KAoS: A Generic Agent Architecture for Aerospace Applications

Jeffrey M. Bradshaw
Stewart Dutfield
Bob Carpenter
Renia Jeffers
Tom Robinson

1. Industrial-Strength Agents for Technical Information Management and Delivery

The complexity of modern engineered systems motivates the requirement for timely access to technical and operational documentation [11; 13]. Documents are both the most valuable and the most expensive knowledge resource in engineering organizations [25]. Product and product-related documents may be intended for use by thousands of people over a life-cycle of many years [52; 56]. Designers, engineers, operators, maintenance technicians, suppliers, and subcontractors often require access to the same documents, but for different purposes and with different perspectives and terminology. Because documentation specialists cannot anticipate all the circumstances and questions that may arise, they try to organize and index text, graphic, and multimedia in a context-free manner. People, however, resist reading manuals that describe system features in a task-neutral way [62]. Instead they use information retrieval strategies that are context-*dependent* [12; 15; 55]. For example, they remember that information about the diameter of a particular rivet was (or was not) relevant to the selection of a tool for repairing the fuselage. They organize their work by posting frequently-referred-to pages of a maintenance manual in prominent places in their work area, thus exploiting situational knowledge not available to the manual's original authors.

The rapidly growing amount and complexity of information available has compounded these problems. Until relatively recently, computing resources were so scarce and the bandwidth of human-computer interaction so low that the major problem was to increase access to online information [57]. Now the amount of data that can be manipulated is so overwhelming and the barriers to access so much more permeable that we need to be seriously concerned about how to actively, selectively keep only the most relevant information at the forefront of user interaction.

The promise of software agents. Software agents have been proposed as one way to help people better cope with the increasing volume and complexity of information and computing resources. 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, and networking capabilities to discover, link, and securely access the appropriate data and services. Documents assembled through the use of such agents are termed "virtual" because they may have never existed as such until the moment they were dynamically composed and presented through the current information lens. They are termed "adaptive" because the tools, content, and user interface learn to tailor themselves over time to the requirements of particular users and situations [23].

A variety of agent theories, architectures, and languages have been proposed.[1] Simple script-based agents have proven themselves useful in repetitive administrative tasks; more complex procedural agents have been applied to applications such as systems or network management [60; 63]. Additional agent work has focused on areas such as Internet resource discovery and

---

[1] See [81] for a survey of agent theories, architectures, and languages, and an annotated list of agent-based systems.

information integration [10; 22; 35; 36; 47; 73; 78; 80], intelligent coordination of distributed problem-solvers [33; 37; 39; 40; 44; 45; 48; 70], and active user assistance [4; 14; 51; 53; 64]. Yet other agent implementations are beginning to appear that will enable mobile agents to perform business transactions in a safe and secure manner [76; 77]. In contrast, the use of agents for context-dependent assembly of virtual documents from distributed information is a relatively new research area. The initial impetus has come from the explosion of distributed information on the public Internet [10], and is now being recognized as a requirement for organizations needing more flexible and dynamic access to private sources of heterogeneous, distributed information.

Problems in scaling, security, and agent interoperability. While several approaches to agent technology are showing significant promise, 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 scaling issues. Moreover, the complete lack of standards has raised concerns about agent interoperability. 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 only accommodate a single native language and a limited set of proprietary services to which it alone can provide access.

To address some of these deficiencies, we are developing KAoS (Knowledgeable Agent-oriented System) as part of an ongoing collaboration to develop an open distributed architecture for software agents. Section 2 provides the background of KAoS. We then present the aims and major components of the KAoS architecture (section 3): agent structure, dynamics, and properties; the relationship between agents and objects; and the elements of agent-to-agent communication. Following this, we briefly summarize past and current applications (section 4).

2. Background

MANIAC and the first version of KAoS. KAoS grows out of work beginning in 1988 on a general purpose interapplication communication mechanism for the Macintosh called MANIAC (Manager for InterApplication Communication) [16; 18]. In 1992, we began a collaboration with the Seattle University Software Engineering program to develop the first version of KAoS [42; 72]. We replaced the integrated planner with a fully object-oriented agent framework, borrowing ideas from Shoham's AGENT-0 work [67]. The following year, a new group of students replaced the MANIAC capability with HP Distributed Smalltalk's version of OMG's Common Object Request Broker Architecture (CORBA) [7; 68]..

Combining KAoS and CORBA. CORBA provides a means of freeing objects and agents from the confines of a particular address space, machine, programming language, or operating system [7]. 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 an *Agent Facility* (see below), and a compound document facility that will likely be based on an enhanced version of the CI Labs OpenDoc specification [58].[1]

During the 1994-1995 academic year, two groups of Seattle University Masters of Software Engineering students developed new implementations of KAoS. One team worked with a single-address-space version of IBM's System Object Model (SOM) [24] as a first step towards an eventual full CORBA-compliant implementation using Distributed SOM. Their prototype application used agents to provide content for OpenDoc parts. Another team worked with DEC's

---

[1] Within the OMG, work is underway to provide interoperability between Microsoft's Component Object Model (which underlies OLE) and CORBA. OpenDoc parts are already OLE-compatible out of the box. Various approaches to providing object system interoperability are discussed in [38].

ObjectBroker, and explored agent interaction with Microsoft Component Object Model (COM) underlying OLE 2 [21]. Both implementations were based on the new version of the KAoS architecture and agent-to-agent protocol described in section 3 below.

Future directions. Figure 1 illustrates the long-range context in which KAoS is being defined. We expect to continue development in both C++ and Smalltalk. Though we have optimized our implementation to address Boeing's current needs, we intended the general architecture to support rapid evolution. KAoS can be enhanced straightforwardly to enable interoperation with new agent-related standards and commercial products as they emerge. We plan to prepare future versions of the basic KAoS implementation that will be placed in the public domain where they can be evaluated and improved upon.

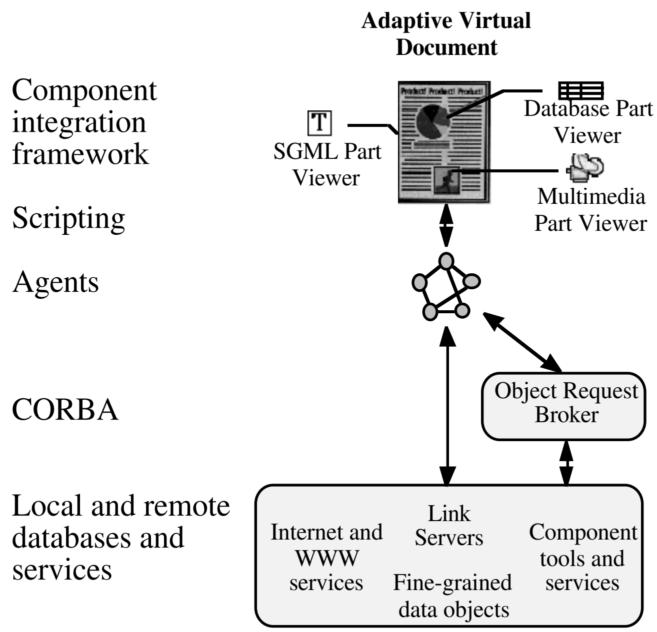

Figure 1. The long-range context in which the KAoS agent architecture is being defined.

As we develop and evaluate elements of the KAoS architecture, we also want to encourage increased cooperation among research teams and product development groups working on agent technology. To this end, we are participating in the definition of the *Agent Facility*, currently being proposed to the common facilities task force of the Object Management Group (OMG) [74]. We have also been active participants in the Hippocrene project of the Aviation Industry Computer Based Training Committee (AICC) and are collaborating with members of the KQML subgroup of the knowledge-sharing initiative to resolve interoperability issues. Agreements are being formalized so that initial work can begin on demonstrations of interoperability between KAoS, ACL [40], Telescript [77], and the LogicWare server which incorporates a version of KQML [73]. 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.

3. KAoS Architecture

3.1. Overview

Aims and scope. The architecture seeks to provide the following:

- a form of agent-oriented programming, based on a foundation of distributed object technology;

- structured conversations between agents, that may preserve their state over time;
- a framework to extend the language of inter-agent communication in a principled manner, taking into account:
  - the repertoire of illocutionary acts ("verbs") available to agents
  - the repertoire of conversation policies available to agents
  - the content of messages;
- an environment in which to design agents to engage in specialized suites of interactions.

Communication, content, and context in KAoS messages. Like KQML [37], we make a distinction between communication, content, and contextual portions of agent messages. The communication portion encodes information enabling proper message routing, such as the identity of the sender and recipient. The content portion contains the actual gist of the message (e.g., the specific request or information being communicated), and may be expressed in any notation or form desired, including binary programs. The contextual portion describes the type of message being sent (e.g., request, inform) and how it relates to the larger scope of the particular conversation taking place. Optionally, the message context may also contain other descriptive information, such as the language used to express the content and (if the content is declarative) references to particular ontologies associated with it. The combination of all these features allow agents to properly analyze, route, and deliver messages without requiring interpretation of content.[1]

KAoS as an agent communication meta-architecture. A major challenge is to define an architecture that is general enough to be implemented in many different ways and applied to diverse problems, yet specific enough to support a prime goal of agent interoperability. A prime example 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.

Because KAoS architecture neither dictates particular transport-level protocol, nor the form in which content should be expressed, and since agents can be configured with whatever set of communication primitives is desired, it may be properly regarded as an agent communication *meta*-architecture. By providing a basic framework defining the set of assumptions for agent communication to take place, we hope to lay a robust foundation for the definition of any number of interoperable agent communication language implementations that may prove useful in the future.[2]

Basic characteristics of agents are described in section 3.2. A consistent structure provides mechanisms for managing knowledge, commitments, choices, and capabilities. Agent dynamics are managed through a cycle that includes the equivalent of agent birth, life, and death.

Section 3.3 describes the relationship between agents and objects. Each agent contains exactly one *generic agent instance,* which implements a minimal set of capabilities. 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 or resources.

Messages are exchanged between agents in the context of *conversations* (section 3.4). *Verbs* name the type of illocutionary act (e.g., *request, promise*) represented by a message. 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

---

[1] The KAoS architecture neither requires interpretation of content by broker and facilitator agents, nor disallows it when it is possible and desirable to do so. By way of comparison, Finin's description of KQML [37] states that every implementation "ignores the content portion of the message", whereas Genesereth's ACL (Agent Communication Language) description of KQML currently makes the commitment to KIF (Knowledge Interchange Format) as the content language, so that the content is always available for interpretation [40].

[2] If a particular third-party agent implementation does not conveniently lend itself to direct implementation or emulation in KAoS, one or more mediation agents can be defined to act as a gateway between the disparate agent worlds (see section 3.3.3 below).

frequently recurring interactions of a routine nature simply and predictably. *Suites* provide convenient groupings of conversation policies that support a set of related services (e.g., the *Service Broker suite*). A starter set of suites is provided in the architecture, but can be replaced or extended as needed.

Extensibility. The current version of the architecture aims only to specify those generic capabilities which are basic to agent communication. The fundamental feature of the architecture is that it provides a robust, extensible foundation for agents needing to communicate with anything, anywhere. Additional agent features and capabilities may be defined within specific agent subtypes. The implementation inheritance features of some ORBs (e.g., DSOM) and object-oriented programming languages allow new agents to be defined simply, by adding or redefining only the differences between themselves and their more general ancestors.

Scope of the current work. The current architecture does not address some important areas of agent research, including:

- end-user authoring
- mobile agents
- semantics of agent communication
- joint intention
- planning
- complex negotiation
- vague goal specification
- learning and adaptive behavior
- anthropomorphic presentation
- message translation

A discussion of these and other issues and potential enhancements is beyond the scope of this paper.

3.2. Basic Characteristics of Agents

Agent-oriented programming [67] is a term that has been proposed for the set of activities necessary to create software agents. In the context of KAoS, an agent can be thought of as an extension of the object-oriented programming approach, where the objects are typically somewhat autonomous and flexibly goal-directed, respond appropriately to some basic set of speech acts (e.g., request, offer, promise), and ideally act in a way that is consistent with certain desirable conventions of human interaction such as honesty and non-capriciousness.[1] From this perspective, an agent is essentially "an object with an attitude."

But it is important to note that an agent's "attitude" is not really an *attribute* but rather an *attribution* on the part of a person. That is what makes coming up with a once-and-for-all definition of an agent so difficult: one person's agent is another person's "smart object"; and today's "smart object" is just a few years away from being seen tomorrow as just another "dumb program." The key distinction is in our point of view. For agent proponents, the claim is that just as some algorithms can be more easily expressed and understood in an object-oriented representation than a procedural one [46], so it sometimes may be easier for developers and users to think in terms of intentional agents instead of passive objects [31].[2]

3.3. Agents and Objects

The KAoS architecture defines a basic structure and core speech-act-based agent-to-agent protocol to be shared among all agents. To this basic structure and protocol, specific extensions

---

[1] It is still too early to tell if agent-oriented programming will require fundamentally different models of software development [59] and user-interface design [34; 41].

[2] Russell and Norvig [66, p. 821] discuss the fact that while the concept of an intentional stance might help us avoid the paradoxes and clashes of intuition, the fact that it is rooted in a relativistic folk psychology can create other sorts of problems. Resnick and Martin [54; 61] describe examples of how, in real life, people quite easily and naturally shift between the different kinds of descriptions of designed artifacts. See Erickson [34] for an additional useful perspective on the advantages and disadvantages of encouraging users to think in terms of agents.

and capabilities can be added as needed through standard object-oriented mechanisms. Communication between agents takes place through the use of *messages*. A message consists of a packet of information, generally sent asynchronously, whose type is represented by a *verb* corresponding to some kind of illocutionary act (e.g., *request, inform*). Messages are exchanged by agents in the context of *conversations*. Each message is part of an extensible protocol—consisting of both message names and conversation policies—common to the agents participating in the conversation. Figure 3, modified from [67], enumerates distinctions between communication in classical object-oriented programming and in the agent-oriented architecture defined here.

	Objects	**Agents**
Basic Unit	instance	agent
Parameters defining state of basic unit	unconstrained	beliefs, commitments, capabilities, choices,…
Process of computation	operations	messages
Message types	defined in classes	defined in suites
Message sequences	defined implicitly	defined in conversations
Social conventions	none	honesty, consistency,…

Figure 3. Objects versus agents.

Figure 4 identifies the characteristics of an operation, and compares these to the characteristics of a message. By "operation," we mean the invocation of a procedure or a method.

Operation	**Message**
Operation name	Verb
Signature	Conversation Parameters Content
Return Value	(none)

Figure 4. Message characteristics

Though operations may take place in isolation, messages only occur in the context of a conversation. The meaning of a given operation may vary between instances of different classes, but a message always has a meaning defined by its place in a particular conversation (section 3.4). For example, a *decline* message in the context of an *Offer* conversation, may mean something different than the same message in the context of a *Request.*

The parameters of a message contain any necessary meta-information about message processing (e.g., maximum response delay, whether acknowledgment is required) and the message content (e.g., content language). The message content portion itself may contain either declarative knowledge structures (embedded verbs) or program code. A detailed discussion of the content falls outside the scope of this document.

3.3.1. Composition of Agents

An agent contains an instance of the generic agent class (or of a subclass of that class) which is called the *generic agent instance*. The generic agent class implements a minimal set of capabilities, sufficient to participate in a small set of basic conversations. Figure 5 shows intercommunicating agents that contain instances of the generic agent class. A composite agent is an agent that is comprised of more than one agent.

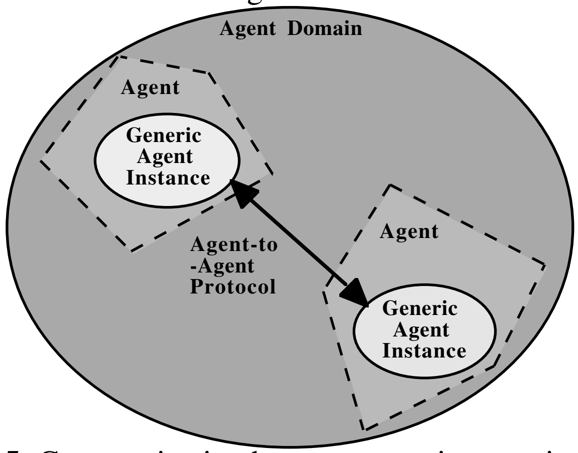

Figure 5. Communication between generic agent instances.

The agent *domain* denotes the extent of inter-agent messaging; that is, no agent can communicate directly with any agent across the bounds of an agent domain. For example, in a CORBA environment the bounds of the agent domain would be the bounds of the distributed ORB.

Specific capabilities of particular agents may be defined by any combination of *inheritance* (i.e., by creating specialized subclasses of the generic agent class) or *aggregation* (by incorporating instances of other classes).[1] Figure 6 shows an agent implemented as an aggregation of two owned object instances with a single instance of the generic agent class. Some owned instances may be very active, others may function passively as inert data holders; they all have in common the inability to communicate directly using the agent-to-agent protocol.

Examples of owned object instances might include:

- reusable encapsulations of capabilities such as inference engines
- encapsulations of internal resources over which the agent has exclusive control, such as knowledge and commitments
- representations of external resources over which the agent has exclusive control, such as a mailbox

[1] In object-oriented programming literature, aggregation means one of two things: 1. An alternative to inheritance, used by COM, in which a class picks and chooses attributes and methods—or groups thereof—from other classes [21]. The new class has a subset of the union of the attributes and methods from the other classes, together with any attributes and methods which the new class introduces; 2. The sense in which we use it here—namely, the composition of an entity (such as an agent) from several instances.

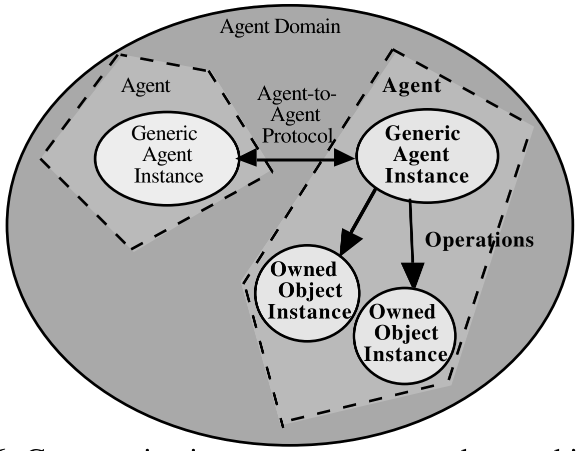

Figure 6. Communication among agents and owned instances

If an instance is not part of an agent, then any instance which is part of any agent may interact with it directly. Examples of this may include:

- encapsulations of internal resources over which the agent does not have exclusive control. For example, instances of agent conversation objects might, in a particular KAoS implementation, be instances shared between the participating agents
- representations of external resources over which the agent does not have exclusive control, such as a display or an ODBMS.

Agents with non-exclusive control over external resources are called *mediation agents* (see section 3.3.3).

3.3.2. Agent Environments: Bridging Domains through Proxy Agents

An agent *environment* is comprised of the set of all agent domains that fall within the range of the agent-to-agent protocol, and is thus potentially unbounded. The agent-to-agent protocol may extend beyond a particular distributed object environment through the use of *proxy agents* (figure 7). Proxy agents are useful in cases where two agent domains share the same implementation of agent-to-agent messaging but cannot communicate because their distributed object environments are disjoint. Separate agent domains, whether similar or not, may use proxy agents that communicate through E-mail or through some other mechanism. Any extension of the range of the agent-to-agent protocol beyond the bounds of an agent domain by definition uses a proxy agent.

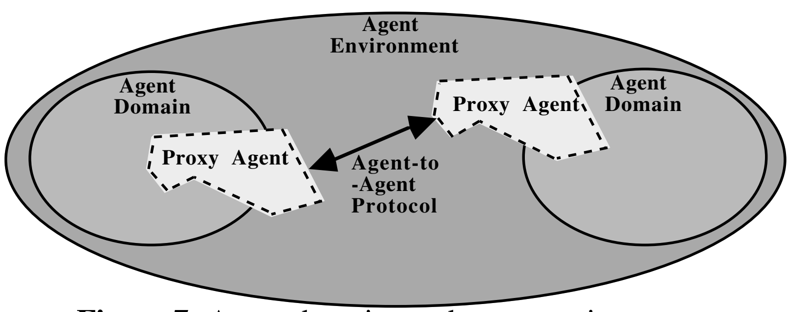

Figure 7. Agent domains and agent environments

To extend the range of the agent-to-agent protocol beyond the agent domain requires that:

- agents in both domains understand the agent-to-agent protocol.
- one or more agents in each domain are capable of transmitting and receiving the agent-to-agent protocol over some form of connection between the two domains, and in so doing act as gateways to their counterparts in the remote domain.

3.3.3. Mediation Agents

A *mediation agent* is any agent that communicates with external entities or resources. Hence, a proxy agent is a special case of a mediation agent. A mediation agent provides in essence a gateway or wrapper for external non-agent entities, allowing them to access resources via the agent domain, and in turn allowing other agents to make use of them through normal agent-to-agent protocols. A single mediation agent may manipulate many external resources, and several mediation agents may share a single external resource. Figure 8 illustrates some of the kinds of resources that mediation agents might manipulate.

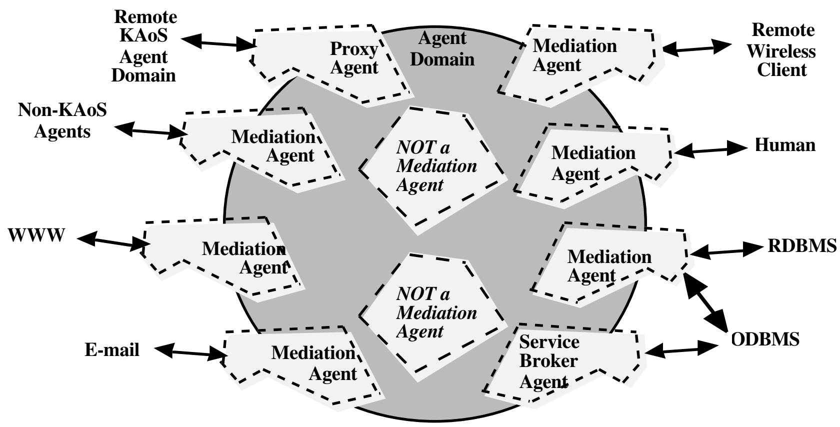

Figure 8. Mediation agents.

In a typical KAoS application, most or all agents perform some form of mediation. Since external resources can be shared, the system designer need not design a single agent as a dedicated resource manager (otherwise that resource manager can become a bottleneck for an otherwise distributed system). Instead several agents may be allowed—where appropriate—to access a given resource through an external resource management facility (for example, a database API).

3.3.4. Service Brokers

The *Service Broker* is usually implemented as a special case of a mediation agent. It can access information about the object identifier and location of the generic agent instance for any agent within the domain that has advertised its services. The Service Broker performs a similar rôle to a KQML "agent server" facilitator which uses the advertise and recommend performatives [37]. In a distributed object environment, the agent domain could be implemented with a single object repository manipulated by a Service Broker agent. In a CORBA environment, the OMG *trading facility* could be used in support of the Service Broker function.

At initialization time, agents must be able to locate the Service Broker. An agent *advertises* a service if it is prepared to respond to messages from other agents wishing to use that service. An agent desiring to use a service may *recruit* from the Service Broker, any agents that have previously advertised that service.

The Service Broker does not keep track of agents that consume services but do not provide them. Neither does it directly provide a repository for shared knowledge—if required, this should be implemented by a separate mechanism such as a blackboard.

Specific message types used for communication with the Service Broker are discussed in section 3.5.7 below.

3.4. Agent-to-Agent Communication

3.4.1. Conversations

Unlike most agent communication architectures,[1] 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 [79, p. 64].*

A major issue for designers of agent-oriented systems how to implement policies governing conversational and other social behavior among agents. Walker and Wooldridge [75] 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 [40], KAoS provides an explicit set of mechanisms encoding message sequencing conventions[2] 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.[3] Shared knowledge about message sequencing rules enables agents to coordinate frequently recurring interactions of a routine nature simply and predictably.

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: such a conversation is said to be *embedded* in the older, *containing conversation*. Activity may occur simultaneously in a containing conversation and any conversations embedded within it. However the embedded conversation must terminate no later than the containing conversation does.

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 protocol.

3.4.2. Conversation Policies

Conversation policies[4] prescriptively encode regularities that characterize communication sequences between users of a language. A conversation policy explicitly defines what sequences of which messages are permissible between a given set of participating agents.

A state transition diagram is used to represent each conversation policy.[5] Every transition leads to exactly one state. All transitions lead to a state labeled with a unique identifier such as a number. The scope of the identifier is confined to the conversation policy—that is, no similarity can be inferred between states of the same number in different conversation policies. Exactly one transition (the first transition) in each conversation policy does not originate in a state. Each transition represents a message and is labeled with the originator and recipient, and each but the

---

[1] Notable exceptions are Barbuceanu and Fox's COOL [5], Kuwbara's AgenTalk [49], and the GOAL cooperation service framework [29]. Labrou and Finin [50] have suggested a scheme by which a future version of KQML could implement conversation policies.

[2] For an excellent discussion on the role of convention in language use, see [32].

[3] Where more complex interactions demand it, predefined communication conventions can be supplemented by emergent ones (see section 5.1.5).

[4] A concept similar to our conversation policies is that of *dialogue grammars* [27]. We discuss the limitations of dialogue grammar models for agent communication in section 5.1.5 below.

[5] Ackroyd describes an object-oriented design for a finite state machine in [1].

first transition is labeled with the message name. All states have transitions entering them. Any state with no transition leaving it is a final state; reaching a final state ends the conversation.

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 core suite currently consist of: *Inform, Offer, Request, Conversation for Action (CFA),* and *Query* .[1]

Inform. The simplest case of a conversation is a single message in one direction with no response. This is analogous to sending a piece of information by E-mail with no expectation of an answer. In such a case, the conversation policy reduces to the kind of atomic message sending encountered in most agent communication languages. A slightly more complex example would allow Agent A to optionally require Agent B to acknowledge receipt of the information, as in the *Inform* conversation policy shown in figure 9. The option of whether or not an acknowledgment is required is conveyed as part of the parameters of the initial message.

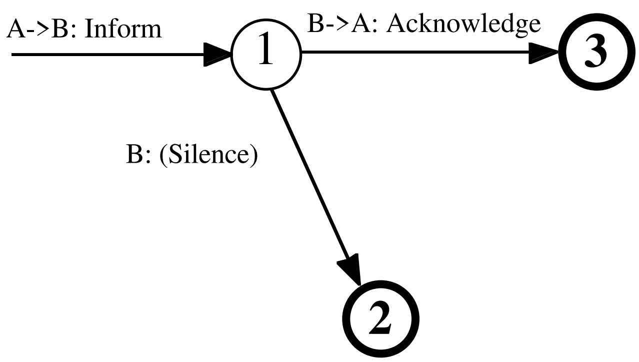

Figure 9. The *Inform* conversation policy.

Offer. Whereas an *inform* message carries the information with it, we define an *offer* as a proposal to do something in the future. Hence an offer is something that can be *declined*, while it is impossible to decline to be informed once one already has received some information (figure 10). As an example, a monitoring agent could initiate an *Offer* conversation with another agent that seemed to need its help.

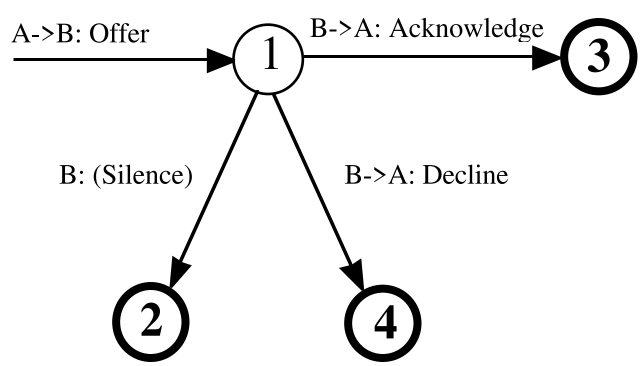

Figure 10. The *Offer* conversation policy.

Request. The conversation policy for a *Request* is shown in figure 11. This kind of conversation policy is well suited to an agent that can reliably fulfill its commitments, or where 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

---

[1] The *Query* conversation policy is described in section 3.4.7.

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.

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. Careful semantic analysis of agent communication must take into account both the individual message and its function in the larger conversation. This is an issue which cries out for more careful study.

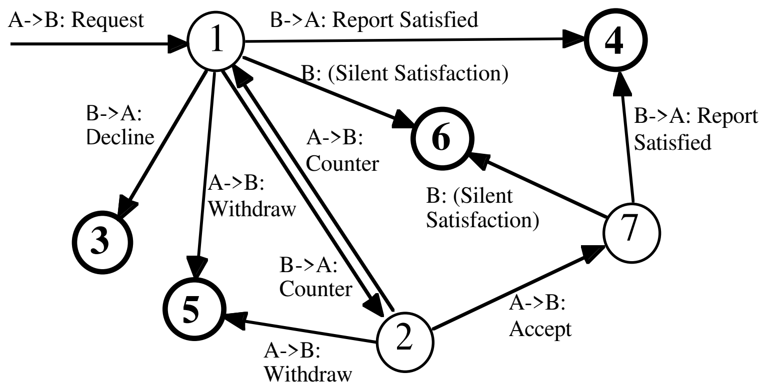

Figure 11. The *Request* conversation policy.

Conversation For Action. We regard Winograd and Flores' conversation for action (cfa) [79] as a more complex sort of request (figure 12). We include a slightly modified version of the *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.[1] In contrast to the simple request, the conversation for action provides a more complex mechanism to handle commitments that persist over time and may not be reliably fulfilled. Embedded conversations may well occur, 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.

[1] We are not, however, claiming that the conversation for action model is necessarily well-suited for human-to-human conversation [9; 27; 28; 30; 65; 69].

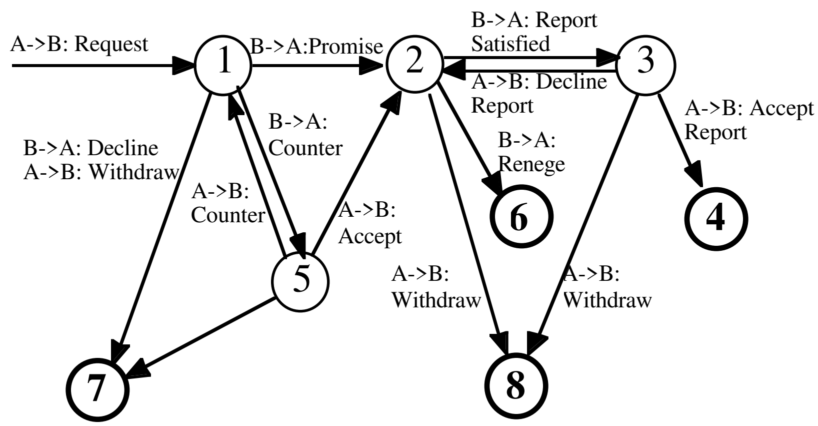

Figure 12. The *Conversation For Action* conversation policy.

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.

3.4.3. Verbs

Verbs name the type of illocutionary act represented by a message. All verbs fall into one or both of the following categories:

- the name of the initial message in a conversation.
- a named state transition in one or more conversation policies.

That is, some verbs appear only inside existing conversations, some only initiate conversations, and some may occur in either context.

The agent's capacity to understand any verb which may occur during a conversation is implicit in its capacity to process the conversation policy for that conversation. The capability of understanding a verb which initiates a conversation (an *initial verb*) entails:

- understanding the initial verb
- implementing the conversation policy that the verb uses.

3.4.4. Suites

A *suite* provides a convenient grouping of conversation policies that support a set of related services.[1] For example, Figure 13 shows a core suite of initial verbs and conversation policies, available to all agents. In addition to this core suite, special agents such as the Service Broker would be expected to process at least one additional set of conversations (i.e., the *Service Broker suite*).

The table below represents the basic elements of the *Core suite,* omitting the Query conversation policy which is described later in section 3.4.6. 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.

[1] There is an analog to Apple Event Suites, which group high-level interprocess events supporting a functional area [2].

	Inform	Offer	Request	CFA
inform	I			
acknowledge	(S)	(S)		
offer		I		
decline		(S)	S	S
request			I	I
counter			S	S
accept			S	S
withdraw			S	S
promise				S
report satisfied			(S)	S
accept report				S
decline report				S
declare satisfied				S
renege				S

Figure 13. The basic elements of the *Core suite*, omitting *Query*

The conversation models and verbs in the Core suite are all available to other suites. In this way, new suites can be treated as extensions to previous ones. Similarly, if an agent class is aware of particular suites, then all instances of that class and its subclasses will also be aware of those suites.

3.4.5. 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 Service Broker without first advertising their existence, and since service-requesting agent cannot access the required services without first recruiting the appropriate agent through the Service Broker, every agent must have access to the Service Broker suite. However, there is an important difference between non-Service-Broker and Service Broker agents in how they will participate in such conversations: the former will only need to know how to *initiate* advertising and recruiting 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, a generic agent could always perform Advertiser or Recruiter rôles in the Service Broker suite, but only the Service Broker agent will act in the rôle of Service Broker.

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 service requester and one for the service provider). Where appropriate, agents may be permitted to play more than one possible rôle for a given conversation policy. For example, a Service Broker may act as a service provider during the course of processing a *recruit* 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 an embedded *recruit* conversation with another Service Broker in order to have its assistance in locating service providers consistent with the original containing *recruit* request.

From figure 13, we see that the *Core suite* provides the following combinations of initial verb, conversation policy, and rôle for agents which originate conversations:

- *inform*, *Inform*, informer
- *offer, Offer,* offeror
- *request, Request,* requester
- *request, CFA,* requester.

From figure 16, we see that the Service Broker suite provides the following additional combinations:

- *advertise*, *Offer*, advertiser
- *retire*, *Inform*, retiree
- *recruit*, *Request*, recruiter
- *recruit, CFA,* recruiter.

The initial verb of a conversation determines the rôle for the agent originating the conversation. In the Service Broker example, any agent generating an *advertise* or *retire* verb acts as an advertiser or retiree, and the agent receiving either of these messages must adopt the Service Broker rôle.

Requirements for service requester and service provider agents. The developer of service-providing agent may design it to support several suites, from which the requesting agent makes a selection. To use a service, the requester may support one or more suites associated with that service. To support a suite, an agent must be capable of adopting at least one rôle in that suite. To adopt a rôle, a requesting agent must do the following:

- for at least one combination of conversation policy, initial verb, and rôle:
  - implement the conversation policy
  - generate the initial verb
- implement any conversation policies that may arise from either agent initiating a related embedded conversation (section 3.4.8 discusses this point in more detail).

To adopt a rôle, a service provider must do the following:

- implement all conversation policies that the requesting agent may specify
- implement any conversation policies that may arise from its initiating related embedded conversations with the requesting agent.

3.4.6. Extending the Protocol by Adding a Query Conversation Policy

Though the starter set of conversation policies defined in KAoS will be sufficient for many sorts of agent interactions, there will often be a need to add to the protocol. We will illustrate how this is done by adding a *Query* conversation policy to complete the partial *Core suite* shown in figure 13. 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 14). The major difference between the *Query* and *Request* conversation policies is that the *B:A report satisfied* message is not optional, and it must contain some result (i.e., a response to the query) as part of its content.

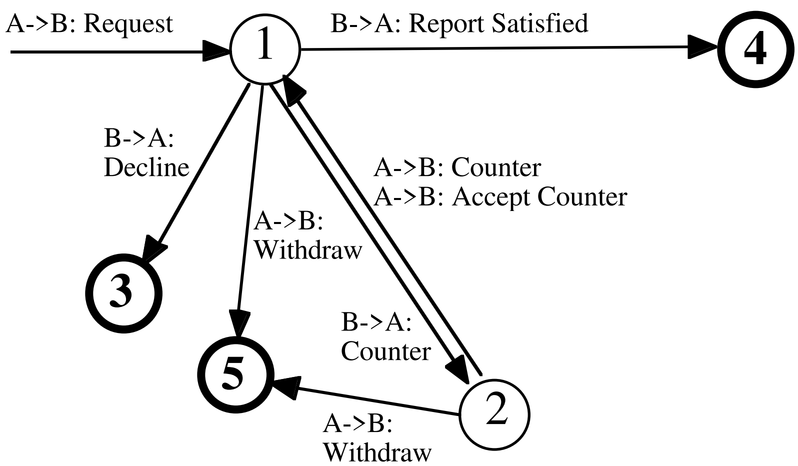

Figure 14. The *Query* conversation policy.

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 15). The shaded cells show what has been newly added: one conversation policy, one verb, and the participation information.

	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

Figure 15. Completing the *Core suite* by adding the *Query* conversation policy.

3.4.7. Embedded Conversation Example and the Service Broker Suite

As discussed earlier, a conversation between a service requester and a service provider may necessitate further embedded conversations between the two agents. For example, a *Request* may give rise to embedded *Inform* conversations as we illustrate below.

Our example is based on the *Service Broker suite* shown in figure 16. The suite adds three new verbs: *advertise, retire,* and *recruit.* The *advertise* message is sent to the Service Broker by any agent wishing to offer services. It uses the *Offer* conversation policy with a more specific verb.[1] The *retire* message is used by an agent to withdraw its offer of services. It uses the *Inform* conversation policy. The *recruit* message is used by an agent to request the Service Broker's help in finding an agent to perform some set of services. *Recruit* will use either the *Request* or *CFA* conversation policies.

	Inform	Offe r	Reque st	CF A	Quer y
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
advertise		I			
retire	I				
recruit			I	I	

Figure 16. The *Service Broker suite*.

We give two examples of how the Service Broker could handle *recruit.* The notation follows that of transitions within conversation policies, but with a simplified data portion of the messages specified as two encapsulated arguments, the second of which is the content portion.

Recruit example without embedded conversations. One way of handling *recruit* would be for the Service Broker to return the list of agents providing the requested services in the content portion of the single *report satisfied* message:

A:B recruit ((conversation identifier) (conversation policy, timeout)) (service)

B:A report satisfied (conversation identifier) (((service) (agent) (suites)) ((service) (agent) (suites))...)

Recruit example with embedded conversations. Another way of handling the message would be to return the list of requested agents one by one as a series of embedded *Inform* conversations, each one identifying one service provider. This might be useful if the search for all required agents might take a significant amount of time. Agent A could evaluate the agents returned by the Service Broker one by one until it determines that it has received enough information. Then it can send *withdraw* in order to terminate the containing recruitment conversation.

Embedded conversations are shown as indented. Note that the embedded conversation is an *Inform* without acknowledgment:

A:B recruit ((conversation identifier) (conversation policy, timeout)) (service)

    B:A inform ((conversation identifier) (conversation policy))

---

[1] The reason that an advertisement uses the *Offer* conversation policy rather than the one for *Inform* is to give the Service Broker an opportunity to refuse the services of the agent, if it deems it necessary for some reason (e.g., the credentials of the advertising agent are not acceptable).

((service) (agent) (suites))
B:A inform ((conversation identifier) (conversation policy))
((service) (agent) (suites))
...
A:B withdraw (conversation identifier)

4. Applications

4.1. Initial Prototypes

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 the battleship game, defining specializations of the generic agent class for one or many cooperating ship captains on each team, a game board service broker, an Excel spreadsheet mediation agent, and a referee [3; 72]

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 [8]. Generic agent capability was specialized to create a supervisor agent, a job administration agent, a user administration agent, and a client mediation agent that handled interaction between OpenDoc "clients" and a KAoS agent domain.

A scheduling environment prototype showed how KAoS could be used to implement assistants to aid in the process of scheduling meetings and meeting rooms [6]. A simulation of interaction with the agent system through electronic mail and agent learning of user preferences were also shown. The scheduling environment consisted of a set of scheduling agents, a scenario agent, a mail mediation agent handling interaction between a MAPI mail application and the agent domain, and an OLE journaling mediation agent that communicated with Microsoft Excel.

4.2. *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 [19; 43]. A new version of KAoS is being incorporated into one such prototype of an intelligent performance support system. The system, named *Gaudi,*[1] 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.

Five requirements guide *Gaudi's* evolution:

1. **Compose your tool set any way you want it.** 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 traditional monolithic applications to distributed component-based software (e.g., WWW, OpenDoc, OLE, Java) must also be provided.
2. **Link to anything (without requiring markup).** SGML and HTML-based software typically provides for linking based on embedded markup of textual data. However this becomes problematic:
   - where context-sensitive linking is needed, since appropriate links may vary according to the user, task, or situation;

---

[1] The system is named for the Spanish artist and architect, Antonio Gaudi (1852-1926), who is most widely known for his work on the *Sagrada Familia* temple in Barcelona [71]. This monumental unfinished structure , on which construction still continues after more than a hundred years, epitomizes our desire to produce an architecture capable of outliving its designers and of providing a suitable foundation for unanticipated additions of significant new features. We believe that complex, long-living structures are something that need to be started by designers, but continually "finished" by users [20].

- 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.

3. **Run it everywhere.** This requirement underlines the necessity of developing a cross-platform approach (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.
4. **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.
5. **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.

As part of a collaboration with NASA-Ames and Johnson Space Center, we are sharing design information about external linking architectures, and working to incorporate their adaptive link engine [26; 55] into *Gaudi*.

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 strengths 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 and OLE;
- 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 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.

Acknowledgments

We appreciate the all those whose contributions have made KAoS a reality: especially Jack Woolley (advisor) and members of the 1992-93 Seattle University KAoS team (Nick George, Rob Jasper, Monica Rosman LaFever, Katrina Morrison, Katrina Morrison, Dan Rosenthal, Steve Tockey), the 1993-94 team (David Atler, Mike Bingle, Tim Cooke, John Morgan, Dan Van Duine, Michael Zonczyk), and the 1994-95 KAoS-CORBA (Dwight Barker, Pete Benoit, Jim Tomlinson, Sheryl Landon) and TAINT (Mary Bos, Robert Boyer, Stewart Dutfield, E. J. Jones) teams. Thanks also to Ian Angus, John Boose, Guy Boy, Kathleen Bradshaw, Chuck Buchanan, Alberto Cañas, Bob Carpenter, Dick Chapman, Phil Cohen, Dave Cooper, Stan Covington, Ron Deiss, Al Erisman, Megan Eskey, Ken Ford, Peter Friedland, Brian Gaines, Kirk Godtfredsen, Roger Guay, Pat Henderson, Pete Holm, Earl Hunt, Renia Jeffers, Dick Jones, Oscar and Sharon Kipersztok, Cathy Kitto, Vicki Lane, David Madigan, Bill McDonald, Mark Miller, Nathalie Mathé, Peter Morton, Ken Neves, Steve Poltrock, Judy Powell, Alain Rappaport, Tyde Richards, Tom Robinson, Kurt Schmucker, John Schuitemaker, Doug Schuler, Dick Shanafelt, Kish Sharma, Mildred Shaw, David C. Smith, Jim Spohrer, Keith Sullivan, Steve Tanimoto, Sankar Virdhagriswaran, Helen Wilson, and Debra Zarley.

For information on availability of KAoS, contact the Jeff Bradshaw at jbrad@grace.rt.cs.boeing.com, (206) 865-6086. A more complete discussion of KAoS may be found in [17].

References

[1] Ackroyd, M. (1995). Object-oriented design of a finite state machine. *Journal of Object-Oriented Programming*(June), 50-59.

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

[3] 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.

[4] Ball, G., Ling, D., Kurlander, D., Miller, J., Pugh, D., Skelly, T., Stankosky, A., Thiel, D., Dantzich, M. V., & Wax, T. (1995). Lifelike computer characters: The Persona project at Microsoft Research. In J. M. Bradshaw (Ed.), *Software Agents.* (pp. in preparation). Cambridge, MA: AAAI/MIT Press.

[5] Barbuceanu, M., & Fox, M. S. (1995). COOL: A language for describing coordination in multi-agent systems. V. Lessor (Ed.), *Proceedings of the First International Conference on Multi-Agent Systems,* (pp. 17-24). San Francisco, CA, , Menlo Park, CA: AAAI/MIT Press ,

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

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

[8] 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.

[9] 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 ,

[10] Bowman, C. M., Danzig, P. B., Manber, U., & Schwartz, M. F. (1994). Scalable Internet resource discovery: Research problems and approaches. *Communications of the ACM*, 37(8), 98-107, 114.

[11] Boy, G. (1992). Computer integrated documentation. In E. Barrett (Ed.), *Sociomedia: Multimedia, Hypermedia, and the Social Construction of Knowledge.* (pp. 507-531). Cambridge, MA: MIT Press.

[12] Boy, G. A. (1991). Indexing hypertext documents in context. *Proceedings of the Third ACM Conference on Hypertext,* (pp. 1-11). San Antonio, Texas, , ,

[13] Boy, G. A. (1991). *Intelligent Assistant Systems*., London: Academic Press.

[14] Boy, G. A. (1995). Software agents for cooperative learning. In J. M. Bradshaw (Ed.), *Software Agents.* (pp. in preparation). Cambridge, MA: AAAI/MIT Press.

[15] Boy, G. A., & Mathé, N. (1993). Operator assistant systems: An experimental approach using a telerobotics application. In K. M. Ford & J. M. Bradshaw (Ed.), *Knowledge Acquisition as Modeling.* (pp. 271-286). New York: John Wiley.

[16] Bradshaw, J. M., Boose, J. H., Covington, S. P., & Russo, P. J. (1988). How to do with grids what people say you can't. *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop,* (pp. 4.1-4.20). Banff, Alberta, Canada, , SRDG Publications, University of Calgary, Department of Computer Science ,

[17] Bradshaw, J. M., Dutfield, S., Benoit, P., & Woolley, J. D. (1995). KAoS: Toward an industrial-strength generic agent architecture. In J. M. Bradshaw (Ed.), *Software Agents.* (pp. in preparation). Cambridge, MA: AAAI/MIT Press.

[18] Bradshaw, J. M., Holm, P., Kipersztok, O., Nguyen, T., Russo, P. J., & Boose, J. H. (1991). Intelligent interoperability in DDUCKS. *Working Notes of the AAAI-91 Cooperation Among Heterogeneous Intelligent Systems Workshop,* . Anaheim, California, , ,

[19] 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, , ,

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

[21] Brockschmidt, K. (1994). *Inside OLE 2*., Redmond, WA: Microsoft Press.

[22] Brown, C., Gasser, L., O'Leary, D. E., & Sangster, A. (1995). AI on the WWW: Supply and demand agents. *IEEE Expert*, 10(4), 50-55.

[23] Browne, D., Totterdell, P., & Norman, M. (Ed.). (1990). *Adaptive User Interfaces.* London: Academic Press.

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

[25] Carter, J. (1992). Managing knowledge: The new systems agenda. *IEEE Expert*(June), 3-4.

[26] Chen, J. R., & Mathé, N. (1995). Learning subjective relevance to facilitate information access. *Submitted to CIKM-95,* . , , , ,

[27] 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 ,

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

[29] Cunningham, J. (1995). GOAL Cooperation Service Framework. V. Lessor (Ed.), *Proceedings of the First International Conference on Multi-Agent Systems,* (pp. addendum). San Francisco, CA, , Menlo Park, CA: AAAI/MIT Press ,

[30] 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 ,

[31] Dennett, D. C. (1987). *The Intentional Stance*., Cambridge, MA: MIT Press.

[32] Deuchar, M. (1990). Are the signs of language arbitrary? In H. Barlow, C. Blakemore, & M. Weston-Smith (Ed.), *Images and Understanding.* Cambridge, England: Cambridge University Press.

[33] Englemore, R. (Ed.). (1988). *Blackboard Systems.* Reading, MA: Addison-Wesley.

[34] Erickson, T. (1995). Designing agents as if people mattered. In J. M. Bradshaw (Ed.), *Software Agents.* (pp. in preparation). Cambridge, MA: AAAI/MIT Press.

[35] Etzioni, O., & Weld, D. (1994). A softbot-based interface to the internet. *Communications of the ACM*, 37(7), 72-76.

[36] Etzioni, O., & Weld, D. S. (1995). Intelligent agents on the Internet: Fact, fiction, and forecast. *IEEE Expert*, 10(4), 44-49.

[37] Finin, T., Labrou, Y., & Mayfield, J. (1995). KQML as an agent communication language. In J. M. Bradshaw (Ed.), *Software Agents.* (pp. in preparation). Cambridge, MA: AAAI/MIT Press.

[38] Foody, M. (1995). Providing object system interoperability with middleware. *Cross-Platform Strategies: Supplement to SIGS Publications*, June, 11-21.

[39] Gasser, L. (1991). Social conceptions of knowledge and action: DAI foundations and open systems semantics. *Artificial Intelligence*, 47, 107-138.

[40] Genesereth, M. R. (1995). An agent-based framework for interoperability. In J. M. Bradshaw (Ed.), *Software Agents.* (pp. in press). Cambridge, MA: AAAI/MIT Press.

[41] Gentner, D., & Nielsen, J. (1995). The anti-Mac: Violating the Macintosh human interface guidelines. *Proceedings of CHI-95,* (pp. 183-184). Denver, CO, , New York: ACM ,

[42] George, N., Jasper, R., LaFever, M. R., Morrison, K., Rosenthal, D., Tockey, S., Woolley, J., Bradshaw, J. M., Boy, G., & Holm, P. D. (1994). KAoS: A knowledgeable-agent-oriented system. *Proceedings of the AAAI Spring Symposium on Software Agents,* (pp. 24-30). Stanford, CA, 21-23 March, , ,

[43] 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, , ,

[44] Hanks, S., Pollack, M. E., & Cohen, P. R. (1993). Benchmarks, test beds, controlled experimentation, and the design of agent architectures. *AI Magazine*, Winter, 17-42.

[45] Hewitt, C., & Inman, J. (1991). DAI betwixt and between: from “intelligent agents” to open systems science. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6), 1409-1419.

[46] Kaehler, T., & Patterson, D. (1986). A small taste of Smalltalk. *BYTE*, August, 145-159.

[47] Knoblock, C. A., & Ambite, J.-L. (1995). Agents for information gathering. In J. M. Bradshaw (Ed.), *Software Agents.* (pp. in preparation). Cambridge, MA: AAAI/MIT Press.

[48] Kuokka, D., & Harada, L. (1995). On using KQML for matchmaking. V. Lesser (Ed.), *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95),* (pp. 239-245). San Francisco, CA, , Menlo Park: AAAI/MIT Press ,

[49] Kuwabara, K. (1995). AgenTalk: Coordination Protocol Description for Multiagent Systems. V. Lessor (Ed.), *Proceedings of the First International Conference on Multi-Agent Systems,* (pp. addendum). San Francisco, CA, , Menlo Park, CA: AAAI/MIT Press ,

[50] 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 ,

[51] Maes, P. (1995). Agents that reduce work and information overload. In J. M. Bradshaw (Ed.), *Software Agents.* (pp. in preparation). Cambridge, MA: AAAI/MIT Press.

[52] 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, , ,

[53] Malone, T. W., Grant, K. R., & Lai, K.-Y. (1995). Agents for information sharing and coordination: A history and some reflections. In J. M. Bradshaw (Ed.), *Software Agents.* (pp. in preparation). Cambridge, MA: AAAI/MIT Press.

[54] Martin, F. (1988) *Children, cybernetics, and programmable turtles*. Unpublished Masters Thesis, Massachusetts Institute of Technology Media Laboratory.

[55] Mathé, N., & Chen, J. (1994). A user-centered approach to adaptive hypertext based on an information relevance model. *Proceedings of the Fourth International Conference on User Modeling (UM ‘94),* (pp. 107-114). Hyannis, MA, , ,

[56] Nelson, P. R., & Schuler, D. (1995). *Managing engineering information with hypermedia*. Seattle WA: Boeing Commercial Airplane Group, April 17.

[57] Nelson, T. (1980). Interactive systems and the design of virtuality, Parts 1 and 2. *Creative Computing*, November & December, 56-62; 95-106.

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

[59] Raccoon, L. B. S. (1995). The chaos model and the chaos life cycle. *ACM SIGSOFT Software Engineering Notes*, 20(1), 55-66.

[60] Reinhardt, A. (1994). The network with smarts. *BYTE*, 10, October, 50-64.

[61] Resnick, M., & Martin, F. (1990). *Children and artificial life*. E&L Memo No. 10. Cambridge, MA: MIT Media Laboratory,

[62] Rettig, M. (1991). Nobody reads documentation. *Communications of the ACM*, 34(7), 19-24.

[63] Richman, D. (1995). Let your agent handle it. *InformationWeek*, April 17, 44-56.

[64] Riecken, D. (1995). M: An architecture of integrated agents. In J. M. Bradshaw (Ed.), *Software Agents.* (pp. in preparation). Cambridge, MA: AAAI/MIT Press.

[65] 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.

[66] Russell, S., & Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*., New York: Prentice-Hall.

[67] Shoham, Y. (1993). Agent-oriented programming. *Artificial Intelligence*, 60(1), 51-92.

[68] Soley, R. M. (1994). Role of object technology in distributed systems. In R. Khanna (Ed.), *Distributed Computing: Implementation and Management Strategies.* (pp. 469-478). Englewood Cliffs, NJ: Prentice-Hall.

[69] 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 ,

[70] Tambe, M., Johnson, W. L., Jones, R. M., Koss, F., Laird, J. E., Rosenbloom, P. S., & Schwamb, K. (1995). Intelligent agents for interactive simulation environments. *AI Magazine*, 16(1), 15-39.

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

[72] 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), .

[73] Virdhagriswaran, S. (1994). Heterogeneous information systems integration: An agent-messaging-based approach. T. Finin (Ed.), *Proceedings of the CIKM-94 Workshop on Intelligent Agents,* . Gaithersburg, MD, , ,

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

[75] 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 ,

[76] Wayner, P. (1995). Free agents. *BYTE*, March, 105-114.

[77] White, J. (1995). Mobile agents. In J. M. Bradshaw (Ed.), *Software Agents.* (pp. in preparation). Cambridge, MA: AAAI/MIT Press.

[78] Wiederhold, G. (1992). Mediators in the architecture of future information systems. *IEEE Computer*(March), 38-49.

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

[80] Woelk, D., Huhns, M., & Tomlinson, C. (1995). Uncovering the next generation of active objects. *Object Magazine*, 4, July-August, 33-40.

[81] Wooldridge, M. J., & Jennings, N. R. (1995). Agent theories, architectures, and languages: A survey. In M. J. Wooldridge & N. R. Jennings (Ed.), *Intelligent Agents: ECAI-94 Workshop on Agent Theories, Architectures, and Languages.* (pp. 1-39). Berlin: Springer-Verlag.