Experience with the InfoSleuth Agent Architecture

Marian Nodine* and **Brad Perry*** and **Amy Unruh†,***
{nodine, bperry, unruh}@mcc.com
*Microelectronics and Computer Technology Corporation (MCC)
Austin, Texas 78759
†Systems and Software Lab, DSPS R&D, Texas Instruments
PO Box 655303, MS 8378, Dallas, TX, 75265

Abstract

The MCC InfoSleuth project is developing an architecture and toolkit for deploying agent applications that focus on *information gathering and analysis* over diverse and dynamic networks of multimedia information sources[1]. In this paper, we present the structure of the *layered agent shell* that we use for rapid and consistent insertion of agents in the InfoSleuth environment. The shell attempts to aggregate and make available the common functionalities and services found across all classes of agents in our applications. The intent of the shell is to allow developers to concentrate on the unique aspects of each agent while inheriting common, configurable functionality.

Two critical findings are highlighted in our presentation of the agent shell. First, KQML can be used for interagent messaging, but progress in agent interactions requires a conversational metaphor to guide conversation policies among agents. Second, there are multiple "semantic levels" at which the same set of agents may want to converse over the lifetime of an application task. These levels are clearly delineated in our shell and can be realized using combinations of the appropriate transfer protocols. The structure of the InfoSleuth layered agent shell provides valuable insight and lessons for the ongoing definition, development, and acceptance of tools for open agent-based systems.

Introduction

An agent system is a set of cooperating processes distributed across a network or internet. Each agent is a specialist in a particular task or subtask. Agents are similar to traditional distributed systems in that they farm out subtasks to other agents as needed. However, unlike in a distributed system, agents are often developed by different groups of people, and therefore implementing agents that cooperate well can be more difficult. One further complicating factor is that agents typically interact with other agents on a semantic basis, rather than on a set of shared interfaces such as might be found in, say, a CORBA-based distributed system.

InfoSleuth (InfoSleuth 1998) is an architecture and toolkit for deploying agent systems that is undergoing active research and development at MCC. The InfoSleuth environment focuses on *information gathering and analysis* over diverse and dynamic networks of multimedia information sources. The emphasis behind InfoSleuth is to establish a stable agent infrastructure and interaction machinery such that disparate groups and organizations can independently develop agents that meet and work together in the context of an InfoSleuth application. Our working environment is such that we develop the base InfoSleuth architecture and agent creation/monitoring tools at MCC and then release these artifacts for deployment by our investing corporations and government sponsors. With the aid of these participants, we have deployed the InfoSleuth architecture in several application domains, including healthcare, environmental protection, semiconductor manufacturing, content-based image dissemination, and military logistics.

In this paper, we present the *layered agent shell* that we use for rapid and consistent creation and monitoring of agents in an InfoSleuth environment. This shell allows for a clean, multi-layered approach to interagent organization and communications. First, it has made implementing and installing new agents into the system easy, as the syntax and semantics of many message exchanges are not only well-specified, but also implemented for you. Second, it provides a forum for leveraging off of existing, more sophisticated technologies for syntax-level support of message exchange, while maintaining the semantic nature of the overall communication. We standardized the use of an agent shell within InfoSleuth applications starting in September, 1996. Our experience with InfoSleuth has led us to believe that this layering is the correct approach to supporting communication in

[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".

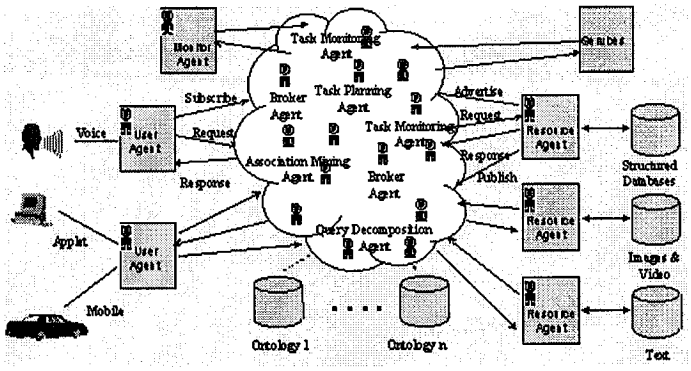

Figure 1: InfoSleuth: Dynamic and Broker-based Agent Architecture.

an open and dynamic agent-based system.

The structure of this paper is as follows. We begin with a brief overview of the InfoSleuth architecture and then proceed to expand on the concept of an "agent shell" and issues with interagent communication and layered architectures that we have encountered during the development and evolution of the InfoSleuth agent system. Next we summarize the effort that was required to deploy InfoSleuth in three different domain. This is followed by a comparison of InfoSleuth to other related technologies. Finally, we conclude with current research issues and concluding remarks.

The InfoSleuth System

The InfoSleuth model (Bayardo *et al.* 1997; Nodine & Unruh 1997) defines a proven framework for loosely collecting agents based on semantic advertisements and then dynamically composing agents based on application needs. Figure 1 depicts the agent architecture currently defined and deployed in the MCC InfoSleuth project. Its current implementation is in Java^{TM} and the agent tools have been written once and used, without change, to intermix agents from several Unix^{TM} and Windows^{TM} platforms. A few notable aspects of this architecture are:

- Agents advertise their "information gathering and monitoring" capabilities, using semantic constructs from the InfoSleuth agent *capability ontology*. The capability ontology supports descriptions of an agent's performance, properties, services, and content. An agent's advertisement of content is in terms of portions of domain-specific *common ontologies*, so that an agent may constrain its capability advertisement to apply to only a select set of concepts, relationships, or instances from a particular application domain.
- At any point, a relevant set of available agents can be discovered, via the semantic constraint satisfaction services of the broker, over the active set of agent advertisements. The InfoSleuth system dynamically constructs information gathering agent communities, based on brokering and planning principles, to satisfy given tasks as best as possible.

The general class of applications where the InfoSleuth system has been deployed can be classified as *information gathering and analysis* over diverse and dynamic information networks. In this application class, the primary activity of the agents is to accumulate and transform data from disparate sources into ontological, or domain, abstractions. A simplified view of the InfoSleuth information gathering and analysis process is as follows:

> *The domain ontology defines a set of "domain events and activities" that drive decision making in the application. The ontology relates these events in a graph-structure where each "node" of the graph has a distinct and periodic information need. The InfoSleuth agents cooperate to populate the information needs of the domain events by accumulating and transforming data from a myriad of disparate data sources.*

InfoSleuth offers a metaphor beyond its counterparts in distributed information gathering, such as (Arens, Knoblock, & Shen 1996; Garcia-Molina *et al.* 1997; Levy, Srivastava, & Kirk 1995), in that InfoSleuth is focused on loosely coupled components performing complex analysis and mined associations over the information space.

Additional highlights of the InfoSleuth paradigm related to agent interaction are:

- *Community orientation*: the agents in an InfoSleuth application form *communities* based the agents' advertised domains of interest, capabilities, and current tasks. InfoSleuth brokers are used to facilitate interconnection within a community and bridge the gap into other related communities as application needs expand beyond a particular community focus.
- *Semantic matchmaking*: when one InfoSleuth agent requires services from another agent, the InfoSleuth broker reasons over advertised semantic constraints on agents' behaviors as well as syntactic constraints of their interfaces. This is in contrast to CORBA interface brokering and keyword-based matchmaking, as further discussed in the section on related work.
- *Beyond KQML*: KQML (Knowledge Query and Manipulation Language) (Finin, Labrou, & Mayfield 1997) is an effort to define a standard useful set of speech acts, or *performatives*, that agents can use to exchange information; as well as the (semi-formal) semantics behind the performatives. The performatives each have a number

of fields, or parameters associated with them – in addition to the *content* of the performative, other *contextual* parameters specify e.g., the language and ontology of the message. Routing parameters specify the sender and receiver. One could view InfoSleuth as a *large-scale testbed, focused on industrial applications, exercising the soundness and viability of the KQML specification.*

Agent Shells and Their Use in InfoSleuth

At the base of our InfoSleuth system is a software support module termed the *agent shell.* This section discusses the motivation in defining an agent shell and then presents the details of the InfoSleuth agent shell.

What is an agent shell?

An agent shell is an extensible template that attempts to aggregate and make available common functionalities and services found in agent environments. The creation of a new agent involves subclassing the agent shell, inheriting select functionalities and services from the shell, and linking the agent-specific logic into the "agent application API" of the shell. A layered agent shell attempts to align these functionalities and services behind well-defined interfaces and into semantically related layers. In other words, a layered agent shell provides structured support for the rapid creation and easy deployment of agents in large-scale multi-agent applications. An agent shell should encapsulate the basic functionalities common across agents, including: Agent startup procedures, advertisement, monitoring and control, validation and security, agent community-specific activities, agent interoperation, and agent abilities such as mobility and aggregation with other agents. The support for agent interoperation in an agent shell takes place at a variety of levels, including

- understanding the same interagent message protocols (e.g., KQML),
- sharing the same expectations of agent interaction with respect to subtasks (conversations),
- forwarding and/or broadcast of conversations to other agents,
- communicating evolving task- and conversation-related specifications, and
- monitoring and possibly controlling the operation of other agents.

Why use agent shells?

Agent shells provide a reusable common operating environment for developing agents within a given agent community. This type of architectural design follows the successful use of layering in other types of systems where interoperation is integral, such as data communication networks.

The InfoSleuth agent shell facilitates *rapid and consistent* development of agents by multiple disjoint parties. The *consistent* development of agents is accomplished by capturing the interaction patterns and syntactic and bookkeeping issues of agent interaction in the layers of the agent shell. The *rapid* development of agents is accomplished by collecting a set of specialized *agent support services.*

Our initial experience within InfoSleuth showed us that the KQML specification is loose enough that it can be subject to a variety of (sometimes conflicting) interpretations. Even though we were working closely as a group, the ability of our agents to interoperate was impeded by these interpretational differences. The agent shell provided a common standard that we could all adhere to. Currently, our developers can focus on the needs of their individual agent application, and rely on the shell to provide for the appplication-independent services. When we need to extend the basic functionality of all agents, for instance by allowing some to be mobile, this involves only extending the agent shell; the rest is transparent to the applications.

Functional layering within an agent shell, with a clearly defined interface for each layer, facilitates the integration of and plug-and-play experimentation with application and/or third-party software in an agent support environment. For instance, our shell has a *message layer* which currently allows the agent to run over a variety of transport mechanisms, including TCP and CORBA. Inserting SMTP as a transport mechanism involves developing an SMTP-generating version of the message layer. Similarly, generating a shell for an agent community that uses, say, market-based control strategies does not require any changes to layers which support messaging.

From the outside of an agent community, an agent shell provides a standard interface to all agents in the community. Because of this, the use of a shell allows the developers of interoperating agents to bypass common initial discussions such as how to structure messages and what types of conversational exchanges to support.

The Infosleuth Agent Shell

Figure 2 shows the original agent shell layering defined and implemented in the InfoSleuth project. Each layer has a clean semantics and a well-defined interface. This section provides a succinct summary of the functionality in the existing InfoSleuth agent shell.

Message Layer The *agent message layer* maps logical KQML requests made by the conversation layer into and out of physical network exchanges with other agents. Our current message layer maps

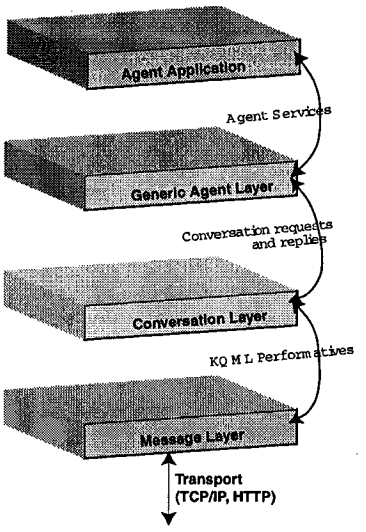

Figure 2: InfoSleuth agent layers.

logical KQML requests into network exchanges over the TCP/IP, HTTP, and CORBA communication protocols. The message layer encapsulates three basic functionalities:

- Maintains a mapping of recently-used logical agent names to their physical network access addresses. Name lookup failures result in the message layer automatically contacting an InfoSleuth broker to resolve the address.
- Ensures that all agents employing it are consistent in their syntactic use of KQML. This consistency involves two aspects: (1) assuring the creation of valid KQML messages from logical requests made by the conversation layer; and (2) parsing and checking messages received from other agents into valid KQML messages, then propagated to the conversation layer. This functionality frees our agents from worrying about syntactic inconsistencies in their use of KQML.
- Standardizes the process of transferring KQML over TCP/IP and HTTP communication links in InfoSleuth.

Conversation Layer The *conversation layer* of the InfoSleuth architecture defines and enforces conversation policies for a group of cooperating agents. A set of standard messages, e.g. in KQML, representing the available speech acts, can serve as a basis for very simple communication among agents. However, messages do not get sent in isolation; rather, there are often ongoing dialogs among two or more agents. Within a dialog, the interpretation of an individual message may depend on the *context* of the dialog in which they are participating. A "conversation" is a partially-ordered set of messages transmitted among a set of agents, all of which relate conceptually to an initiating speech

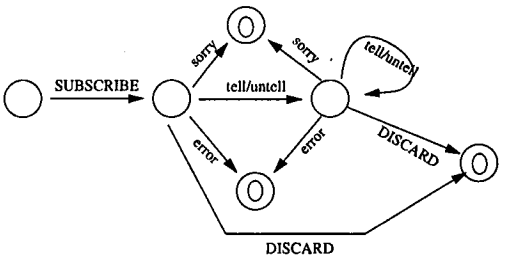

Figure 3: A finite-state model describing a SUBSCRIBE conversation. The messages from the initiating agent are in uppercase; the messages from the responding agent are in lowercase.

act. A *conversation policy* is a formal and deterministic specification of the ordering of speech acts exchanged between agents during a conversation. Conversation policies may be *presriptive* or *emergent*. A *prescriptive* conversation policy is one that is defined *a priori*, and used to enforce the agents' communications. Emergent conversations are those in which the agents are not following specific external conversational policies; but rather where the performatives they use are determined by their internal functionality. These actions may be (but are not necessarily) driven by the agent's semantic understanding of the discourse taking place (Smith & Cohen 1996).

Open agent communication requires that any agents that communicate share a common set of conversation policies. However, in an open system, one cannot make assumptions about the agents' common understanding of a semantics for discourse. Instead, agents need to both expect and provide similar responses in similar contexts. We believe that support of interoperation requires imposition of prescriptional structure on at least some functional subset of the agents' conversational policies. This does not preclude the use of dynamically- and incrementally-specifiable conversation policies, however.

Approach and Implementation. In InfoSleuth, we currently support a default set of system-wide pairwise conversation policies The conversation policies specify the allowable messaging interactions that may occur among a set of collaborating agents. They guarantee consistent interaction between agents and enable the conversing agents to maintain a reliable context, or session, throughout their interactions. While the use of shared prescriptive conversations does not guarantee semantically consistent actions by the agents, it facilitates consistency in an open system by providing the agent with an implicit semantics for the content of its messages.

Finite-state machines are used to specify the allowable pairwise conversations between agents in the system. Conversation polices are currently identified based on the initial performative, so that a distinct conversation policy exists for each possible initiating message. Other work has used a similar approach (Bradshaw *et al.* 1997; Cost *et al.* 1998; Labrou 1996; FIPA).

Figure 3 shows an example of a conversation policy. The transition from the start states in all models correspond to the sending of the conversation-initiating message. The remaining transitions define acceptable message between the two agents at a given time. Final states indicate the end of the conversation. Conversation support is multi-threaded, for both the initiator of a conversation and the recipient. Each thread maintains state, performs error checking, and communicates with the agent for the duration of its conversation.

Generic Agent Layer The *generic agent layer* embodies the services crucial to the operation of all agents that participate in an InfoSleuth community. Within InfoSleuth, there are two such services. The first is the semantic matchmaking service, which enables an agent that is requesting a service to locate another agent that can provide that service. The second service provides information on the knowledge accessible through the *ontologies* defined within the community.

Approach and Implementation. InfoSleuth implements these generic agent services as separate agents within the community. *Broker agents* collect agent advertisements and implement semantic matchmaking. *Ontology agents* store the different domain models and answer questions concerning them.

The generic agent layer of the InfoSleuth agent shell implements the following interfaces to these generic agent services:

- Standard advertisement interface to broker agents. This includes the agent's initial advertisement as well as incremental changes to that advertisement. This interface also implements the ability for the agent to unadvertise itself before it goes offline.
- Standard querying interface to broker agents. This provides abilities for the agent to query a broker about other agents whose services may be available to it. It also parses the result from the broker into a meaningful structure and inserts any new agents into the agent's address book.
- Standard querying interface to ontology agents. This allows the agent to query an ontology agent about the different domains in the system and the types of knowledge that they represent.

Agent Application Layer The agent application layer implements the functionality specific to the agent itself. It is the intent of the layered agent shell architecture to allow the agent developer to focus his efforts primarily on the agent application layer, and to inherit directly the agent shell capabilities and the optional services that it supports.

In InfoSleuth, agent applications such as the broker, the multiresource query agent, the resource agents, and the data mining agents each implement their own specific capabilities within the agent application layer. The process of collaboration among the agent applications is that of speech-act-based conversational requests and replies. Conversation requests embody the speech act, ontology, content language and specific content of the different inter-agent service requests and assertions. In this way, we abstract details about operational issues such as message passing mechanisms and agent location issues from the purview of the application developer.

Issues with Agent Shell Development

In this section, we discuss some of the issues that have emerged during the design, implementation, and use of our agent shell. This section is organized from the top down in the layering of Figure 2.

Conversation Issues

Our experiences with InfoSleuth thus far have pointed out a number of areas in which our current support of conversation policies needs to be extended.

Speech Acts vs. Performatives. We believe it is important to restrict the agents' conversation policies to operate over "true" speech acts instead of KQML performatives (Finin, Labrou, & Mayfield 1997). The conversations should be described at a semantic level, and should not include performatives designed to support data flow management, such as `stream-all`, `ready`, and `next`; instead of `ask`. Data flow management and indirect data references should be transparent to the conversation structure, and supported by a lower architectural level in the agent architecture. This implies a refinement of the layering in Figure 2, with a new layering that separates flow control and conversational structure. An agent's support for flow control should be represented in the system's *capability ontology* and advertised by the agent, thus allowing an agent to determine what data management protocols it uses – including, potentially, KQML data-management performatives – to talk to another agent. This layering of functionality allows robust and open specification of conversation policies independent of the way a particular agent implements its data management.

Declarative Conversation Policies. We need to support declarative and dynamically-specifiable conversation structures. As the complexity of the tasks support by the InfoSleuth system grows, the

set of required conversation policies grows as well. Declaratively-specifiable state machines will make conversation specifications more robust, and will allow conversation policies to be added dynamically to an agent (sub)system. Furthermore, if InfoSleuth's conversation policies are declaratively specifiable, it will be possible to support incremental definition of policies via message exchange between agents. For example, an agent might specify the set of speech acts from which it expects a reply to a message to be drawn. (Cost *et al.* 1998) describes one approach in support of declarative conversation templates.

Decoupling Conversations from Message Types. As InfoSleuth supports increasingly more complex activities, it has become evident that the conversation structures must be decoupled from the performatives with which they are initiated, so that the same performative can start more than one conversation. As an example, an `ask` can potentially start a simple `query/reply` conversation, or a negotiation.

If performatives are not decoupled from conversations, it would either a) be necessary to create a new performative for each new conversational structure; or b) require the individual agent applications to maintain context between smaller conversations — in effect, to move some of the conversation management to the agent application level. We do not believe that either of these options are an appropriate solution for an open system. Thus, we will assume that the same performative can start more than one conversation, and that in some cases additional context may be sent in the initiating message to identify the conversation policy.

Multicasting, Forwarding, and Delegation. The conversation structures supported by InfoSleuth must grow more complex in other dimensions as well. An agent must be able to send a message to more than one recipient agent, where the recipients are identified by either their logical name, or a semantic description in support of associative multicasting. An agent must be able to forward a request to another agent for handling and reply. Similarly, an agent must be able to delegate by initiating a request but specifying that the recipient send its reply elsewhere.

In addition, agents must be able to receive replies to a request from more than one agent. Such a situation often arises in complex tasks, where an agent handles part of a request itself, but delegates some aspect of it to another. For example, InfoSleuth supports knowledge mining activities in which a *task execution* agent receives a knowledge mining task from a user agent. As part of accomplishing that task, the task execution agent sends a series of messages to a *knowledge mining* agent. The knowledge mining agent's replies, as well as additional replies by the task execution agent, are all returned

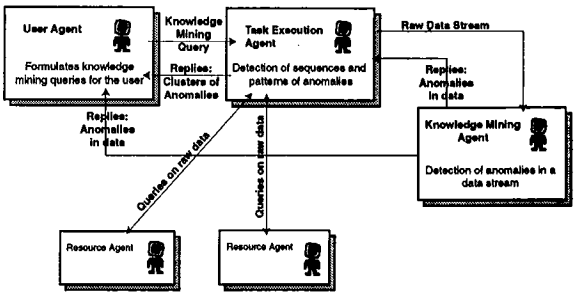

Figure 4: Delegation in support of a query, and replies to the query returned to a user agent from both a task execution agent and a knowledge mining agent.

Agent Message Layer	Conversation Layer	Agent Layer
:perf-type :sender :receiver	:perf-type :sender :reply-with :in-reply-to	:content :language :ontology :aspect :code :comment

Table 1: KQML Performative Properties and Layer Dependencies

to the user agent in response to its original message. Figure 4 shows this process.

KQML Issues

Our diverse deployment of InfoSleuth has given us an in-depth analysis into the use of KQML in real applications under significant workloads. This section highlights our most significant findings and suggestions for how to carry KQML into a "prime-time" agent communication standard.

Performatives as Property Lists. KQML'93 defines a message as a flat property list with string valued properties. During our development of agent interactions in InfoSleuth, we found an interesting "allocation" of the KQML properties to logical layers in our agent shell. Different properties are linked to different functional aspects of agent communication. If we group the properties in a KQML message with the layer that uses them, we get the allocation in Table 1.

Since a KQML message is a flat property list, the agent message layer *must* process and parse each incoming KQML message in its entirety before channeling the message into the conversation layer. This "all or nothing" parsing is quite inefficient and introduces a noticeable performance detriment in the InfoSleuth applications we have tested under realistic application loads. We believe that each layer should be involved in extracting only those properties that are needed for that layer's processing. Thus, the agent's message layer should be able to

ignore :perf-type, as it has nothing to do with the sending and receiving of the messages. This would also mean that properties such as :ontology could be totally ignored until they reach the agent layer. Some of the mechanisms that allow delaying the parsing and extracting of properties were present in an earlier version of KQML (Finin & Wiederhold 1991).

"Open content exchange" standards, such as HTTP, have realized this need for explicitly defining message layers to support early termination and deferred parsing of messages. Following this approach would lead to a more structured agent communication message, where the message is nested according to where the information is needed in a layered architecture. A more structured version of a KQML (or agent communication) message would look similar to that in Figure 5. The structure in Figure 5 is motivated by the layering in the InfoSleuth agent shell. We believe the layering is sound and stress that the agent communication world would benefit from adopting some form of agent message layering.

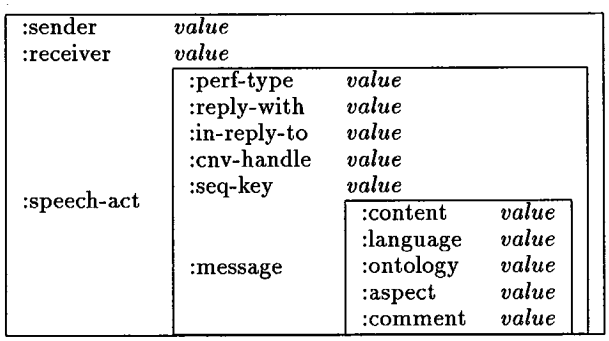

Figure 5: A Nested Version of a KQML Message

Sending Knowledge vs. Sending Information. Another fallacy we found in the KQML specification is the unnecessary and confusing mixture of knowledge and information in agent communications. We believe that (1) agents should *use KQML to exchange knowledge* about what they know and what information they have available; and (2) agents should operate outside of KQML to negotiate the actual exchange of bulk, or complex, information products. Unfortunately, the KQML specification mixes knowledge exchange with data exchange without delineating the difference between these paradigms. We feel that KQML is effective when used compactly at the knowledge-level and agent systems should use existing protocols for streaming large information products between agents once they have agreed, by conversing in KQML, on the need to exchange an information product. This separation of "sending knowledge packets" from "streaming information products" has proven to be quite straightforward and natural in our large-scale agent application experiences.

Infosleuth Application Experiences

Creating a new application from InfoSleuth involves iteration over the following three activities:

- Develop an ontology describing the "domain events and activities" to be populated from the information network.
- Reuse any number of the "generic" agents provided by the InfoSleuth base (i.e., the broker, ontology agent, user agent, etc.).
- Connect any application-specific logic to the InfoSleuth agent shell and advertise it in the network as an agent ready to provide information services on fragments of the domain ontology.

The intent of our agent shell is to make the third item as straight-forward as possible and enable new logic to easily connect into larger application environments. In the next few paragraphs, we give an overview of various applications that have been constructed from InfoSleuth and give a summary of the effort that was required (or saved) by using the InfoSleuth infrastructure.

Evironmental Data Exchange Network (EDEN). The EDEN project (Pitts & Fowler 1998) is a multinational program between the United States and Europe aimed at sharing information about environmentally contaminated sites and technologies used to remediate the conditions. The program is using InfoSleuth to provide access to dynamically changing sets of information resources and to perform associational queries across multiple resources. In this application a resource is a database documenting sites, remediation technologies, or both. The application need is to answer ontological requests at the level of "what sites are contaminated in Texas?" and "notify me when any site uses phyto-remediation for treatment." In terms of InfoSleuth, EDEN required the creation of two classes of agents. (1) Resource agents for individual resources (i.e., databases) that map and advertise their content in the EDEN ontology. (2) Value mapping agents that translate between the instance-level terms found in the EDEN databases. InfoSleuth provides a specialization of the agent shell termed the *resource agent shell* that encapsulates (and parameterizes) the conversations a resource may have with the other agents in the network. Using this shell and its associated tools, EDEN resource agents are usually constructed in about one day. The value mapping agents are implemented as resources too (i.e., a database of translation tables). Overall, the EDEN application is primarily aimed at integrated multiresource information gathering from an evolving resource base.

The InfoSleuth agent shell enables new data resources to be quickly and autonomously injected into the ongoing application instance.

Content-based Image Dissemination. At Hughes Research Laboratories (HRL), InfoSleuth is being used to perform integrated content-based image dissemination from collections of satellite image repositories (Shek *et al.* 1998). In terms of InfoSleuth, the HRL application required performing the followin two tasks. (1) Interfacing the conversational interface of the agent shell with the search facilities of the image repositories. (2) Constructing an agent that performed "fuzzy joins" over multiple image resources. Connecting the interfaces of the image repositories to the conversational interface turned out to be a straight-forward task and we were able to "agentify" a repository in a matter of days. The more difficult task was providing a mapping from the logical ontological requests a conversation would contain into image processing patterns in the underlying repository. It took about one month to construct this mapping and bring two different image repositories online and advertising their content as resource agents.

To construct an information request in this application, end-users provide two things (a) a set of sample images portraying weather patterns of interest; and (b) a set of concepts, or hints, from an ontology of geoscientific phenonema defining the user's semantic focus in the sample images. Answering a request in this application is a 3-step process: (1) use the InfoSleuth broker to find the set of active resources that "best match" the user request; (2) forward appropriate fragments of the request to the identified resources; and (3) perform a fuzzy join (or integrated union) of the results to arrive at a ranked answer list. To summarize, it took about one month to bring the first version of the system online, and about three months to demonstrate its scalability across a large number of diverse satellite image repositories. Again, the consistency of the InfoSleuth agent shell and its existing information gathering components made this application a highly productive exercise.

Healthcare Data Analysis. The InfoSleuth team has had an ongoing relationship with the NIST healthcare "community" to provide data analysis capabilities across multiple healthcare institutions and providers (Fowler & Martin 1997). One of the more interesting applications of InfoSleuth in this domain was that of *detecting deviations in healthcare encounter costs.* This is best understood by providing an example of the ontological request being satisfied in the network: "Notify when the cost of reattaching major limbs deviates from the expected norm; also notify when this event happens at hospitals in the same HMO group." There were three essential components this application added to the InfoSleuth base. (1) The resource agents that monitor hospital data sources for encounter information. (2) A deviation detection agent (DDAgent) that computes "expected norms" from encounter data emanating from multiple hospitals and checks new encounter events for deviations from the expected norm. (3) A pattern collection agent that subscribes to deviations computed in the DDAgent and looks for temporally correlated events from "related hospitals." As with the EDEN application, the resource agents interfaced with structured databases and each was constructed in less than one week. The DDAgent used the "subscription conversations" in the InfoSleuth agent shell to setup "encounter event notifications" with appropriate resources recommended by the broker. The pattern collection agent also used subscription conversations to setup "cost deviation notifications" from the DDAgent. For both of these analysis agents, the only work required was to interface the specific algorithms themselves to the event-based conversations in the agent shell. This application is quite intriquing because it demonstrates coordinated information gathering and analysis being performed at multiple levels (i.e., resources, multiresource norms, deviations, correlated patterns) and occurring in an event-driven manner in a dynamic network of information sources. It took three weeks to hook the application-specific functionality into the agent shell and bring the initial demonstration of this application online and interacting with actual data sources.

This section gave a feeling for the applications that have been built with the InfoSleuth infrastructure. In each, adding "new application logic" to the InfoSleuth information gathering paradigm required a straight-forward interfacing with the conversational interface of the agent shell. InfoSleuth itself is being supported by 6 commercial companies and several government entities. In all cases, we are experiencing very rapid "time to deployment" of complex and domain-specific information gathering and analysis applications.

Related Work

There are a number of areas of agent research related to our development of the InfoSleuth architecture. These include work on agent communication languages; development of conversation policies as well as support of conversation policies via agent tasks; and agent architectures and frameworks.

Agent Communication Languages. The concept of *speech acts*, or *illocutionary acts*, has grown out of philosophy and linguistics research (Cohen & Levesque 1995). These actions include requesting, promising, offering, acknowledging, asserting, etc. It is suggested that human utterances are the observable byproduct of such actions. Speech act theory has been proposed recently as the foundation

for inter-agent communication. The use of a standard set of speech acts in open systems provides a structure to agent discourse in which the intended meaning of the content of the speech act (what is being said) can be interpreted more easily, and provides a semantic structure to the messages intuitive to the human users of a system.

It has been observed (Smith & Cohen 1996) that the speech acts in existing agent systems fall primarily into two general categories: *requests* and *assertions*. This suggests — and has been borne out by our own observations – that there is a strong overlap in the speech acts required by many agent systems, and that a small comprehensive set would be sufficient for many multiagent systems. Currently, there are several efforts towards defining this standard, comprehensive set of speech acts (Finin, Labrou, & Mayfield 1997; FIPA).

Conversation Policies and Task Planning. A few other systems share InfoSleuth's use of conversation policies as state machines (Bradshaw *et al.* 1997; Cost *et al.* 1998; Finin, Labrou, & Mayfield 1997), although we are not aware of any which allow dynamically and incrementally specifiable policies. In addition, work in DAI, e.g. (Decker & Lesser 1993), has explored the use of "toolkits" of interaction policies.

Task specification is closely related to specification of conversation policies, since tasks must often incorporate speech-act actions. Efforts such as (SPAR) support the exchange of declarative task information between agents.

Some agent systems do not employ a conversation layer, but enforce a conversational policy via rule-based agents which all share the same conversation rules (Barbuceanu 1997; Chauhan 1997). This approach works very well in a system of agents that all support rule-based reasoning. The InfoSleuth architecture allows agents to exploit rules for conversation when they have that ability, but does not require this ability in an open system — the conversation layer enforces conversational consistency even in purely procedural agents.

Agent Shells and Agent Architectures. There are a number of other agent systems targeted towards information-gathering tasks, which share similarities with InfoSleuth in the functionality of the agents as well as the system organization. Most contain agents which "wrap" resources, matchmaking services of varying degrees of complexity, middleware agents to access and combine information, and user agents which interact with the user and maintain information about their preferences and task history. None provide the same set of functionality as does InfoSleuth, with respect to heterogeneous and global query processing, knowledge and data mining, the use of conversation policies, semantic and constraint-based brokering, and task planning. Table 2 provides a comparison of some of these systems.

There are number of architectures that support agent mobility as well (IBM 1996; Dartmouth ; ObjectSpace 1997). These architectures are at a somewhat lower level than the top layers of the InfoSleuth architecture, in that they provide a possible foundation on which to build agents that both exchange data; and move and dock when appropriate. We are investigating the feasibility of incorporating an existing mobility package, such as (ObjectSpace 1997), into the InfoSleuth architecture.

Discussion and Conclusions

In this paper we reviewed and discussed the InfoSleuth layered agent shell used for rapid and consistent creation of agents in an open environment. The shell aggregates and makes available the common functionalities and services found across all agents and various classes of agents in our applications. The intent of the shell is to allow developers to concentrate on the unique aspects of each agent while inheriting common, configurable functionality.

At the core of our experiences are three critical conclusions. First, progress in agent interactions requires *more* than context-free syntactic standards. Indeed, the successful deployment of the many InfoSleuth applications can be traced to our introduction of a *conversation layer* oriented around enforcing conversation policies among interacting agents. Second, we have developed and deployed one of the most advanced testbeds exercising the KQML proposed agent communication standard. We concluded that KQML itself was only a step in the right direction and must be used within the context of a conversational metaphor. Furthermore, we have cataloged a set of lessons-learned with the KQML specification, including inconsistencies, inefficiencies, and confusing mixtures of "semantic levels" in its speech-act performatives. Finally, it became necessary to define a clean layering of agent functionalities and services in InfoSleuth.

Our layered approach to agent shell development resulted in an easily configured and extensible agent shell. An important aspect of this approach is that each layer is predicated on a well-defined set of interfaces, allowing plug-and-play experimentation with special purpose or third-party implementations of a layer's interface. The experiences described in this paper provide valuable insight and lessons for the ongoing definition, development, and acceptance of tools for open agent-based systems.

References

Arens, Y.; Knoblock, C. A.; and Shen, W. 1996. Query reformulation for dynamic information integration. *JIIS*.

Agent Architectures	Global Query Processing	Conversation Policies	Semantic Brokering	Task Plans	Layered Architecture
RETSINA (RETSINA 1998)			√	√	√
KaOs (Bradshaw *et al.* 1997)		√	√		√
JatLite (Petrie & Jeon)					√
JAFMAS (Chauhan 1997)				√	√
GMU (Kerschberg 1997)	√				
OOA (Martin *et al.* 1997)	√		√		√
Lockheed-Martin (McKay *et al.* 1996)	√				√
InfoSleuth	√	√	√	√	√

Table 2: Agent architectures and their relationship to InfoSleuth

Barbuceanu, M. 1997. Coordinating agents by role based social constraints and conversation plans. In *Proceedings of AAAI '97*, 16–21.

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: Agent-based semantic integration of information in open and dynamic environments. In *Proceedings of SIGMOD '97.*

Bradshaw, J.; Dutfield, S.; Benoit, P.; and Woolley, J. 1997. KAoS: Toward an industrial-strength open agent architecture. In Bradshaw, J., ed., *Software Agents.* AAAI Press. chapter 17.

Chauhan, D. 1997. *JAFMAS: A Java-based Agent Framework for Multiagent Systems Development and Implementation.* Ph.D. Dissertation, ECECS Department, University of Cincinnati.

Cohen, P., and Levesque, H. 1995. Communicative actions for artificial agents. In *ICMAS-95.*

Computer Science Dept., Dartmouth. Agent TCL. <www.cs.dartmouth.edu/agenttcl.html>.

Cost, R. S.; Finin, T.; Labrou, Y.; Luan, X.; Peng, Y.; Soboroff, I.; Mayfield, J.; and Boughannam, A. 1998. Jackal: a java-based tool for agent development. In this volume.

Decker, K., and Lesser, V. 1993. Designing a family of coordination algorithms. In *Proceedings of the 11th National Conference on Artificial Intelligence.*

Tate, A., et al.
Shared planning and activity representation. <www.aiai.ed.ac.uk:80/~arpi/spar/>.

Sycara, K., et al. 1998. RETSINA. <www.cs.cmu.edu/~softagents/retsina/ans/java/docs/ansHTML/index.html>.

Finin, T., and Wiederhold, G. 1991. An overview of KQML: A knowledge query and manipulation language. Stanford CS Dept TR.

Finin, T.; Labrou, Y.; and Mayfield, J. 1997. KQML as an agent communication language. In Bradshaw, J., ed., *Software Agents.* AAAI Press.

FIPA. <www.cselt.stet.it/fipa>.

Fowler, J., and Martin, G. 1997. The healthcare administrator's associate: An experiment in distributed healthcare information systems. In *Proceedings 1997 AMIA Annual Fall Symposium*, 548–552.

Pitts, G., and Fowler, J. 1998. Collaboration and knowledge sharing of environmental information: The EDEN project. In *IEEE International Symposium on Electronics and the Environment (ISEE).*

Garcia-Molina, H.; Papakonstantinou, Y.; Quass, D.; Rajaraman, A.; Sagiv, Y.; Ullman, J.; Vassalos, V.; and Widom, J. 1997. The TSIMMIS approach to mediation: Data models and languages. *JIIS* 8(2).

ObjectSpace, Inc. 1997. Voyager technical review. <http://www.objectspace.com/voyager/voyager_white_papers.html>.

InfoSleuth. 1998.
<www.mcc.com/projects/infosleuth>

Kerschberg, L. 1997. The role of intelligent software agents in advanced information systems. In *British National Conference on Databases (BNCOD 97).*

IBM Tokyo Research Laboratory. 1996. IBM aglets: Programming mobile agents in Java. <www.ibm.co.jp/trl/aglets/whitepaper.htm>.

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

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

Martin, D. L.; Oohama, H.; Moran, D.; and Cheyer, A. 1997. Information brokering in an agent architecture. In *Proceedings of the Second International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology.*

McKay, D.; Pastor, J.; McEntire, R.; and Finin, T. 1996. An architecture for information agents. In *AIPS-96.*

Nodine, M., and Unruh, A. 1997. Facilitating open communication in agent systems: the infosleuth infrastructure. In *Proceedings of ATAL-97.*

Petrie, C., and Jeon, H. JatLite.
<http://java.stanford.edu/java_agent/html/>.

Shek, E.; Vellaikal, A.; Dao, S.; and Perry, B. 1998. Semantic agents for content-based discovery in distributed image libraries. In *IEEE Workshop of Content-Based Access of Image and Video Databases.*

Smith, I., and Cohen, P. 1996. Toward a semantics for an agent communications language based on speech-acts. In *Proceedings of AAAI-96*, 24–31.