Scalable Semantic Brokering over Dynamic Heterogeneous Data Sources in InfoSleuth™

Marian Nodine, William Bohrer*
MCC

Anne Hee Hiong Ngu†
The University of New South Wales

Anthony Cassandra
MCC

September 13, 1999

Abstract

InfoSleuth[1] is an agent-based system for information discovery and retrieval in a dynamic, open environment. This paper discusses InfoSleuth's multibroker design and implementation. InfoSleuth's brokering function combines reasoning over both the syntax and semantics of agents in the domain. The broker must reason over explicitly advertised information about agent capabilities to determine which agent can best provide the requested services. Brokering in InfoSleuth is a match-making process, recommending agents that provide services to agents requesting services. Robustness and scalability issues dictate that brokering must be distributable across collaborating processes. Our multibroker design is a peer-to-peer system that requires brokers to advertise to and receive advertisements from other brokers. Brokers collaborate during matchmaking to give a collective response to requests initiated by non-broker agents. This results in a robust, scalable brokering system.

Keywords: Multibrokering, Semantic Matching, Facilitation, Multiagents, Information Agents, Heterogeneous Systems

1 Introduction

Distributed architectures partition the execution of tasks over processes spread out over a computer network. A particular process passes off some subtask to another process either by sending it a message and waiting for the response, or via a remote procedure or method call. The brokering function in a distributed system matches a request for a specific service with a remote process that can perform that service. Both distributed object systems and agent systems require the use of the brokering function. A broker or matchmaker is a process that implements and executes the brokering function. In a brokered system, agents advertise themselves to a broker, then the broker responds to queries about agents.

*Now at Athens Group

†This work was conducted while the author was on sabbatical leave at MCC

[1]The InfoSleuth Project ended June 30, 1997, and is currently in phase two, called the InfoSleuthII Project. Some of the work described in this paper has come under the auspices of both projects. However, in the remainder of the paper we refer to both projects as simply "InfoSleuth".

In this paper, we examine brokering from the perspective of tailoring it to add robustness, scalability, and flexibility to either an agent community or a distributed object system. In a distributed object architecture such as CORBA (Common Object Request Broker Architecture), a process can run a remote method by first accessing a broker (e.g., a CORBA ORB) to determine which remote process offers the procedure or method call. One basic assumption that these systems make is that the definition of the procedure or method interface uniquely defines its semantics. That is, ORBs do not check to see if the local implementation actually does what the caller intended.

Syntactic brokering uses the structure or format of a task specification to match a requester with a service provider, matching requests to object interfaces or query/scripting languages to decide which service providers to recommend. For example, "myRelationalQueryAgent" may advertise that it takes its input according to SQL 2.0 syntax. Any request for SQL 2.0 query processing could then be directed to this process. By this definition, CORBA currently provides syntactic brokering services.

Agent systems, like distributed object systems, offer the ability to partition the execution of a task over several processes distributed across a network. The InfoSleuth agent system [2, 7, 18] adds *semantic* brokering functions to complement the syntactic brokering process. Semantic brokering uses the intended operation and accessed information of the request to match it with the meaning of the offered services of the providing agent. Introducing the notion of semantic brokering allows a broker to recommend services based on the semantics that define the sub-task. For example, "myRelationalQueryAgent" might advertise or register with the broker that it has the capability to do query processing of relational algebra queries, but it cannot do any statistical aggregation within those queries. Any time some other agent requires a query to be run over some known set of relational data, and the query fits those constraints, myRelationalQueryAgent could be tapped to do the job.

Brokering can be provided by a single process or agent that saves all agent advertisements, and processes all requests for services. In our experience, this architecture works well for an agent system with dozens of agents. However, the single broker approach presents a single point of failure and a limit to scalability, especially as we envision an open agent system containing agents with differing capabilities and varying levels of intelligence. In InfoSleuth, we implemented a *multibrokering* approach where many (specialized) brokers collaborate to provide brokering services. We show that our multibrokering approach increases the robustness and scalability of InfoSleuth without compromising performance.

The remainder of this paper is organized as follows. In the following sub-section, we briefly review InfoSleuth, our agent-based information discovery and retrieval system where multibrokering is a core function. In Section 2 we discuss the nature of the brokering process that matches a service requester with a service provider. We argue that the brokering process is insufficient unless it incorporates both the semantics of the request and the syntax in which the request is made. Sections 3 and 4 describe important multibrokering principles and how they are implemented in InfoSleuth. In Section 5, we discuss the experiments that we conducted to confirm the robustness and the scalability of our multibrokering

approach. Finally, we discuss related work in Section 6 and conclude in Section 7.

1.1 InfoSleuth

The InfoSleuth system consists of a set of collaborating agents that work together for information discovery and retrieval in a dynamically changing environment such as the World Wide Web. Some typical InfoSleuth information gathering and analysis examples are:

- Notify me when the cost of hospital stays for a Caesarian delivery significantly deviates from the expected cost.
- Notify me when my competitors have many more news articles about object-oriented databases than normal.
- Tell me when the following pattern of unexpected events occurs over these different computers, as it may indicate an unauthorized intrusion.

The above complex queries cannot be performed by currently available search engines or multidatabases. They cannot be parsed by search engines because they deal with information at a semantic (rather than keyword) level, and also are specified over multimedia information sources. Similar problems occur with multidatabases, because the information may not be kept in database systems. Also, these queries deal with trends and collections of information. Thus, a more integrated, semantically-based, multimedia approach is required.

The strengths of InfoSleuth are that it can adapt to a wide range of information retrieval and analysis tasks, and that it can also adapt itself to changes in the availability and capability of the different agents in a given InfoSleuth community. Some of the types of retrieval and analysis tasks include:

- gathering information via complex queries from a changing set of databases and semi-structured text repositories distributed across an internet,
- performing polling and notification for monitoring changes in data,
- analyzing gathered information using statistical data mining techniques and/or logical inferencing, and
- noticing patterns in how information is changing that may indicate new trends or problems.

In addition to offering such a broad spectrum of information services, InfoSleuth communities also must be able to adapt to changes in their composition, including the addition, removal, or changing capabilities of agents, users, and information resources. Much of this adaptability is accomplished by the use of semantic brokering.

Figure 1 depicts the InfoSleuth agent architecture currently defined and deployed in InfoSleuth.

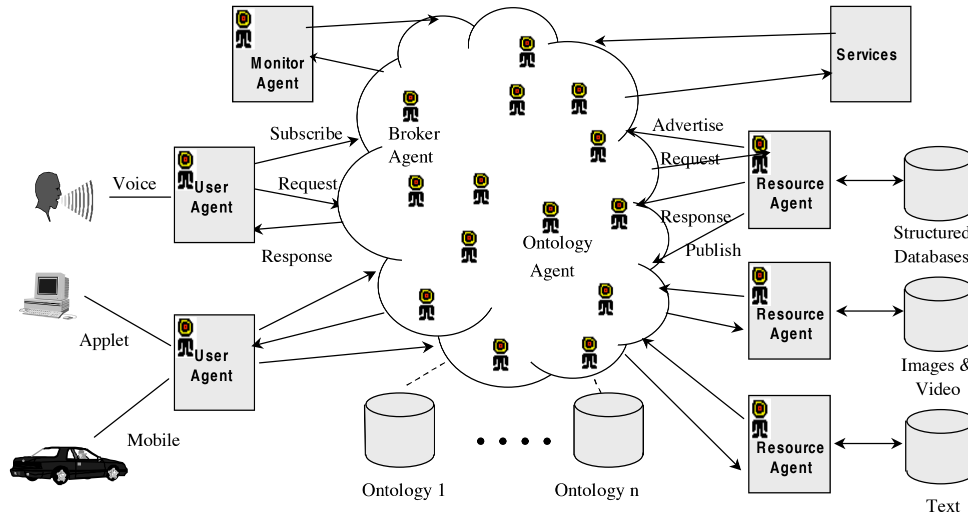

Figure 1: InfoSleuth: Dynamic and Broker-based Agent Architecture

The InfoSleuth agents are organized as core agents (those enclosed by the cloud) that provide basic information subscription, filtering and fusion capabilities, resource agents (those appearing to the right of the cloud) that serve as interface to external information sources, and user agents (those appearing on the left of the cloud) that act as proxies for individual users or groups of users.

In a given community of InfoSleuth agents, the core agents (broker agent, task planning agent, multiresource query agent, data mining agent, ontology agent) work together to connect users with the information resources that they need. These agents service requests over a set of common ontologies, accessed via the ontology agents.

One key element of an InfoSleuth community is its ability to adapt to the changing composition of the agents that make up the community. InfoSleuth uses a sophisticated brokering process to match agents to different parts of the information retrieval and analysis tasks posed by its users. In the following section, we discuss this brokering process and indicate how it helps to enhance the flexibility and robustness of the agent community.

2 Brokering

In this section, we discuss the combined syntactic and semantic-based brokering in InfoSleuth. We also describe a common service ontology that allows specification of both syntactic and semantic knowledge for agents to advertise to or query the broker.

2.1 Broker Agents

The broker agent maintains a knowledge base of information that other agents have advertised about themselves, and uses this knowledge to match agents with requested services. For instance, a number of different agents may advertise that they can answer data requests in SQL. However, one agent may advertise that it

contains a wealth of information about the healthcare domain, another that it is familiar with the aerospace industry, and yet a third that its sole function is to do complex query processing to assemble related information from different sources. When an agent requests resources that speak SQL and contain healthcare information, the broker's task is to return only the potentially relevant resources, in this case ruling out all but the agents that have healthcare information. Another important function the broker provides is to do some reasoning on constraints on the information content of an agent. If for instance, an agent has advertised that it knows about the healthcare domain model, it can also advertise the fact that its subsection of the domain model is restricted to podiatrists in Dallas and Houston. If the broker receives a request for information resources that do not overlap this sub-section of the healthcare domain, it will not recommend this agent.

The reasoning engine of InfoSleuth's broker agent is unique in that it can reason over different kinds of constraints expressed over the service ontology. For example, it can return all matched slots from classes that are fragmented. It can reason over class-subclasses and derived concepts relationships. The latter is particularly important because both content and capabilities are often represented hierarchically according to containment relationships. For example, if an agent does all query processing, then it certainly does relational query processing and could process a simple select query over a single relation. However, just because an agent can process a simple select query does not mean that it can do any relational query. We can represent a partial hierarchy of query capabilities as shown in Figure 2.

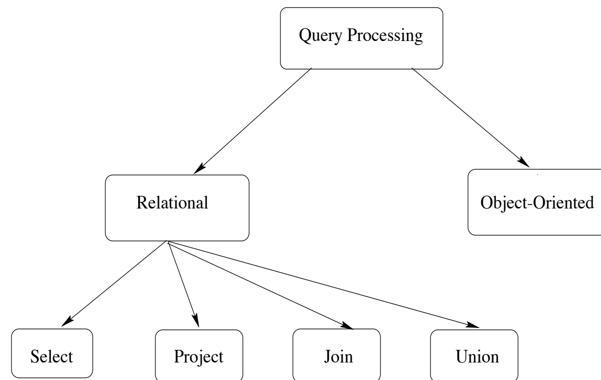

Figure 2: A Capability Hierarchy for Query Processing

2.2 The Brokering Process in InfoSleuth

The broker agent provides a *matchmaking* service to the InfoSleuth agents, matching agents that require services from other agents with agents that can provide those services. One of the primary jobs of a broker is to maintain a *repository* containing current and correct information about operational agents and the services they can provide. When an agent comes online, it announces itself to some broker by *advertising* to it, using the terms and vocabulary described in its *service ontology* (see Section 2.3). This is shown in Figure 3. The broker stores all of the advertised information in its repository. When an agent's set of available services

changes, the agent may update its advertisement, and the broker will update the information in its repository. When an agent goes offline, it first *unregisters* itself from the broker. Also, the broker periodically pings each of the agents that have advertised to it, to discover any agents that have failed. The broker removes from its repository all information about agents that have failed or unregistered themselves.

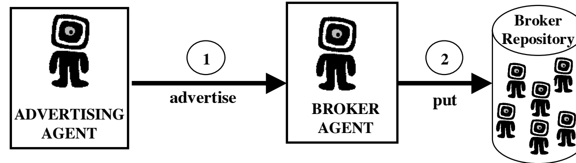

Figure 3: Advertising to the broker

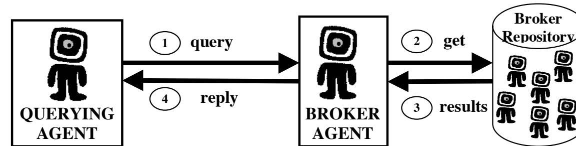

Figure 4: Querying the broker

Brokers also receive *queries* from agents that are looking for other agents that can provide specific services. These queries are specified in terms of the service ontology. The broker uses a rule-based reasoning engine implemented in LDL [25] to reason over the query and advertisements to determine which agents have advertised services that match those requested in the query. This set of matching agents found by the reasoning engine is returned by the broker to the requesting agent. This process is shown in Figure 4.

In InfoSleuth, the broker agents play a critical role in maintaining an up-to-date repository of all of the agents available for access within an agent system. Without the presence of brokers (or other agents that have similar capabilities), an agent would be unable to locate new agents that could provide services to it. This locational task is critical either in the case where the agent knows about no other agents that can provide a needed service or in the case where an agent is looking for *all* agents that can provide a service. For instance, when the agent system's purpose is to provide information, the broker is needed to ensure that *all* available agents that can contribute information are located, even in a system where such agents may come on- and off- line frequently.

To illustrate this, we will show how the brokering is used to process a simple query over multiple resources in a single broker system. As each agent comes online, it advertises its capabilities as shown in Figure 5. Thus, after the actions in this figure, the broker has a repository containing four advertisements, one for each of agents "mhn's user agent", "MRQ agent", "DB1 resource agent" and "DB2 resource agent".

At some point in time, user "mhn" submits the SQL query "select * from C2" to her user agent. At this point, "mhn's user agent" must locate a query processing agent that can assemble all of the information that this community knows about class "C2", as shown in Figure 6. In this figure, "mhn's user agent" first forwards a query to the broker agent for one multiresource query processing agent that can accept and

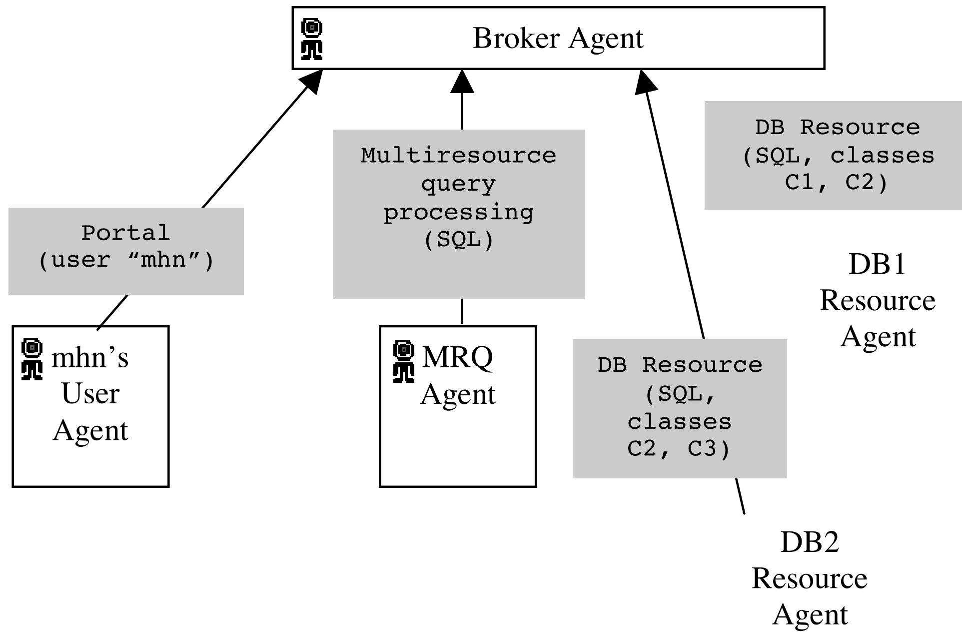

Figure 5: Agents advertising to the broker

process SQL queries. The broker replies, stating that ”MRQ agent” fits the description in the request. The user agent then forwards the query to ”MRQ agent”.

When agent ”MRQ agent” receives the query from the user agent, it looks at the query to determine which classes are required to answer the query, and discovers that it needs to look for class ”C2”. It then queries the broker for all resource agents that can answer an SQL query involving class ”C2”. This is shown in Figure 7. The broker processes the query and returns the responses ”DB1 resource agent”, ”DB2 resource agent”. ”MRQ agent” then forwards a query to these two agents, receives the responses, assembles the result, and forwards it back to ”mhn's user agent”. Note that if the original query had been for class ”C3”, then only the response ”DB2 resource agent” would have been returned tot ”MRQ agent”.

Let us assume for the moment that a new agent, ”MRQ2 agent” comes online and advertises to the broker, stating that it is a multiresource query agent that speaks SQL and specializes in queries over the class ”C2”. If now user ”mhn” poses the same query to her user agent, then agent ”MRQ2 agent” would be recommended to the user agent because it has a better semantic match to the request than does agent ”MRQ agent”.

2.3 Brokering Knowledge

Syntactic brokering is the process of matching requests to agents on the basis of the syntax of the incoming messages. A classic example of syntactic brokering is found in CORBA (Common Object Request Broker Architecture) [9], which locates processes that can execute a method call that has a specific signature defined in its Interface Definition Language. In the agent-based community, KQML (Knowledge Query Manipulation Language) [23] specifies agent advertisements as templates for KQML messages representing requests

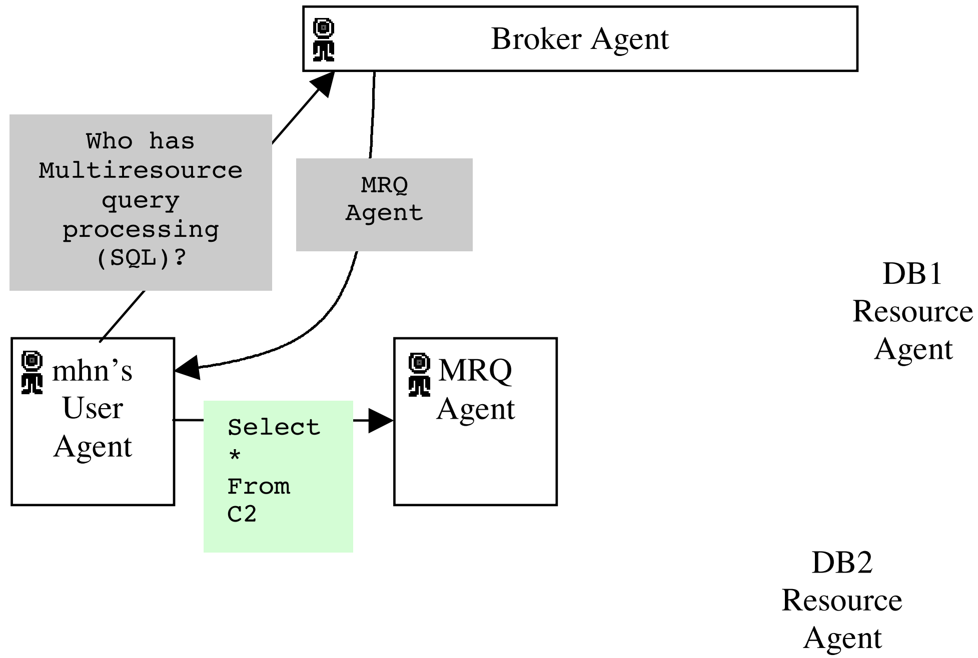

Figure 6: User agent querying the broker for general-purpose SQL query agents.

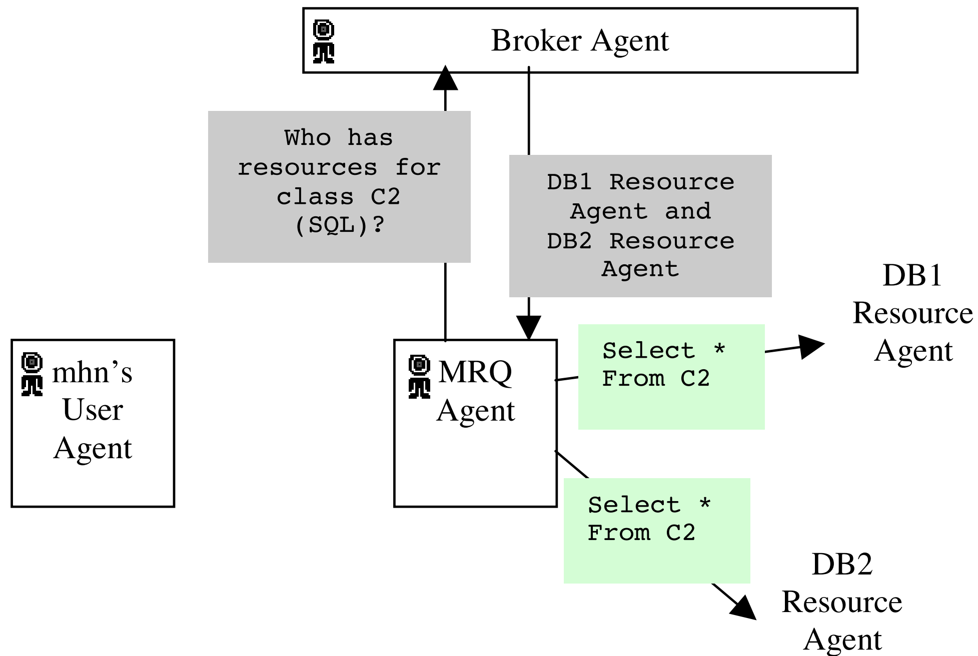

Figure 7: MRQ agent querying the broker for resource agents.

for services. Requesting agents must send request messages that effectively "fill in" these templates in order for the request to match the advertisement. This approach also relies only on syntactic-level brokering. Figure 8 lists some of the syntactic information that the broker may use.

Agent name and location

- Unique identifier for the agent
- Directions on how to contact the agent (host, port, transport protocol)
- Agent type (e.g., query agent, resource agent)

Agent syntactic knowledge

- Communication languages/services (e.g., CORBA)
- Content languages (e.g., SQL, LDL)

Figure 8: Syntactic agent service ontology information

In a general agent system, brokering needs to take into account knowledge beyond the syntactic considerations. Consider the following case: A query packaged in a KQML's ask-all performative enters the agent community and is passed to the task execution agent for task planning. This agent takes the user context and the query, and plans the general execution of the query. This plan includes deciding where to forward the query for processing and whether to cache the result or analyze it. The same query is then forwarded in the same performative to the multiresource query agent (MQA), which decides whether or not the query involves data from multiple resources. If it only involves one resource, the MQA will forward the same query in the same performative to the resource agent, which runs the query against its database and returns the results. Here, we have three different agents taking the same performative as input (i.e., the same interface), but doing quite different things with the embedded query. Thus, the syntactic information contained in the `ask-all` performative is not enough to decide at which point in the processing the agent should receive the query – the broker agent must also understand whether the agent's semantics for the performative match the current requirements.

Semantic brokering is the process of matching requests to agents on the basis of the requested capabilities or services, and/or (in an information system) constraints on the information that an agent can provide. This agent knowledge is expressed independently of syntax, using the common service ontology. Figure 9 lists some of the different semantic information that the broker may use.

As with syntactic brokering, semantic brokering in and of itself is not sufficient for correct assignment of agents to requests. Consider a case where there are multiple query processing agents, all of which process queries specified in languages that are based on relational algebra, but one agent expects its input in SQL, while the other expects its input in a relational subset of OQL (Object-oriented Query Language). In this case, the semantics are not sufficient to distinguish which agent to select for processing a specific relational query.

In InfoSleuth, we take into account both syntactic and semantic brokering knowledge when matching service requests to agent advertisements. This is because, if a broker fails to take into account syntactic

Agent capabilities

- Conversation types the agent can participate in (e.g., ask-all, subscribe, emergent)
- Agent's functionality (e.g., O-O query processing)
- Restrictions on those capabilities (e.g., multimedia joins, vertical fragmentation)

Agent content

- Supported ontologies (e.g., healthcare)
- Restrictions on ontologies (e.g., patients over 65)

Agent properties

- Adaptivity (e.g., cloneable, mobile)
- Processing statistics (e.g., throughput, estimated processing time)

Figure 9: Semantic agent service ontology information

constraints, the recommended agent will be unable to understand the message it receives. If a broker fails to take into account semantic constraints, the recommended agent may perform some action different than the one intended. Specifications such as CORBA and KQML that define purely syntactic brokering make the assumption that syntax implies semantics - an assumption that fails as the agents become more and more autonomous. We also may take into account additional pragmatic properties like the processing capacity of the agent or its ability to move.

2.4 An Example

In this section, we show an example of how an advertisement and a query to a broker are specified using the InfoSleuth service ontology described above.

Resource agents are the back-end agents within InfoSleuth which act as proxies for structured or semi-structured repositories. A resource agent, let's name it "resourceAgent5", can send an advertisement to the broker with the following content:

Agent name and location
agent address: tcp://b1.mcc.com:4356
agent name: ResourceAgent5
agent type: resource
Agent syntactic knowledge:
agent interface query language: SQL 2.0
agent communication language: KQML
Agent capabilities:
supported conversations: subscribe, update, ask-all
capabilities: relational query processing, subscription
Agent content:
supported ontology name:healthcare
supported ontology classes: diagnosis, patient
supported ontology slots: diagnosis-code, patient-age
supported class key: patient-id
constraints description:patient_age between 43 and 75
Agent properties:
properties:non-mobile
estimated response time:5

The advertisement above tells the broker that the sending agent can be contacted via the `tcp` transport protocol at port `4356`, on the host machine `b1.mcc.com`, using `KQML` as the inter-agent message

exchange language, and accepting `SQL 2.0` queries. Under the semantic knowledge category, it specifies that the agent can perform `relational query processing`. It accepts `subscriptions` (i.e., allows the user to monitor certain events or changes in data). It has knowledge about the fragment of the `healthcare` domain model that deals with `diagnosis` and `patient` classes and the patient data is restricted to `patients between the age of 43 and 75`. It is a `non-mobile` agent and can return the answer within `5 seconds`.

Upon receipt of this advertisement, the broker validates and translates the advertisement into a format that its reasoning engine can understand and asserts it in its repository. The content of an `ask-all` query to the broker to find which resource agents can answer QueryAgent2's request for patients between the age of 25 and 65 with diagnosis code 40w is shown below:

Agent name and location:
agent address: ?agent-address
agent name: ?agent-name
agent type: resource
Agent syntactic knowledge:
agent interface query language: SQL 2.0
Agent content:
ontology name: healthcare
ontology classes: ?available-classes
ontology class slots: ?available-class-slots
ontology class keys: ?class-keys
data constraints: (patient_age between 25 and 65)
AND (patient.diagnosis_code = '40W')
Agent properties:
estimated response time: ?response-time
Result format:
?agent-address, ?agent-name, ?class-keys
?available-classes, ?available-class-slots
?response-time

The information that is being queried for is specified in the result format clause. The syntactic or semantic information that the agent does not care about is not specified in the content. The semantic information that this agent is particularly interested in is the contents (`healthcare`) and its data constraints (`patients between the age of 25 and 65,diagnosis code 40W`).

The reasoning engine matches the constraints on agent type, agent interface query language, ontology name, and data constraints with the advertisements that it has stored. Note that the reasoning engine would match the agent that advertised knowledge about patients between 43 and 75 (i.e., ResourceAgent5).

3 Multibrokering

Multibrokering allows the process of matching service agents to requests to be distributed across multiple brokers, each representing a different set of agents. That is, brokers collaborate with each other in making recommendations to requesting agents for specific services that other agents have advertised. When a broker receives a request for an agent with specific capabilities, it looks for matches in its own repository of agent information and may also query other brokers to find external agents with needed capabilities.

In this section, we discuss the limitations of single broker systems, and present some principles required

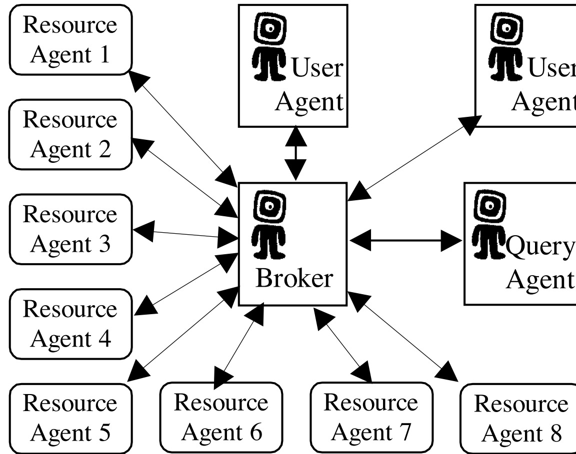

Figure 10: Single Broker Architecture

for building a multibrokering system. These multibrokering principles include methods for brokers to exchange information in a collaborative peer-to-peer fashion rather than arranging themselves hierarchically, organization of information about both agents and other brokers, proposed methods for discovering brokers, and policies for initiating inter-broker searches.

3.1 Single Broker Architecture

In a single broker architecture, the broker is the central repository for information about available agents and resources in the system. Rather than caching information about the entire system, when agents come online they need only know the location of the broker and how to query it for information to locate other agents in the system. Each agent then advertises itself to the broker and queries the broker when it needs to locate other agents. The single broker architecture and its relationship to the InfoSleuth agents that advertise and query to it is illustrated in Figure 10. In this view of the system, there are eight resource agents (beer1-4, rabfA1-4) advertised to the broker and three requesting agents who queried the broker.

A single broker represents a single point of failure. If the broker cannot be located, no inferencing on the data domain can be performed and the agents in the community will be unable to locate other agents accurately. A single broker system also represents a hard limit to scalability. While a single InfoSleuth broker can easily keep track of dozens of agents participating in the system, there is no doubt that at some point when a single broker trying to store and process information about thousands of agents will be unable to respond in a reasonable amount of time.

3.2 Principles for Multibrokering

The major goals of multibrokering center around robustness, flexibility and scalability. With multiple brokers, processing load can be more evenly distributed around the system. In the current system, the knowledge processing must be handled by one centralized broker which must know how to reason about all domains. Allowing for multiple brokers allows for parallel development of more precise reasoning over

narrower domains.

Peer-to-peer Architecture. We use a peer-to-peer topology for inter-broker connectivity. Since this allows any broker to freely advertise/unadvertise to any other broker in the system, it is more scalable. More importantly, this topology ensures that there isn't a single point of failure.

A hierarchical architecture for multi-brokering, in which one broker plays the role of "super-broker", has the same robustness and scalability problems as single-broker systems. First, the super-broker represents a single point of failure. Second, when the load on the super-broker becomes too large, the old super-broker must split into two brokers and a new super-broker added at the top. Hierarchical brokering represents a postponement of, rather than a solution to, scalability. Peer-to-peer brokering circumvents these scalability issues. So long as brokers may freely advertise and unadvertise themselves to other brokers , entry to and exit from the group of brokers is easy. Newly advertised brokers' data will be integrated into each new search as if the data had always existed, and brokers which have unadvertised themselves will simply cease to exist so far as the rest of the system is concerned.

The only major disadvantage of a peer-to-peer architecture is the cost of inter-connection. When the number of brokers become very large, the connectivity cost could be significant. However, we may be able to reduce the connectivity cost on a per-search basis by only propagating requests along a spanning tree of the current broker digraph.

Non-broker agents must advertise.

Non-broker agents must advertise their capabilities to at least one broker. Robustness increases if agents advertise redundantly to several brokers. This ensures that an agent is still available if one of the brokers it has advertised to goes down. It is the agents' responsibility to ensure that the different copies of their advertisements are kept consistent in all the brokers to which they advertise.

In a multibroker system, we need to also define how a non-broker agent connects to the broker(s). In theory, it is possible for every non-broker agent to connect or advertise to every broker in the system. This would be very robust, but is impractical for large numbers of agents and brokers.

Brokers may specialize. In a world with multiple systems and brokers interoperating, an agent should take care to ensure that it advertises to brokers that best represent its interests. For example, if a food supplier agent advertises to a broker that only brokers healthcare information, the broker should forward it to a broker that can deal with food suppliers. If no such broker exists, it may reject the advertisement. With this approach, the brokers will need metrics to measure how well the advertisement fits within the broker's advertised purpose. When brokers specialize in certain domains, it is possible to develop optimized reasoning over a narrower domain and hence lead to better performance when the number of agents and brokers becomes very large. To prevent the possibility of all brokers rejecting some agent whose advertisement fits in with no broker, each group of cooperating brokers should contain at least one general-purpose broker for queries not covered by the specialized brokers.

3.3 Multibroker Architecture

In a multibroker environment, a broker not only keeps information about other agents in its repository, it also keeps information about other brokers that it knows about. Brokers are connected in a directed graph structure, with the nodes representing brokers and the arcs representing knowledge of other brokers' current advertisements. Thus if Broker1 has advertised itself to Broker2, there is an arc from Broker2 to Broker1. The connectivity should be sufficient to ensure that each broker is either directly or indirectly connected to all the other brokers in the system (i.e. no disconnected sub-network of brokers).

Collaborative Reasoning. Each broker maintains, in addition to its repository of advertised agent information, a reasoning engine that matches queries to its own set of agent advertisements. This reasoning engine may also match broker queries to other brokers that may contain advertisements for other agents that can service the query. Each broker request is forwarded to relevant other brokers, which then may propagate the requests further. The response to the broker query contains the union of all agents which have advertised to some broker that the broker query reached, and which match the request. Searches are restricted internally to prevent undesirable propagation of requests, e.g., cyclical propagation and prolonged search in a very large system.

A *broker consortium* is a set of brokers that are fully interconnected, such as the one shown in Figure 11. This figure shows four brokers with eight resource agents, and some user and query processing agents. When Broker1 receives a request, it analyzes the content and uses the rules in its rule base to reason over its agent repository for any possible matches. It also initiates an inter-broker search (if asked by the requesting agent using the search policy option). If the search policy is to always expand the search to other brokers, Broker1 will forward the request simultaneously to all the other brokers that it knows about (Broker2, Broker3, Broker4). Each of these brokers then use the same procedures as were used by Broker1 to match for potential agents that can provide the requested service. Eventually, the other brokers return their results to Broker1, which combines them with its own (possibly empty) list of providing agents, eliminating duplicated entries. Broker1 then returns the combined list to the requesting agent.

Consortia may be configured explicitly, or may form naturally among brokers whose agents share a common set of goals or interests. A given broker may belong to more than one consortium; therefore, a set of interconnected brokers that can collaborate takes the form of a connected network of broker consortia. Figure 12 shows interconnection of several consortia of brokers. Note that in both diagrams we use bidirectional arrows between brokers that know about each other.

Multibroker Service Ontology. In Section 2.3, we described a service ontology that non-broker agents use for advertising and querying the broker. Under a multibroker architecture, we need to expand that ontology to allow brokers to specify their broker reasoning capabilities and their specializations, as well as their inter-broker communication abilities to other brokers.

For the most part, broker advertisements should be specified against the same ontology as the agent

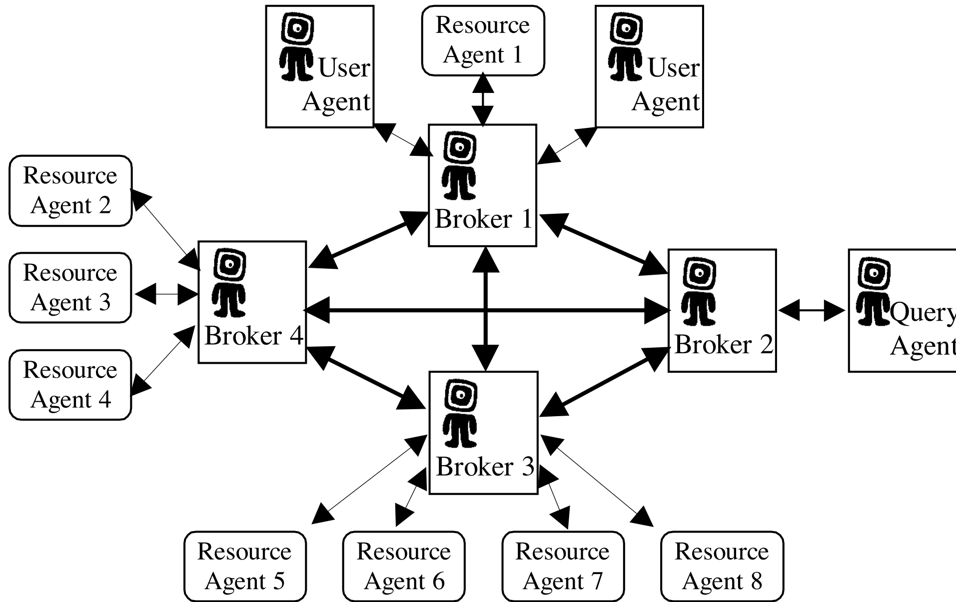

Figure 11: Multibroker architecture

advertisements. However, the ontology may need to be extended to take care of such items as the broker's consortium memberships and the types of agents the broker has in its repository. Figure 13 shows example extensions of the service ontology with respect to multibrokering capabilities:

4 Implementation of Multibrokering

Aspects that are crucial to the robust implementation of a multibroker system include:

1. integrating new agents and brokers into the broker network,
2. ensuring all brokers and agents remain interconnected, and
3. ensuring that the brokers process queries collaboratively and thoroughly.

4.1 Discovering Brokers

In this section, we briefly outline a broadcast approach for agents and brokers to use to locate (other) brokers. With this approach, each broker is a member of at least one consortium. The consortia overlap their membership in such a way that they all interconnect. A broker advertises its location and capabilities to all the other brokers in each of its consortia. By default a broker must advertise at least its locational information. However, when a broker also advertises its capabilities to another broker, a broker can reason over the other brokers' capabilities and eliminate brokers that definitely should not be contacted during an inter-broker search. This improves the processing time by ruling out unnecessary queries.

Broker Objectives and Advertisements. With independent brokers, each broker may have a specific objective for the type of agent information it maintains. This objective is reflected in the capabilities that

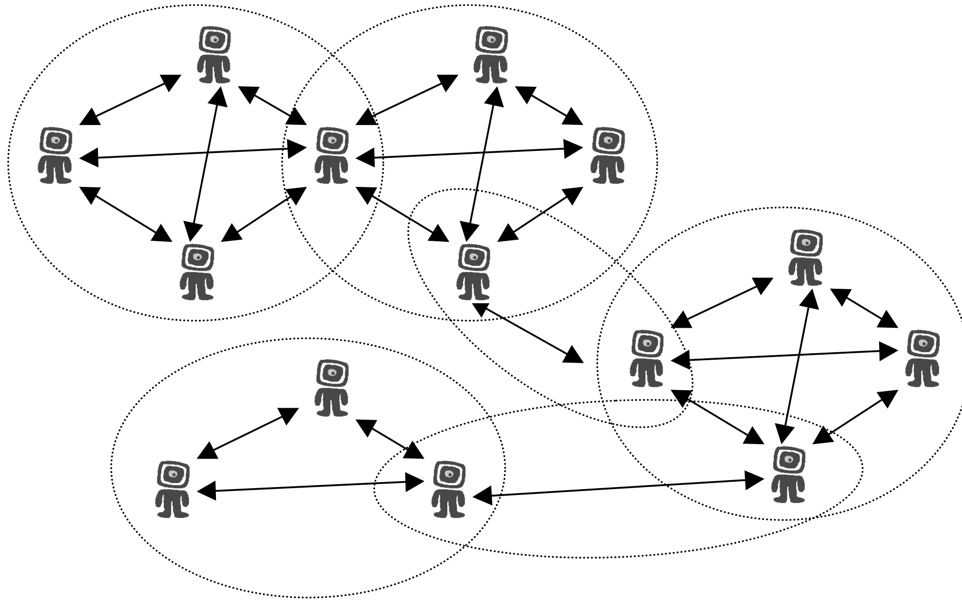

Figure 12: Interconnected consortia of brokers

it advertises to other brokers. Additionally, a broker should be able to determine what advertisement it is willing to accept in an open agent system based on its objective. If the objective of a broker is to provide services to a wide variety of requesting agents, then it should accept as many non-overlapping advertisements as possible. If the objective is to develop a specialty in brokering over certain chosen domains, then it should only accept advertisements that overlap with its chosen domains. A broker may also modify its objective based on, for instance, an analysis of the queries it is receiving.

Brokers Discovering Other Brokers. When a new broker starts up, it needs to know which broker consortia it wishes to join. This may be specified either via the broker's configuration parameters, or may be implicitly specified using a well-known port for each consortium. Once that is identified, it can advertise to all the brokers in each such consortium. The new broker may also query the other brokers it has advertised to for their lists of broker advertisements that may fit its own specialization, so that it can select and pull interesting advertisements into its own repository.

A broker receiving an advertisement may accept or reject it, or pass it on to other potentially-interested brokers. The sending broker will receive a confirmation message from any broker that accepts its advertisement. If no brokers accept the advertisement, the broker that received the original advertisement will reply with a `sorry` message. The sending agent may then try to locate other brokers via some external mechanism such as published lists or bulletin boards.

Agents Discovering Brokers. Each non-broker agent is configured with one or more preferred brokers to connect to on startup. This represents its initial entry point(s) into the brokering system.

Once operational, the agent may decide to change to a different broker. To do this, it sends a query to

Broker Syntactic Information

- Contact directions
- Community membership
- Consortial membership
- Communication language
- Content Language

Broker Semantic Information

- broker conversation types (e.g., delegation, forwarding)
- broker's functionalities/services
- restrictions on services
- broker's specializations (e.g., agent types in respository)
- supported ontologies
- restrictions on ontologies
- performance

Figure 13: Multibroker service ontology information

the preferred broker for one or all of the brokers that are available in the system with the capabilities and data domain that it is interested in. It then picks one in the list to use. Alternatively, the agent might use the preferred broker and keep a history of how this broker handles its request. If over a period of time, the user discovers that its preferred broker always forwards the request to a specific broker or set of brokers, then he could reconfigure his agent to add the new broker to its list of preferred brokers.

4.2 Maintaining Connectivity

Redundant advertising and robust connectivity are keys to maintaining a reliable network of brokers.

4.2.1 Redundant Advertising

Redundant advertising is the practice of agents and brokers making identical advertisements of their services to more than one broker. Every agent or broker maintains a configuration parameter defining how many brokers that agent or broker should advertise to. In addition all agents, including broker agents, keep track of two lists of brokers: a list of brokers that they know about (known-broker-list), and a list of brokers they have successfully advertised to (connected-broker-list). The connected-broker-list is a subset of the known-broker-list.

Each agent or broker advertises to brokers on the known-broker-list but not on the connected-broker-list. When an advertisement is successful, the broker that kept the advertisement is added to the connected-broker-list. Once the number of such connected brokers reaches the configured number of redundant advertisements, the advertisement process stops.

In the event that a broker should unexpectedly leave the agent community, it is the responsibility of each agent to detect that the broker has left and re-initiate the advertising process. In the meantime, given that there was a redundant advertisement, the agent will still be visible to other agents in the system via the

remaining brokers that it has advertised to.

During operation, an agent may also discover more brokers that it deems appropriate to advertise to. In this case, the agent adds them to its known-broker-list.

4.2.2 Robust Connectivity

Agents periodically test to see whether all of the brokers they have advertised to still know about them. At some (configurable) periodic interval, agents will cycle through the connected-broker-list, and query each broker in turn to see if it still knows about them. We call this a “broker ping”.

If the broker has died, either the transport layer will fail to make the connection to the broker or the broker will fail to respond. In either case, the transport layer of the agent detects this, and the broker ping fails. In the event that a broker is alive but does not have information about the agent that is doing the querying, the broker will receive a reply containing no matches from the broker that it queried, and it will remove this broker from its connected-broker-list.

Once an agent has successfully traversed its connected-broker-list, it checks to see if it needs to re-advertise, as described in the previous section. If so, it re-initiates the advertisement process. If at the end of the re-advertisement the agent is connected to no brokers, the agent will enter a dormant state and wait until the next polling interval and attempt to reconnect.

4.3 Broker Query Processing Policies

Given that we can locate other brokers, the question remains as to when to start looking at other brokers when processing a brokering request. Suppose a broker gets a request for information about database resources dealing with the healthcare industry in Dallas, Texas, but finds that it doesn’t have any information stored about agents that would meet these particular criteria. The requesting agent can then specify the policies under which it wishes for the broker to initiate an inter-broker search. This policy needs to be passed along when one broker forwards a message to another broker. If the requesting agent did not specify any policy, the default policy set by a broker will be used.

Our implementation of the inter-broker search policy follows closely those defined for the trading service in CORBA [20]. It is a property list consisting of the following items:

- hop count - this is the maximum number of hops between brokers that the request will traverse. It can be overridden by the broker’s max_hop_count. The default is set to one, which limits the search to the broker’s own consortium and other directly-connected brokers.

- follow option - this indicates whether the matchmaking process should only consider the local broker’s repository, or all repositories, or as many repositories as are needed to find a single match. If the request is for a single agent, this defaults to the “until you find a single match” policy; otherwise it defaults to the “all repositories” policy.

When a broker forwards a request to a second broker and a second broker forwards that request to a third broker and the third broker passes it on again, the request might end up back where it started. To prevent such a loop, we keep a list of brokers that a request has been forwarded to and pass this list along with the message.

5 Experimental Results

This section presents two types of empirical evaluations of InfoSleuth's multi-brokering mechanisms. In Sub-section 5.1 we present some experiments done using the InfoSleuth system directly. However, because running multiple, large-scale experiments with InfoSleuth poses some difficulty, in 5.2 we present some simulation-based results.

5.1 Multibrokering behavior in InfoSleuth

We performed a set of experiments using the InfoSleuth system to determine if multibrokering was feasible and if specialization helps. We also did some limited tests on scalability.

We tested the response time for queries under different configurations of resource agents, number of brokers, number of query agents and user agents. Each experiment consisted of issuing SQL statements (encapsulated in KQML messages) to an InfoSleuth agent community and processing them according to the procedure described in Section2.2, The response time is the total time for the user to get the result displayed on the screen from the time the query is submitted. This includes CPU, disk I/O, communication among agents and graphical display of results.

Table 1: Experimental Query Streams

name	# RAs[2]
SA (single agent)	1
DA (double agent)	2
4A (four agent)	4
VF (vertical fragmentation)	4
CH (class hierarchy)	4
FH (fragmentation & class hierarchy)	4

The types of queries submitted are characterized in Table 1, and are representative of the majority of the types of queries that InfoSleuth currently handles. We use the query streams shown in Figure 2 for running the queries. In this multibroker case, each broker is running on a different Sparc Ultra 1 machine. The single-broker variant of each experiment has all the agents running on a single broker on a single Sparc Ultra machine. All of the Sparc Ultra 1 machines were running the SunOS 2.5 operating system.

All queries are executed by a single query agent.

Each experiment is repeated 3 times. Table 3 shows the average response time expressed as a ratio of multibroker/single broker for each of the above type of queries. A ratio of less than 1.0, implies improved

Table 2: Experimental configurations

Expt	SA	DA	4A	VF	CH	FH	#RAs
1			√				4
2	√	√	√				4
3	√	√	√	√			8
4	√	√	√	√	√		12
5	√	√	√	√	√	√	16

performance of multibrokering over single brokering.

Table 3: Experimental results

Expt	4A	DA	SA	VF	FH	CH
1	1.00					
2	1.04	1.05	1.01			
3	1.12	1.01	1.05	0.85		
4	0.98	0.95	0.91	0.77	0.86	
5	0.3	0.31	0.47	0.76	0.63	0.67

When the system is underloaded (Experiment 1-3), the response time for queries is slightly better in a single broker versus a multibroker system (the ratio is greater than 1.0). However, the difference is less than 0.1 in most cases. Thus we can conclude that the response time for a query did not degrade with an increased number of brokers. On the other hand, when the system is loaded (Experiment 4-5), the response time in multibroker systems is better for all the queries.

We also conducted a sixth experiment to check the effect of specialization of brokers in a multibroker environment. This experiment used the same agents and query streams as Experiment 5, but with all the resources associated with a given query stream kept at a single broker. Table 4 shows the average response time expressed as a ratio of multibrokering with specialization/multibrokering without specialization.

Table 4: Experiment 6 Results

Expt	4A	DA	SA	VF	FH	CH
6	0.86	0.86	0.87	0.74	0.6	0.29

This experiment shows that there is an improvement in response time for all the above type of queries with specialization of brokers (ratio less than 1.0). Intuitively, this is because the individual brokers reason over less information, and therefore the reasoning is more straightforward and less costly.

5.2 Simulation-based Experiments

There are many obstacles involved with running large scale experiments using actual agent applications, many of which we experienced while conducting the experiments of the previous sub-section. The mecha-

nisms for managing very large numbers of agents do not exist in the current InfoSleuth system nor do they exist in any other agent system at present. This makes the set-up, execution, monitoring and result gathering for the experiments extremely difficult, especially as the number of agents to be managed increases. Additionally, to run experiments with hundreds of agents requires enough resources for all the agents to run; including both hardware and time. Furthermore, if these are real agents, they will each need to be configured, possibly requiring fabricating enough data for all the agents to use. Another factor in trying to evaluate an isolated portion of the InfoSleuth system, which is a complex prototype, is that non-essential components with less than optimal implementations can degrade the performance to the point where it masks the true impact of the system characteristic being evaluated. Finally and more generally, evaluation of agent system properties is often desired at design-time before any application exists.

A simulation-based approach to evaluating an agent system overcomes all of these obstacles. A simulation gives complete control over all of the agents; it doesn't need to run in real time; each agent requires far fewer resources; there is no need for real data; and you can model only the relevant characteristics that affect the performance of the system, eliminating any negative effects caused by non-critical components.

At present, the resource constraints have made it very difficult to run controlled experiments in the existing InfoSleuth system, especially when there are more than a few dozen agents. Because of this and the desire to demonstrate that multi-brokering is robust and can scale up to non-trivial numbers of agents, we have used a simulation-based approach for evaluation. The agent simulation is built upon a discrete-event simulator, modeling both machine characteristics and network connections. The broker behaviors were implemented to closely mimic the behaviors of the brokers in the actual InfoSleuth system. Below, we present a broad overview of the simulator, since space constraints of the paper prevents a more comprehensive description.

There were fewer types of agents used in the simulation experiments than were used in the InfoSleuth experiments. Since we wanted to focus on the broker characteristics, we limited the types to broker, resource and query agents. The query agents are simply a mechanism for putting a load on the brokers, while the resource agents simply defined the amount and type of information the brokers have to reason about. The focus of the experiments is on the performance of the brokers under various configurations.

5.2.1 The Simulator

The simulator used was built in-house at MCC as part of an independent project. It is meant to be more general than simply a simulation of the InfoSleuth system. However, because of its generality, we were able to set the parameters and behaviors of the agents to closely match those of the agents in the InfoSleuth system. Below we discuss the simulator, which also serves to describe our experimental set-up.

Processor Model At the lowest level of the simulator is a model of a processor, which is the fundamental unit on which agents can run. While multiple agents can be put on the same processor to mimic realistic resource sharing, for the experiments conducted here, all agents are assumed to be running on separate

processors.

The processors' main parameter is the speed, which is a relative measure of how fast they can compute. This can be set on a processor-by-processor basis to mimic a non-homogeneous computing environment, though the experiments shown here used identical settings for all processors.

Network Model Each processor has a model of its connection to other agents. While the simulator allows an agent to have multiple network connections, the experiments used here assume each agent has a single connection to the network. The main parameter for the network is its speed or bandwidth. For these experiments all network connections have the same speed which we set to a fairly conservative value. Although the effective bandwidth is very difficult to measure, we chose something that is on the high side of 10 megabit Ethernet connection: 250 kilobytes per second.[3] We also modeled the network latency time to account for the overhead required independent of the message sizes. In these experiments the latency was a very conservative 0.1 seconds.

Hardware Reliability Both the processor and network connection models admit to being unreliable. We assume an exponential distribution for the time to failure and a separate exponential distribution for the time to repair. For all experiments except those concerning system robustness, the hardware was set to be perfectly reliable. For the robustness experiments we varied the mean time to failure of the brokers' processors only, where the means are shown in the experimental results section.

Common Agent Model For the three types of agents in the simulator (query, broker and resource), there are some common parameters that are shared by all three, though not all agents use all of the parameters. For agents that need to constantly check the existence of other agents, there is a ping interval defining the maximum length of time it will allow to pass without any contact with another agent. This was set to 300 seconds. Additionally, there is a time-out period defined to limit the amount of time an agent will wait for a reply from another agent. This too was set at 300 seconds.

Resource Agent Model To mimic the presence of ontologies where different agents have data from different ontologies, each resource agent is defined to have a particular data domain. The total number of distinct data domains in a given simulation run was one fourth of the total number of resource agents for all but the robustness experiments below. Thus, a query over a particular data domain would have four separate resources that satisfied the query. For the robustness experiments, each resource agent had its own unique domain, which helps to track exactly how often a query was satisfactorily answered.

In addition to a data domain, the resource agent model defines the amount of data it has, the size and complexity of the advertisement it sends to the broker, how quickly it can answer queries over its data as a function of the size of its data, the complexity of the query being asked and the size of the resulting

[3] See `http://www.isi.edu/lsam/publications/http-perf/#nets` for some discussion of effective network speeds.

answer. Note that the size of the query answer is defined by how much data the query covers and the size of the resource's data. Aside from the query engine speed and the queries complexity factor, the resources query answering speed will also be scaled by by the speed of the processor the agent is running on. For the experiments in this paper, the base query answering speed of all resources was set to be 1 second per 100 megabytes of data.

A resource agent will advertise their data to all of the brokers it is connected to. For the experiments where resources were connected to only a single broker, the broker was chosen uniformly randomly from among all the brokers in the system at start-up, to prevent any regular distribution pattern of data domains over the brokers, which tended to skew some of our earlier experimental results. For the robustness experiments, we used redundant advertising where the amount of redundancy was varied as discussed in the results sub-section below.

Query Agent Model A query agent in the simulator serves only to put a load on the system. Each query agent is independent and generates queries to a broker at times that are exponentially distributed. If the query agent is connected to multiple brokers, it uniformly randomly chooses a broker on each query issued. For our experiments, we used a single query agent connected to all the brokers in the system to represent the overall system load on the brokers.

The querying of the brokers are done independently according to an exponential distribution, but upon a reply from the brokers, the agent will query any matching resource agents that the broker returns. Thus, the queries to the resource agents are correlated with the queries to the brokers.

Each query, whether to a broker or a resource agent, has its data domain, complexity and coverage set randomly. The data domain is selected uniformly over all the available data domains (defined by the resource agents) and will be used by the brokers to match to the resources which have advertised over that domain.

The complexity of the query is a relative measure of how much time the broker or resource agent will require to answer the query; more complex queries will require more processing time. A query with complexity 0.5 can be processed twice as fast a query with complexity 1.0. The complexity is randomly generated according to bounded Gaussian distribution; we put bounds on the Gaussian to ensure we always get a positive number. For all the experiments shown here, the complexity is set to be 1.0 (i.e., mean of 1.0 and variance of 0.0).

The coverage is only used by the resource agents and defines the size of the result of the query relative to the amount of data a resource agent has. A query whose coverage is set to 0.1 means that after the resource agent processes the query, the size of the result will be one-tenth of the data the resource agent is defined to have. This too is generated using a bounded Gaussian distribution, this time to ensure that values stay between zero and one. For the experiments conducted here, the coverage used had a mean of 0.001 and variance of 0.001.

Broker Agent Model A broker will accept advertisements and add them to its list of advertisements. Each advertisement has a size and upon reception of the advertisement the memory consumption required by the broker is incremented. The brokers also respond to queries from the query agent. There is a parameter that defines the base speed of the reasoning engine which helps decide how long the broker will take to answer a query. For all of the experiment here, this value is set to 1.0 second per megabyte of advertisements it has in its list of advertisements. This base time is scaled according to the relative complexity of the query and the relative speed of the processor.

The size of the result to the query agent is a function of the number of resource agents that advertised the same domain as contained in the query. For the experiments in this paper, a broker result is set to be 10 kilobytes per agent that matches the query.

Multi-brokering The multi-brokering behaviors in the simulator were tailored to mimic those of the InfoSleuth system. In particular, is can simulate the effects of the various "follow options" and hop counts. For all experiments here, the "all repositories" option is used. For those experiments where the brokers communicate, since the broker network is fully connected, the hop-count was set to 1.

5.2.2 Simulation Results

In this section we present some results from our simulation experiments. First we show the inherent problems with single broker networks and how broker specialization compares to a system where there are simply multiple replicated brokers, but where each broker is a replication of the others; i.e., all brokers maintain complete knowledge of all other agents. The second set of experiments explores the scalability of a multi-brokering system with the behavioral characteristics of the InfoSleuth brokers. Multi-brokering imposes extra overhead due to the communication between the brokers, and we want to ensure that as the number of agents in the system increases that this overhead does not degrade the overall performance. The final set of experiments demonstrates how a multi-brokering system provides robustness. In this experiment, brokers fail using an exponential distribution with varying means.

Each individual experiment was the simulation of 4 hours of system execution time. Because the simulations are based upon pseudo-random inputs, we ran each set of experiments 10 times and averaged the results. This helped ensure that we were not reporting results from a particular anomalous pseudo-random number sequence.

Single versus Multiple Brokers In this experiment was compared three different brokering strategies, using eight different system query loads. Figure 14 shows the results of our experiments where there are 24 resource agents and 20 brokers. By far, the worse performance is in the single broker arrangement. Because there are 24 resource agent advertisements in the single broker's repository, it will take a minimum of 24 seconds to respond to a query. As Figure 14 shows, query rates faster than its processing time completely saturates the broker. In contrast, having multiple brokers, divides the overall system load and thus yields

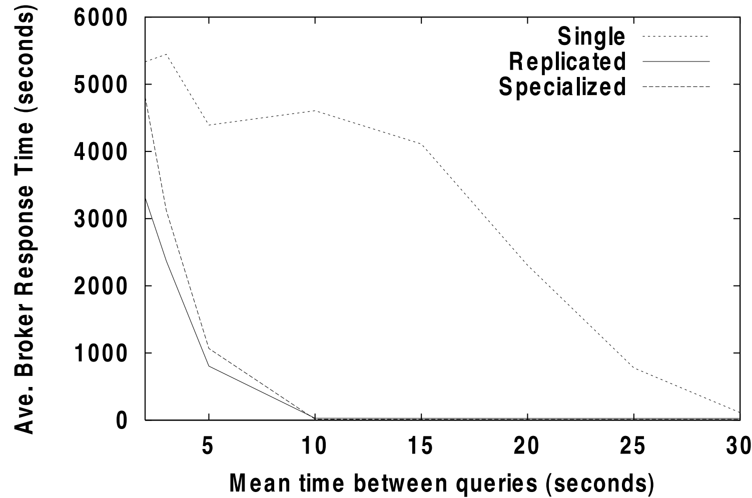

Figure 14: Single brokering versus multiple brokering.

better response times.

Replication versus Specialization The replicated broker case in Figure 14 consists of 20 brokers each with identical copies of all 24 resource agents' advertisements. Thus, they still require a minimum of 24 seconds to process a query. This is an advantage over the single brokering case since it distributes the query load across multiple agents. However, in the specialized broker case, each resource agent has only one broker it has advertised to, so to provide an answer to a query, the brokers must communicate with one another. As Figure 14 shows, for high query frequencies, the extra over-head in broker communication outweighs any advantage gained by parallelizing the computation across multiple brokers. However, the scale of Figure 14 masks the differences between the replicated and specialized cases. Figure 15 shows a close-up of the comparison between the two for mean query intervals of 10 and greater. Here, the gains in computing the answers in parallel across multiple brokers outweighs the extra overhead involved with the broker communication.

Figure 16 shows the same experiments as Figure 15 except that here there are only 4 brokers in the system, though still 24 resource agents. This shows that even with a higher resource-to-broker ratio, specialization of the brokers helps.

Note that "specialized" in these simulation experiments simply means that not all resources have advertised to all brokers. Thus, all brokers must still contact all other brokers to answer a query. In the InfoSleuth system, brokers can advertise their capabilities to other brokers which means that a broker can know in advance which brokers it can immediately rule out from a query. Though we did not conduct any simulation experiments for this case, this sort of specialization would only help to improve the response times provided that the extra time cost in reasoning over broker advertisements was less than the communication time between the brokers.

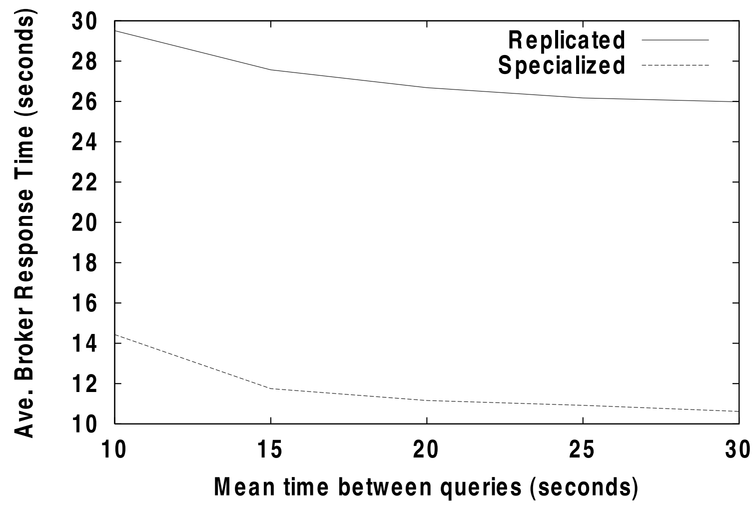

Figure 15: Replicated brokering versus specialized brokering with 20 brokers and 24 resource agents.

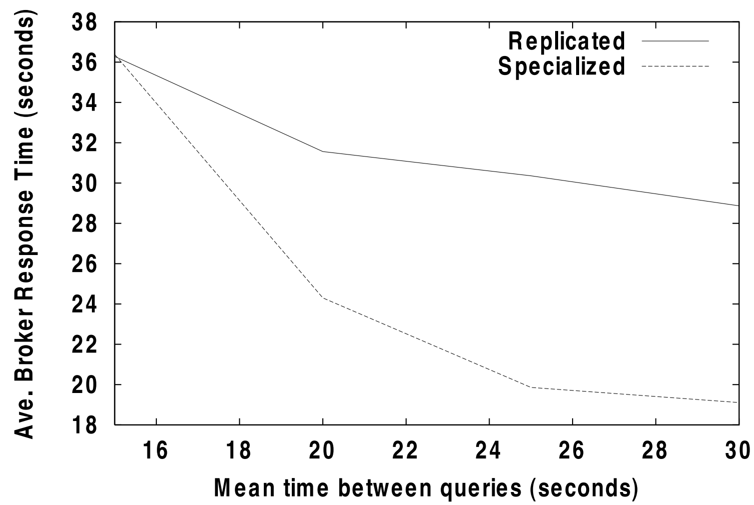

Figure 16: Replicated brokering versus specialized brokering with 4 brokers and 24 resource agents.

Scalability This set of simulation-based experiments varies the number of agents in the system, while maintaining all other system parameters. We simulated systems with the following numbers of resource agents: 12, 24, 48, 72, 96, 120, 144, 192, and 240. Since our focus is on the inter-agent communication overhead, we needed to ensure that the broker agents' local computations remained the same across this range. Thus, we defined that each broker would, on average, have the advertisements for 12 resources. Thus the number of brokers for each of the above resource agent sizes are 1, 2, 4, 6, 8, 10, 12, 16, and 20 respectively.

Each resource agent's advertisement size was set to 1 megabyte and the processor speed and broker reasoning engine speeds were set to require one second of processing time for each megabyte of advertisements. Thus, on average, a broker will need 12 seconds to compute the query answer based on its local advertisements. This presents a theoretical lower bound on the response time in the multi-brokering system. Note that if you simply wanted to have multiple brokers each with identical copies of all advertisements, then the response times would definitely not scale well with the number of resource agents, since it will take each broker one second per resource agent to answer a query or a total of 240 seconds for the largest case we look at here.

The metric of interest here is the average response time to the query agent from the brokers. Unlike the InfoSleuth experiments which had some extra overhead for the processing and rendering of the result, this simulation data is purely the time between when the query is issued to the broker and when the reply is received from the broker.

Figure 17 shows the results of varying the number of agents in the system and for varying query frequencies ("QF" is the mean time between queries.) If the overhead of communication presented an obstacle to scalability, then one would expect the response times to get dramatically worse as the number of agents (both brokers and resources) increased. However, as the data in Figure 17 shows, the response times tend to level off, and certainly do not show any catastrophic behavior.

The results of this experiment shows that multi-brokering systems, despite the extra overhead, do scale up nicely. With a multi-brokering system, the gains achieved by distributing the query processing exceed the overhead incurred as the number of agents in the system increases.

Robustness In this set of experiments, we fixed the number of brokers and resources at 5 and 20 respectively. The query frequency was fixed to have a mean query time of once every 60 seconds to ensure that the system was operating in a range that did not saturate it processing capabilities. The parameters we vary are the mean failure time of the brokers and the amount of redundancy in the number of brokers that each resource agent sends their advertised to. The mean failure rates used are 1000000, 3600, 1800, and 900 seconds. We vary the number of brokers each agent advertises to from 1 to 5.

Table 5 shows the obvious result for the number of replies from the broker to the number of queries asked of the broker (expressed as a percentage). Naturally, as the failure frequency goes up, the more likely we are to contact a broker that does not respond. Aside from the variation due to the random nature of

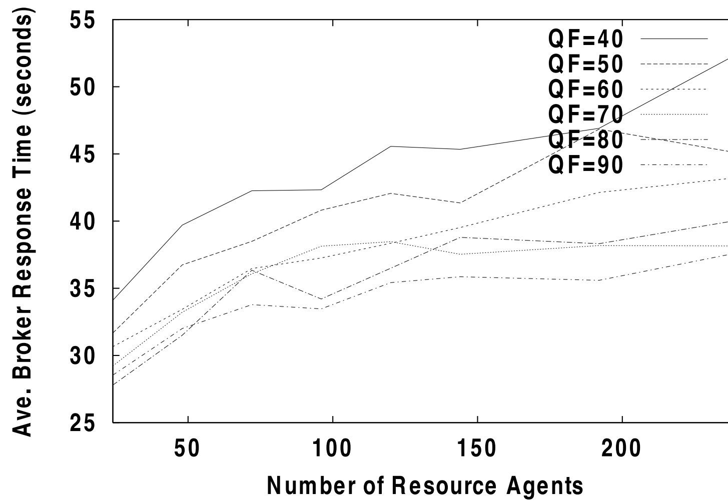

Figure 17: Scalability of broker specialization across a range of number of resource agents and system query frequencies (QF).

Failure Mean (secs.)	**Advertisement Redundancy (Number of Brokers)**				
	1	2	3	4	5
1000000	99.56%	97.37%	100.00%	99.14%	100.00%
3600	77.64%	70.71%	69.87%	61.26%	63.45%
1800	37.50%	44.40%	46.69%	44.64%	59.41%
900	34.05%	26.47%	17.87%	22.90%	16.79%

Table 5: Percentage of queries that brokers reply to.

Failure Mean (secs.)	Advertisement Redundancy (Number of Brokers)				
	1	2	3	4	5
1000000	100.00%	100.00%	100.00%	100.00%	100.00%
3600	75.00%	92.90%	92.22%	97.42%	100.00%
1800	75.86%	85.44%	95.58%	100.00%	100.00%
900	20.25%	76.19%	69.05%	86.67%	100.00%

Table 6: Robustness experiments: percentage of queries successfully answered.

the experiments, these percentages should be independent of the redundancy of the advertisements, since it only measures whether a broker replies not whether it actually located an agent to satisfy the query.

The robustness of the system is evaluated by looking at only those queries for which the broker responded. In these cases we want look at the quality of the broker's response. In this particular experiment each resource has a unique data domain, so there is exactly one agent that should match each query. The quantity of interest is the number of resources queried versus the number of broker replies that are received, since this is directly correlated to the number of times the proper resource agent was located by the broker network. As seen in the first row of Table 6, when the brokers are very unlikely to fail, the number of resource agents queried will be identical to the number of broker replies (i.e., 100% of the queries answered had found the matching resource agent.)

The last column shows that with complete redundancy, you can always find the agent if you get a reply at all. In this case, the brokers have no real reason to communicate since all brokers know about all resources. However, for the other cases where inter-broker communication is needed to answer the queries, you can see the definite trend that the more redundancy there is, the more robust the system is to failures.

6 Related Work

There are several research projects and systems that address the issue of integrating heterogeneous systems. These systems rely on some form of brokering or mediation to achieve semantic integration. We evaluate two issues when comparing brokering in InfoSleuth to brokering in other systems – syntactic vs. semantic brokering and single vs. multibroker architectures.

One area where brokering has been used extensively is within distributed object systems such as CORBA [9, 20]. CORBA provides a framework in which distributed, heterogeneous objects can interact. CORBA objects describe their interfaces to a broker (ORB), using a common interface description language called IDL. When a CORBA object makes a call to another object, it determines the signature of the method being invoked, and places a request to the ORB to locate an object with that signature. The ORB looks through its list of interface descriptions (and possibly those of other related ORBs), and matches the signature to some method in some object's IDL description, and returns the matched object. The requesting object can then invoke the method on the matched object.

DISCO [24] uses CORBA brokers to do their brokering. The CORBA Trading Object Service [20]

provides mechanisms for incorporating some semantic brokering, however, they have not actually incorporated any reasoning. In fact the level of semantic brokering is equivalent to the look up of yellow pages in a telephone book. The knowledge that can be expressed in a yellow page is a structured list of properties. It is not possible to describe constraints such as range of data involved, relationships between input and output, subsumption relationships between concepts, and correctness and completeness of data.

CORBA traders also implement multibrokering through internetworking among traders; we borrowed some of their ideas on inter-broker search policies. CORBA's traders are passive due to the lack of reasoning power. This means that the federation of traders is always formed statically with a fixed topology. Our peer-to-peer multibroker architecture allows broker consortium to evolve or self-organize depending on the goal/objective of each individual broker. Only recently have commercial implementations of CORBA traders been available (e.g., [10]). No information is available regarding the scalability and robustness of any of the CORBA traders.

Different agent communication languages [23, 6] have developed special messages to send to a broker agent, with a basic assumption of some underlying brokering process. KQML defines advertise messages, as well as broker and recruit messages, which allow an agent to advertise its services and ask a broker about other services. In these messages, the service is expected to be described in terms of a second KQML message, possibly with wild cards. A match between a request and an agent takes place when the agent's advertisement unifies with the performative specified in the broker or recruit message. Again, this implements a syntactic match, as the primary concern is matching the structure of the agent's interface.

Syntactic brokering uses the structure or format of a task specification to match a requester with a service provider. This involves using mainly syntactic properties such as object or method interfaces or query/scripting languages to decide which service providers to recommend. For example, a process QA may advertise that it takes its input according to SQL 2.0 syntax. Any request for SQL 2.0 query processing could then be directed to QA. Syntactic brokering functions are sometimes also incorporated into commercial agent frameworks such as Zeus [19]. One basic, but problematic assumption that syntactic brokering systems such as the CORBA ORB make is that the definition of the procedure or method interface uniquely defines its semantics. That is, ORBs do not check to see if the local method definition that conforms to the interface actually does what the caller intended. Thus, semantic mismatches are possible even when there is a syntactic match between a requester and a provider.

Several information retrieval systems use semantic brokering with respect to information sources. These systems include SIMS [1], TSIMMIS [17], InfoMaster [8] and Information Manifold [14, 15]. They all evolved from research in multidatabases where a canonical model (global ontology) is used. The way these systems work is to define a common vocabulary, or ontology, to define the objects in their information domain. Individual information sources that contain these objects then describe constraints on the objects that they can provide, in terms of this common vocabulary. The broker then uses these constraints to determine how to process queries from users that involve one or more of these resources. The capabilities

of these systems are similar to InfoSleuth's in that they reason over the information content of the agents and constraints over that content; however, their idea of a service ontology only encompasses information and not any other agent capabilities. Thus, these systems implicitly do syntactic brokering when matching resources, as they expect that each resource supports a single underlying query language.

The SHADE project [16, 12, 13] at Lockheed Palo Alto Research Labs extended the notion of syntactic brokering by including information about services represented using KIF, a knowledge interchange method that represents first-order logic expressions. Queries concerning agents were matched with these advertisements using unification. Additionally, SHADE provides some facility to broker over constraints on the values of the data, similar to the semantic brokering over data that we have done in InfoSleuth. However, we have not found a clear description of how this facility works. Since SHADE relies on a shared ontology to represent data, Kuokka and Harada also propose a companion matchmaker named COINS that uses TF/IDF (term frequency / inverse document frequency) filtering to match document characteristics to free text [13].

The LARKS matchmaking system [22] in RETSINA [21, 4, 5], has attempted to address the issue of semantic brokering by providing input-output descriptors and term frequency measures to determine whether or not a semantic match occurs between a requested service and a service provider. RETSINA matchmakers describe the semantics of their offered services both in terms of signatures (inputs and outputs), but also in terms of the relationships between the inputs and the outputs. They also use TF/IDF techniques to categorize the semantic relevance of a query to an advertisement. The term frequency measures are similar to those used in COINS. The matching process in RESTINA is structured such that users can select a trade-off between performance verses quality of matching.

In addition, there are a few general papers of interest on brokering and agent architectures. These include a second developed agent framework similar to InfoSleuth and its approach to brokering (though different in other aspects) in [3] and some general discussions in information agents [13, 11].

7 Conclusions

The brokering function matches specific requests for services with providers that can satisfy those requests. Syntactic brokering does this match based on purely syntactic characteristics of the requested service such as method signatures, query languages or input forms. Semantic brokering takes into account the nature and characteristics of the requested service - what functions do you want to perform and what data do you want to access. A good brokering system, such as InfoSleuth's, should implement both syntactic and semantic brokering.

We described our approach to defining a service ontology - a shared vocabulary that agents can use to describe themselves to the broker. Advertising and querying is done in terms of logical expressions and constraint programming. This enables the broker to both access stored information gleaned from agent advertisements and reason over that information when determining which agents provide a set of requested services.

A single broker can accomplish much in the way of recommending resources and assisting in maintaining a dynamic set of distributed computing and information resources. However, a single broker architecture presents a barrier to scalability and robust operation. We presented a multibrokering peer-to-peer architecture with brokers belonging to different consortia. Brokers maintain up-to-date information about other brokers as well as other agents, couched in terms of a multibroker service ontology. We described how robust multibrokering can be implemented, especially in term of how broker discovers other brokers and how it implements inter-broker searches. We showed empirically the feasibility of multibrokering, and explored its effectiveness. Our preliminary experiments were encouraging in that they showed the feasibility of and hinted at the scalability of the multibrokering system.

Because it was impractical to study multibrokering in very large agent-based systems, we made use of an agent simulator developed here at MCC to analyze the robustness and scalability of our multibrokering approach as the agent community grows in size. We analyzed the scaling behavior of our specialized brokering approach and showed its superior scalability with respect to replicated brokering systems. We experimented with various topologies and connectivity properties of the brokers, enabling us to determine in the future what the most efficient tradeoff is between connectivity and brokering speed and robustness. However, we do recognize the drawbacks of a simulation-based approach to such experimentation as well, including the need to validate the behavior of the simulator against the real-life system (whether implemented or not), and the accuracy of the settings of the myriad of simulation parameters such that the model faithfully represents the real-life components. Our confidence in our simulator and our configuration settings should continue to grow as we gain more simulation experience.

Acknowledgments

The authors would like to acknowledge the members and sponsors of the InfoSleuth group for their help in the brokering ideas presented in this paper, including Amy Unruh, Gale Martin, Ray Shea and Marek Rusinkiewicz for their helpful comments on the text.

References

[1] Y. Arens, C.A. Knoblock, and W. Shen. Query reformulation for dynamic information integration. *Journal of Intelligent Information Systems*, 6(2), 1996.

[2] R. Bayardo and et.al. InfoSleuth: Agent-based semantic integration of information in open and dynamic environments. In *Proc. ACM SIGMOD Int'l Conference on Management of Data*, 1997.

[3] D. Moran D. L. Martin, H. Oohama and A. Cheyer. Information brokering in an agent architecture. In *Proc. Int'l Conference on the Practical Application of Intelligent Agents and Multi-agent Technology*, 1997.

[4] K. Decker and K.P. Sycara. Intelligent adaptive information agents. *Journal of Intelligent Information Systems*, 9(3), 1997.

[5] K. Decker, M. Williamson, and K. Sycara. Matchmaking and Brokering. In *Proc. Int'l Conference on Multi-Agent Systems*, 1996.

[6] FIPA. http://www.fipa.org.

[7] Jerry Fowler, Marian Nodine, Brad Perry, and Bruce Bargmeyer. Agent-based semantic interoperability in infosleuth. *Sigmod Record*, 28(1), March 1999.

[8] M.R. Genesereth, A. Keller, and O.M. Duschka. Infomaster: An Information Integration System. In *Proc. ACM SIGMOD Int'l Conference on Management of Data*, 1997.

[9] Object Management Group and X/Open. *The Common Object Request Broker: Architecture and Specification, Revision 1.1*. John Wiley and Sons, 1992.

[10] IONA. White paper on orbix trader. Technical report, IONA Technologies, http://www.iona.com/, 1999.

[11] L. Kerschberg. The role of intelligent software agents in advanced information systems. In *Proc. British National Conference on Databases*, 1997.

[12] D. Kuokka and L. Harada. On using KQML for matchmaking. In *ICMAS*, pages 239–254, 1995.

[13] D. Kuokka and L. Harada. Integrating information via matchmaking. *Journal of Intelligent Information Systems*, 6(2), 1996.

[14] A. Levy, D. Srivastava, and T. Kirk. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems*, 5(2), 1995.

[15] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. Int'l Conference on Very Large Data Bases*, 1996.

[16] McGuire, Kuokka, Weber, Tenenbaum, Gruber, and Olsen. SHADE: Technology for knowledge-based collaborative engineering. *Journal of Concurrent Engineering: Research and Applications*, 1(3), 1993.

[17] Garcia Molina and et.al. The TSIMMIS approach to mediation: Data models and languages. *Journal of Intelligent Information Systems*, 8(2), 1997.

[18] Marian Nodine, Jerry Fowler, and Brad Perry. Active information gathering in infosleuth. In *Proceedings of the International Symposium on Cooperative Database Systems for Advanced Applications*, 1999.

[19] Hyacinth Nwana, Divine Ndumu, Lyndon Lee, and Jaron Collis. ZEUS: A tool-kit for building distributed multi-agent systems. *Applied Artifical Intelligence Journal*, 13(1):129–186, 1999.

[20] OMG. OMG trading object service specification. Technical Report 97-12-02, Object Management Group, http://www.omg.org/corba, 1997.

[21] Katia Sycara, Matthias Klush, Seth Widoff, and Jianguo Lu. Dynamic service matchmaking among agents in open information environments. *SIGMOD Record*, 28(1), March 1999.

[22] Katia Sycara, Jianguo Lu, Matthias Klusch, and Seth Widoff. Matchmaking among heterogeneous agents on the internet. In *Proceedings of the AAAI Spring Symposium on Intelligent Agents in Cyberspace*, 1999.

[23] Y. Labrou T. Finin and J. Mayfield. KQML as an agent communication language. In J.M. Bradshaw, editor, *Software Agents*. AAAI Press, 1997.

[24] A. Tomasic, L. Raschid, and P. Valduriez. Scaling heterogeneous databases and the design of DISCO. In *Proc. Int'l Conference of Distributed Computing Systems*, 1996.

[25] C. Zaniolo. The logical data language (LDL): An integrated approach to logic and databases. Technical Report STP-LD-328-91, MCC, 1991.