Information Brokering in an Agent Architecture

David Martin
Artificial Intelligence Center
SRI International
martin@ai.sri.com

Hiroki Oohama
Laboratory of Information Technology
NTT Data
hama@lit.rd.nttdata.co.jp

Douglas Moran
Artificial Intelligence Center
SRI International
moran@ai.sri.com

Adam Cheyer
Artificial Intelligence Center
SRI International
cheyer@ai.sri.com

Abstract

To date, document identification based on keyword matching strategies has been the basis of most efforts to provide assistance in accessing information resources on the Internet. However, in view of the limitations on human browsing time and the evolution of more capable software agents, we can expect rapid expansion in the use of fully queryable information sources (such as databases and knowledge bases) and *semiqueryable* sources (such as form-based query pages on the World Wide Web, and collections of Web pages that are structured by textual markups).

The ability to obtain information from a wide variety of queryable and semi-queryable sources is a prerequisite for the success of many types of software agents. Providing access to these sources — whether for end users or for software agents acting on behalf of users — poses a number of interesting challenges, many of which can themselves be addressed using agent-based approaches.

This paper describes a working prototype Information Broker system, developed within the Open Agent Architecture framework, that provides transparent access to a variety of information sources, each encapsulated as an independent agent. In this system, a broker *meta-agent* provides flexible mediation services, accepting queries expressed in broker or source schemas, gathering and integrating all available responses from the relevant sources, and allowing for the addition or deletion, at runtime, of participating information sources. Other broker features support the use of conversions, normalizations, and other basic domain knowledge in queries, and *persistent queries* (for which the Broker notifies requestors of changes in information sources that could affect their results). Source agents implement caching and retrieval strategies that alleviate problems related to the long access times and unreliability of Internet sources.

1 Introduction

The rapid growth of the World Wide Web and other forms of Internet and intranet access, and the need for automated assistance in making use of these resources, are now widely appreciated. The most immediate need, and the one receiving the greatest attention to date, is in the area of document retrieval; that is, the identification of textual documents that are relevant to some topic of interest. The topic of interest is generally indicated by an expression containing keywords, and the goal of the retrieval is to locate documents from which a human reader can extract useful information. A number of systems — such as NTT Data's *InterInfo*[1] in the commercial realm and *Amalthaea* [11] in the realm of agent research — are providing increasingly sophisticated approaches to document retrieval.

1.1 Structured and Semistructured Information Sources

An equally important set of challenges, but one that has not yet become as widely recognized, exists in the area of structured and semistructured information sources. By *structured* information sources, we mean sources that can be queried by using well-defined, general query languages, such as relational databases, object-oriented databases, and knowledge bases.

By *semistructured* information sources, we include a variety of sources that contain sufficient structure to be treated as databases, even though they do not provide the full generality and power of a query language. In the context of the World Wide Web, this structure is usually provided in one (or both) of two ways: by an informal form-based query interface or by the presence of HTML (HyperText Markup Language) markups and other textual markers used according to site-specific conventions.

We refer to a semistructured source that provides an informal form-based query interface as a *semiqueryable* source. It should be apparent that, even when such an interface is provided, its "query" capability usually falls far short of the power and generality of a structured information source. When a source provides structure by textual markers, we call that a *semistructured textual* source.

An example of a semistructured source — in which both types of structure are present — begins with a Web page that allows one to ask for hotels by filling in one or more of the following items: a location, a hotel chain, or a class of accommodation (economy, standard, luxury). This page is *semi*queryable, because even though it allows for the construction of certain simple queries, there are many others that cannot be constructed.[2] The result of this query is a list of hypertext links to hotel pages, each of which contains the same basic data in more or less the same format. Each of these pages is a *semistructured textual* source of information, and each can be transformed into a set of data records, using parsing techniques, given that the format is known.

[1]All product names mentioned in this document are the trademarks of their respective holders.

[2]For instance, one cannot ask, in a single query, for luxury hotels in Amsterdam and economy hotels in Paris.

1.2 The Need for Information Brokering

An *information brokering* system is one that provides coordinated access to a heterogeneous collection of structured and semistructured information sources. There are three key reasons why structured and semistructured information sources — and thus, information brokering systems — will be of rapidly increasing importance in the Internet and intranet worlds. First, human browsing time is both limited and, at least in the workplace, extremely expensive. Whereas *document retrieval* returns a body of text from which a human browser can extract some required data, *data retrieval* from structured and semistructured sources returns the required data itself. Thus, in performing tasks where the data requirements are well defined, there is significant economic pressure to make use of structured and semistructured sources.

Second, enterprises have large investments in legacy databases, and naturally want to leverage these in the context of the World Wide Web. This is especially true in light of the decentralization of corporate resources, and other trends in business process engineering. One straightforward way to leverage existing databases is simply to make them accessible to employees via Web interfaces. But the potential for leverage goes much further, when one considers the variety of ways in which legacy databases can be used in combination with other enterprise information resources, and with information from sources on the Internet. One role of information brokering systems is to facilitate the creation and maintenance of applications that rely on such combinations of information sources.

Third, queryable information sources are an essential requirement for the evolution of software agents that provide services within the context of the Internet or an intranet. By *services*, we refer not just to document retrieval, but to the full gamut of services that can be built around networked information. To provide a travel planning service or a stock tracking service, an agent must be able to retrieve data about a specific domain, in a form it can make use of — which requires access to structured and semistructured information. (Although natural language processing technologies have made impressive advances in recent years, there is still a long way to go before a program could reliably obtain this required data from *unstructured* text.) Moreover, many agents will need to access a wide and changing variety of information sources (consider, for example, an agent that finds the best available price for some product). As new sources become available, they will need to quickly acquire access to those sources. Information brokering systems can provide this access in a way that can be reused by numerous application agents.

1.3 Information Brokering and Agent Technology

Software agents will be very active *consumers* of the services provided by information brokering systems. We also argue that agent technology yields a promising approach to constructing the *providers* of these services. That is, the individual information sources as well as the brokering component(s) can very naturally be instantiated as agents, and their efforts coordinated by using the mechanisms of an agent architecture.

One other general observation may be made here regarding the relationship between information brokering and agent technology: challenges encountered in coordinating access to information agents may be viewed as special cases of those encountered in coordinating the activities of all types of agents. Thus, many of the techniques developed to coordinate the satisfaction of a query by multiple agents may well be applicable to the more general problem of facilitating the activities of agents in completing various other types of tasks.

To summarize, the role of queryable information sources will expand rapidly in the context of Internet and intranet resources. Their use in these contexts poses several interesting challenges. Addressing these challenges is the focus of the Information Broker project, developed within the framework of the Open Agent Architecture (OAA). Following a brief discussion of these challenges in the next section, the remainder of this paper describes the system that has been developed in the initial work on this project. Section 3 provides an overview of the system, with background information about the OAA. Section 4 describes the system's architecture, and subsequent sections describe its individual components.

2 Challenge Areas

Our focus in this work has been on providing capabilities in two areas: *mediation* allows for the transparent interoperation of heterogeneous information sources; flexible *retrieval strategies* address issues such as long access times and unreliability of Internet resources.

2.1 Mediation

Mediation is a process that permits a requestor to get information from a wide variety of sources, without having to be aware of the identities, locations, schemas, access mechanisms, or contents of those sources. A component that performs mediation presents a single schema to its requestors, accepts queries expressed in that schema, and handles all the details of getting the appropriate data from the relevant information sources — each of which is likely to operate with a different schema.

The capabilities provided by a mediator may be broken down into three subareas: delegation, translation, and optimization. *Delegation* is the process of selecting the appropriate sources from which to satisfy each subquery of a given query. *Translation* of each subquery into the schemas of the selected sources must take place before the subqueries are sent to the sources — and results returned from the sources must be translated back into the schema of the original query. *Optimization* results in a query execution plan that obtains results from the selected sources as efficiently as possible, and exploits parallelism wherever possible.

It should be noted that, in their essence, the problems of mediation are not unique to networked environments. Some of these problems have already been studied for some time, under the heading of *heterogeneous databases*. However, the emergence of the Internet has triggered a renewed interest in these problems.[3] This is not only due to the massive volume

[3]Other projects currently exploring these issues are described in [1], [5], [6], [7], [8], [12], and [13].

of data that is now becoming accessible over the Internet, but also because of new aspects of these problems that occur in the context of the Internet.

For example, the rapid creation (or discovery) of new information sources on the Internet, and frequent restructuring of existing sources, make it very valuable for a mediator to be able to accommodate new sources without going offline for reconfiguration. The absence of any centralized control over information sources implies that a mediator must be able to accommodate considerable variation in their schemas. The overhead involved in making HTTP connections to a number of different sites requires that a mediator be able to plan a retrieval so that it requires a minimal number of different sources.

2.2 Retrieval Strategies

In addition to the problems of mediation, Internet and intranet environments pose a number of other challenges for integrated information systems. For instance, the *unreliability* and *uneven quality* of Web sources may call for the exploitation of redundant or overlapping sites. Wide variation in the access mechanisms of *semistructured* sources suggests that strategies are needed to insulate the mediation component from this variation.

The use of semistructured textual sources on the Web also introduces a new set of time factors, because first, a very large number of URL (Universal Resource Locator) accesses may be required to process a single query, and second, parsing the text from a single site can itself be time consuming.

Our approach to addressing these issues includes the use of caching for Web-based sources, and several related retrieval strategies, which may be specified by information requestors. The use of caching raises additional issues, such as when cache updates should take place, and at what level of granularity.

3 The Information Broker System

To explore relevant issues, we created an information brokering architecture that integrates elements from software agent technology, database technology, and Internet retrieval technology. To demonstrate this approach, we constructed a prototype system, in which a Broker agent coordinates access to a variety of information resources in the domain of travel.

The prototype system illustrates two different ways in which the Broker agent can be used:

- A direct query interface allows the user to enter specific queries about hotels, relays the queries to the Broker, and formats the results for the user. In the prototype system, this interface is provided by a Web page.
- A service-providing agent insulates the user from query details, and can employ a user model in obtaining and evaluating the information needed for a specific task. In the

prototype system, this is demonstrated by the Travel Planner Agent (TPA), which accepts some basic trip constraints from the user, obtains hotel and flight data from the Broker, and uses that data to formulate desirable itineraries.

In both cases, the Broker obtains and integrates the requested information from several heterogeneous sources, each of which participates in the system as an independent agent. These include several Web-based sources providing hotel information, a relational database of flight information, a Web-based source that provides the latest available rates of currency exchange, and a corporate database listing rates for hotel chains which grant a discount.

Broker features allow for expression of queries in broker or in source schemas, the use of conversions, normalizations, and other basic domain knowledge in queries, flexibility in the means of retrieval, and *persistent query* capabilities. These features are explained in greater detail below.

The following subsection gives a brief overview of the OAA, the framework in which the agents of the brokering system operate.

3.1 The Open Agent Architecture

The Open Agent Architecture provides a framework for integrating a society of software agents, each possessing a high degree of independence and autonomy, within a distributed environment. A collection of agents satsifies requests from users, or other agents, by acting cooperatively, under the direction of one or more *facilitators* (which are themselves agents of a special type).

The system's architecture uses a hierarchical configuration in which each application agent connects as a client of a facilitator. Facilitators provide content-based message routing, global data management, and process coordination for their set of connected agents. Facilitators can, in turn, be connected as clients of other facilitators.

Agents share a common communication language and a number of basic structural characteristics and capabilities. An agent library provides this common functionality. For example, every agent can install local or remote triggers on data, events or messages; manipulate global data stored by facilitators; and request solutions for a set of goals, to be satisfied under a variety of different control strategies. In addition, the agent library provides functionality for parsing and translating expressions in the Interagent Communication Language, and for managing network communication using TCP/IP.

The OAA's Interagent Communication Language (ICL) is the interface language shared by all agents, no matter what machine they are running on or what computer language they are programmed in. The ICL has been designed as an extension of the Prolog programming language, in order to take advantage of unification and backtracking during interactions among agents.

The OAA is described in greater detail in [3], [9], and [10].

4 System Architecture

As shown in Figure 1, the central functionality of the system is provided by the *Information Broker* agent, working in close cooperation with the OAA *Facilitator*. The Broker accepts requests (queries) from either a direct query interface or a service-providing (helper) agent, as shown at the left of the figure. While the demonstration system includes only one of each type, there is no limit in principle to the number of requestors the Broker can serve, except for the constraints imposed by processing time and communications bandwidth.

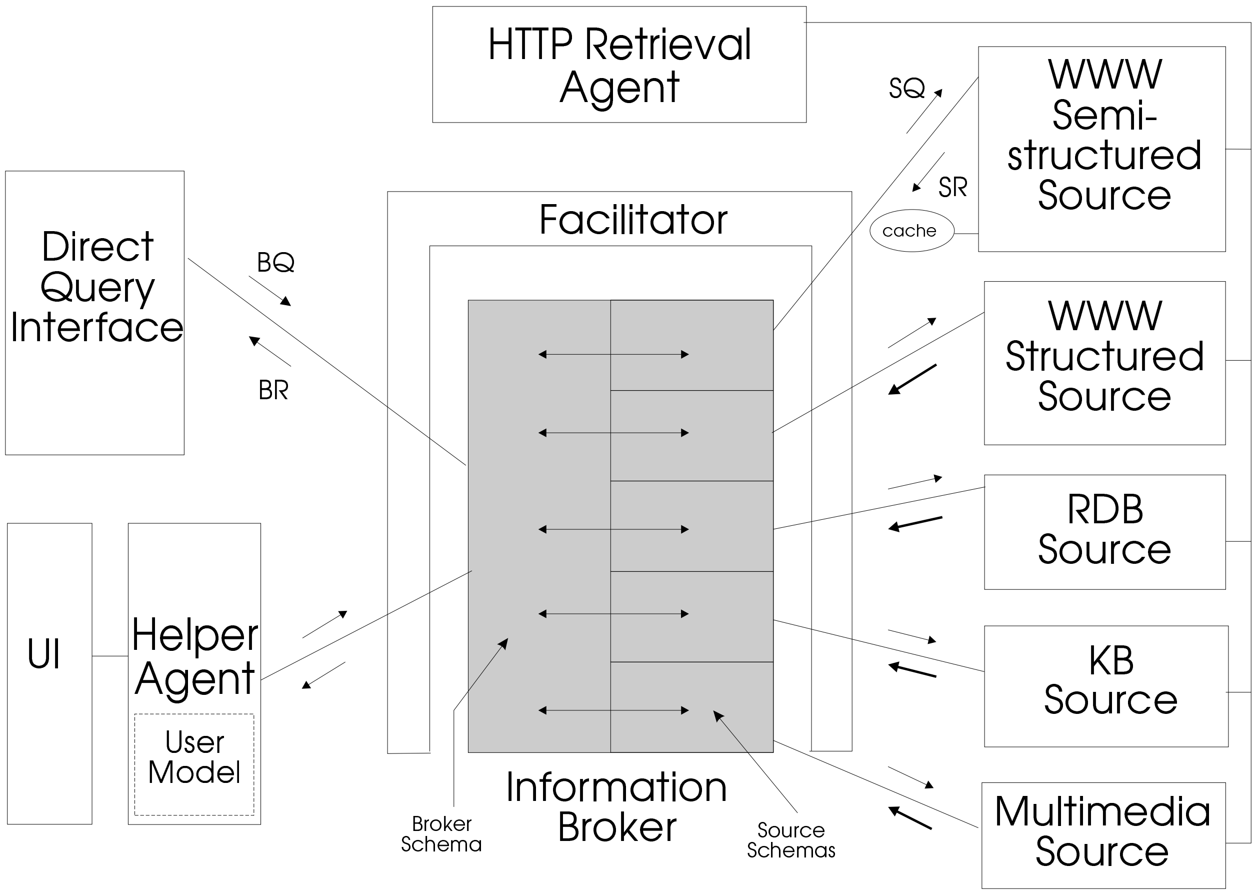

Figure 1: Information Broker system architecture

The Broker delegates, translates, and relays the appropriate subqueries to the available *source agents* (shown at the right of Figure 1), and then accepts the results and reintegrates them for return to the requestor. Each source agent is an encapsulation, as an OAA agent, of some information source. Web-based source agents make use of an HTTP retrieval agent, which is shown at the top of the figure.

In the figure, *BQ* (Broker Query) and *BR* (Broker Response) refer to items expressed in the broker schema, whereas *SQ* (Source Query) and *SR* (Source Response) refer to items expressed in a source schema — as explained in the next section. *RDB* abbreviates Relational

DataBase, and *KB* abbreviates Knowledge Base.[4]

5 The Broker Agent

The Broker agent provides transparent access to a collection of heterogeneous information sources in a given domain. *Transparent* means that an information requestor (a query interface agent or some other agent) need not be concerned with any details regarding the Broker's information sources. The Broker publishes a fixed schema, the *broker schema*, that is used in constructing queries, and this schema is all that is needed by a requestor to make use of the Broker's services.

Schema, here, is essentially the same as a relational database schema. From the developer's point of view, equivalently, it may be thought of as a collection of Prolog predicates.[5] With respect to the broker schema, some of these predicates are implemented locally, at the Broker, whereas others are implemented remotely, at the sources.

Queries submitted to the Broker are expressions in the Interagent Communication Language (ICL), the language of the OAA. Each query is syntactically the same as a Prolog goal, usually a compound goal. For an OAA developer using the Broker's services, the ICL has two advantages: first, it is easily understood, and familiar to anyone who knows Prolog or the logical style of expressing database queries; second, it allows the developer to take advantage of Prolog-style unification and backtracking in expressing and processing queries.

When the Broker receives a query, it uses *schema mapping rules* to determine which parts of the query should go to which sources, and then to translate each subquery into the schemas of those sources to which it will be sent. (This process is described in greater detail below.) After the responses to the subqueries are received, the Broker translates them into the broker schema and integrates them into a single response, which is then returned to the requestor.

Thus, the Broker insulates an information requestor from the heterogeneity of schemas that is likely to be present in any collection of information sources. The Broker has this core functionality in common with heterogeneous database systems. However, the Broker goes further than many existing systems, in that it allows for the addition or removal of information sources at runtime. This can include sources of which the Broker has no prior knowledge.

To permit this dynamism in the participation of available information sources, the Broker makes use of capabilities provided by the OAA. When an information source agent comes online, it *registers* its presence with the Broker by calling one of the Broker's agent interface procedures, *broker_register_source*. As one of the arguments to this procedure, the source agent passes to the Broker a set of schema mapping rules, which contain the knowledge that

[4]The prototype system does not currently include knowledge base or multimedia sources.

[5]For this reason, we use the term *predicate*, where some readers with a database background might prefer *relation*. In most places in this paper, these two terms can be used interchangeably.

is needed by the Broker to translate between the broker schema and the source schema.[6]

This approach to schema mapping means that the Broker need have no prior knowledge of the schema of any of the participating information sources. What *is* required is that the developer of the source agent be aware of the Broker's schema, which, as mentioned earlier, has been fixed and "published". Thus, to allow for the dynamic participation of sources, an important part of the Broker's knowledge originates with the sources, each of which provides the mapping rules for its schema.

At the same time, because the schema mapping rules are *used* at the Broker, rather than at the source agent, it is possible to bring a source online with minimal changes to the original data or to the original capabilities of the source. This may be done by creating an agent *wrapper* around the original source component — an approach that is supported by the underlying OAA libraries and development tools.

Source agents that need to go offline may do so in an orderly fashion by calling another of the Broker's agent interface procedures, *broker_unregister_source*, to inform the Broker that the source is no longer available. However, it is also important to allow for sources that may die or become unavailable unexpectedly. Here again, the Broker uses mechanisms provided by the OAA Facilitator. To do this, the Broker installs a *trigger* on the Facilitator, for each source agent, which is set to fire whenever that source agent dies or becomes unavailable for any reason. The effect of the trigger is to send a notification message to the Broker, so that it can unregister the source and retract its schema mapping rules.

5.1 Domain Specialization

A Broker agent may be specialized for a given domain, by incorporating domain knowledge. To illustrate, our prototype Broker agent is specialized with basic knowledge relevant to travel, such as distances between major cities — data that is not likely to be included in any of the participating sources, but that is very likely to be useful in conjunction with the sources' data. Questions about this domain knowledge can then be included in queries submitted to the Broker. For example, using the direct-query hotel interface, it is possible to ask about hotels that are in a given city C, or in nearby cities (cities within a certain distance from C). By filling in knowledge gaps in this way, it is possible for the Broker to provide a retrieval capability that is greater than the sum of the individual information sources.

Procedures for converting and normalizing units represent another type of domain knowledge that is useful in a Broker. For example, in the travel domain, our prototype Broker provides procedures for converting and normalizing units of currency and the formatting of addresses, which vary widely from country to country. These procedures can be used in schema mapping rules, to ensure that results are returned in units and formats that are appropriate for the current user. This alleviates the need for each information source to provide these conversions

[6]These and other communications between the Broker and the source agents are handled through the services of the OAA Facilitator. We illustrate this, in Figure 1, by showing the Facilitator partially surrounding the Broker.

and normalizations, and thus makes it easier to prepare information sources to work with the Broker.

5.2 Schema Mapping Rules

Delegation and translation of queries by the Broker are determined by schema mapping rules, which are provided at runtime by each information source that connects to the Broker.

There are several types of schema mapping rules, the most important of which is the *broker predicate* rule. A broker predicate rule, which provides a partial definition of some predicate in the broker schema, has the syntax of a Prolog rule, and essentially the same semantics. Each mapping rule takes the form

$$B(\overline{X}) \leftarrow S_1(\overline{Z}_1), \ldots, S_n(\overline{Z}_n)$$

where

1. $B(\overline{X})$ is a predicate in the broker schema.
2. Each of $S_1, \ldots, S_n$ may be taken from any of the following sets:
   - predicates in the source schema (i.e., the source providing the rule)
   - broker domain predicates
   - ICL (Interagent Communication Language) built-in predicates, including
     - standard comparison operators such as $</2$, $\leq/2$, $>/2$, and $\geq/2$
     - a small set of utility predicates such as *member*/2 and *append*/2
3. As in an ordinary Prolog clause, each element of the argument lists $\overline{X}, \overline{Z}_1, \ldots, \overline{Z}_n$, may be a variable or an instantiated value.

Broker domain predicates provide domain-specific knowledge, and most are purely extensional. For example, in the travel domain of the prototype system, $city\_distance(City1, City2, Miles)$ provides a table of distances between given cities.

An information source may be associated with multiple rules defining the same broker predicate, and multiple sources can define the same broker predicate.

Although the body of the broker predicate rule is characterized as a conjunction of predicates, it should be noted that in practice, these rules, like ordinary Prolog rules, are not strictly limited to conjunction. Disjunction, negation (that is, Prolog-style negation as failure), and a few other control operators are also allowed, but for any rule body containing these, standard equivalences of logic may be used to find an equivalent set of conjunctive rules.

5.3 Query Processing

The Broker's strategy in handling a query is to provide all possible answers, given its current schema mapping rules. It does this in such a way as to maintain Prolog semantics in performing query evaluation, and incorporates several straightforward optimizations that are important, considering the distributed nature of the information sources.

Of these optimizations, the most important is called *chunking*. The goal of chunking is simply to identify the largest subqueries that can be sent to an information source (after translation, as explained below) without changing the meaning of the original query. A *chunk* may contain broker predicates that are known to be satisfiable by the source, and ICL built-in predicates, but not broker domain predicates.

Chunking has three advantages. First, it requires that fewer communications occur between the Broker and an information source, reducing the overhead required to establish and complete communications. Second, it allows the information source to make better use of whatever optimization capabilities it may provide. Finally, it can significantly reduce the amount of data returned from a source, by allowing joins and other operations to be performed at the source, rather than at the Broker.

When the Broker receives a query, it takes the following steps to produce a *query execution plan*:

- Determine, for each predicate in the query that is not an ICL built-in, what set of sources provides solutions for that predicate.
  - If the predicate is a broker predicate, this is done by testing for unification of the predicate against the heads of the broker predicate rules. If it unifies with the heads of one or more of the rules associated with a given source, that source is placed in the set of sources for the predicate.
  - If the predicate is a broker domain predicate, then the Broker itself is regarded as the sole source of solutions.
- Determine which are the largest subqueries that can be treated as chunks, and which sources can handle each chunk. In this process, as an optimization, ICL built-in predicates (including arithmetic comparisons) are included with chunks to be solved by sources, wherever possible, rather than being solved by the Broker itself.
- For each chunk, rewrite it as a disjunction of translated subqueries, where each disjunct is the translation of the subquery for one of the sources that can handle that chunk. The translation of a chunk for a given source is obtained by substituting, for each broker predicate in the chunk, the disjunction of the bodies of the broker predicate rules, for that source, whose heads unify with the predicate.

In the resulting query execution plan, each translated subquery is labeled with the name of the source by which it is to be solved. The plan is then interpreted according to Prolog semantics, except that each translated subquery is solved remotely, by the appropriate information source.

5.4 Persistent Queries

An extremely useful service provided by the Broker agent is the *persistent query*. A persistent query is one that the Broker initially answers in the normal way, but then remembers and monitors against changes in the available information sources. When an information source is added, removed, or updated, the Broker checks the persistent queries to see if their results may have been affected by the change.

As mentioned in section 5, the Broker detects additions of sources by calls made to *broker_register_source*, and removals by calls to *broker_unregister_source*, or by the firing of a trigger installed on the Facilitator. Updates of sources are brought to the Broker's attention, with an indication of the source predicates involved, by means of triggers installed on the sources. In each of these cases, the Broker performs a straightforward analysis of each persistent query, using the schema mapping rules for the source in question, to determine if it may have been affected.

If so, the Broker sends a change notification to the user or agent that submitted the persistent query. For example, in the case of our direct-query interface, the result of a persistent query change notification is an email message to the user who entered the query. The email message informs the user that the relevant data has changed, reconstructs the user's query input, and allows him to resubmit his query with a single button push, so as to obtain the latest available results for that query. (The interactive nature of the email message is achieved by using a multipart MIME format for the message, and reading it using the mail handling component of a Web browser.)

The TPA deals with persistent query change notifications in a more comprehensive fashion, as explained in section 7.

5.5 Queries Expressed in a Source Schema

The Broker also provides the capability of answering queries that are expressed in the schema of an information source, rather than in the broker schema. This capability can have great value in systems where one or more information sources are provided by legacy databases.

The need for this capability arises when a legacy database provides a well-developed user interface to which users are accustomed, and it is desirable to retain that interface, while at the same time integrating the legacy database with the Broker's services. In this case, the goal is for the user to formulate a query using the legacy interface (which generates queries in the source schema of the legacy database), but to have that query answered by all the Broker's information sources, with the results returned to the legacy interface. This is made possible, with a minimum of new engineering effort, by arranging for the Broker to handle queries in that source schema.

In this situation, the legacy database becomes one of the Broker's sources, and participates along with other sources in answering queries submitted to the Broker in the broker schema. But in addition, this legacy source may also act as an information requestor, submitting queries that are formulated in terms of its source schema. (A syntactic translation may be needed to place the query in ICL syntax, but in most cases, this is relatively straightforward to implement.) To make this possible, the legacy source must first submit, to the Broker, a collection of *source predicate rules*, which contain the knowledge needed by the Broker to translate queries from the source schema into the broker schema. These rules are similar to broker predicate rules, except that their heads are predicates from the source schema, and their bodies are compositions of predicates from the broker schema.

When a query expressed in a source schema is submitted, the Broker uses the source predicate rules to translate it into a query in the broker schema, and then solves that query in its usual way, with respect to all sources other than the legacy source. These results are then translated into the schema of the legacy source. The legacy source is not included in the Broker's normal mediation algorithms. Rather, the Broker submits the original query back to the legacy source just as it was. Finally, the Broker integrates all the responses and returns them to the source.

6 Source Agents

An information source agent that works with a Broker agent may be of several different types. The primary requirements are first, that it be able to handle the Prolog-style query syntax that the Broker generates, and second, that it be able to characterize its contents in terms of schema mapping rules. The first requirement is not a very restrictive one, as it is considered straightforward to translate from a Prolog-style syntax to SQL or other relational query languages. The second requirement means, roughly, that a source can be presented in terms of a relational model. (Although it should be possible to accommodate some sources that are inherently object-oriented or knowledge-based, this support has been left for future work.)

In meeting the second requirement, one must consider the origin and access mechanisms of the source's data, which can vary widely between information sources. For instance, some may be traditional relational databases, whereas others may contain data extracted from the Web. In the case of Web data, several types of data sources may be identified for our purposes. First, there are fully structured sources — relational databases that are accessible via a completely general query language, such as SQL. These are relatively uncommon at present. Second, there are semistructured sources. As mentioned in the Introduction, these may be either *semiqueryable* (accessible via a form-based query page), *semistructured textual*, or some combination of the two. semistructured textual sources are Web sites that include large collections of pages, each having enough structure, in HTML markups, headings, and/or other textual clues, to allow for extraction of a set of data elements.

In addition to traditional relational databases, implemented as Prolog databases, the prototype system makes use of several semistructured textual Web sources. To extract the data from the relevant pages, parsing techniques, based on context-free grammars, are used. A library of reusable code has been developed in support of these techniques.

6.1 Caching and Retrieval Strategies

As mentioned earlier, Web-based sources introduce challenges for data retrieval. Foremost among these are long access times (especially relevant to semistructured sources) and low reliability. In the Information Broker system, these are addressed through a combination of caching and flexible retrieval strategies.

6.2 Caching

Caching is accomplished both by batch update procedures and by update procedures that result from individual queries. A batch update procedure is one that runs offline (that is, independently of the source's activities in response to queries), and regenerates a complete cache. For a semistructured data source, a batch update could involve accessing hundreds or thousands of pages and parsing each one for relevant data, so this could easily take hours. Each of the Web-based information sources has been initialized in this way, and is set up so that query processing can continue normally by one agent, while a batch update is performed by a separate agent.

Cache updates can also be triggered by normal query processing. Whenever a query is satisfied (or partially satisfied) with data retrieved directly from the Web, this data is used to update the cache.

6.3 Retrieval Strategies

Flexible retrieval strategies are used to determine whether a query is satisfied entirely from a cache, directly from the Web at query time, or from a combination of the two. A request that comes in to the Broker can include, in addition to the query itself, a specification of a retrieval strategy, which will be propagated to each source agent that is employed in answering the query.

Data records in a cache are time-stamped, and some strategies are implemented in relation to these time stamps. In addition, a source agent can specify a default longevity for each predicate that it caches. For example, since room rates are relatively stable for hotels in our Web sources, the longevity for the relevant predicates is specified as two weeks. For the agent that provides current monetary exchange rates, however, longevities are set at 24 hours.

Currently, four retrieval strategies are supported. *All-from-web* calls for retrieval of all data from the Web, at query time, regardless of how long it might take. *All-from-cache* calls for retrieval exclusively from the cache. *From-web-if-expired* means that for each data element that is accessed in answering the query, its time-stamp is checked, and if it is older than the default longevity for that predicate, then the element is updated from the Web. *From-web-if-older-than(N)* is similar, except that the longevity is specified (as N seconds) for that particular query.

Consistent implementation of caching and retrieval strategies by the various source agents is supported by a library of reuseable code.

7 The Travel Planner Agent

Even though a Broker provides a general, transparent means of accessing multiple data sources with a single query, there is still much that can be done to assist the user in making good use of the information provided by the Broker. A variety of service-providing components can be envisioned, which access the Broker on behalf of the user, thus insulating her from the details of query construction altogether. The instantiation of the Broker as a software agent helps to make this possible, because the Broker's services are thus made readily available to a wide variety of agents that come online.

The Travel Planner Agent provides an example of a high-level service-providing agent that insulates the user from formulating Broker queries. TPA's function is to formulate a list of possible itineraries for a trip, given a few basic trip constraints that are supplied by a user. These trip constraints include such things as dates of departure and return, destination city, and overall budget. Each itinerary prepared by TPA includes complete details about a possible choice of flights and hotel stays for the trip, and itineraries are scored and ranked according to user preferences.

To evaluate alternative itineraries, TPA maintains a user model indicating the user's preferences about a variety of travel parameters. For example, with respect to air travel selections, these parameters include such things as airline, class of travel, alternative airports, and time of day for departure. The presence of the user model also allows TPA to minimize the amount of new information required from the user when formulating an information request.

To minimize the user's effort in maintaining the user model, TPA has the capability of questioning the user about her experiences after a trip has been completed. Once a trip has been scheduled, TPA remembers the return date. Shortly after that date, using services provided by other OAA agents, TPA will send a questionnaire to the user by email, asking that she provide ratings for some of the parameters of the trip, such as a specific hotel or hotel chain. If a trip parameter has previously been rated by the user, she will be given an opportunity to update the rating; otherwise, she will be prompted to select a new rating.[7] The questionnaire is presented as an HTML form, and a "Submit" button causes the user's

[7]For most parameters, ratings are integers between -5 and +5.

responses to be returned to TPA, for incorporation into the user model.[8]

TPA also provides a persistent query service, based upon that provided by the Broker. When it receives a persistent query change notification from the Broker, the TPA determines what set of itineraries was based on that query, automatically resubmits the queries relevant to those itineraries, obtains the new results, and recomputes the requested itineraries based on these results. It compares these itineraries to the results of the original request, which it has saved, to determine whether the resulting itineraries have been affected in any way. If so, it sends the new itineraries to the end user by email.

Note that TPA's persistent query service is more comprehensive than that of the Broker, in that compares its newly generated itineraries against the previous ones, before generating a user notification. By contrast, when the Broker provides a persistent query change notification, it is based on advice from one of the Broker's sources that its data has changed in some way, and indicates that the query results *may have been affected* by the change. That is, the Broker does not determine with certainty whether the query results have been affected, but it gives the requesting user or agent the opportunity to make that determination. The reason for this difference is that the Broker is intended to handle large numbers of queries from (potentially) a large number of agents. Thus, it is not reasonable to expect the Broker to save the results of all queries handled.

8 Future Directions

Although it has not yet been exploited, our architecture allows for cooperation between multiple broker agents. This can be done by having one broker register as an information source with respect to another, and present mapping rules that allow its broker schema to be regarded as a source schema. These relationships between brokers could be organized around the structure of the relevant domains. For example, whereas our Broker directly uses sources in the subdomains of air travel and accommodations, among others, it would be natural to have these two subdomains handled by independent brokers.

The storage and use of broker predicate rules at the Broker, rather than at the individual sources, raises an opportunity for further simplifying the introduction of new information sources. Currently, each source has to be given the ability to process the ICL query syntax. Although this is easy when working with an implementation language for which the agent library is available, it could be a significant obstacle in other contexts. As a substitute, if a source agent exists without this ability, it would be possible for the Broker to perform the interpretation of the query syntax on behalf of that agent, in such a way that the agent need only respond to atomic requests (that is, where each request consists only of a single predicate). On the other hand, for those source agents which *do* handle the ICL syntax, but do not possess the ability to perform any optimizations, the Broker could provide more sophisticated optimizations of the compound requests that go to those agents.

[8]As of this writing, the implementation of the questionnaire capability is incomplete.

More can be done, in the Broker, to ensure effective utilization of information sources. Currently, the use of a source in answering a subquery is determined by unification of the subquery predicates against the heads of broker predicate rules. Although this provides a useful degree of discrimination of sources, greater discrimination could be achieved by attaching additional characterizations of source contents. For example, it should be possible to specify that a source selection depends on an argument of a subquery predicate being a member of some fixed set of values, or within a certain range, if numeric. In addition, rules could be annotated with an approximate indication of what portion of the available data the source can provide, if known. This would allow the Broker, in some cases, to avoid sending requests to two or more sources that are likely to return the same data.

Finally, we would like the Broker to provide estimates of retrieval times to interested requestors. That is, it should be possible for a requestor to find out how long the wait might be for the results to a a given query, with a given retrieval strategy. This would assist the requestor in selecting a retrieval strategy, and, assuming the requestor is a software agent, would allow it to manage its interactions with users more effectively.

9 Related Work

The growth of the Web has contributed to a renewal of interest in the problem of providing a uniform interface to multiple information sources, which has received considerable attention recently in both the AI and Database literature. Within this body of work, the Information Broker project is characterized by its emphasis on the following: dynamic availability of sources, supported by modular sets of schema mapping rules; ease of bringing new and legacy information sources online; transparent access to semistructured sources, supported by caching and retrieval strategies; and thorough integration with an existing agent framework.

In the database community, several systems are being built around the notion of a *mediator*, including CARNOT [4], TSIMMIS [2], and HERMES [14]. Broadly speaking, for each of a chosen set of queries, these systems provide a procedure to answer the query using the available sources. Given a new query, these systems attempt to answer it by relating it to the set of known queries. In the Information Broker, sources are described independently of the queries for which they will be used, which is less restrictive as to which queries are answerable, and also makes it easier to add or remove sources. TSIMMIS has also given attention to the use of semistructured sources, with an approach based on *yacc*-style recognition grammars.

The Information Manifold [8] shares our emphasis on dynamic addition and removal of sources. It too employs modular sets of source description rules, but relies on a different form of rule, where source schema predicates are *characterized* in terms of broker predicates (whereas, in the Information Broker, broker predicates are *defined* in terms of source predicates). There are interesting tradeoffs between the two approaches. In particular, although Information Manifold characterizations provide the broker with more information regarding the contents of each source, and thus allow for greater optimization in the selection of the sources that are relevant to a query, their use is computationally expensive, and the form of the rules must be more tightly restricted.

In the AI community, systems such as SIMS [1] and InfoMaster [6] are also based on explicit representations of the contents of information sources. In SIMS, these are based on mappings of source schemas to classes and attributes specified in the description logic language LOOM, and planning technology is used to generate query plans appropriate for the sources. SIMS' transformation rules, which guide the planning process, do not guarantee that a query-answering plan will be found if one exists. In addition, defining mappings using arbitrary logic rules, as in the Information Broker, provides greater flexibility than in SIMS. In the RETSINA (Reusable Task Structure-based Intelligent Network Agents) architecture [15], *information agents* play a role similar to that of our Broker agents, and use KQML in much the same way that our agents use ICL. In InfoSleuth [7], which employs KQML and KIF for communications, this role is implemented jointly by an Execution Agent and a Broker Agent.

10 Summary

Structured and semistructured information sources will be increasingly important as we learn to use the Internet and intranets in more sophisticated ways. Transparently coordinated access to heterogeneous, rapidly changing collections of these sources is a prerequisite for the evolution of many types of software agents.

We have described an approach and a prototype system that provide this access, within the context of an agent framework, the Open Agent Architecture. Information consumers and providers are instantiated as agents, and interact through the services of a Broker agent, which include mediation, dynamic addition and deletion of sources, and persistent queries. Provider agents present a variety of information sources as queryable databases, hiding details of access and storage, and provide caching and flexible retrieval strategies.

Our approach to information brokering fits naturally into the agent paradigm, and benefits from the autonomy it allows for individual providers and consumers, and the flexibility of interactions between them. In addition, the agents are able to take advantage of specific capabilities provided by the agent framework, including communication and coordination mechanisms, the Interagent Communication Language, triggers, and support provided for creating wrappers around legacy applications implemented in a variety of languages.

References

[1] 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.

[2] Sudarshan Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey Ullman, , and Jennifer Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *Proceedings of the IPSJ Conference*, Tokyo, Japan, October 1994.

[3] P. R. Cohen, A. Cheyer, M. Wang, and S. C. Baeg. An open agent architecture. In O. Etzioni, editor, *Proceedings of the AAAI Spring Symposium Series on Software Agents*, pages 1–8, Menlo Park, California, March 1994. American Association for Artificial Intelligence.

[4] C. Collet, M. N. Huhns, and W Shen. Resource integration using a large knowledge base in CARNOT. *IEEE Computer*, 55(62), 1991.

[5] Hector Garcia-Molina, Dallan Quass, Yannis Papakonstantinou, Anand Rajaraman, Yehoshua Sagiv, Jeffrey D. Ullman, and Jennifer Widom. The TSIMMIS approach to mediation: Data models and languages. In *Proceedings of the Second International Workshop on Next Generation Information Technologies and Systems (NGITS '95)*, Naharia, Israel, June 1995.

[6] Infomaster home page. Available via World Wide Web URL `http://infomaster.stanford.edu/`.

[7] Infosleuth project index. Available via World Wide Web URL `http://www.mcc.com/projects/infosleuth/`.

[8] Thomas Kirk, Alon Y. Levy, Yehoshua Sagiv, and Divesh Srivastava. The information manifold. In *Proceedings of the AAAI Spring Symposium on Information Gathering in Distributed Heterogeneous Environments*, Stanford, California, March 1995.

[9] David L. Martin, Adam Cheyer, and Gowang-Lo Lee. Agent development tools for the open agent architecture. In *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, pages 387–404, Blackpool, Lancashire, UK, April 1996. The Practical Application Company Ltd.

[10] Douglas B. Moran, Adam J. Cheyer, Luc E. Julia, and David L. Martin. The open agent architecture and its multimodal user interface. In *Proceedings of the 1997 International Conference on Intelligent User Interfaces (IUI97)*, Orlando, Florida, 6-9 January 1997.

[11] Alexandros Moukas. *Amalthaea*: Information discovery and filtering using a multiagent evolving ecosystem. In *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, pages 421–436, Blackpool, Lancashire, UK, April 1996. The Practical Application Company Ltd.

[12] Xiaolei Qian. Semantic interoperation via intelligent mediation. Final Report CDRL A009, Computer Science Laboratory, SRI International, Menlo Park, California, 1996. Unpublished project report.

[13] Michael Stonebraker, Paul M. Aoki, Avi Pfeffer, Adam Sah, Jeff Sidell, Carl Staelin, and Andrew Yu. Mariposa: A wide-area distributed database system. Sequoia 2000 Technical Report 95/63, University of California, Berkeley, CA, June 1995. Appeared in *VLDB Journal* 5, 1 (January 1996), pp. 48 – 63.

[14] V.S. Subrahmanian, S. Adali, A. Brink, R. Emery, J. Lu, A. Rajput, T. Rogers, R. Ross, and C. Ward. HERMES: A heterogeneous reasoning and mediator system. Technical report, University of Maryland, 1995.

[15] K. Sycara, K. Decker, A. Pannu, M. Williamson, and D. Zeng. Distributed intelligent agents. In *IEEE Expert*, December 1996.