (dashed lines in the figure) indicates that they represent the same class of information. More precisely, that their extensions are equivalent. Thus, if the user (of the `Sea Agent`) requests *all* seaports, that information can be retrieved from the concept `Harbor` of the `Harbor_Agent`. Note that the domain model may include relationships that involve concepts coming from different agents (like the role `docked-at` of the `ship` concept) but are not explicitly present in any one information source.

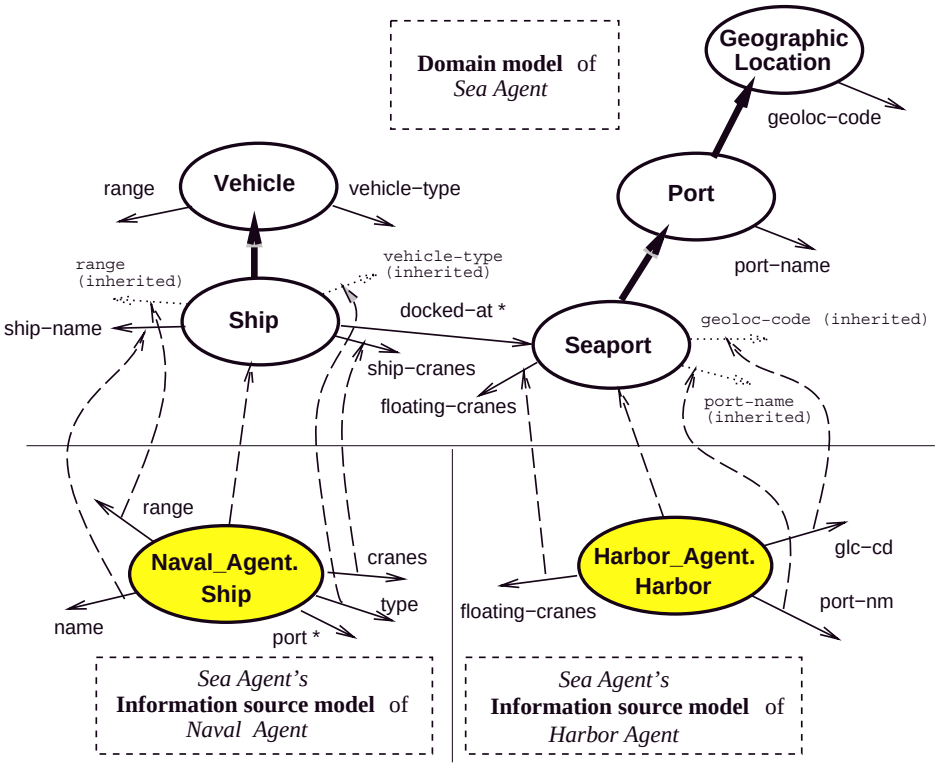

Figure 1.3
Relating an Information-Source Model to a Domain Model (in the `Sea Agent`)

1.4 Communication Language and Protocol

We use a *common* language and protocol to communicate among agents (in order to avoid the n^2 translation problem). Strictly speaking, there are two different aspects in agent communication. The content of the communication and the particular communicative act that is intended. This is reflected, respectively, in the choice of Loom as the language in which to describe the desired information requested by agents, and the Knowledge Query and Manipulation Language (KQML)[Finin *et al.*, 1992, Finin *et al.*, in press] as the protocol to organize the dialogue among agents.[2]

Queries to an information agent are expressed in a subset of the Loom query language. These queries are composed of terms of its 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 (by reasoning with the mappings introduced in Section 1.3.2).

Figure 1.4 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 (analogous to the projection operation in the relational algebra). 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 values of `?port` and `?port_name`. More precisely, this query requests all `seaport-name` and `ship-type` pairs where the depth of the port exceeds the draft of the ship.

In addition to sending queries to other agents, the agents also need the capability to send back information in response to their queries. We use an implementation of KQML to handle the interface protocols for transmitting queries, returning the appropriate information, and building the appropriate internal structures. Messages among SIMS agents, and between SIMS agents and the LIM agents, which provide access to relational databases, are all uniformly expressed in KQML. Recall that in order to make

[2] However, the use of KQML in our agent network is transparent to the user.

```
(retrieve (?port_name ?ship_type)
          (: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 1.4
Example Loom Query

an existing database or other application program available to the network of agents requires building a wrapper around the existing system. This wrapper should include the capability of handling the relevant KQML performatives and understand the expressions of the common content language, i.e., the Loom query language. Currently, the operations supported by SIMS are retrieve, update, insert, delete, and notify.

To summarize, the communication among agents proceeds through the following phases. Once the example of Figure 1.4 has been translated into queries in terms of each information source, each subquery (in its Loom query language form) will be enclosed in a KQML message and transmitted to the appropriate information source. Then, the wrapper of the receiver will unpack it, translate the Loom expression into the language originally handled by that agent (for example, SQL in the case of a relational database), collect the results, and send them back as a KQML reply. Recall also that only one such wrapper would need to be built for any given type of information source, which reduces the complexity of the translation among heterogeneous systems from quadratic to linear in the number of different data description languages.

1.5 Query Processing

A critical capability of an information agent is the ability to flexibly and efficiently retrieve and process data. The query processing requires developing an ordered set of operations for obtaining the requested set of data. This includes selecting the information sources for the data, the operations for processing the data, the sites where the operations will be performed, and the order in which to perform them. Since data can be moved around between different sites, processed at different locations, and the operations can be performed in a variety of orders, the space of possible plans is quite large.

We have developed a flexible planning system to generate and execute query access plans. The planner is based on an implementation of UCPOP [Barrett *et al.*, 1993].

We augmented this planner with the capability for producing parallel execution plans [Knoblock, 1994], added the capability of interleaving planning and execution [Ambros-Ingerson and Steel, 1988, Etzioni *et al.*, 1994], and added support for run-time variables [Ambros-Ingerson and Steel, 1988, Etzioni *et al.*, 1992] for gathering information at run time. This work extends previous work on interleaving planning, execution, and sensing with the ability to perform these operations in parallel and applies these ideas to query processing. In the context of query processing, it allows the system to execute operations in parallel, augment and replan queries that fail while executing other queries, gather additional information to aid the query processing, and accept new queries while other queries are being executed.

This section describes how this planner is used to provide flexible access to the available information sources. First, we describe how we cast a query as an information goal. Second, we describe how an agent *dynamically* selects an appropriate set of information sources to solve an information goal. Third, we present our approach to producing a flexible query access plan. Fourth, we describe how the interleaving of the planning and execution can be used to execute actions in parallel, employ sensing operations to gather additional information for planning, handle new information requests as they come in, and replan when actions fail. Finally, we describe how an agent optimizes queries using semantic knowledge about the contents of other information sources.

1.5.1 An Information Goal

A planning problem consists of a goal, an initial state, and a set of operators that can be applied to transform the initial state into the goal. In this subsection, we will describe the goal and initial state and in the following two subsections we present the operators used in the planning process.

For information gathering, the goal of a problem consists of a description of a set of data as well as the location where that data is to be sent. For example, Figure 1.5 illustrates a goal which specifies that the set of data be sent to the OUTPUT device of a `sims` agent. The data to be retrieved is defined by the query expressed in the Loom knowledge representation language, as described in Section 1.3.

The initial state of a problem defines the information agents that are available as well as which server they are running on. The example shown in Figure 1.6 defines three available agents. Each clause defines the name of the agent and the machine it is running on. For example, the first clause defines the Naval_Agent, which is running on the machine `isd12.isi.edu`. In addition, there is also the domain and information models, that are static (for the duration of the query processing) and are accessed directly from a Loom knowledge base.

```
(available output sims
           (retrieve (?port_name ?ship_type)
                     (:and (seaport ?port)
                           (port_name ?port ?port_name)
                           (has-channel ?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 1.5
Example Planner Goal

```
((source-available Naval_Agent isd12.isi.edu)
 (source-available Harbor_Agent isd14.isi.edu)
 (source-available Port_Agent isd14.isi.edu))
```

Figure 1.6
Example Initial State

1.5.2 Information Source Selection

An information goal sent to an agent is expressed in terms of the domain model of that agent. Part of the planning for an information goal requires selecting an appropriate set of information sources (other information agents or data repositories). To select the information sources, a set of reformulation operators are used to transform the domain-level terms into terms about information that can be retrieved directly from an information source [Arens *et al.*, 1995]. If a query requests information about ports and there is a single information source that provides such information, then the mapping is straightforward. However, in some cases there may be several information sources that provide access to the same information and in other cases no single information source can provide the required information and it will need to be drawn from several different sources.

Consider the fragment of the knowledge base shown in Figure 1.7, which covers the knowledge relevant to the example query in Figure 1.5. The concepts `seaport, channel` and `ship` have links to information source concepts, shown by the shaded circles, which correspond to information that can be retrieved from some information agent. Thus, the `Harbor_Agent` contains information about both seaports and channels, and the `Port_Agent` contains information about seaports.

The system has a number of truth-preserving reformulation operations that can be

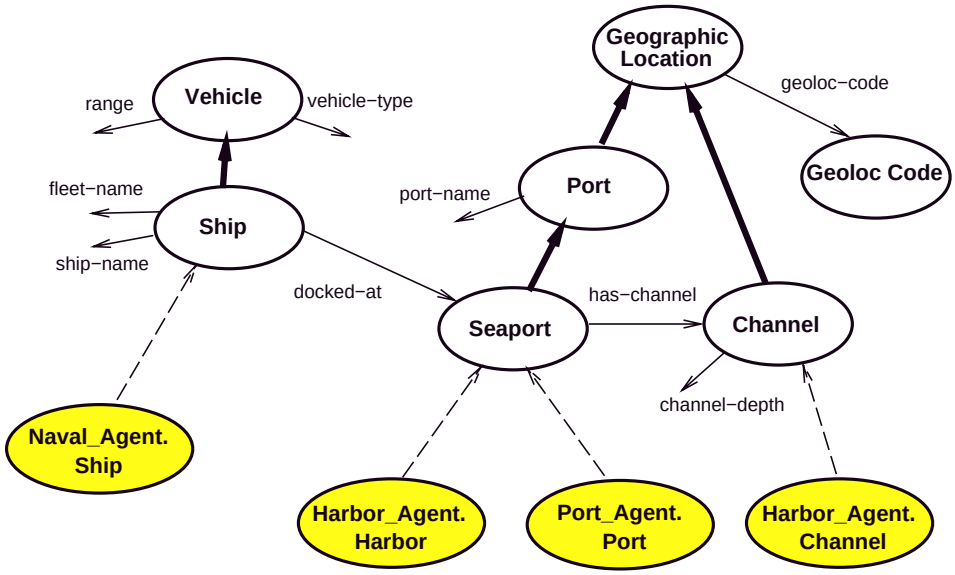

Figure 1.7
Fragment of the Domain and Information-Source Models

used for reformulating a domain-level query. The basic operations include:

- **Information Source Selection** 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 several 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 executing the query.
- **Concept Generalization** uses knowledge about the relationship between a concept and a superconcept 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 seaports, but the information sources that correspond to the seaport concept do not contain the requested information, then it may be possible to generalize seaport 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 `seaport` and `port`.

- **Concept Specialization** 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 `seaports` with an elevation greater than 300 feet, it can be reformulated into a request for all `inland-waterway seaports` using knowledge in the model that only inland-waterway seaports have an elevation above 300 feet.
- **Definition Substitution** replaces a relation defined between concepts in the domain model with equivalent terms that are available in the information-source models. For example, `has_channel` 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 occurs both in `seaport` and `channel`.

For example, consider the query shown in Figure 1.5. 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 `has_channel`. Thus, the system must reformulate `has_channel`, using the substitute definition operator. This expresses the fact that `has_channel` can be materialized by performing a join over the keys for the seaport and channel concepts. The resulting reformulation is shown in Figure 1.8.

```
(retrieve (?port_name ?ship_type)
      (: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 1.8
Result of Applying the Definition Substitution Operator to Eliminate `has_channel`

Another step in the reformulation process is to select information sources for the information requested in the query. This can be done using the select-information-source operator, which selects among the available information sources. Figure 1.9 shows a reformulation of a query for information about seaports, which could be provided by

either the `Harbor_Agent` or `Port_Agent`. In this case `Harbor_Agent` is selected because the information on channels is only available in the `Harbor_Agent`.

Domain-Level Query:

```
(retrieve (?port_name ?depth)
      (:and (seaport ?port)
            (port_name ?port ?port_name)
            (geoloc_code ?port ?geocode)
            (channel ?channel)
            (geoloc_code ?channel ?geocode)
            (channel_depth ?channel ?depth)))
```

Source-Level Query:

```
(retrieve (?port_name ?depth)
      (:and (harbor_agent.harbor ?port)
            (harbor_agent.port_nm ?port ?port_name)
            (harbor_agent.glc_cd ?port ?glc_cd)
            (harbor_agent.channel ?channel)
            (harbor_agent.glc_cd ?channel ?glc_cd)
            (harbor_agent.ch_depth_ft ?channel ?depth)
```

Figure 1.9
Result of Selecting Information Sources for Channels and Seaports

1.5.3 Generating a Query Access Plan

In addition to selecting the appropriate information sources to solve an information goal, the planner must also determine the appropriate data manipulation and ordering of those operators to generate the requested data. Therefore, besides the operators for source selection, there are five operators for manipulating the data:

- **Move** – Moves a set of data from one information agent to another.
- **Join** – Combines two sets of data using the given join operation.
- **Retrieve** – Specifies the data to be retrieved from a particular information source.
- **Select** – Selects a subset of the data using the given constraints.
- **Compute** – 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 specifying the original query.

Consider the operator shown in Figure 1.10 that defines a join performed locally. This operator is used to achieve the goal of making some information available in the local

knowledge base of a SIMS agent. It does this by partitioning the request into two subsets of the requested data, getting that information into the local knowledge base of an agent and then joining that data together to produce the requested set of data. The operator states that (1) if a query can be broken down into two subqueries that can be joined together over some join operator, and (2) the first set of data can be made available locally, and (3) the second set of data can also be made available locally, then the requested information can be made available. The predicate `join-partition` is defined by a program that produces the possible join partitions of the requested data.

```
(define (operator join)
    :parameters (?join-ops ?data ?data-a ?data-b)
    :precondition (:and (join-partition ?data ?join-ops
                                        ?data-a ?data-b)
                        (available local sims ?data-a)
                        (available local sims ?data-b))
    :effect (:and (available local sims ?data)))
```

Figure 1.10
The Join Operator

To search the space of query access plans efficiently, the system uses a simple estimation function to calculate the expected cost of the various operations. Using this evaluation function in a branch-and-bound search, the system will produce a plan that has the lowest overall parallel execution cost. In the example, the planner leaves the join between the `harbor` and `channel` to be performed by the `Harbor_Agent` since this will be cheaper than moving the information into the local knowledge base of an agent and joining it together.

The plan generated for the example query in Figure 1.5 is shown in Figure 1.11. This plan includes the source selection operations as well as the data manipulation operations since these operations are interleaved in the planning process. In this example, the system partitions the given query such that the ship information is retrieved in a single query to the `Naval_Agent` and the seaport and channel information is retrieved in a single query to the `Harbor_Agent`. All of the information is brought into the local knowledge base of the agent originating the query, 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 to the agent or application that requested the information.

1.5.4 Interleaving Planning and Execution

The previous two sections described the operators for both selecting an appropriate set of information sources and for manipulating the data retrieved from those information

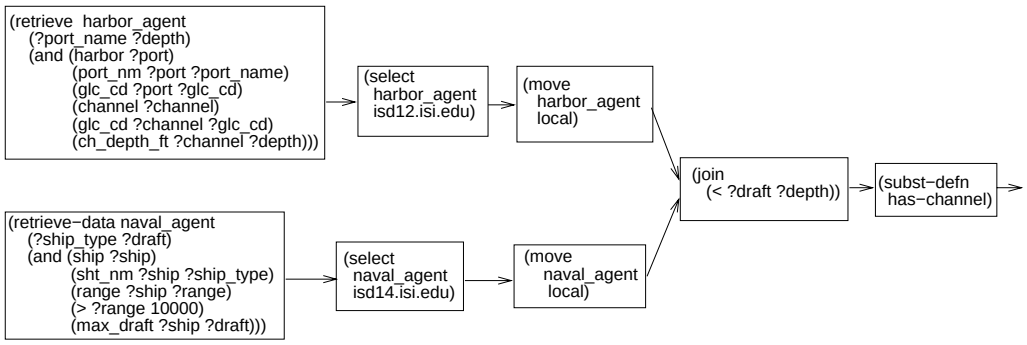

Figure 1.11
Parallel Query Access Plan

sources. This section describes how this planning is tightly integrated with the execution to provide the ability to flexibly and efficiently process queries [Knoblock, 1995].

The interleaving of the planning and execution provides a number of important capabilities for the agents:

- An agent can run continuously, accepting queries and planning for them while it is executing other queries.
- If a failure occurs, an agent can replan the failed portion of the plan while it continues to execute queries that are already in progress. After replanning, the system will redirect the failed subquery to a different agent or information repository.
- An agent can issue sensing actions to gather additional information for query processing. This allows an agent to gather additional information to formulate more efficient queries to other information sources. Information gathering can also help to select among a number of potentially relevant information sources [Knoblock and Levy, 1994].

Rather than having a separate execution module, the execution is tightly integrated in the planner. This is done by treating the execution of each individual action as a necessary step in completing a plan. Thus, the goal of the planner becomes producing a complete and executed plan rather than just producing a complete plan. This allows the planner to interleave the planning process with the execution, which makes it possible to handle new goals, replan failed goals, and execute actions to gather additional information.

The execution of an action is viewed as a commitment to the plan in which the action occurs. This means that the planner will only consider the plan from which the action is executed and all valid refinements of that plan. Since execution of an action commits to

the corresponding plan, we would like the planner to be selective in choosing to execute an action. This is achieved by delaying the execution of any action as long as possible. The idea is that the planner should find the best complete plan within some time limit before any action is executed. Then once execution has begun, it would resolve any failed subplans or new goals before executing the next action.

Execution of an action may take considerable time, so the planner does not execute an action and wait for the results. Instead the system spawns a new process to execute the action and then that process notifies the planner once it has completed. At any one time there may be a number of actions that are all executing simultaneously.

On each cycle of the planner, the system will check if any executing actions have completed. Once an action is completed then the executing action is removed from the agenda. If it completes successfully then the action is left in the plan and marked as completed. Other actions that depend on this action may now be executable if all of their other dependencies have also been executed. If an action fails, the system produces a refinement of the executing plan that eliminates the failed portion of the plan. This replanning can be performed while other actions are still executing.

If a new goal is sent to the planner, the system simply inserts an open condition for that new top-level goal. The additional open condition will be handled before initiating execution of any new operators, so the planner will augment the existing plan to solve this new goal in the context of the existing executing plan.

The planner also supports run-time variables [Ambros-Ingerson and Steel, 1988, Etzioni *et al.*, 1992], which allow the planner to perform sensing operations in the course of planning. These variables appear in the effects of operators and essentially serve as place holders for the value returned by the operator when it is actually executed. Run-time variables are useful because the result can be incorporated and used in other parts of the plan.

1.5.5 Semantic Query Optimization

Before executing any actions in the query plan, the system first performs semantic query optimization to minimize the overall execution cost [Hsu and Knoblock, 1993]. The semantic query optimizer uses semantic knowledge about the information sources to reformulate the query plan into a cheaper, but semantically equivalent query plan. The semantic knowledge is learned by the system as a set of rules (Section 1.6.2).

Consider the example shown in Figure 1.12. 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 `ship` and check the values of the `range` to retrieve the answer.

An example semantic rule is shown in Figure 1.13. This rule states that all ships with

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

Figure 1.12
Example Subquery

range greater than 10,000 miles have a draft greater than 12 feet. Based on these rules, the semantic query optimizer infers a set of additional constraints and merges them with the input query. The final set of constraints left in the reformulated query is selected based on two criteria: reducing the total evaluation cost, and retaining the semantic equivalence. A detailed description of the algorithm is in [Hsu and Knoblock, 1993]. 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. The system can reformulate a query by adding, modifying or removing constraints. The resulting query is shown in Figure 1.14.

```
(:if (:and (naval_agent.ship ?ship)
           (naval_agent.range ?ship ?range)
           (naval_agent.fuel_cap ?ship ?fuel_cap)
           (> ?range 10000))
(:then (> ?draft 12)))
```

Figure 1.13
An Example Semantic Rule

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

Figure 1.14
Reformulated Query

We can reformulate each subquery in the query plan with the subquery optimization algorithm and improve their efficiency. However, the most expensive aspect of queries to multiple information sources is often processing and transmitting intermediate data. In the example query plan in Figure 1.11, 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 optimization algorithm to be able to propagate the constraints along the data flow paths in the query plan. We developed a query plan optimization algorithm which achieves this by using the semantic rules to derive possible constraints and propagating these constraints around the query plan.

Consider an example of reformulating the query plan shown in Figure 1.11. The system is given the fact `41` $\leq$ `?depth` $\leq$ `60`. The subquery optimization algorithm can infer from the constraint `(< ?draft ?depth)` a new constraint `(< ?draft 60)` and then propagate this constraint to constrain the maximum draft. The algorithm will insert the new constraint on `?draft` in that subquery.

The resulting query plan is more efficient and returns the same answer as the original one. The amount of intermediate data is reduced because of the new constraint on the attribute `?draft`. The entire algorithm for query plan optimization is polynomial. Our experiments show that the overhead of this algorithm is very small compared to the overall query processing cost. On a set of 26 example queries, the query optimization yielded significant performance improvements with an overall reduction in execution time of 59.84% [Hsu and Knoblock, 1995].

1.6 Learning

An intelligent agent for information gathering should be able to improve both its accuracy and performance over time. To achieve these goals, the information agents currently support three 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. Finally, an information agent can analyze the contents of its information sources in order to refine its domain model to better reflect the currently available information. All these forms of learning can improve the efficiency of the system, and the last one can also improve its accuracy.

1.6.1 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 [Arens and Knoblock, 1994]. An elegant feature of using Loom to model the domain is that cached information can easily be 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 concept 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.

1.6.2 Learning rules for Semantic Query Optimization

The goal of an information agent 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 [Hsu and Knoblock, 1994, Hsu and Knoblock, 1995].

The learning is triggered when an agent detects an 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 optimization. These learned rules form an abstract model of the information provided by other agents or data repositories.

1.6.3 Reconciling Agent Models

So far we have assumed that the domain and information-source models of an agent are perfectly aligned. That is, the mappings among concepts in these models perfectly correspond to the actual information. In a network of autonomous agents this assumption will not hold in general. First, the designer of the models might not have had a complete understanding of the semantics of the information provided by each agent. Second, even if at design time the models were accurate, the autonomy of the agents will cause some concepts to drift from their original meaning. The dynamic nature of the information implies that we need to provide mechanisms to detect inconsistency and/or incompleteness in the agent's knowledge. In this section we describe an approach to automatically reconcile agent models, which will improve both the accuracy of the represented knowledge and the efficiency of the information gathering. It consists of three phases. First, an agent checks for misalignments between the domain and source models. Second, it modifies the domain model to represent the new classes of information detected. Third, if possible, it learns from the actual data a description that declaratively describes these new concepts.

We will illustrate the main ideas of this approach through an example from the domain of the `Sea_Agent` (Figure 1.7). Assume that initially both `Harbor_Agent.Harbor` and `Port_Agent.Port` contain the same information about major commercial seaports, which for the purposes of the application is in agreement with the intended semantics of the concept `Seaport`. However, the `Port_Agent` evolves to contain information about recreational, small fishing harbors, etc. The `Harbor_Agent` and `Port_Agent` are no longer equivalent providers of `Seaport` information.

First, analyzing their actual extensions, our agent will notice that `Harbor_Agent.Harbor` is now a subset of `Port_Agent.Port`. Second, the domain model is automatically modified as shown in Figure 1.15. A new concept `Commercial-Seaport` is added to the domain model as a subconcept of the original `seaport`. `Harbor_Agent.Harbour` will map now into `Commercial-Seaport`. Third, we apply machine learning algorithms (currently ID3 [Quinlan, 1986]) in order to obtain a concise description of this new concept. For example, it might construct a description that distinguishes commercial seaports from generic seaports by the number of cranes available. With this refined model, a query like "retrieve all the seaports that have more than 15 cranes and channels more than 70 feet deep", which describes information only satisfied by commercial seaports, could be appropriately directed to the `Harbor_Agent`, saving both in communication (less data transmitted) and processing costs (less data considered in any subsequent join), because

the concept `Harbor` has a smaller number of instances. Moreover, a query about a small-craft harbor will not be incorrectly directed to the `Harbor_Agent`, but to the `Port_Agent` which is the only one that can provide such information.

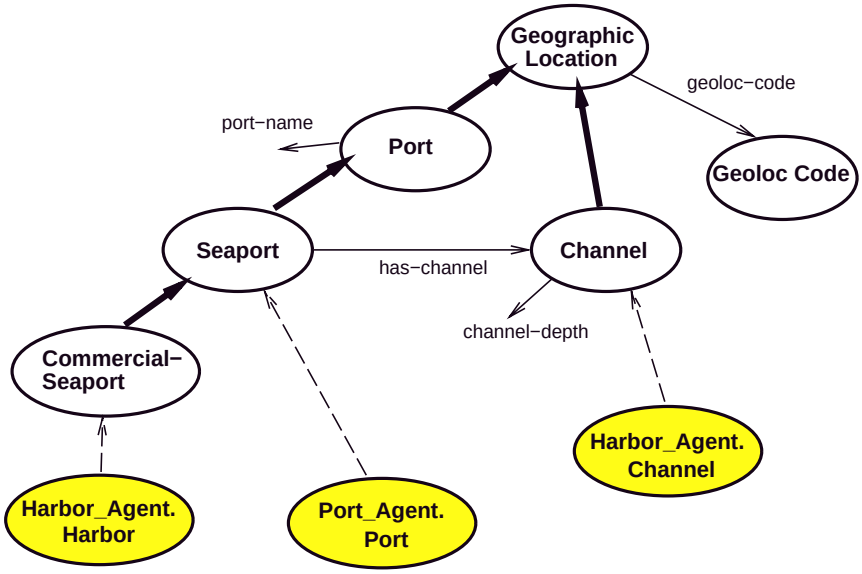

Figure 1.15
Reconciled Model

A more detailed explanation of these techniques can be found in [Ambite and Knoblock, 1994], including other cases in which the extensions are overlapping or disjoint, or deal with more than two information source concepts.

The benefits of the reconciliation are twofold. First, increased accuracy of the knowledge represented in the system. These new concepts provide a more precise picture of the current information available to an agent system. This mechanism adapts automatically to the evolution of the information sources, whose contents may semantically drift from the original domain model mappings. Also, human designers may revise these concepts to both refine the domain model and detect errors. Second, increased efficiency of query processing. A SIMS agent will use those concepts that yield a cheaper query plan. These new concepts provide better options for retrieving the desired data.

1.7 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, (see [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 [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. [Etzioni *et al.*, 1992, Etzioni *et al.*, 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. In contrast to this work, the focus of our work is on flexible and efficient retrieval of information from heterogeneous information sources. Since these 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.

Levy el al. [Levy *et al.*, 1994] are also working on building agents for information

gathering. The focus of their work has been on developing a framework for selecting a minimal set of sites to answer a query. In contrast, SIMS integrates the site selection process in the query planning process in order to provide greater flexibility. This integration allows SIMS generate a query plan and a corresponding set of sources that will produce the requested information most efficiently.

1.8 Discussion

This paper described the SIMS architecture for intelligent information 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, (4) efficiency in terms of minimizing the overall execution time for a given query, and (5) adaptability in terms of being able to track semantic discrepancies among models of different agents. We will discuss each of these features in turn.

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 optimization 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.

Fifth, the ability to compare and reconcile models of different agents make the agents **adaptable** to changes in the information sources. Using the detailed semantic models of the information sources, an agent can track changes in the information sources and update its own models appropriately. This information is critical for both the accuracy and efficiency of the query processing.

To date, we have built information agents that plan and learn in the logistics planning domain. These agents contain a detailed model of this domain and extract information from a set of relational databases. The agents generate query access plans to the appropriate information sources, execute the queries in parallel, and learn about the information sources. Future work will focus on extending the planning and learning capabilities of these agents.

Acknowledgments

We gratefully acknowledge the contributions of the other members of the SIMS project, Yigal Arens, Chin Chee, Chunnan Hsu, Wei-Min Shen, and Sheila Tejada for their work

on the various components of 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.

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. J.L. Ambite is supported by a Fulbright/Spanish Ministry of Education and Science Scholarship. 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, the Fulbright program, the Government of Spain, or any person or agency connected with them.

Bibliography

[Ambite and Knoblock, 1994] Jose Luis Ambite and Craig A. Knoblock. Reconciling agent models. In *Proceedings of the Workshop on Intelligent Information Agents*, Gaithersburg, MD, 1994.

[Ambros-Ingerson and Steel, 1988] Jose Ambros-Ingerson and Sam Steel. Integrating planning, execution, and monitoring. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 83–88, Saint Paul, Minnesota, 1988.

[Arens and Knoblock, 1994] Yigal Arens and Craig A. Knoblock. Intelligent caching: Selecting, representing, and reusing data in an information server. In *Proceedings of the Third International Conference on Information and Knowledge Management*, Gaithersburg, MD, 1994.

[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.

[Arens *et al.*, 1995] Yigal Arens, Craig A. Knoblock, and Wei-Min Shen. Query reformulation for dynamic information integration. In Preparation, 1995.

[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.

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

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

[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.

[Etzioni *et al.*, 1992] Oren Etzioni, Steve Hanks, Daniel Weld, Denise Draper, Neal Lesh, and Mike Williamson. An approach to planning with incomplete information. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, pages 115–125, Cambridge, MA, 1992.

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

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

[Finin *et al.*, in press] Tim Finin, Yannis Labrou, and James Mayfield. KQML as an agent communication language. In J. Bradshaw, editor, *Software Agents*. AAAI/MIT Press, Menlo Park, CA, in press.

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

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

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

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

[Knoblock and Levy, 1994] Craig A. Knoblock and Alon Levy. Efficient query processing for information gathering agents. In *Proceedings of the Workshop on Intelligent Information Agents*, Gaithersburg, MD, 1994.

[Knoblock *et al.*, 1994] Craig A. Knoblock, Yigal Arens, and Chun-Nan Hsu. Cooperating agents for information retrieval. In *Proceedings of the Second International Conference on Cooperative Information Systems*, Toronto, Canada, 1994.

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

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

[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.

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

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

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

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

[Quinlan, 1986] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[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.

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