An Architecture for Information Retrieval Agents*

Craig A. Knoblock and Yigal Arens
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292, USA
{KNOBLOCK,ARENS}@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 needed. In this paper we present an architecture for building such agents that addresses the issues of representation, communication, problem solving, and learning. We also describe how this architecture supports agents that are modular, extensible, flexible and efficient.

Introduction
With the current explosion of data, retrieving and integrating information from various sources is a critical problem. We are addressing this problem by building information retrieval agents. This paper describes how these agents communicate with one another, model the information that is available from other agents, efficiently retrieve information from other agents, and improve their performance through experience.

The architecture presented in this paper is based on the SIMS information server, which was previously described in [Arens et al., 1993]. This paper reviews the relevant details of SIMS, focusing on the aspects and features that support multiple collaborative agents. Each information agent is a separate SIMS server. Thus, each 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 identifies an appropriate set of information sources, generates a plan to retrieve and process the data, uses knowledge about the data to re-formulate the plan, and then executes it. This paper describes our approach to the issues of representation, communication, problem solving, and learning, and describes how this approach supports multiple, collaborating information retrieval agents.

Representing the Knowledge of an Agent
Each information agent is specialized to a particular area of expertise. This 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.

An agent contains a model of its domain of expertise, which provides the terminology for interacting with agents, as well as models of all information sources that are available to that agent. Both the domain and information source models are expressed in the Loom knowledge representation language [MacGregor, 1988, MacGregor, 1990]. The domain model provides class descriptions, subclass and superclass relationships, roles for 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.

Modeling the Domain
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 material from one location to another using aircraft, ships, trucks, etc.

The application domain models are defined using a hierarchical terminological knowledge base (Loom)
with nodes representing each class of objects, and relations on each node that define the relationships between the objects. For example, Figure 1 shows a fragment of the domain model in the transportation planning domain. In this figure, the nodes represent classes, the thick arrows represent subclass relationships, and the thin arrows represent relations between classes.

The classes defined in the domain model do not necessarily correspond directly to the objects described in any particular information source. The domain model is intended to be a description of the application domain from the point of view of providing access to users or other information agents that need to obtain information about the particular application domain. However, the domain model is used as the language with which 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 objects contained in the information source, as well as the relationship between these objects and the objects in the domain model. The mappings between the domain model and the information source model are needed for transforming a domain-level query into a set of queries to actual information sources.

Figure 2 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 a SIMS information agent are expressed in Loom. These queries are composed of terms in a general domain model, so there is no need to know or even be aware of the terms or language used in the underlying information sources. Instead, the information
agent translates the domain-level query into a set of queries to the underlying information sources. These information sources may be other information agents, databases, knowledge bases, or other application programs.

Figure 3 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 subclause 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 subclause is an example of a relation expression and states that the relation port.name holds between the value of ?port and the variable ?port.name. This query describes a class of seaports and a class of ships, and requests all seaport and ship pairs where the depth of the port exceeds the draft of the ship.


Figure 3: Example Loom Query

Given a query such as the one described above, an information agent identifies the appropriate information sources and issues queries to these information sources to obtain the requisite data for answering the query. The queries to the underlying information sources are also expressed as Loom queries. This simplifies the overall system since it only needs to handle one underlying language, although each information source may have different processing capabilities. Since all of the information agents use Loom as the query language, queries can be sent to another information agent without any translation. To handle existing information sources systems such as databases, a wrapper for each type of system must be constructed, which takes a Loom query as input and maps the query into the language of the underlying information source.

Communication between information agents is done using the Knowledge Query Manipulation Language (KQML)[Finin et al., 1992]. This language handles the interface protocols for transmitting queries, returning the appropriate information, and building the appropriate structures. This language can also be used to combine a database system with an appropriate wrapper to form a limited type of information agent. This agent is limited in the sense that only it has access to its own internal database. This approach has been successfully applied to building a set of restricted agents that can access relational databases using the Loom Interface Manager (LIM) [Pastor et al., 1992]. The agents described in this paper use these LIM agents for accessing the data in a set of relational databases.

Planning to Access Information Sources

The core part 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, we need a system that dynamically generates plans for information retrieval, and monitors and updates the plans as they are executed.

To process a domain-level query requires first selecting an appropriate set of information sources that can be used to answer a query and then generating an appropriate query plan for processing the data. This section describes these two steps in turn.

Selecting Information Sources

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

Consider the fragment of the knowledge base shown in Figure 4, which covers the knowledge relevant to the example query in Figure 3. The concepts Seaport, Channel and Ship have subconcepts, shown by the shaded circles, that correspond to concepts whose instances can be retrieved directly from some information agent or database. Thus, the GEO information source contains information about both seaports and channels, and the PORT information source contains information about only seaports. Thus, if the user asks for seaports, then the query must be translated into one of the information source concepts — GEO.Seaports or PORT.Ports.

Reformulation Operations. 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 re-
Figure 4: Fragment of Domain Model

This information is used to replace a requested concept with a set of concepts that cover it.

**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 that occurs in both seaport and channel, which is `geoloc-code` in this case.

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

For example, consider the query shown in Figure 3. There are two concept expressions - one about ships and the other about seaports. The first step is to translate the seaport expression into an information-source-level expression. Unfortunately, none of the information sources contain information that defines `channel_of`. Thus, the system must first reformulate `channel_of` using the decompose operator. This is done using 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 5.

The next step is to reformulate the seaport concept into a corresponding information source concept. The channel and ship concepts would then be similarly reformulated. The final query, which is the result of reformulating all of these concepts, is shown in Figure 6.
of data, where the data is specified in the same terms

Each of these operators manipulates one or more sets

out the processing of a query:

There are three basic operators that are used to plan

out the processing of a query:

- **Move-Data** – Moves a set of data from one informa-

  tion source to another information source.

- **Join** – Combines two sets of data into a combined

  set of data using the given join relations.

- **Retrieve-Data** – Specifies the data that is to be re-

  trieved from a particular information source.

Each of these operators manipulates one or more sets

that are used for communicating with an agent and

between agents. This simplifies input/output, since

there is no conversion between languages required at a

later time.

The job of the planner is to find a partially ordered

sequence of operations that can be used to retrieve the

set of data specified in the given query. The plan-

ner is implemented in a version of UCPOP [Penberthy

and Weld, 1992, Barrett et al., 1993] that has been ex-

tended to generate parallel execution plans [Knoblock,

1994]. The system searches through the space of possi-

ble plans using a best-first search until a complete plan

is found. The plan generated for the example query in

Figure 6 is shown in Figure 7.

Since the information retrieval agent must act au-

tonomously in the real world, it builds a plan that is
tailored to the current world model. In order to handle

failures and unexpected results, we plan to tightly in-

tegrate the planner with an execution and monitoring

component. This will enable the information agent to

react intelligently to problems in retrieving data. For

example, other agents, networks, or databases may be

down, the system may get back more data than ex-
pected, or it may not get back any of the data it ex-
pected and it must retrieve it from somewhere else. So

while the system is executing an information retrieval
plan, it will constantly monitor the status and consider

alternative plans. If part of a plan fails, then the sys-

tem will immediately replan that part, or if another

plan appears to be more promising than the one cur-

rently executing, it will switch to the new plan.

**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 infor-

mation. 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 Loom knowledge base
and retrieved directly from Loom. An elegant feature of
using Loom to model the domain is that caching the
data fits nicely into this framework. The data is cur-
rently brought into Loom for local processing, so cach-
ing is simply a matter of retaining the data and

recording what data has been retrieved.

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

Once the system has defined a new class and stored the data under this class, the cached information becomes a new information source. 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. See [Arens and Knoblock, 1993] for a detailed description of the caching performed by the agents.

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 supports 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. Separate models for the domain and the contents of the information sources.
3. A flexible planning system that selects an appropriate set of information sources.
4. Generation of parallel query access plans.
5. A learning system that constantly attempts to improve 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 Loom query as input and returns the data requested by the query. The separate domain models provide a modular representation of the contents of each of the information agents. 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 separation of the domain and information source models and the dynamic planning of information requests 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 is not necessary to integrate the information with the other information source models. Similarly, changes to the schemas of information sources require only changing the model of the schemas. 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 planning 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, the generation of parallel access plans, the caching of retrieved data, and the ability to learn about information sources provide efficient access to large amounts of distributed information. The planner generates plans that minimize the overall execution time by maximizing the parallelism in the plan to take advantage of the fact that autonomous information sources can be accessed in parallel. 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 [Georgeff, 1990]. 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 specifically designed and built to perform particular tasks. Etzioni et al. [1992, 1993] 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 this work has been on efficient query processing with Horn rules.

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 [Reddy et al., 1989, 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.

Conclusion

This paper described the SIMS architecture for intelligent information retrieval agents. As described above, 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 [Arens et al., 1993]. 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.

References


