Capability-based Agent Matchmaking*

Anthony Cassandra, Damith Chandrasekara and **Marian Nodine**
Microelectronics and Computer Technology Corporation (MCC)
Austin, TX 78759
{arc, damith, nodine}@mcc.com

Abstract

In an agent-based system, where different agents form dynamic associations for the purposes of collaborative processing, there is a basic need for agents to be able to describe themselves to other agents. This allows agents to locate other agents that can provide them with needed capabilities to accomplish specific tasks at a given time. An agent may describe itself in terms of its interfaces, the services it can provide, the tasks it can accomplish, the data it can provide, etc. Practical, implemented systems that perform some of this functionality include those based on CORBA and other distributed object services, but their representation is too limited to function effectively in an agent system with many agents that serve similar functions. At the other end of the spectrum are research systems that allow an agent to specify its capabilities in terms of reasonably complete formal logic specifications; however these descriptions have the potential to be very unwieldy and error-prone and may require too much computation for it to be practical in a large, diverse agent system.

In this paper we present an approach to agent capability description and capability matching that is expressive enough to capture complicated agent functionality, yet simple enough to be scalable to large and diverse agent systems. This approach relies on shared, focused ontologies that provide a common vocabulary for describing information and services. An agent can then advertise itself in terms of the focused ontologies, and query about other agents using these same ontologies. We show by example how specific, implemented agents describe their capabilities in a way that distinguishes them from other agents. We contrast our approach to the existing ones and point out where it is superior.

Introduction

Our experiences with the InfoSleuth[1] (Bayardo *et al.* 1997; Nodine & Unruh 1997; Nodine, Fowler, & Perry 1999) agent system led to the discovery of many weaknesses in existing matchmaking systems. The InfoSleuth project began matching mostly over data and agent types, with limited use of the capabilities of the underlying reasoning engine. As the basic framework of the system matured, new agent types were developed, and the need for more complete reasoning over the *capabilities* of agents became more apparent.

In examining other systems that were either agent-based or used service matchmaking, we found no real system that was suitable for describing capabilities in enough detail, while being abstract enough that the actual capabilities themselves were not bogged down by being unnecessarily descriptive.

For example, distributed object systems such as CORBA (OMG & X/Open 1992) provide the ability to describe services in terms of interfaces, effectively reducing the semantic expressiveness of the capability descriptions to a series of method call signatures. This was inadequate for some InfoSleuth applications because we could come up with many instances where two or more agents presented the same interface, but each agent did something different when the interface was "called". Also, systems such as CORBA grew out of the distributed object world, thus its use in an agent-based system will restrict the agents' interactions to remote procedure calls. Interactions between agents often take a more complex form encompassing a series of method exchanges.

Within the agent community, agent communication languages such as KQML (Labrou 1996) attempt to define how an agent advertisement should look. For example, KQML defines an *advertise* performative that allows an agent to advertise its capabilities in terms of the KQML performatives it is able to accept. Thus, an *advertise* performative has (one or more) nested performative(s) that define, in effect, the agent's message interface. One unfortunate problem with this approach is that, given that agent interactions do take a more complex form that encompasses a series of method exchanges (*conversations*), an agent usually does not want to advertise incoming messages that are interpreted in the context of already ongoing conversations. A second, more fundamental problem is that this approach also

*Also appears as MCC Technical Report SRI-072-99.

[1]InfoSleuth™ is a trademark of Microelectronics and Computer Technology Corporation.

reduces the ability to describe agent semantics to the ability to describe the interface to that agent, and thus has the same interface-based issues that were present in the distributed object systems.

These matchmaking issues have been discussed and worked on within the context of several agent systems. Of these systems, perhaps the most thought-through approach is that of the LARKS (Sycara *et al.* 1999) matchmaker. This matchmaker uses descriptions of services that are based on formal logic specification techniques. This approach, while general, leads to overly detailed advertisements. This calls into question how well it will scale in increasingly diverse agent systems.

Our experience with the InfoSleuth system led us to the conclusion that we needed to represent agent capabilities at multiple levels, including:

- The agent conversations that are used to communicate about the service, in terms of a conversation ontology,
- The interface to the service, in terms of a language ontology,
- The semantics of what the service does, in terms of a service ontology,
- The information a service operates over, in terms of a domain ontology.

Note that this approach relies on the availability of multiple, focused ontologies that describe specific conversations, languages, services and domains. These ontologies codify different aspects of specific services into a set of agreed-upon terms that can be shared among agents (and users). In some sense, this is a generalization of the original InfoSleuth approach, which reasoned over focused domain ontologies. However, the methodology for couching the ontologies and advertisements in the form we describe in this paper is a significant step beyond the representation of information this way. Additionally, an important property of our framework is its independence from the ontologies and concepts being reasoned over, which allows for future extensions.

In this paper we present our framework for representing these non-trivial agent capabilities as well as how to query over these capabilities. We also discuss how this leads to a very general reasoning mechanism which is simple both conceptually and computationally. We begin by discussing our somewhat pragmatic view of ontologies, which is followed by the description of our new framework. We then present some example applications for which existing systems are inadequate and show how they can be easily represented in our framework. We conclude with discussion of related work.

Ontologies

For an agent-based system to be functional, it is not enough to define a syntax of communication or even a semantics of the types of conversations that occur between agents. The information the agents exchange must also be semantically consistent. An ontology provides a convenient mechanism for agents to express and exchange information about their knowledge and capabilities. The ontology must have enough concepts and give enough detail to allow the agents to provide useful information, but it cannot provide too much semantic detail, since this will quickly become overwhelming in terms of the labor required to create the representations and the computational resources required to communicate and reason about the representations.

```
ontology service
   class subscription
      slot computation in {direct, indirect}
      slot adaptability in {dynamic, static}
      slot response-type in {bulk, incremental}

   class query
      slot computation in {direct, indirect}
      slot response-type in {bulk, first-solution}

   class data-response
      slot delivery in {inline, http}
      slot language
            in {tuple-format, xml-annotated}
      slot annotations in {source-tagging, tracing}
```

Table 1: Simple service ontology example.

With the current state-of-the-art in knowledge representation, it is infeasible to have a single ontology that captures every concept any agent will ever need. Instead, we use many small, domain-specific ontologies where any particular agent can state exactly which of these ontologies it supports or even which subset of the ontology it can support. Using these ontological pieces we get the flexibility of applying our framework to any domain, while keeping the complexity of the knowledge representation task manageable.

For our purposes, an ontology consists of an ontology name and contains a set of classes and their slots. Each class has a name and classes can be arranged hierarchically using class-subclass relationships. Each slot within a class has a name and a type and can also define the set of possible values allowed for the slot.

As an example, Table 1 shows a simple ontology of agent services which contains three classes and no subclass relationships. The subscription class has slots that define characteristics of an agent which supports subscriptions. The first slot, computation, specifies whether the data that is being subscribed to comes directly from that agent or if it is obtained through another agent; the set of possible values for that slot are direct or indirect. Similarly, the query and data-response classes have slots which define sets of possible values and which represent qualities of an agent's query capabilities and characteristics of its responses.

```
ontology service
   class subscription
      slot computation in {direct}
      slot adaptability
      slot response-type in {incremental}
```

Table 2: Ontology fragment example.

Ontology Fragments

Although an ontology will contain all the concepts for a particular domain, no agent is required to support the entire domain and often agents will only support or understand some subset of the ontology. Subsets of ontologies that are specific to an agent we term *ontology fragments*.

Like a full ontology definition, an ontology fragment consists of the ontology name and contains a set of classes where each class can contain a set of slots. The semantic interpretation of the combination of classes within a fragment and slots within a class will depend on the specific context where the fragment is used (in a query, or in an advertisement), and is discussed in subsequent sections.

The set of possible values that can be specified in the ontology definition serve to define the domain of possible values. Agents are able to further constrain[2] a slot's possible values by including constraints on the slots in an ontology fragment. Table 2 shows an ontology fragment for an agent which can only support subscriptions, gets its data directly and provides only incremental results. Note that the slot adaptability is unconstrained, which means it supports all the possible values for that slot.

Framework

The basic requirement that led to the framework outlined in this paper was the need to perform matchmaking over a richer set of agent capabilities. Our framework provides a mechanism for agents to advertise their abilities and characteristics to a matchmaking agent as well as providing mechanisms to query the matchmaker for sets of agents meeting specific criteria.

Agent Capability

Our framework is built around the concept of an agent's capability. Although individual domain-specific ontologies are advantageous for simplifying the representation problem, these concepts from separate domains interact, and it is important to capture these interactions to properly describe an agent's capability. For example, consider an agent that offers subscriptions over concepts from a particular information domain ontology. An agent would like to describe its capabilities not only in terms of the concepts it can handle in the information domain, but also in the form of the types of subscriptions it can handle. In this example, we could put all the subscription concepts into the information domain ontology, but this would not help another agent that wants to do something similar, but which uses a different information domain. Our solution keeps the subscription concepts in a separate ontology and allows other ontologies to be combined with it. Thus, in this case, the capability of the agent would be the conjunction of a fragment of the service ontology and a fragment of an information domain ontology.

```
capability example-capability
   ontology service
      class subscription
         slot computation in {direct}
         slot adaptability
         slot response-type in {incremental}
   ontology environment
      class site
         slot state in {Texas}
         slot contaminant
         slot remedial-response
         slot report
   ontology conversation
      class conversation
         slot language in {kqml}
```

Table 3: An example of an agent capability.

The generalization of this is to allow an agent to describe itself using a conjunction of any number of separate ontology fragments. This is our precise definition of an *agent capability*. We recognize that, although we have generally defined the inter-ontology relationships to be conjunctive, there is usually a deeper semantic relationship between the ontologies. Furthermore, different advertised ontology combinations can have differing semantics for how the various ontologies are related. We have chosen not to explicitly represent this relationship, since this would add a significant amount of complexity to the entire framework. As with the concepts in the ontologies themselves, we assume these deeper semantics are understood by the agents that use them.

Table 3 shows an example of an agent capability consisting of three ontology fragments[3]. The service ontology fragment defines the characteristics of subscriptions the agent can handle. The environment fragment defines the available environmental information, while the conversation fragment defines the conversational support of the agent.

Note that our decomposition of capabilities into individual ontology fragments is a very general structure which can provide us with a uniform view of informa-

[2] Although we will only present constraints on elements of a set in this paper, more general constraints are supported, e.g., numeric intervals.

[3] This example is derived from the ontologies used in the EDEN Project (Fowler *et al.* 1999).

tion and services. This makes the representation and reasoning over the representation extremely general.

Agent Advertisement

We have defined an agent's capability as a conjunction of ontology fragments and have defined an ontology fragment as the subset of the ontology supported by the agent. An *agent advertisement* is what an agent uses to declare its abilities to a matchmaking agent and it consists of one or more agent capabilities. When more than one agent capability is advertised, the semantic interpretation is similar to a disjunction: the agent is capable of any one of the advertised capabilities, and each capability is considered independently.

Within the ontology fragments of a capability will appear one or more classes from the ontology. The semantic interpretation is that the agent supports any subset of those advertised classes. An advertised class can contain one or more advertised slots which is interpreted semantically as the agent supporting any subset of these slots. Finally, there can be constraints on an advertised slot, where the interpretation is a disjunction or union of all the constraints. The slot is considered to be unconstrained when there are no advertised constraints.

Queries and Reasoning

The ability for an agent to advertise its capabilities is only useful if there are agents that are interested in asking questions about other agents' capabilities and a reasoning component that can match queries to advertisements. A query is structurally very similar to a capability that exists in an advertisement. A query is a conjunction of ontology fragments, where each fragment can specify one or more classes, slots from each class and constraints on the slots. However, since this appears in the context of asking a question, it will have slightly different semantics in the reasoning engine.

Matching Queries with Advertisements At the top-most level, the set of ontology fragments in a query is interpreted as a conjunction. An advertised capability matches this query if it has advertised at least all those ontology fragments and each of the individual queried fragments match the advertised fragments, where the matching criteria is discussed below. Note that the advertisement can contain other ontology fragments that do not appear in the query, but all the ones that do appear in the query must have a matching fragment in a particular advertised capability.

Matching Ontology Fragments First, and most obviously, the names of the ontologies these fragments are associated with must be the same. Next, the advertised classes must match the queried classes, but here the query has the option of specifying the set of classes as either a conjunction or a disjunction. For a conjunction of queried classes, the advertised ontology fragment must contain at least all of the queried classes and the classes must match (the matching criteria is defined below). Note that the advertised ontology fragment may have more classes than are queried. For a disjunction of classes, it is enough for any one of the queried classes to match one of the advertised classes.

Matching Ontology Classes A queried class can contain one or more queried slots. As with classes within an ontology fragment, a queried class has the option of querying on a conjunction or disjunction of slots. A conjunction will mean that the advertised class must support and match at least all those queried slots. A disjunction means that at least one of the queried slots must match in the advertised class.

With the ontology fragments, the first criteria for matching required the ontologies to have the same name. However, for the matching between classes, the queried class name and the advertised class name may be different. The reason for this is that the ontologies are actually a hierarchy of class-subclass relationships. Thus, if class radioactive-site is a subclass of site then agents that support class radioactive-site might be useful for an agent that asks about information from class site. Thus, the reasoning algorithm that matches queries to advertisements reasons over the ontology class hierarchy.

In a queried capability, every class specified has a taxonomic inference depth which is used by the reasoning engine to help determine which advertised classes match the queried class. The inferencing is always from the queried class to its subclasses, since matching on super-classes could result in agents whose data does not answer the query being asked. The inference depth represents the maximum number of class-subclass relationships to traverse while checking if an advertised class is a subclass of the queried class. A depth of 0 (zero) means that the advertised class name must be identical to the queried class name. An infinite depth means that there is no limit, while all other values represent some finite limit.

Matching Ontology Slots For a slot to match it must have the same name and there must be a non-empty intersection between the queried constraints and the advertised constraints. Both the advertised and queried constraints represent a subset of all the possible values a slot can have, so a slot that matches with a non-empty intersection semantically means that it is possible that it will have relevant information or services.

Retrieving Information from a Query While being able to match advertisements is a necessary function, there is more to the matching than simply saying "match" or "no match". The agent that asks the query will likely need to have more information about the matching process. For example, an agent that queries for all matches to an ontology fragment with any one

of three classes might need to know exactly which of the three classes were present in the matched advertisement.

One possible solution to making sure an agent gets all the information about the match is to simply return all the matching advertisement capabilities. While this is the cleanest solution conceptually, in a real agent system this tends to pose problems. One problem concerns the volume of information that has to be exchanged as the result of a query. Also, if the querying agent actually needs information about the matching process, by returning the whole advertisement the querying agent will be required to duplicate many of the reasoning steps. As a consequence, our framework has allowed parts of the queries to be annotated with attributes that explicitly define the type and amount of information to return in a query result. Space has precluded us from presenting these details in the paper, but the exact form of these attributes is not as crucial as recognizing the need for returning information about the matching process and controlling its size.

Application and Example

Now that we have presented the basic framework for our matchmaking system, we present an example of how it is used within the context of a simplistic, but real application. We note that this example shows one of a myriad of matchmaking decisions that have been made all along in the InfoSleuth system, but it illustrates some of the salient concepts that have made matchmaking so difficult all along.

Advertising In this application, we begin with an example of a resource agent named tx-env-resource that provides environmental contamination information related to contaminated sites in Texas. The example advertisement is shown in Table 4.

This agent advertises some generic capabilities – the ability to be monitored, the ability to answer pings – available in all agents. These are named the monitor-subscribe-capability and the ping-capability, respectively. Next, the agent advertises two capabilities related to how its own information can be accessed – one related to subscriptions (the env-subscription-capability) and one related to queries (the env-query-capability). This advertisement is interpreted to mean that the agent tx-env-resource has several capabilities, any one of which can match a given query.

Only the env-subscription-capability is described entirely in Table 4. This capability consists of a set of related ontology fragments. The fragment of the conversation ontology specifies that the agent accepts conversations of the form used by subscriptions, using the language KQML. The fragment of the sql ontology specifies that the query should be specified in SQL syntax. The fragment of the environment ontology specifies the classes and slots that have data in the agent. The fragment of the service ontology specifies that the service

```
advertisement
  capability monitor-subscribe-capability
    ...
  capability ping-capability
    ...
  capability env-subscription-capability
    ontology conversation
      class conversation
        slot conversation-type in {subscribe}
        slot language in {kqml}
    ontology sql
      ...
    ontology environment
      class site
        slot contaminant
        slot remedial-response
        slot city
        slot state in {Texas}
      class contaminant
        ...
      class remedial-response
        ...
    ontology services
      class data-response
        slot language in {tuple-format}
        slot delivery in {http, inline}
        slot annotations in {source-tagging}
      class subscription
        slot computation in {direct}
  capability env-query-capability
    ...
```

Table 4: Advertisement for the resource agent tx-env-resource.

is direct (accessed locally within the agent), as well as other service properties.

Now, assume that we have a large collection of environmental resources, all of which contain related information concerning contaminants and sites in various locations throughout the world. Each resource has its own resource agent, maintained locally, that makes the information available in the form of the environment ontology. Users may wish to access a combined view of the information using the subscription paradigm. Therefore, we can add to the agent system one or more agents that takes the same kind of subscription request and subscribes in turn to several resources, merging their results into a unified view of the information. Its advertisement would be as shown in Table 5.

This agent advertises the same generic capabilities, monitor-subscribe-capability and the ping-capability, that the resource agent advertised. The first additional capability, the subscription-capability, states that it can respond to subscribe conversations in KQML where the information to be subscribed to is specified as any SQL query over any other ontology, and the response is de-

```
advertisement
   capability monitor-subscribe-capability
      ...
   capability ping-capability
      ...
   capability subscription-capability
      ontology conversation
         class conversation
            slot conversation-type in {subscribe}
            slot language in {kqml}
      ontology sql
         ...
      ontology services
         class data-response
            slot language in {tuple-format}
            slot delivery in {http, inline}
         class subscription
            slot computation in {indirect}
            slot adaptability in {dynamic}
      ontology data-independent
         ...
```

Table 5: Advertisement for the subscription agent my-subagent.

livered either inline in the KQML message or off-line using http. The indirect specification in the services indicates that the actual information is retrieved from other agents, and the dynamic adaptability indicates that its answers can take into account new information from new resource agents. The last ontology fragment, data-independent, captures the concept that this agent's services are not dependent on any particular data domain ontology; i.e., it is a general subscription agent that can provide subscriptions over any data domain.

Querying Within the context of an agent system, visualize a user agent that supports the users in inserting subscriptions into the agent system and processing the results. Suppose that the user presents a subscription request:

SELECT contaminant
FROM site
WHERE site is in Austin, TX.

The user agent begins by looking for some agent that can do subscriptions, and can process the request. The query it presents to the matchmaker is shown in Table 6.

This query will match to the resource agent tx-env-resource, but not to the subscription agent my-subagent. It also will match to any other direct information resource agent that has some compatible capability advertised, so the agent that sent the query may receive a list of multiple agents in response to its query.

Suppose that, in the case where multiple resources are available to answer the query, there are more matching agents that the user agent wants to deal with. The

```
query
   capability subscribe-to-capability
      ontology conversation
         class conversation
            slot conversation-type in {subscribe}
            slot language in {kqml}
      ontology sql
         ...
      ontology environment
         class site
            slot contaminant
            slot city in {Austin}
            slot state in {Texas}
      ontology services
         class data-response
            slot language in {tuple-format}
            slot delivery in {inline}
         class subscription
            slot computation in {direct}
```

Table 6: Query that looks for agents that have information about new contaminants at sites in Austin, TX as they are reported.

user agent then decides to use a general subscription agent to return an amalgam of information from multiple resources as a single subscription; it is looking for a data independent subscription resource that will collect and merge the requested information. In this case, the query is nearly the same, except the computation mode is indirect and the environment ontology is replaced with the data-independent ontology. With this new query, the matchmaker would match the request to my-subagent. When the agent my-subagent receives the subscription request, it in turn asks the matchmaker for all direct suppliers of the requested information, using the same subscription mechanism. It subscribes to all agents that are returned from the matchmaker, assembling the results together into a single result stream. When new resources come online that have relevant information, my-subagent subscribes to those agents as well, merging their results into the same result stream.

Discussion This is an example of one of many real problems we have encountered trying to do matchmaking among agents in real, large-scale applications. Other issues that have motivated us towards this capability-based approach for advertising and querying include:

- The need to provide a finer level of resolution on the aspects of standard query languages that a particular agent actually supports.
- The need to specify detailed aspects of fairly complex, data analysis services.
- The need to describe the interface to an agent at multiple levels – one at the level of the agent communi-

cation language and one at the level of the content language.

- The need for agents to advertise long-term tasks such as data analysis and document extraction tasks, thereby providing the information necessary to multiplex similar requests into single processing tasks.
- The need for individual agents to provide different classes of people various types of access to the agents' information and services through the use of different ontologies over the same domain.

Related Work

Syntactic matchmaking uses the structure or format of a task specification to match a requester with a service provider. Approaches to syntactic matchmaking include interface-based matchmaking and syntactic unification. This type of matchmaking involves using mainly syntactic properties such as object or method interfaces or query/scripting languages to decide which service providers to recommend. The weakness of syntactic matching is that it relies on the assumption that the scope and operation of a given service can be adequately described in terms of the interface to that service - an assumption that does not always hold within complex, diverse agent-based systems.

Interface-based matchmaking has been used extensively is within distributed object systems such as those based on CORBA (OMG & X/Open 1992; OMG 1997). CORBA uses 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 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.

Different agent communication languages (Labrou 1996; FIPA 1997) have developed special messages to send to a facilitator agent such as a matchmaker. KQML defines *advertise*, *broker* and *recommend* messages, which allow an agent to advertise its services, and ask about other agents' services. In these messages, the service is 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 broker or recruit performative. Again, this implements a syntactic match, as the primary concern is matching the structure of the agent's interface.

Some agent systems have pushed their matchmaking capabilities beyond those of syntactic matchmaking. For example, several information retrieval systems reason over the content of different information sources. These systems include SIMS (Arens, Knoblock, & Shen 1996), TSIMMIS (Molina *et al.* 1997), InfoMaster (Genessereth, Keller, & Duschka 1997) and Information Manifold (Levy, Srivastava, & Kirk 1995; Levy, Rajaraman, & Ordille 1996). These systems define a common ontology for describing 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 matchmaker then uses these constraints to determine how to process queries from users that involve one or more of these resources. The matchmaking capabilities of these systems do not extend beyond the use of information content.

The SHADE project (McGuire *et al.* 1993; Kuokka & Harada 1995) represents services using KIF, a knowledge interchange method that employs first-order logic expressions. Queries concerning agents are matched using unification. Additionally, SHADE provides some facility to match over constraints on the values of the data, similar to the early matchmaking in the InfoSleuth system. Since SHADE relies on a shared ontology to represent data, Kuokka and Harada also propose a companion matchmaker named COINS that uses TF/IDF filtering to match document characteristics to free text (Kuokka & Harada 1996). In this paper, we do not address the use of such filtering, because using filtering for matchmaking is orthogonal to our use of ontologies.

The LARKS matchmaking system (Sycara *et al.* 1999) was developed within the context of the RETSINA (Decker & Sycara 1997) project. 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. LARKS implements a multilevel matchmaker that reasons over both.

In some sense, LARKS representation and reasoning provides too much generality, since it is based on a first-order logic. This can present a formidable knowledge representation task. Also, reasoning over this representation may require significant computational resources. However, because of its generality, one could essentially represent our framework in a first-order logic by defining the appropriate predicates for the various pieces of our ontologies. The LARKS reasoning engine could then reason over these higher-level concepts in a similar manner to the one we use in our own reasoning engine.

The InfoSleuth project (Bayardo *et al.* 1997; Nodine & Unruh 1997; Nodine, Fowler, & Perry 1999) also did some early work on matching (Nodine, Bohrer, & Ngu 1999). The limitations that we encountered in this initial implementation of matchmaking served as a motivation for the development of the approach presented in this paper.

Conclusions

This paper has presented a novel framework for agents to advertise their capabilities and to query and reason over these capabilities. We have shown that representing non-trivial agent capabilities in other agent-based

or distributed object-based frameworks is either impossible or impractical.

In response, we have developed a new framework that relies on the use of focused ontologies and constraints over those ontologies in describing requested or provided agent capabilities. This framework presents a relatively uniform view of advertisements, queries and query results. This uniformity greatly reduces the complexity of the matchmaking reasoning mechanism.

We have a developed a prototype implementation of this framework using InfoSleuth agents. This prototype has shown that the addition of more expressive advertisements, queries and reasoning enables a wide range of enhancements. This increases the functionality and efficiency of the system dramatically. In particular, the query processing aspects of the system have been greatly improved since the query can be more specific and focused. This has greatly reduced the amount of extra reasoning that is required in the query processing agents as well as eliminating a significant amount of extraneous work.

Finally, we note that while our focus is on having a matchmaker agent that performs this reasoning (matching a query against a repository of agent advertisements) there are other situations where general reasoning is needed. For instance, there could be a blackboard system where queries are posted and advertisements are matched to a repository of queries.

Acknowledgments We thank all of the members of MCC's InfoSleuth group for the many discussions and experiences they have shared over the years. Parts of this paper must surely contain an amalgam of the their thoughts and ideas.

References

Arens, Y.; Knoblock, C.; and Shen, W. 1996. Query reformulation for dynamic information integration. *Journal of Intelligent Information Systems* 6(2):99–130.

Bayardo, R.; Bohrer, W.; Brice, R.; Cichocki, A.; Fowler, J.; Helal, A.; Kashyap, V.; Ksiezyk, T.; Martin, G.; Nodine, M.; Rashid, M.; Rusinkiewicz, M.; Shea, R.; Unnikrishnan, C.; Unruh, A.; and Woelk, D. 1997. InfoSleuth: An agent-based semantic integration of information in open and dynamic environments. *Proceedings of the ACM SIGMOD International Conference on Management of Data.*

Decker, K., and Sycara, K. 1997. Intelligent adaptive information agents. *Journal of Intelligent Information Systems* 9(3):239–260.

FIPA. 1997. http://www.fipa.org.

Fowler, J.; Nodine, M.; Perry, B.; and Bargmeyer, B. 1999. Agent-based semantic interoperability in InfoSleuth. *Sigmod Record* 28(1).

Genessereth, M.; Keller, A.; and Duschka, O. 1997. Infomaster: An information integration system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 539–542.

Kuokka, D., and Harada, L. 1995. On using KQML for matchmaking. In *Proceedings of the International Conference in Multi-agent Systems (ICMAS)*, 239–254.

Kuokka, D., and Harada, L. 1996. Integrating information via matchmaking. *Journal of Intelligent Information Systems* 6(2):261–279.

Labrou, Y. 1996. *Semantics for an Agent Communication Language.* Ph.D. Dissertation, University of Maryland at Baltimore County.

Levy, A. Y.; Rajaraman, A.; and Ordille, J. J. 1996. Querying heterogeneous information sources using source descriptions. In *Proceedings of the International Conference on Very Large Databases*, 251–262.

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

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

Molina, G.; Papakonstantinou, Y.; Quass, D.; Rajarman, A.; Sagiv, Y.; Ullman, J.; and Widom, J. 1997. The TSIMMIS approach to mediation: Data models and languages. *Journal of Intelligent Information Systems* 8(2):117–132.

Nodine, M. H., and Unruh, A. 1997. Facilitating open communication in agent systems: the InfoSleuth infrastructure. In *Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages.*

Nodine, M.; Bohrer, W.; and Ngu, A. H. H. 1999. Semantic multibrokering over dynamic heterogeneous data sources in InfoSleuth. In *Proceedings of the International Conference on Data Engineering.*

Nodine, M.; Fowler, J.; and Perry, B. 1999. Active information gathering in InfoSleuth. In *Proceedings of the International Symposium on Cooperative Database Systems for Advanced Applications.*

OMG, and X/Open. 1992. *The Common Object Request Broker: Architecture and Specification, Revision 1.1.* John Wiley and Sons.

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

Sycara, K.; Lu, J.; Klusch, M.; and Widoff, S. 1999. Matchmaking among heterogeneous agents on the Internet. In *Proceedings of the AAAI Spring Symposium on Intelligent Agents in Cyberspace.*