KAoS: An Open Agent Architecture Supporting Reuse, Interoperability, and Extensibility

Jeffrey M. Bradshaw
Research and Technology, Boeing Information and Support Services
P.O. Box 3707, M/S 7L-44, Seattle, WA 98024, (206) 865-6086
jbrad@redwood.rt.cs.boeing.com

Abstract

The long-term objective of the KAoS (Knowledgeable Agent-oriented System) agent architecture is to address two major limitations of current agent technology: 1. failure to address infrastructure, scalability, and security issues; and 2. lack of semantics and extensibility of agent communication languages. The first problem is addressed by taking advantage of the capabilities of commercial distributed object products (CORBA, DCOM, Java) as a foundation for agent functionality, and supporting collaborative research and statndards-based efforts to resolve agent interoperability issues. The second problem is addressed by providing an open agent communication meta-architecture in which any number of agent communication languages with their accompanying semantics could be accommodated. Unlike most agent communication architectures, KAoS explicitly takes into account not only the individual message, but also the various sequences of messages in which it may occur. Shared knowledge about message sequencing conventions (conversation policies) enables agents to coordinate frequently recurring interactions of a routine nature simply and predictably.

1. INTRODUCTION

1.1. The Promise of Software Agents

Current trends have made it clear that automation of dynamic, real-time environments will dramatically increase in the coming decades. The complexity, real-time constraints, and distributed nature of such tasks require that software not merely respond to requests for information but intelligently anticipate, adapt, and actively seek ways to support users. Not only must these systems assist in coordinating tasks among humans, they must also help manage cooperation among distributed programs.

Software agents have been proposed as one way to help people better cope with the increasing volume and complexity of information and computing resources (Bradshaw, 1996) . Researchers are hopeful that this approach will help restore the lost dimension of individual perspective to the content-rich, context-poor world of the next decade. What will such agents do? At the user interface, they will work in conjunction with compound document frameworks and document management tools to select the right data, assemble the needed components, and present the information in the most appropriate way for a specific user and situation. Behind the scenes, agents will take advantage of distributed object management, database, workflow, messaging, transaction, searching, indexing, and networking capabilities to discover, link, and securely access the appropriate data and services.

1.2. Limitations of Current Approaches

While several approaches to agent technology are showing significant promise, many critical issues remain

unsolved. For one thing, agents created within one agent framework can seldom communicate with agents created within another.

Semantics. KQML has been proposed as a standard communication language for distributed agent applications (Finin, Labrou, & Mayfield, 1996; Genesereth, 1996) . The core concept is that agents communicate via "performatives", by analogy with human performative sentences and speech acts (e.g., "I hereby request you to send me file ABC.TEX"). Unfortunately, KQML developers have not yet reached full consensus on many issues. Agent designers are free to add new types of performatives to the language. However, there exist a number of confusions in the set of performatives supplied by KQML and no constraints are provided to agent designers on what can be a performative (Cohen & Levesque, 1996) . Without a clearly-defined semantics of individual performatives as they are employed within particular types of agent-to-agent dialogue, developers cannot be sure that the communication acts their agents are using will have the same meaning to the other agents with whom they are communicating. Such a semantics is needed to determine the appropriateness of adding new performatives to a particular agent communication language, and to define their relationship to preexisting ones.

Infrastructure, scalability, and security. In addition to the current limitations of agent communication languages, the potential for large-scale, cross-functional deployment of general purpose agents in industrial and government settings has been hampered by insufficient progress on infrastructural, architectural, security, and scalability issues. Considerable research has been done on these issues by the distributed computing community, and in some cases commercial products exist that could address many of them, yet up till now relatively little effort has been made to incorporate these technologies into agent development frameworks.

The current lack of standards and supporting infrastructure has prevented the thing most users of agents in real-world applications most need: agent interoperability (Gardner, 1996; Virdhagriswaran, Osisek, & O'Connor, 1995) . A key characteristic of agents is their ability to serve as universal mediators, tying together loosely-coupled, heterogeneous components-the last thing anyone wants is an agent architecture that can accommodate only a single native language and a limited set of proprietary services to which it alone can provide access.

1.3. KAoS: An Agent Architecture for Reuse, Interoperability, and Extensibility

To address some of these problems, we are working in partnership with Seattle University to develop KAoS (Knowledgeable Agent-oriented System), an open distributed architecture for software agents (Bradshaw, Dutfield, Benoit, & Woolley, 1996) . Our experience with KAoS to date leads us to believe that an approach of this type can become a powerful and flexible basis either for implementing diverse types of agent-oriented systems, or for interoperation with non-KAoS agent frameworks.

Providing infrastructure, scalability, and security through a foundation of distributed object technology. To the extent KAoS can take advantage of architectures such as CORBA, we can concentrate our research efforts on the unique aspects of agent interaction rather than on low-level distributed computing implementation issues. CORBA provides a means of freeing objects and agents from the confines of a particular address space, machine, programming language, or operating system (Betz, 1994) . The Interface Definition Language (IDL) allows developers to specify object interfaces in a language-neutral fashion. Object Request Brokers (ORBs) allow transparent access to these components and services without regard to their location. The CORBA 2.0 specification extends the architecture to deal with the problem of interoperability between ORBs from different vendors. A set of system services is bundled with every ORB,

and an architecture for "common facilities" of direct use to application objects is being defined. Among these common facilities will be a compound document facility based on an enhanced version of the CI Labs OpenDoc specification (Orfali, Harkey, & Edwards, 1995) .
Our collaborations with SU have produced increasingly sophisticated versions of KAoS that are designed to take advantage of the capabilities of commercial distributed object products as a foundation for agent functionality. To date, we have investigated the use of two object request broker (ORB) products: IBM's System Object Model (SOM) (Campagnoni, 1994) , and Iona's Orbix. We have also explored agent interaction with Microsoft Component Object Model (COM) underlying OLE (Brockschmidt, 1994) , and are currently extending our investigations to Distributed COM, ActiveX, and Java.

We are encouraged by the increased cooperation among research teams and product development groups working on agent technology. For example, we are closely following the progress of the Mobile Agent Facility, currently being defined by the common facilities task force of the Object Management Group (OMG) (Chang & Lange, 1996; Lange, 1996; Virdhagriswaran, Osisek, & O'Connor, 1995) . We have also been active participants in the Hippocrene project of the Aviation Industry Computer-Based Training Committee (AICC) (Bradshaw, Madigan, Richards, & Boy, 1993; Bradshaw, Richards, Fairweather, Buchanan, Guay, Madigan, & Boy, 1993) and are working with members of the KQML subgroup of the knowledge-sharing initiative to better understand and resolve interoperability issues. As research progresses, we will continue to advocate industry-wide agent interoperability standards that are neutral with respect to particular hardware platforms, operating systems, transport protocols, and programming languages.

Providing an extensible language semantics through an agent communication meta-architecture. It is challenging to define an architecture that is general enough to be implemented in many different ways and applied to diverse problems, yet specific enough to guarantee support for the requirement of agent interoperability. A prime example of this difficulty is the CORBA specification, which required successive refinement over a period of years until sufficient experience and consensus was attained that cross-vendor interoperability could be assured.

The KAoS architecture dictates neither the particular transport-level protocol, nor the form in which content should be expressed, and allows agents to be configured with whatever set of communication primitives is desired. For this reason, it may be properly regarded as an open agent communication meta-architecture.

While not incompatible with languages such as KQML, KAoS provides a more flexible and robust foundation for industrial-strength agents. We have optimized the architecture for extensibility so that new suites of protocols and capabilities can be straightforwardly accommodated as needed. Our goal is not to lead the invention of new languages and methods of agent interaction but rather to anticipate and easily adapt to new research, standards, and domain-specific enhancements as they emerge in the future. If desired, for example, the set of KQML "performatives" or the communication primitives of some future specialized agent language could easily be implemented within KAoS.

2. OVERVIEW OF KAoS

2.1. KAoS Agents

A consistent structure provides mechanisms allowing the management of knowledge, commitments, choices, and capabilities. These structures are shown in the box on the right of figure 1. Knowledge is defined as a collection of facts and beliefs. Facts are simply beliefs about the agent and the environment in which the

agent has complete confidence. Facts or beliefs may be held privately or potentially made public (e.g., using a blackboard). Desires represent the goals and preferences that motivate the agent. Intentions represent the commitment of the agent to being in a state where it believes it is about to actually perform some set of intended actions (Cohen & Lesvesque, 1990) . All agents are required to appropriately handle external requests to provide information about their structure. An appropriate response might be sometimes simply, "I am unable to give you the information you request."

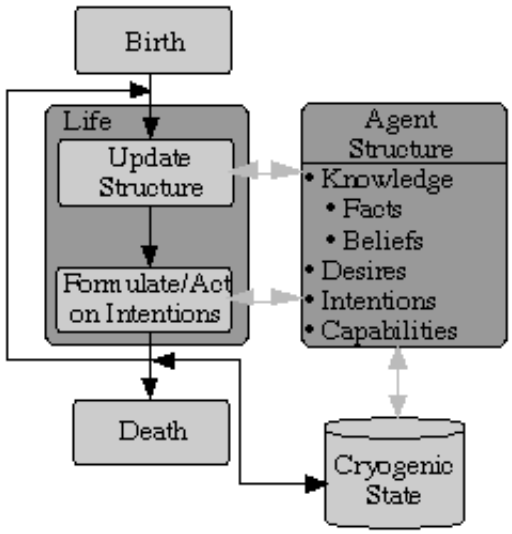

Figure 1. Structure and dynamics of agents. The black arrows represent state transitions, and the gray arrows data flows.

While the KAoS architecture provides the "hooks" for implementing sophisticated agents based on these structures and related mechanisms, it does not require that agents use these hooks in an "intelligent" fashion. The minimal requirement is that agents be able to carry out successful conversations related to services they are requesting or ones which they have advertised-the determination of the mechanisms by which this is accomplished is left to the agent designer.

Capabilities are the services or functions that an agent can provide as defined in specific extensions to the generic agent implementation. Our goal is to allow as much flexibility as possible in how agent capabilities are defined. For example, on the Windows platform, generic agents are currently implemented as OLE servers, and on the Macintosh platform generic agent functionality is exposed through Apple Events. A Java implementation of KAoS is currently being designed. Because KAoS relies on these popular messaging schemes for communication with the generic agent, agent capabilities can be defined or extended straightforwardly using any combination of standard programming languages, general-purpose scripting languages (e.g., AppleScript, Visual Basic, Tcl, Perl, JavaScript) and declarative logic-based programming languages (e.g., KIF, Prolog). We see this kind of extensibility as being a positive step toward the eventual (more ambitious) goal of powerful visual end-user authoring environments wherein complete agents can be defined.

Each agent goes through the equivalent of birth, life, and death. At birth, agents instantiated and initialized with some amount of innate structure. During their lives, agents go through a continuous cycle of reading, processing, and sending messages. Agent death poses special problems. Depending on the application, it may be necessary to include domain-specific procedures for dealing with it. These may include notification of other agents, transfer of any pending commitments, or transfer of knowledge. KAoS agents that are declared

as persistent must be able to go into a form of “suspended animation” (called cryogenic state). Each persistent agent is responsible for saving the aspects of its structure required allow it to be reactivated when required. The process of saving and restoring structure may also be simple or complex, depending on the situation.

Each agent contains a generic agent instance, which implements as a minimum the basic infrastructure for agent communication (figure 2). Specific extensions and capabilities can be added to the basic structure and protocols through standard object-oriented mechanisms. Mediation agents provide an interface between a KAoS agent environment and external entities, resources, or agent frameworks. The Domain Manager controls the entry and exit of agents in a domain according to policies set by the domain administrator. The Matchmaker can access information about the location of the generic agent instance for any agent that has advertised its services. Proxy agents extend the scope of the agent-to-agent protocol beyond a particular domain.

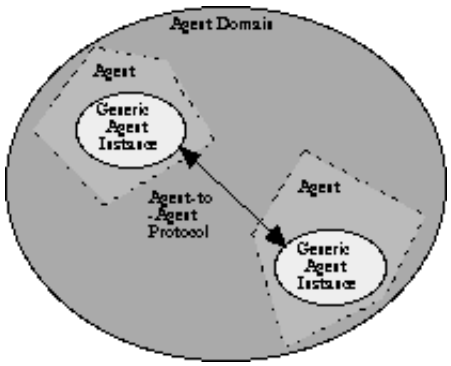

Figure 2. Communication between generic agent instances

2.2. Agent Communication

Conversations. Unlike most agent communication architectures, KAoS explicitly takes into account not only the individual message in isolation, but also the various sequences in which a particular message may occur. We believe that social interaction among agents is more appropriately modeled when conversations rather than isolated illocutionary acts are taken as the primary unit of agent interaction. As Winograd and Flores observe:

> The issue here is one of finding the appropriate domain of recurrence. Linguistic behavior can be described in several distinct domains. The relevant regularities are not in individual speech acts (embodied in sentences) or in some kind of explicit agreement about meanings. They appear in the domain of conversation, in which successive speech acts are related to one another (Winograd & Flores, 1986, p. 64) .

We define a conversation to be a sequence of messages between two agents, taking place over a period of time that may be arbitrarily long, yet is bounded by certain termination conditions for any given occurrence. Conversations may give rise to other conversations as appropriate. Messages occur only within the context of conversations. Each message is part of an extensible protocol common to the agents participating in the conversation. The content portion of a message encapsulates any semantic or procedural elements independent of the conversation policy itself.

Conversation policies. A major issue for designers of agent-oriented systems is how to implement policies governing conversational and other social behavior among agents. Walker and Wooldridge (Walker & Wooldridge, 1995) have termed the two major approaches: off-line design, in which social laws are hard-wired in advance into agents, and emergence, where conventions develop from within a group of agents.

For performance reasons, and also because the deeper logic of conversations has yet to be satisfactorily articulated by researchers, our current architecture provides only for an off-line approach. Just as the KQML agent protocol embodies a separate linguistic messaging layer allowing agents to circumvent the inefficiencies that otherwise would be imposed by the contextual independence of KIF's semantics (Genesereth, 1996) , KAoS provides an explicit set of mechanisms encoding message-sequencing conventions that, in most situations, frees agents from the burden of elaborate inference that otherwise might be required to determine which next message types are appropriate.

Shared knowledge about message sequencing rules enables agents to coordinate frequently recurring interactions of a routine nature simply and predictably. However nothing in the architecture precludes a more sophisticated appraoch, based on an emergent model of agent communication and the notions of joint intention and planning. Cohen and Smith have begun the development of a semantics of agent communication whtat would allow just such an approach (Cohen & Levesque, 1996) , and in the future we expect to build on their work to demonstrate how agent communication protocols can be developed and analyzed using such a semantics.

Messages, verbs, and suites. Facilities for implementing conversation policies and carrying out conversations are built into the generic agent capability. A starter set of conversation policies (the Core suite) is also provided, but can be replaced or extended as needed. The conversation policies of the default core suite currently consist of Inform, Offer, Request, Conversation for Action (CFA), and Query .

Inform. The simplest case of a conversation is Agent A sending a single message to Agent B with the "no response required" option enabled (figure 3). In such a case, Agent B terminates its side of the conversation "silently" and the conversation policy reduces to the kind of atomic message sending encountered in most agent communication languages. A slightly more complex example would be when Agent A requires Agent B to acknowledge receipt of the information. This it does by including a "response required" parameter within the initial message.

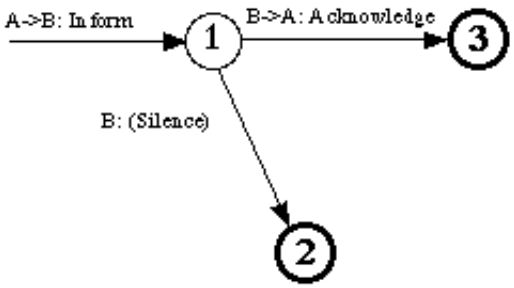

Figure 3. The Inform conversation policy

Offer. Whereas the effect of an inform message is immediate, an offer is future-oriented. Hence an offer is something that can be declined, while it is impossible to decline to be informed once one already has processed the content of an inform message (figure 4). As an example, a monitoring agent could initiate an Offer conversation with another agent that it perceived could benefit from its assistance.

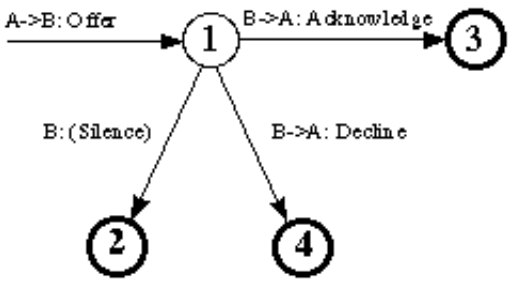

Figure 4. The Offer conversation policy

Request. The conversation policy for a Request is shown in figure 5. This kind of conversation policy (as opposed to the Conversation for Action policy below) is best suited to an agent that known to reliably fulfill its commitments, or for which the consequences of its failure to do so are slight. In the simplest case, Agent B can simply perform the request of Agent A, with an optional acknowledgment. The request may also be declined or countered by Agent B. Agent A can in turn counter again, accept the request, or withdraw it at any time. Once the request has been carried out by B, it sends the report satisfied message to A with results returned in the content portion.

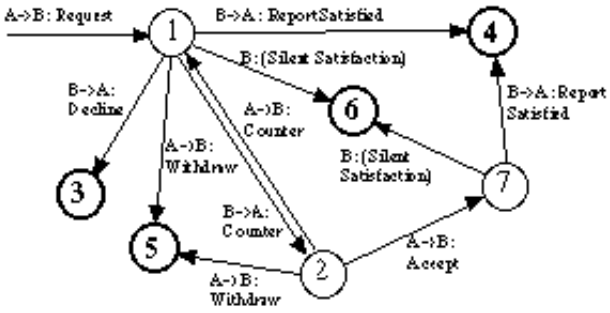

Figure 5. The Request conversation policy

We note here that there is a tradeoff between economy of verb types and "naturalness" of expression within a given conversation policy. For example, one could argue that acknowledge (in the Offer policy) and report satisfied (in the Request policy) should be replaced by simple inform messages. On the other hand, it is clear that the use of the more specific verbs makes it easier to infer the function of the messages in the context of their respective conversation policies.

This tradeoff between economy and naturalness of expression is an issue which cries out for additional study. Based on our informal analysis, we believe that the most common types of more specific verbs can be straightforwardly derived from the formal definitions of a small number of basic speech acts.

Conversation For Action. We regard Winograd and Flores' Conversation For Action (CFA) (Winograd & Flores, 1986) as a more complex variant of request (figure 6). We include a slightly modified version of their Conversation For Action in our core set of conversation policies, since it seems well-suited to many of the requests both that agents make of each other and that humans make of agent systems.

In contrast to the Request conversation policy, Conversation for Action provides a more complex mechanism to handle commitments that persist over time and may not be reliably fulfilled. Additional conversations may well be generated, as the agent negotiates with others to fulfill its commitments. The important feature to note

in the state-transition diagram is that communication about commitments is handled explicitly: a definite promise must be communicated if B accepts A's initial request, and if B does not intend to fulfill its commitment, it must send a renege message to A. A in turn must declare explicitly that it either will accept or decline the report from B that the request has been satisfied.

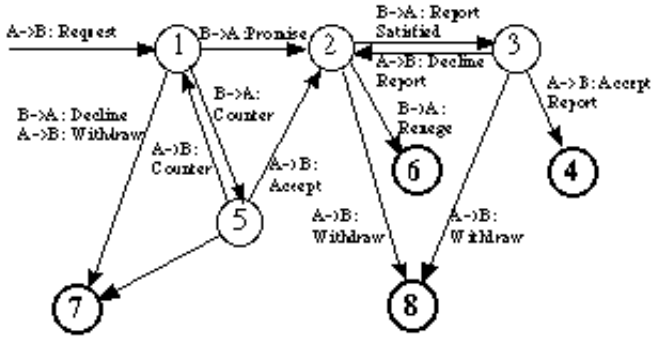

Figure 6. The Conversation For Action (CFA) conversation policy

Asynchronies in conversations. The implementation of a conversation policy must account for asynchronies in conversations (Bowers & Churcher, 1988) . The major asynchrony of concern is between the time of transmission of a particular message and the time of response.

To handle asynchronies, a conversation policy must be designed to prevent a conversation entering a state from which it cannot process an incoming message. An asynchrony will manifest itself as an attempt to effect an invalid transition on a conversation, and should occur only when more than one participant in a conversation can instigate a valid transition from a state. For example, in figure 12, transitions from state 2 allow messages from either A or B. Each transition from such a state will conform to one of the following rules:

- the transition leads directly to a final state, in which the conversation will no longer exist to process another incoming message. For example, B:A renege in figure 6 leaves the conversation in state 6, from which an A:B withdraw is irrelevant.
- the transition leads to a non-final state from which any message from another participant valid in the originating state is still valid-for example, because A:B withdraw is valid from both states 1 and 2 in figure 12, it will have the desired effect even if B:A promise moves the state from 1 to 2.

Conversation policy implementation requirements. The agent initiating a conversation specifies the opening verb and a conversation policy for a conversation, and the responding agent must indicate in return that it is capable of processing both the opening verb and the conversation policy. In implementing a conversation policy, all agents which participate in a conversation will-by definition-correctly generate and interpret all subsequent messages in the conversation.

The capability to implement a conversation policy entails:

- recognizing incoming messages correctly
- generating appropriate outgoing messages
- making the correct state transitions

Suites. A suite provides a convenient grouping of conversation policies that support a set of related services.

The default Core suite of initial verbs and conversation policies is normally available to all agents. In addition to the Core suite, special agents such as the Matchmaker would be expected to process at least one additional set of conversations (i.e., the Matchmaker suite).

The table in figure 7 below represents a conceptual model of the relationship between the basic elements of the Core suite, omitting the Query conversation policy which is introduced in section 3.4.6 below. Information about the relationship between a verb and a conversation policy is shown within the cells: an I (initial) shows that the verb may act as an initial verb and specify the conversation policy for a new conversation; an S (subsequent) shows that the verb may be used during the course of an existing conversation. An S in parentheses indicates that the use of the verb within a given conversation policy is optional in some contexts (e.g., acknowledgment of inform messages is not always required).

Figure 7. The basic elements of the Core suite, omitting Query

Rôles. In a typical conversation, the agent requesting a service will select the suite to be used for the conversation. The agent providing the service must have already advertised the service and the set of suites which it requires. Having done so, the two agents may then participate in a conversation, using an appropriate conversation policy in the selected suite.

Since a service-providing agent cannot make its services known to the Matchmaker without first advertising their existence, and since a service-requesting agent cannot access the required services for the first time without having the Matchmaker recommend an appropriate agent, every agent must have access to the Matchmaker suite (described below). However, there is an important difference between non-Matchmaker and Matchmaker agents in how they will participate in such conversations: the former will only need to know how to initiate advertising and recommending conversations in the rôle of a service requester, while the latter must how to process them as a service provider.

Rôles serve to partition the available messages, such that a given agent need not implement verbs and conversation policies in ways that it will never use. For example, most KAoS agents will be capable of playing advertiser or requester rôles in conversations with the Matchmaker, but only the Matchmaker agent itself will need to implement capabilities and roles relevant to the processing of advertise and recommend messages generated by others.

Rôles and suites. A suite maintains the permissible combinations of initial verb, conversation policy, and rôle. It must specify at least two rôles (e.g., one for the initiator of the conversation and one for the respondent). Where appropriate, agents may be permitted to play more than one possible rôle for a given conversation policy. For example, a Matchmaker may act as a service provider during the course of processing a recommend conversation for a requesting agent. However, in order to carry out the request, it may subsequently act in the rôle of a service requester by initiating a recommend conversation with another Matchmaker in order to have its assistance in locating service providers consistent with the original recommend request.
From figure 7, we see that the Core suite provides the following combinations of initial verb, conversation policy, and rôle for agents which initiate conversations:

- inform, Inform, informer
- offer, Offer, offerer
- request, Request, requester

- request, CFA, requester

The initial verb of a conversation determines the rôle for the agent originating the conversation. For example, any agent generating an inform or request verb necessarily acts as an informer or requester, and the agent receiving either of these messages will automatically adopt the rôle or rôles needed to process these incoming messages.

Requirements for conversation initiators and respondents. To allow communication with other agents, each agent must be designed to support one or more conversations. Being a conversation initiator or respondent requires an agent to do the following:

- for one or more combinations of suite, conversation policy, initial verb, and rôle:
- implement the conversation policies
- implement the capabilities necessary to process messages appropriate to its rôles in the conversations
- if an initiator, generate the initial verb.

Requirements for agents providing a suite of services. Providing a suite of services entails that an agent must be capable of adopting an appropriate rôle for each conversation in that suite. In other words, an agent must do the following:

- implement all the suite's conversation policies
- implement the capabilities necessary to process messages appropriate to its rôles as a service provider within instances of those conversation policies.

Example of Adding a New Conversation Policy: Query. Though the starter set of conversation policies defined in KAoS seems adequate for many common sorts of agent interaction, there will often be a need to add new ones. We will illustrate how this is done by adding a Query conversation policy to complete the partial Core suite shown in figure 7. The query verb can initiate either a CFA conversation policy whose state transitions are identical except for the initial verb, or a new Query conversation policy (figure 8). The major difference between the Query and Request conversation policies is that the B:A report satisfied message is not optional, and it must by definition contain some result (i.e., a response to the query) as part of its content.

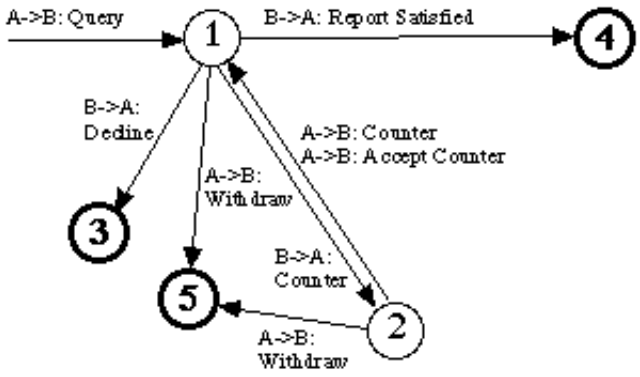

Figure 8. The Query conversation policy

Consistent with the state transition diagram, the table below shows that the query conversation protocol is identical to the request conversation protocol except that the use of the report satisfied verb is required rather than optional (figure 9). The shaded cells show what has been newly added: one conversation policy, one

verb, and the participation information.

Figure 9. Completing the Core suite by adding the Query conversation policy

Example of Conversation Policy Reuse: The Matchmaker Suite. One challenge addressed by the KAoS architecture is how to enable developers and consumers of agent services to add a new suite with minimal effort. For example, if a request for a new service could be made by reusing an existing conversation policy combined with a new initial verb, developers could often be spared the trouble of creating a whole new conversation policy and making it available to each potential requester. In the simplest case, any agent desiring access to the service which had already implemented the conversation policy being reused would simply have to extend its data about supported suites with a new initial verb. In many cases, not only the conversation policy but also many of the agent-specific handlers that process the messages of the conversation policy (e.g., countering) could be reused.

As an example, the Matchmaker suite is shown as the shaded area of figure 10. The suite is implemented by combining existing conversation policies with three new initial verbs: retire, advertise, and recommend. The advertise message is sent to the Matchmaker by any agent wishing to offer services. It uses the Offer conversation policy with a more specific verb. The retire message is used by an agent to withdraw its services. It uses the Inform conversation policy. The recommend message is used to request the Matchmaker's help in finding an agent to perform some service. Recommend uses the Query conversation policy.

Core	Inform	Offer	Request	CFA	Query
inform	I				
acknowledge	(S)	(S)			
offer		I			
decline		(S)	S	S	S
request			I	I	
counter			S	S	S
accept			S	S	S
withdraw			S	S	S
promise				S	
report satisfied			(S)	S	S
accept report				S	
decline report				S	
declare satisfied				S	
renege				S	
query				I	I
Matchmaker					
advertise		I			
retire	I				
recommend					I

Figure 10. The Core and Matchmaker suites.

The Matchmaker suite thus provides the following combinations of initial verb, conversation policy, and rôle for agents which originate conversations:

- advertise, Offer, advertiser
- retire, Inform, retiree
- recommend, Query, requester

3. APPLICATIONS

Initial prototypes and agent utilities. Early versions of KAoS were used to build demonstrations of agent-oriented programming and simulations of various agent activities. The first prototype implemented a multi-agent version of a battleship game, defining specializations of the generic agent class for one or many cooperating ship captains on each team, a game board Matchmaker, an Excel spreadsheet mediation agent, and a referee (Atler, Bingle, Cooke, Morgan, Duine, & Zonczyk, 1994; Tockey, Rosenthal, Rosman LaFever, Jasper, George, Woolley, Bradshaw, & Holm, 1995) . A maintenance performance support prototype demonstrated how mediation agents could help coordinate the interaction between airline maintenance mechanics and their supervisors and adapt the presentation of task-related information through a dynamic OpenDoc component interface (Bos, Boyer, Dutfield, & Jones, 1995) . A scheduling environment prototype based on Microsoft OLE technology showed how KAoS could be used to implement assistants to aid in the process of scheduling meetings and meeting rooms (Barker, Benoit, Tomlinson, & Landon, 1995) .

We have created an agent construction kit prototype based on Microsoft Foundation Classes for the Windows platform, and a visual interface construction kit prototype using HyperCard on the Macintosh. We created a conversation monitor to allow particular sets of agent conversations to be logged, passively monitored, or

intercepted. A Service Viewer provides a view on the services currently registered with a Matchmaker, and an Agent Structure Viewer allows one to inspect the persistent state of a particular agent.

Gaudi intelligent maintenance performance support system. The Boeing Company is exploring the use of portable airplane maintenance aids (PMA) to provide training and support to customers (Bradshaw, Richards, Fairweather, Buchanan, Guay, Madigan, & Boy, 1993; Guay, 1995) . A new version of KAoS is being incorporated into one such prototype of an intelligent performance support system (Bradshaw, Robinson, & Jeffers, 1995) . The system, named Gaudi, is designed around the actual processes, activities, and resources of the work environment. It is intended to directly and actively support necessary tasks, adapting information to the requirements of the user and situation.

Seven requirements guide Gaudi's evolution in the long-term:

1. Think tasks, not documents. The current transition in desktop computing is from an application-centric to a document-centric paradigm. Distributed component integration technologies (e.g., WWW, OpenDoc, ActiveX, Java) are fueling this trend. However, as component integration technologies increase in power and flexibility, user interfaces will move beyond a document-centric approach to a task-centric one. Large undifferentiated data sets will be restructured into small well-described elements, and complex monolithic applications will be transformed into a dynamic collection of simple parts, driving a requirement for new intelligent technology to put these pieces back together in a way that appropriately fits the context.

2. Pave where the path is. This phrase comes from the old story of the college planner who built a new campus with no paths built in at all (Brand, 1994, p. 187) . After the first winter, she photographed where people made paths in the snow between the buildings, and paved accordingly in the spring. The lesson is that some elements of the design of the system need to be postponed, and learned instead through actual experience with the user. As part of a collaboration with NASA-Ames, we are working to incorporate their adaptive engine into Gaudi. The adaptive component is described in more detail in section 5.3 below.

3. Make all parts replaceable. The idea is that future users of such a system would be able to easily add to or replace the software applications Boeing provides with applications of their own choosing in conjunction with their own or Boeing-provided data. A migration path from legacy monolithic applications to distributed component-based software must also be provided.

4. Link to anything (without requiring markup). SGML and HTML-based software typically provides for hyperlinking based on embedded markup of textual data. However embedded markup becomes problematic (Malcolm, Poltrock, & Schuler, 1991) :

- where context-sensitive linking is needed, since appropriate links may vary according to the user, task, or situation;
- where linking needs to be added after the fact to data provided in a read-only format such as CD-ROM, or
- where the unpredictable nature of the content requires dynamic query-based links rather than static pre-determined ones.

Additionally, new techniques need to be developed to allow linking to complex data elements

such as individual frames in a video stream or pieces of 3D geometry. We have implemented an agent-assisted external linking facility that implements dynamic links without markup.

5. Run it everywhere. This requirement underlines the necessity of developing a cross-platform approach (i.e., Mac, Windows, Unix). It also requires that progress in wearable and mobile computing platforms and networking approaches (such as developments in wireless communication) be taken into account.

6. Pull data from anywhere. Rather than delivering a closed-box containing a static set of Boeing data, users must be able to dynamically access and integrate data that may reside on a networked server. This data may include anything from a private airline spares database, to a Boeing-managed media server for digital video, to other sources of information residing anywhere on the public Internet. A special requirement is the ability to interoperate with object request brokers and message-based protocols.

7. Let your agents handle the details. The fragmentation of data into smaller-grain-sized objects and the decomposition of large applications into sets of pluggable components could prove a nightmare for users if there is no support to help them put all the pieces together again. KAoS agents will enable intelligent interoperability between heterogeneous system components, and will help filter and present the right information at the right time in the most appropriate fashion to users who would otherwise be overwhelmed by a flood of irrelevant data.

4. ISSUES AND FUTURE DIRECTIONS

Mobile agents. WWe are working on the issue of agent mobility on two fronts: 1) allowing mobile users of small computing devices to interact with a KAoS agent domain residing on a remote machine, 2) integrating the KAoS architecture with mobile agent approaches that permit the physical migration and secure, managed execution of agent programs on "guest" hosts not belonging to the sender of the agent.

With regard to the first issue, we have completed a prototype of a mediation agent serving a mobile client of the Gaudi application by a wireless connection. In the prototype, agents involved in a currently running session can be transferred from one client to another at the request of a user. Though the current session context is preserved in the transfer, the agents are responsible for adapting to the characteristics of the client platform as required. For example, a user can transfer a session running on a desktop with a high-resolution display to a laptop with a low-resolution display. The user interface will adapt to the new hardware configuration, and hyperlinks not appropriate for the new client (e.g., high-resolution multimedia) will be filtered out automatically. A serial connection is maintained only as needed between the mobile client and the machine on which KAoS is running. To preserve power and bandwidth, the connection is an intermittent rather than an exclusive, continuous one.

With regard to the second issue, we are beginning work on a Java implementation of KAoS and on enhancements to the KAoS OLE/ActiveX implementation to take advantage of DCOM. Agents will be able to transport themselves in two ways: 1) by transferring an entire agent from one domain to another (teleportation), or 2) by transferring only a mobile portion of the agent's capabilities (e.g., as an applet) to a different host (telesthesia). In this second scenario, the mobile portion of the agent could execute on remote hosts while remaining in communication with its generic agent in the home domain. Agent communication with other programs may be facilitated by the ability of agents to plug into an open protocol bus hosted as

part of client or server Web application services (e.g., LiveConnect on NetScape). We will incorporate industry standards for agent transfer protocols as they emerge (e.g., the dispatch, retract, and fetch verbs defined in (Lange, 1996)).

Formalizing semantics and modeling dialogue as a joint activity. To date, we have attempted no formal description of the semantics of agent communication. Ongoing progress in such formalizations is summarized by Cohen and Levesque (Cohen & Levesque, 1996) and Labrou and Finin (Labrou & Finin, 1994) . We anticipate continued collaboration with these and other researchers as this work moves forward.

A more general issue concerns the manner in which such a set of social laws (e.g., conversation policies, collaboration strategies, policies governing reconsideration of conventions) comes to exist within an agent society (Durfee, Gmytrasiewicz, & Rosenschein, 1994; Jennings, 1993; Shoham & Tennenholtz, 1992; Wooldridge & Jennings, 1994) . We have noted above the distinction between the approaches of off-line design and emergence of agent social behavior. While the off-line design of social laws generally makes for simpler design and more predictable agent behavior, we see value in allowing for emergent behavior where the situation demands (e.g., complex negotiations (Zlotkin & Rosenschein, 1994) , teamwork (Cohen & Levesque, 1991)).

For example, Cohen and Levesque (Cohen, 1994) discuss the limitations of “state models” of conversations, such as those we have proposed as part of the current KAoS architecture. While many of the problems they describe (nonliteral language, multifunctional utterances, etc.) are more important for human-human or human-agent than for agent-agent interaction using a very restricted language, they make a good case that, over the long term, “state model” (“dialogue grammar”) approaches need to function in concert with more powerful plan-based approaches that require agents to infer one another’s intentions at runtime.

The KAoS architecture assumes that agents identify each other by the services they advertise; such an environment need not treat random encounters between unrelated agents as a primary concern. Accordingly, the concept of “joint intention” is dealt with only implicitly by considering at design time the services and the rules within conversation policies associated with those services. Increasing the flexibility and power of agents will require elaboration of joint action theory.

Cohen and Smith have begun the development of a semantics of agent communication that would allow the rigorous analysis of conversation policies (Cohen & Levesque, 1996; Smith & Cohen, 1995) . Among other things, they have demonstrated that the behavior of the state transition model of the Winograd and Flores CFA policy is consistent with an emergent behavior of agents operating according to the principles in their model of interagent communication. The goal would be to predict the structure of finite-state models of interagent conversations as used in agent architectures such as KAoS. Such a strategy parallels the approach of Rosenschein, who designed a compiler that generates finite state machines whose internal states can be proved to correspond to certain logical propositions about the environment (Kaelbling & Rosenschein, 1990; Rosenschein, 1985) .

5. CONCLUSIONS

The KAoS architecture will succeed to the extent that it allows agents to carry out useful work while remaining simple to implement. Although it is still far from complete, our experience with the current KAoS architecture has shown it to be a powerful and flexible basis for diverse types of agent-oriented systems. The strength of the architecture derives from several sources:

- it is built on a foundation of distributed object technology and is optimized to work with component integration architectures such as OpenDoc, OLE, and Java and with distributed object services such as those provided by CORBA and DCOM;
- it supports structured conversations which:
- preserve and make use of the context of agent communication at a higher level than single messages,
- allow differential handling of messages depending on the particular conversation policy and the place in the conversation where the message occurs,
- permit built-in generic handlers for common negotiation processes such as countering;
- it allows the language of inter-agent communication to be extended in a principled manner, allowing verbs and conversation policies to be straightforwardly reused, adapted, or specialized for new situations;
- it groups related sets of conversation policies into suites supporting a coherent set services;
- it provides facilities for service names, which are registered by agents offering services;
- it provides facilities for agent names, which uniquely identify an agent as long as it persists;
- it is appropriate for a wide variety of domains and implementation approaches and is platform- and language-neutral;
- it supports simple agents to be straightforwardly implemented, while providing the requisite hooks to develop more complex ones;
- it supports both procedural and declarative semantics;
- it is designed to interoperate with other agent frameworks and protocols either by extending or replacing the core agent-to-agent protocol or by defining specialized mediation agents.

We are optimistic about the prospects for agent architectures built on open, extensible object frameworks and look forward to the wider availability of interoperable agent implementations that will surely result from continued collaboration.

References

Apple_Computer (1993). Inside Macintosh: Interapplication Communication., Reading, MA: Addison-Wesley.

Atler, D., Bingle, M., Cooke, T., Morgan, J., Duine, D. V., & Zonczyk, M. (1994). AgONy! Battleship Documentation. Seattle, WA: Seattle University Department of Software Engineering, June 9, 1994.

Barker, D., Benoit, P., Tomlinson, J., & Landon, S. (1995). KAoS CORBA Design Document. Seattle, WA: Seattle University Department of Software Engineering, May 30, 1995.

Betz, M. (1994). Interoperable objects. Dr. Dobb's Journal(October), 18-39.

Bos, M., Boyer, R., Dutfield, S., & Jones, E. J. (1995). TAINT OpenJob Design Document. Seattle, WA: Seattle University Department of Software Engineering, May 30, 1995.

Bowers, J., & Churcher, J. (1988). Local and global structuring of computer-mediated representation: Developing linguistic perspectives on CSCW in COSMOS. Proceedings of the Conference on Computer-supported Cooperative Work, . Portland, OR, , ACM ,

Bradshaw, J. M. (Ed.). (1996). Software Agents. Cambridge, MA: AAAI/MIT Press.

Bradshaw, J. M., Dutfield, S., Benoit, P., & Woolley, J. D. (1996). KAoS: Toward an industrial-strength generic agent architecture. In J. M. Bradshaw (Ed.), Software Agents. (pp. in preparation). Cambridge, MA: AAAI/MIT Press.

Bradshaw, J. M., Madigan, D., Richards, T., & Boy, G. A. (1993). Emerging technology and concepts for computer-based training. Proceedings of the Sixth Annual Florida AI Research Symposium (FLAIRS '93), . Ft. Lauderdale, FL, , ,

Bradshaw, J. M., Richards, T., Fairweather, P., Buchanan, C., Guay, R., Madigan, D., & Boy, G. A. (1993). New directions for computer-based training and performance support in aerospace. Proceedings of the Fourth International Conference on Human-Machine Interaction and Artificial Intelligence in Aerospace., . Toulouse, France, 28-30 September, , ,

Bradshaw, J. M., Robinson, T., & Jeffers, R. (1995). Gaudi: An unfinished architecture for performance support. P. C. Cacciabue (Ed.), Proceedings of the Fifth International Conference on Human-Machine Interaction and Artificial Intelligence in Aerospace (HMI-AI-AS '5), (pp. in press). Toulouse, France, , ,

Brand, S. (1994). How Buildings Learn: What Happens after They're Built., New York: Viking Penguin.

Brockschmidt, K. (1994). Inside OLE 2., Redmond, WA: Microsoft Press.

Campagnoni, F. R. (1994). IBM's System Object Model. Dr. Dobb's, Winter 1994/95, 24-28.

Chang, D. T., & Lange, D. B. (1996). Mobile agents: A new paradigm for distributed object computing on the WWW. Proceedings of the OOPSLA 96 Workshop "Toward the Integration of WWW and Distributed Object Technology", . , , ,

Cohen, P. R. (1994). Models of dialogue. T. Ishiguro (Ed.), Cognitive Processing for Vision and Voice: Proceedings of the Fourth NEC Research Symposium, (pp. 181-203). , , Philadelphia, PA: Society for Industrial and Applied Mathematics ,

Cohen, P. R., & Lesvesque, H. J. (1990). Intention is choice with commitment. Artificial Intelligence, 42(3), .

Cohen, P. R., & Levesque, H. (1996). Communicative actions for artificial agents. In J. M. Bradshaw (Ed.), Software Agents. (pp. in preparation). Cambridge, MA: AAAI/MIT Press.

Cohen, P. R., & Levesque, H. J. (1991). Teamwork. Technote 504. Menlo Park, CA: SRI International, March.

DeMichelis, G., & Grasso, M. A. (1994). Situating conversations within the language/action perspective: The Milan conversation model. R. Furuta & C. Neuwirth (Ed.), Proceedings of the Conference on Computer Supported Cooperative Work, (pp. 89-100). Chapel Hill, NC, USA, , New York: ACM Press ,

Durfee, E. H., Gmytrasiewicz, P., & Rosenschein, J. S. (1994). The utility of embedded communications: Toward the emergence of protocols. M. Klein & K. Sharma (Ed.), Proceedings of the Thirteenth International Distributed Artificial Intelligence Workshop, (pp. 85-93). Seattle, WA, , Seattle, WA: Boeing Information and Support Services ,

Finin, T., Labrou, Y., & Mayfield, J. (1996). KQML as an agent communication language. In J. M. Bradshaw

(Ed.), Software Agents. (pp. in preparation). Cambridge, MA: AAAI/MIT Press.

Gardner, E. (1996). Standards hold key to unleashing agents. Web Week, 5, April 29,

Genesereth, M. R. (1996). An agent-based framework for interoperability. In J. M. Bradshaw (Ed.), Software Agents. (pp. in press). Cambridge, MA: AAAI/MIT Press.

Guay, R. L. (1995). Notebook simulations as electronic performance support tools for airline maintenance. N. R. Hartley (Ed.), Proceedings of the Royal Aeronautical Society Flight Simulation Group Simulation in Aircraft Maintenance Training Conference, . London, England, , ,

Jennings, N. R. (1993). Commitments and conventions: The foundation of coordination in multi-agent systems. Knowledge Engineering Review, 8(3), 223-250.

Kaelbling, L. P., & Rosenschein, S. J. (1990). Action and planning in embedded agents. Robotics and Autonomous Systems, 6(1-2), 35-48.

Labrou, Y., & Finin, T. (1994). A semantics approach for KQML-a general purpose communication language for software agents. N. R. Adam, B. K. Bhargava, & Y. Yesha (Ed.), Proceedings of the Third International Conference on Information and Knowledge Management, (pp. 447-455). Gaithersburg, MD, , New York: The Association for Computing Machinery ,

Lange, D. B. (1996). Agent Transfer Protocol ATP/0.1 Draft 4. IBM Research, Tokyo Research Laboratory, July 29.

Malcolm, K. C., Poltrock, S. E., & Schuler, D. (1991). Industrial strength hypermedia: Requirements for a large engineering enterprise. Proceedings of the Third ACM Conference on Hypertext, . San Antonio, TX, , ,

Orfali, R., Harkey, D., & Edwards, J. (1995). Client/server components: CORBA meets OpenDoc. Object Magazine, Maay, 55-59.

Robinson, M. (1991). Computer supported cooperative work: Case and concepts. In P. R. Hendriks (Ed.), Groupware 1991: The Potential of Team and Organisational Computing. (pp. 59-75). Utrecht, The Netherlands: SERC.

Rosenschein, S. J. (1985). Formal theories of knowledge in AI and robotics. New Generation Computing, 3(4), 345-357.

Shoham, Y., & Tennenholtz, M. (1992). On the synthesis of useful social laws for artificial agent societies. Proceedings of the Tenth National Conference on Artificial Intelligence, (pp. 276-28?). San Jose, CA, , ,

Smith, I. A., & Cohen, P. R. (1995). Toward a semantics for a speech act based agent communications language. T. Finin & J. Mayfield (Ed.), Proceedings of the CIKM Workshop on Intelligent Information Agents, . Baltimore, MD, , ACM SIGART/SIGIR ,

Suchman, L. (1993). Do categories have politics? The language/action perspective reconsidered. Proceedings of the Third European Conference on Computer-Supported Cooperative Work, (pp. 1-14). Milan, Italy, , Dordrecht, The Netherlands: Kluwer Academic Publishers ,

Tarrago, S. (1992). Gaudi., Barcelona, Spain: Editorial Escudo de Oro, S.A.

Tockey, S., Rosenthal, D., Rosman LaFever, M., Jasper, R., George, N., Woolley, J. D., Bradshaw, J. M., & Holm, P. D. (1995). Implementation of the KAoS generic agent-to-agent protocol. Northwest AI Forum (NAIF) Journal(Spring), .

Virdhagriswaran, S., Osisek, D., & O'Connor, P. (1995). Standardizing agent technology. ACM Standards View, in press.

von Martial, F. (1992). Coordinating Plans of Autonomous Agents., Heidelberg, Germany: Springer Verlag.

Walker, A., & Wooldridge, M. (1995). Understanding the emergence of conventions in multi-agent systems. V. Lessor (Ed.), Proceedings of the First International Conference on Multi-Agent Systems, (pp. 384-389). San Francisco, CA, , Menlo Park, CA: AAAI/MIT Press ,

White, J. (1996). A common agent platform. (http://www.genmagic.com/Internet/Cap/w3c-paper.htm). General Magic, Inc., 11 March.

Winograd, T., & Flores, F. (1986). Understanding Computers and Cognition., Norwood, N.J.: Ablex.

Wooldridge, M. J., & Jennings, N. R. (1994). Formalizing the cooperative problem solving process. M. Klein & K. Sharma (Ed.), Proceedings of the Thirteenth International Distributed Artificial Intelligence Workshop, (pp. 403-417). Seattle, WA, , Seattle, WA: Boeing Information and Support Services ,

Zlotkin, G., & Rosenschein, J. (1994). Rules of Encounter: Designing Conventions for Automated Negotiation among Computers., Cambridge, MA: MIT Press.