Chapter 17

KAoS: Toward An Industrial-Strength Open Agent Architecture

Jeffrey M. Bradshaw, Stewart Dutfield, Pete Benoit, & John D. Woolley

The complexity of modern engineered systems motivates the requirement for timely access to technical and operational documentation (Boy 1991, 1992). Documents are both the most valuable and the most expensive knowledge resource in engineering organizations (Carter 1992). Product and product-related documents may be intended for use by thousands of people over a life cycle of many years (Nelson and Schuler 1995; Malcolm, Poltrock, and Schuler 1991). 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 (Rettig 1991) Instead they use information retrieval strategies that are context-*dependent* (Mathé and Chen 1994; Boy and Mathé 1993; Boy 1991). 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.

Agents for Technical Information Management and Delivery

The rapidly growing amount and complexity of information available has compounded the problems of technical information delivery. Until relatively recent-

ly, computing resources were so scarce and the bandwidth of human-computer interaction so low that every effort was made to increase access to online information (Nelson 1980). 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. As Paul Saffo ((1994) expresses it:

> It is not content but context that will matter most a decade or so from now. The scarce resource will not be stuff, but point of view…. The future belongs to neither the conduit nor content players, but those who control the filtering, searching, and sense-making tools we will rely on to navigate through the expanses of cyberspace… Without a doubt, the agent arena will become a technological battleground, as algorithms rather than content duel for market dominance.

What will such agents do? At the user interface, they will work in conjunction with component integration frameworks to select the right data, assemble the needed components, and presentand format the information in the most appropriate way for a specific user and situation. Behind the scenes, additional agents will take advantage of distributed object management, database, document management, workflow, messaging, transaction, searching, indexing, 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 never have 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 (Browne, Totterdell, and Norman 1990).

A variety of agent theories, architectures, languages, and implementations 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 (Reinhardt 1994; Richman 1995). Additional agent work has focused on areas such as Internet resource discovery and information integration (Brown et al. 1995; Etzioni and Weld 1995, 1994; Knoblock and Ambite 1997; Woelk, Huhns, and Tomlinson 1995; Bowman et al. 1994; Virdhagriswaran 1994; and Wiederhold 1992), intelligent coordination of distributed problem-solvers (Genesereth 1997; Kuokka and Harada 1995; Tambe et al. 1995; Hanks, Pollack, and Cohen 1993;

Gassero 1991; and Hewitt and Inman 1991; Finin, Labrou, and Mayfield 1997; O'Hare and Jennings 1996), and active user assistance (Ball et al. 1997; Boy 1997; Maes 1997; Malone, Grant, and Lai 1997; Riecken 1997; and Cypher 1993). Yet other agent implementations are beginning to appear that will enable mobile agents to perform business transactions in a safe and secure manner (Wayner 1995; White 1997). 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 (Bowman et al. 1994) and is now being recognized as a requirement for business organizations needing more flexible and dynamic access to private sources of heterogeneous, distributed information.

Lack of Semantics and Extensibility of Communication Languages

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.

KQML has been proposed as a standard communication language for distributed agent applications (Finin 1997, Labrou and Mayfield 1997; Genesereth 1997). 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 and Levesque 1997). These problems are compounded as agent communication language designers are increasingly concerned with policies for full agent conversations, rather than simple one-way exchange at individual performatives (Labrou 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.

Lack of 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, and 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.

To address some of these problems, we are developing KAoS (Knowledgeable Agent-oriented System), an open distributed architecture for software agents. Although the framework is still far from complete, 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 or integrating diverse types of agent-oriented systems. The following section provides the background of KAoS. We then present the aims and major components of the KAoS architecture: agent structure, dynamics, and properties; the relationship between agents and objects; and the elements of agent-to-agent communication. Following this, we briefly summarize our experience in building KAoS applications and discuss issues and future directions.

KAoS Background

KAoS grows out of work beginning in 1988 on a general purpose interapplication communication mechanism for the Macintosh called MANIAC (Manager for InterApplication Communication) (Bradshaw et al. 1991, 1988). Plans for coordination among MANIAC-enabled applications were modeled and executed by means of an integrated planner we developed using ParcPlace Smalltalk. A later version, NetMANIAC, extended messaging capabilities to other platforms through the use of TCP/IP.

In 1992, we began a collaboration with the Seattle University (SU) Software Engineering program to develop the first version of KAoS (Tockey et al. 1995; George et al. 1994). We replaced the integrated planner with a fully object-oriented agent framework, borrowing ideas from Shoham's (1997) AGENT-0 work. 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) (Siegel 1996).[2]

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 (Siegel 1996). 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, and Edwards 1995).[3]

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 ActiveX/OLE (Brockschmidt 1994), and are developing a Java version of KAoS.

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 and Lange 1996; Lange 1996; Virdhagriswaran, Osisek, and O'Connor 1995). We have also been active participants in the *Hippocrene* project of the Aviation Industry Computer-Based Training Committee (AICC) (Bradshaw, Madigan, et al. 1993; Bradshaw, Richards, et al. 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. Promising standards will replace aspects of or be incorporated into KAoS as appropriate in the future.

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 to assure that cross-vendor interoperability could be achieved.

The KAoS architecture dictates neither a single 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.[4]

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.[5] 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.

Implementation Context

Figure 1 illustrates the primary long-range application context in which KAoS is being defined. Though we have optimized our implementation to address Boeing's current needs, we intended the general architecture to support rapid evolution. We are currently preparing versions of the basic KAoS implementation that will be placed in the public domain where they can be evaluated and improved upon.

KAoS Architecture

The KAoS architecture currently aims to provide the following:

- A form of agent-oriented programming, based on a foundation of distributed object technology
- Structured conversations between agents, which may preserve their state over time
- An approach for extending the language of inter-agent communication in a principled manner, taking into account the repertoire of illocutionary acts ("verbs") available to agents, the set of conversation policies available to agents, the content of messages, and a means for agents to locate and access desired services
- A framework supporting interoperability with other agent implementations as well as with non-agent programs
- An environment in which to design agents to engage in specialized suites of interactions

KAoS Agents. Basic characteristics of KAoS agents are described in the Basic Characteristics of Agents subsection that follows. A consistent structure pro-

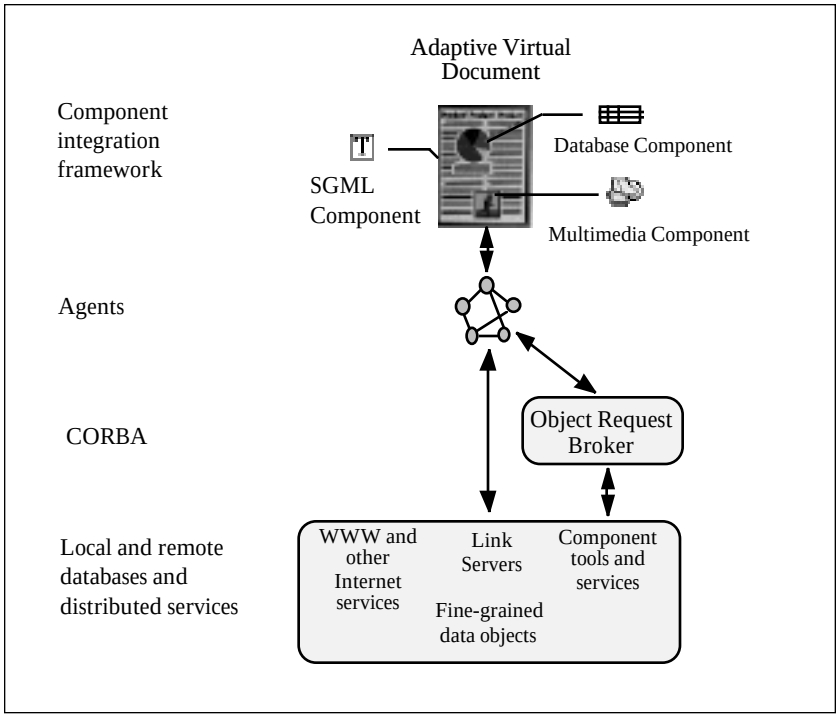

Figure 1. The context in which the KAoS agent architecture is being defined.

vides mechanisms allowing the management of knowledge, desires, intentions, and capabilities. Agent dynamics are managed through a cycle that includes the equivalent of agent birth, life, cryogenic state, and death.

The Agents and Objects subsection describes the relationship between agents and objects. Each agent has a *generic agent instance,* which implements as a minimum the basic infrastructure for agent communication. 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 nonagent entities, resources, or agent frameworks. *Proxy agents* extend the scope of the agent-to-agent protocol beyond a particular agent domain. The *Domain Manager* carries off policies set by a human administrator, such as keeping track of agents that enter and exit the domain. The *Matchmaker* can access information about the location of the generic agent instance for any agent that has advertised its services.

Agent Communication. *Messages* are exchanged between agents in the context of *conversations* (see the Agent-to-Agent Communication subsection). *Verbs* name the type of illocutionary act (e.g., *request*, *promise*) represented by a mes-

sage. 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. *Suites* provide convenient groupings of conversation policies that support a set of related services (e.g., the *Matchmaker suite*). A starter set of suites is provided in the architecture but can be extended or replaced as required.

Scope of the Current Work. The current version of the architecture aims only to specify those generic capabilities which are basic to agent lifecycle management and communication. We are investigating extensions to the architecture to deal with additional issues, including:

- End-user authoring
- Mobile agents
- Semantics of agent communication for emergent conversation behavior
- Joint intention
- Planning
- Complex negotiation
- Vague goal specification
- Learning and adaptive behavior
- Anthropomorphic or other visual presentation
- Message translation

Some of these potential enhancements and technical issues are discussed in the Issues and Future Directions section.

Basic Characteristics of Agents

Agent-oriented programming (Shoham 1997) 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.[6] 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 some person (Van de Velde 1995). That is what makes coming up with a once-and-for-all definition of an agent so difficult: one person's "intelligent 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 in a procedural one (Kaehler and Patterson 1986), so it sometimes may be easier for developers and users to think in terms of intentional agents instead of passive objects (Dennett 1987).[7] Singh (1994) lists several pragmatic and technical reasons for the appeal of viewing agents as intentional systems:

> "They (i) are natural to us, as designers and analyzers; (ii) provide succinct descriptions of, and help understand and explain, the behaviour of complex systems; (iii) make available certain regularities and patterns of action that are independent of the exact physical implementation of the agent in the system; and (iv) may be used by the agents themselves in reasoning about each other."

Agent Structure. The KAoS architecture and generic agent class provide a consistent structure for agents; this includes mechanisms for storing, updating, querying, and "inheriting" knowledge (facts and beliefs), desires, intentions, and capabilities.[8] These structures are shown in the box on the right of figure 2. *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.[9] 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 and Levesque 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."

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 packaged as OLE automation servers or OLE/ActiveX controls, and on the Macintosh platform generic agent functionality is exposed through Apple Events. The Java implementation of KAoS currently relies on sockets. Because KAoS relies on popular messaging schemes for communication between extensions and the generic agent, agent capabilities can be defined or extended straightforwardly using any

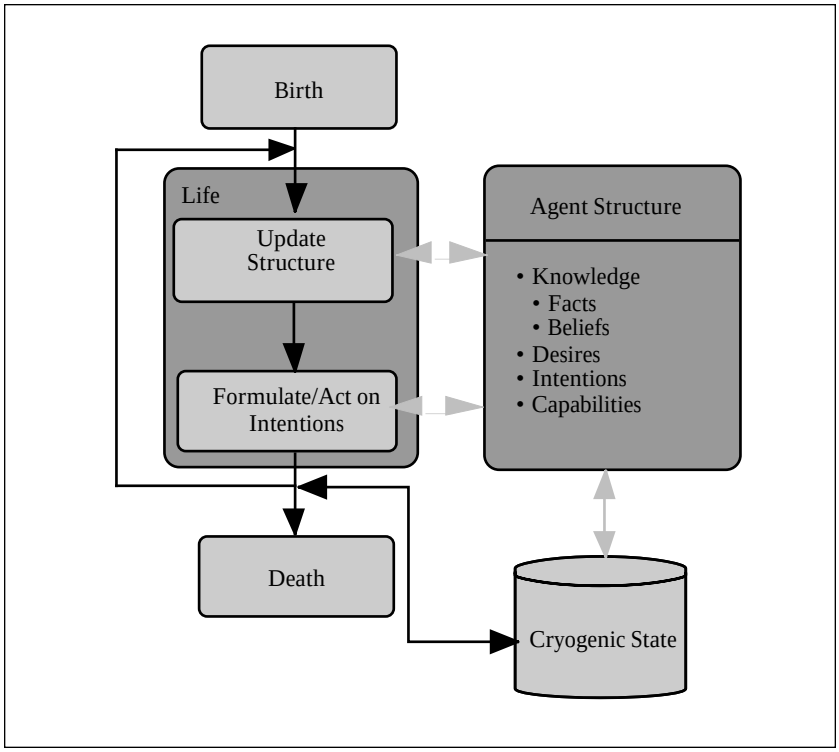

Figure 2. Structure and dynamics of agents (modified from George [1994]). The black arrows represent state transitions, and the gray arrows data flows.

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 (Cypher 1993; Repenning 1993; Spohrer, Vronay, and Kleiman 1991; Smith, Cypher, and Spohrer 1997; Malone, Grant, and Lei 1997).

Agent Dynamics. Figure 2 shows how 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. Agents may acquire additional knowledge, desires, and capabilities as they interact with other agents and with their environment. As messages come in, agents update their structure, formulate their intentions, and send new messages in order to act on them.

In specific applications, agent death may be required to free resources or sim-

ply deal with agents that are no longer useful. 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.[10]

Agents and Objects

The KAoS architecture defines a basic structure and default core speech-act-based agent-to-agent protocol that is normally shared among all agents. To this basic capability, specific extensions and capabilities can be added as needed through inheritance or aggregation. Communication between agents takes place through the use of *messages*. A message consists of a packet of information, usually sent asynchronously, whose type is represented by a *verb* corresponding to some kind of illocutionary act (e.g., *request, inform*).[11] 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. Table 1 enumerates distinctions between communication in classical object-oriented programming and in the agent-oriented architecture.

Like KQML, we make a distinction between communication, content, and contextual portions of agent messages (Finin, Labrou, and Mayfield 1997). 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 executables. The contextual portion describes the type of message being sent (e.g., request, inform) and tells 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 allows agents to analyze, route, and deliver messages properly without necessarily requiring interpretation of content until they reach their final destination.[12]

Table 2 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.

Though operations may take place in isolation, messages occur only in the context of a conversation. The meaning of a given operation may vary between in-

	Objects	Agents
Basic unit	instance	agent
State-defining parameters	unconstrained	knowledge, desires, intentions, capabilities,…
Process of computation	operations	messages
Message types	defined in classes	defined in suites
Message sequences	implicit	defined in conversations
Social conventions	none	honesty, consistency,…

Table 1. Objects versus agents (modified from Shoham 1997).

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

Table 2. Message characteristics.

stances of different classes, but a message always has a meaning defined by its place in a particular conversation (see the Agent-to-Agent Communication subsection). For example, a *decline* message in the context of an *Offer* conversation means something different from 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).

Composition of Agents. Figure 3 shows two agents communicating within a particular agent domain. Each agent contains an instance of the generic agent class (or of a specialization of that class) which is called the *generic agent instance.*[13] The generic agent class implements as a minimum the basic infrastructure for agent communication. It understands conversation policies (see the Conversations Policies subsubsection), and how to initiate and end specific conversations based on a particular policy. The generic agent also monitors the current state of the conversations in which the agent is participating, and is able to judge whether new con-

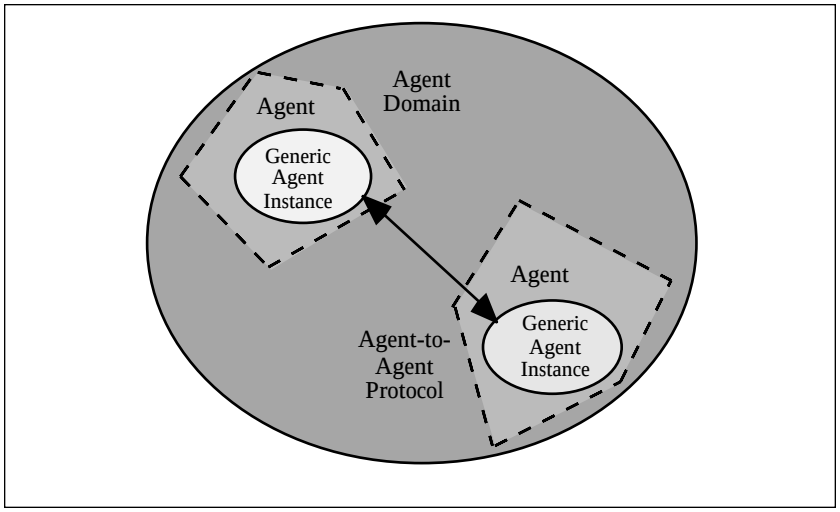

Figure 3. Communication between generic agent instances.

versational moves being proposed at a particular time are permissible.

The agent *domain* denotes the extent of inter-agent messaging; that is, no agent can communicate directly with any other agent across the bounds of an agent domain. For example, in a CORBA environment the bounds of the agent domain would typically be the bounds of a set of communicating ORBs.[14]

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 an extension implemented as a set of separate objects).[15] Figure 4 shows an agent implemented as an aggregation of an agent extension with an instance of the generic agent class. Some extensions may be very active, others may function passively as data repositories.

Defining specialized agents by inheritance generally has the advantage of more efficient performance and tighter integration with the generic agent implementation. Defining them by aggregation has the advantage of allowing the implementation of agent capabilities to be determined dynamically at run-time. For example, a particular agent designed to monitor resource consumption on a set of machines may encounter a situation that requires human intervention. If the agent contains a "hot pluggable" extension, the automated agent capability can be unplugged from the generic agent instance on-the-fly and a human be put in its place without interrupting the ongoing agent conversations, and without the other agents even being aware that it is a person rather than a program that is now directly controlling agent behavior.

Examples of agent extensions might include:

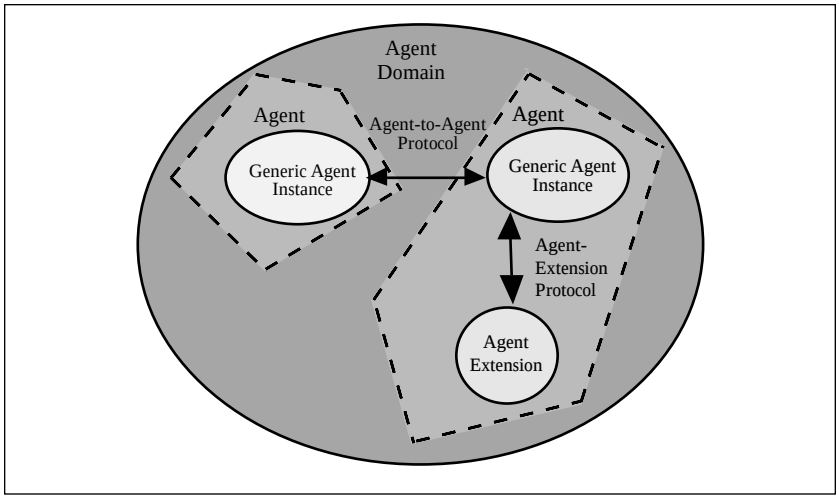

Figure 4. Communication between a generic agent and its extension.

- Unique programs implementing specific agent capabilities
- 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

If an object is not exclusively owned by a particular agent, then any part of any agent may interact with it directly. Examples might include:

- Encapsulations of internal resources over which the agent does not have exclusive control. For example, instances of an "agent conversation object" data structure might, in a particular KAoS implementation, be shared between the agents participating in flat conversation
- External resources over which the agent does not have exclusive control, such as a display or an ODBMS.

Agent Environments: Bridging Domains through Proxy Agents. An agent *environment* comprises 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 extends beyond a particular domain through the use of *proxy agents* (figure 5). Proxy agents are useful in cases where two agent domains share agent-to-agent protocols but cannot communicate because they are implemented within different distributed object environments.[16] For example, communication between different implementations of KAoS (e.g., Java, ActiveX, CORBA) might require proxy agents. Separate agent domains, whether similar or not, may use proxy agents that communicate through

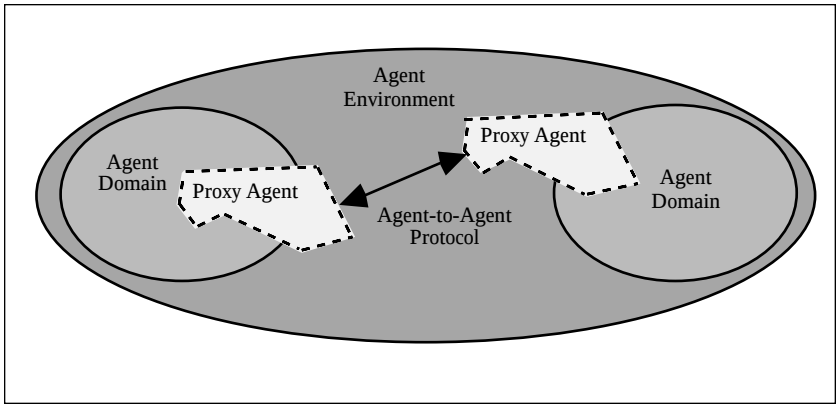

Figure 5. Agent domains and agent environments.

sockets or through some other mechanism.[17] Any extension of the range of the agent-to-agent protocol beyond the bounds of an agent domain by definition uses a proxy.

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.

Mediation Agents. A *mediation agent* is any agent that communicates with external (i.e., nonowned) 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 6 illustrates some of the kinds of resources that mediation agents might manipulate.

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

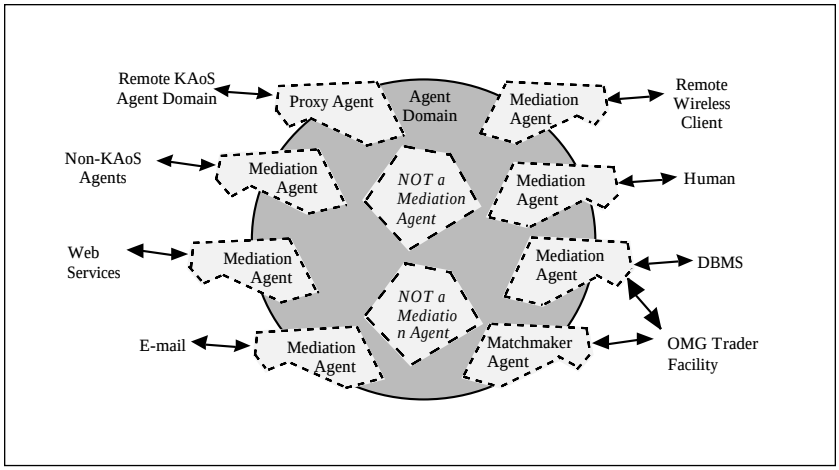

Figure 6. Mediation agents.

Domain Managers and Matchmakers. For an agent domain to become active, two agents must be started. One is the *Matchmaker,* by which agents access information about services within a domain (analogous to the yellow pages in a telephone book). The other is the *Domain Manager*, which controls the entry and exit of agents within a domain, and maintains a set of properties on behalf of the domain administrator.

For an agent to become part of a domain, it first registers itself with the Domain Manager, whose location must be known or accessible to the agent at initialization time. The Domain Manager ultimately lets an agent join a domain, or prevents it from joining based on policies set by the administrator who is responsible for set up of the domain. The Domain Manager relies on a separate naming service (white pages) to associate agent names with implementation-specific object references.

In our implementations to date, Matchmakers have been defined as special cases of mediation agents which use external repositories such as system registries or databases to store data about services. The Matchmaker's major function is to help client agents find information about the location of the generic agent instance for any agent within the domain that has advertised its services, and to forward that request to Matchmakers in other domains where appropriate.[18] In a CORBA environment, an OMG *trader facility* could be used in support of the Matchmaker function.

The Domain Manager provides the address of the Matchmaker to agents within its domain. An agent *advertises* a service to the Matchmaker if it is prepared to respond to messages from other agents wishing to use that service. An

advertise message may specify whether there are any restrictions on which agents may have access to and visibility of the advertised service. For example, certain services may be made available only to client agents within the advertising agent's own domain. An agent desiring to use a service may ask a Matchmaker to *recommend* available agents that have previously advertised that service. A recommend query may involve simple or sophisticated pattern-matching against an arbitrary collection of potential service provider properties.

The Matchmaker does not currently track agents that consume services but do not provide them. Neither does it directly provide a general repository for shared agent knowledge—if required, this could be implemented by a separate mechanism such as a blackboard. Specific message types used for communication with the Matchmaker are discussed in the Matchmaker Suite subsection.

Agent-to-Agent Communication

Conversations. Unlike most agent communication architectures,[19] 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 (1986) 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. (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 (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 because the deeper logic of conversations has yet to be satisfactorily articulated by researchers, the current KAoS 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 1997), KAoS provides an explicit set of mechanisms encoding message-sequencing conventions[20] 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.[21] Shared knowledge about message sequencing rules enables agents to coordinate frequently recurring interactions of a routine nature simply and predictably.

Conversation policies[22] 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.

In current versions of KAoS, state transition diagrams are used to represent each conversation policy.[23] 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 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. Some conversation policies implement silence as a valid transition between states; for example, *Inform* may terminate with an acknowledge message or with silence, depending on what option is selected by the initiator of the conversation (see below). Where silence is appropriate, the conversation terminates immediately after the initial transition.

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* .[24]

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 7). 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.

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 8). As an example, a monitoring agent could initiate an *Offer* con-

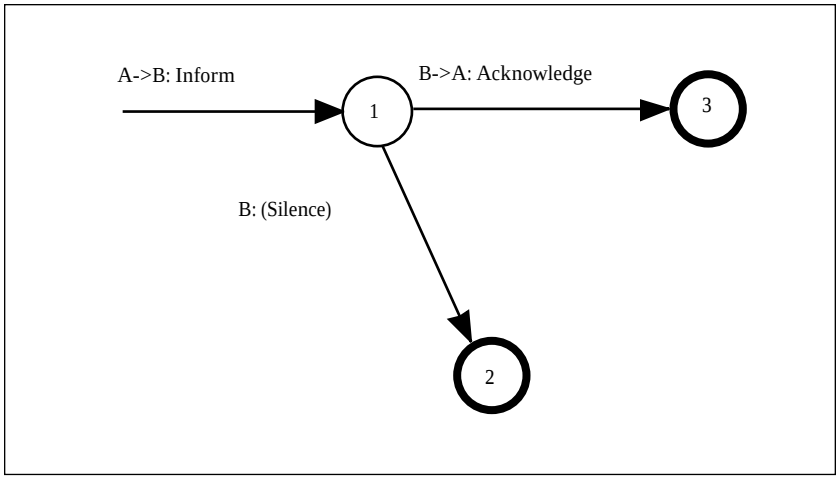

Figure 7. The Inform conversation policy.

versation with another agent that it perceived could benefit from its assistance.

Request. The conversation policy for a *Request* is shown in figure 9. 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 optionally 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.

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 semantics of 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' (1986) *Conversation For Action* (CFA) as a more complex variant of Request (figure 10). 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

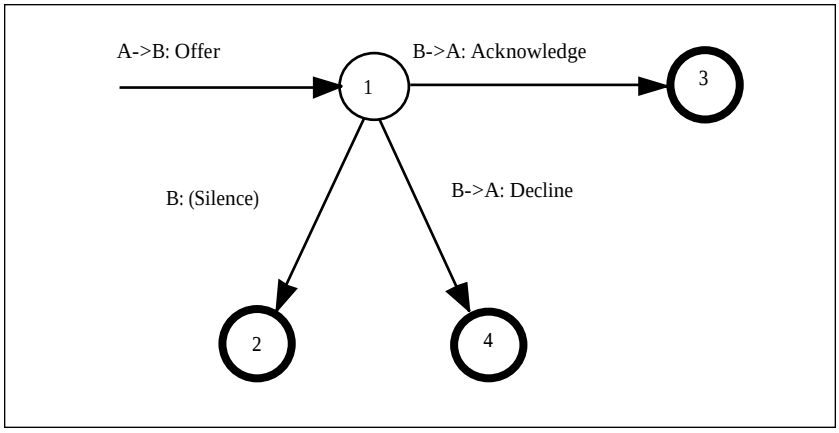

Figure 8. The Offer conversation policy.

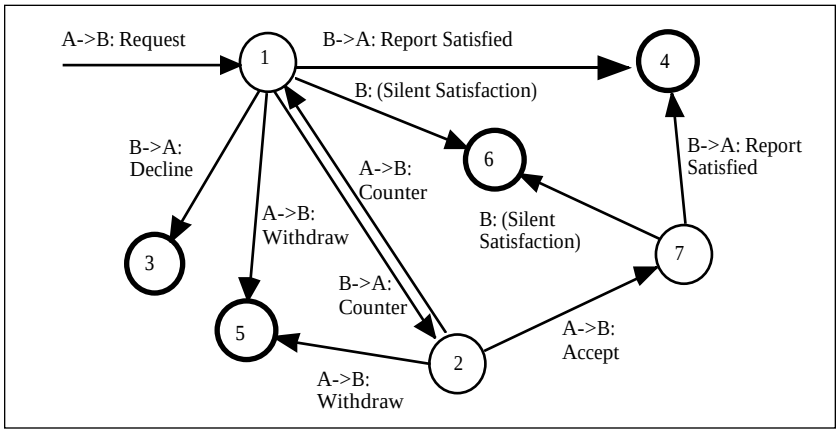

Figure 9. The Request conversation policy.

both that agents make of each other and that humans make of agent systems.[25]

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.

Asynchronies in conversations. The implementation of a conversation policy

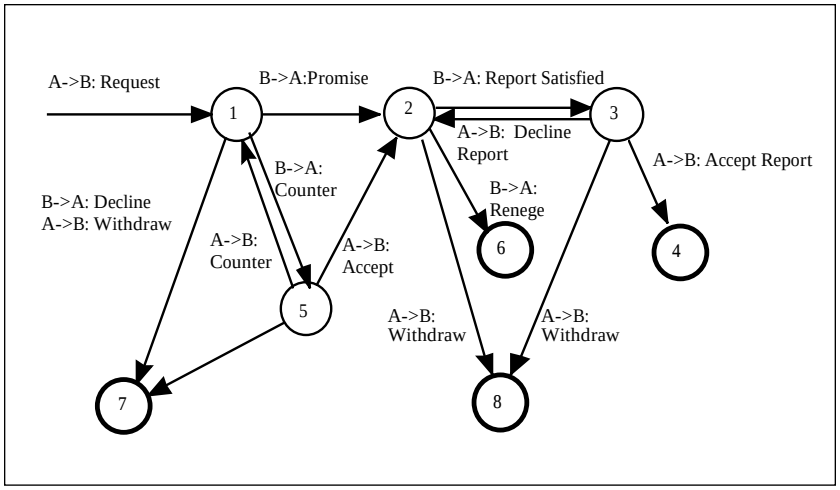

Figure 10. The Conversation For Action (CFA) conversation policy.

must account for asynchronies in conversations (Bowers and Churcher 1988).[26] 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 10, 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 10 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 10, 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

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

Suites. A *suite* provides a convenient grouping of conversation policies that support a set of related services.[27] The default *Core suite* of initial verbs and conversation policies is normally available to all agents. In addition to the Core suite, specialized agents such as the Matchmaker would be expected to process at least one additional set of conversations (i.e., the *Matchmaker suite*).

Table 3 represents a conceptual model of the relationship between the basic elements of the Core suite, omitting the *Query* conversation policy which is introduced in the Query subsection that follows. 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).

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 in the Matchmaker Suite subsection that follows). However, there is an important difference between non-

Core Suite	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

Table 3. The basic elements of the Core suite, omitting Query.

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 table 3, 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 may be 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 table 3. The *query* verb can initiate either a *CFA* conversation policy whose state transitions are identical except for the initial verb,[28] or a new *Query* conversation policy (figure 11). 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.

Consistent with the state transition diagram, table 4 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. The shaded cells

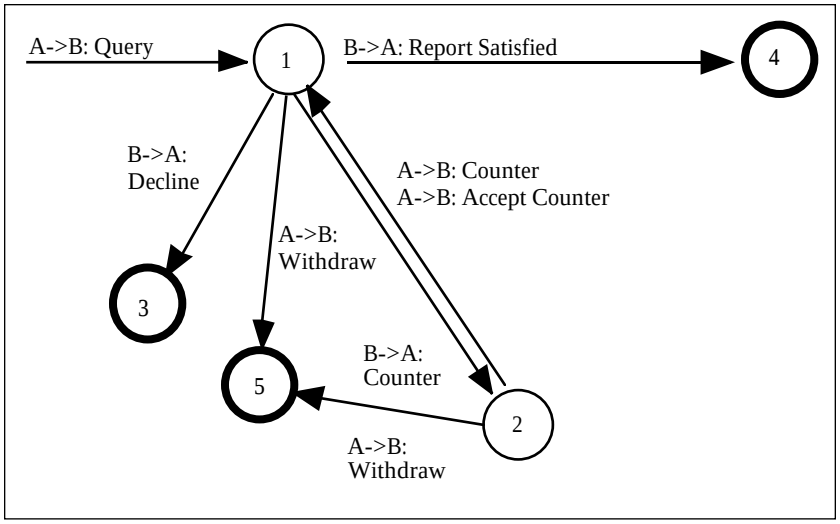

Figure 11. The Query conversation policy.

show what has been newly added: one conversation policy, one verb, and the participation information.

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 table 5. 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.[29] 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.[30]

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

Core Suite	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

Table 4. Completing the Core suite by adding the Query conversation policy.

- *Advertise, Offer*, advertiser
- *Retire, Inform*, retiree
- *Recommend, Query*, requester

Emulating Common KQML Agent Interactions. Using the Core and Matchmaker suites as described, one could straightforwardly emulate common types of KQML interactions described by Finin, Labrou, and Mayfield (1997):

- The simple KQML *ask/tell* sequence is identical to the simplest case of the KAoS *Query* where an *A:B query* message would be followed by a *B:A report satisfied* message.
- The KQML *subscribe* example could be implemented as a *Request* that resulted in a series of *inform* messages sent from state 1 whenever the variable of interest changed. The conversation would continue until B decided to send a *decline* message to end the original request for the subscription service. Alternatively, a new *subscribe* verb and/or conversation policy could be added.
- The KQML *recruit* example could be implemented as three separate conversations: the first as an *advertise* conversation between the Matchmaker and

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

Table 5. The Core and Matchmaker suites.

Agent B, the second as a *recommend* conversation between the Matchmaker and Agent A, and the third, once Agent B was located, as a *query* from A to B. Alternatively, a new, more complex conversation policy for *recruit* or *recommend* could be defined.

- The KQML *broker* conversation would be handled in a similar fashion to *recommend*, except that a new verb would need to be added to the Matchmaker to handle the indirection of the reply.
- The KQML *recommend* example is equivalent to the KAoS Matchmaker's *recommend* conversation.

Applications

In this section we describe some of the applications of KAoS to date.

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 (Tockey et al. 1995; Atler et al. 1994).

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 et al. 1995). 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 (Barker et al. 1995). A simulation of interaction with the agent system through electronic mail and agent learning of user preferences was also created. 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.

Our experience indicated that a set of utilities to aid the construction, debugging, and maintenance of agents would be invaluable for future applications. We 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. Finally, we have explored the use of NASA's CLIPS development environment to represent and operate on Matchmaker knowledge.

Gaudi Intelligent Performance Support Architecture

The Boeing Company is exploring the use of portable airplane maintenance aids (PMA) and online Web-based tools (Boeing OnLine Data—BOLD) to provide training and support to customers (Guay 1995; Bradshaw et al. 1993) A new version of KAoS is being incorporated into one such prototype of an intelligent performance support system (Bradshaw, et al. 1997). The system, named *Gaudi,*[31] is being designed around the actual processes, activities, and resources of the work environment. It is intended to directly and actively support neces-

sary tasks, adapting information to the requirements of the user and situation. A similar architecture is being developed to support large-scale collaberation between medical staff at the Fred Hutchinson Cancer Research Center and primary-care physicians worldwide (Bradshaw et al. 1997).

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 an adaptive engine into *Gaudi*. The adaptive component is described in more detail in the Learning and Adaptivity subsection that follows.
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, and 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. Linking to a variety of live dynamic datafeeds is of particular importance We have implemented an agent-assisted external linking facility that implements dynamic links without requiring 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.
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.

Issues and Future Directions

Work in progress on mobile agents, formalizing semantics and modeling dialogue as a joint activity, and learning and adaptivity are described in this section.

Mobile Agents

We 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 connec-

tion 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 developing a Java implementation of KAoS and are enhancing 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 the agent's extension (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]).[32]

Formalizing Semantics and Modeling Dialogue as a Joint Activity

To date, we have attempted no formal description of the semantics of KAoS agent communication. Ongoing progress in such formalizations is summarized by Cohen and Levesque (1997) and Labrou (Labrou 1996; Labrou and 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, and Rosenschein 1994; Wooldridge and Jennings 1994; Jennings 1993; and Shoham and Tennenholtz 1992). We have noted in the discussion of conversations 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 and Rosenschein 1994], teamwork [Cohen and Levesque 1991]).

For example, Cohen (1994) discusses 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 he describes (nonliteral language, multifunctional utterances, etc.) are more important for human-human or human-agent communication than for agent-agent interaction using a very restricted language, he makes 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. Cohen summarizes the rationale for plan-based dialogue theories as follows:

> Plan-based models are founded on the observation that utterances are not simply

> strings of words, but rather are the observable performance of speech acts… Plan-based theories of communicative action and dialogue assume that the speaker's speech acts are part of a plan, and the listener's job is to uncover and respond appropriately to the underlying plan, rather than just to the utterance. (p. 187).

As an argument he cites Grice's (1975) well-known example of a pedestrian with an empty gas can who asks "Where is the nearest gas station?" The answer "it's two blocks down the road" may be truthful, and a perfectly conformant response given the conversation policy in force, "but would be useless if the speaker knew that the gas station were closed. Rather, what a cooperative dialogue participant is *supposed* to do is provide an answer that addresses the speaker's goals, plans, here, one that directs him to the nearest *open* gas station." (Cohen 1994, p. 187)

Cohen and Levesque (1991) develop the concept of a "joint intention" that applies to a set of agents involved in cooperative dialogue.[33] According to theory, team behavior is more than coordinated individual behavior: it involves the mutual adoption of beliefs, intentions, and goals. For example, when an agent agrees to respond to the query for the nearest gas station, a helpful answer should be seen in terms of its having adopted some subset of the requester's goal structure, agreeing not only to fulfill the manifest request but also to do whatever is reasonable to help satisfy the questioner and meet his objectives. This may include, for example, finding and removing obstacles to the success of the plan, or even recommending a different plan if it is known that the current one will fail.

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. Smith and Cohen (1996) have begun the development of a semantics of agent communication that would allow the rigorous analysis of conversation policies such as those described in this chapter (see also Cohen and Levesque 1997). 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. We expect to accommodate an emergent model of agent communication in a future version of KAoS.[34]

Learning and Adaptivity

It is very difficult to write successful general-purpose agents. That is why some of the most successful applications of adaptive agents (Maes 1997) and programming-by-demonstration environments (Cypher 1993) have been for applications such as mail and calendar managers, where the learning algorithms could operate within a strong task-model that defined context.

Unlike more routine applications, the complexity and dynamic nature of many aerospace problems make it very difficult to anticipate situations or contexts that agents will be encountering. Thus not only the situation patterns, but also much of the task model and contextual knowledge must be acquired online and incrementally as agents perform tasks in the real world.

We intend to provide a framework for the acquisition of situational knowledge by reimplementing and refining the Situation Recognition and Analytical Reasoning (SRAR) model (Bradshaw 1994; Boy and Mathé 1993; Boy 1991; Mathé 1990). The SRAR model was originally developed in 1986 as part of a project to aid astronauts in diagnosing faults in the orbital refueling system of NASA's space shuttle. It has subsequently been applied at NASA Ames to develop a suite of computer-integrated documentation (CID) (Boy 1992; Chen and Mathé 1992) and telerobotics (Mathé and Kedar 1992) applications. Working in collaboration with the originators of this approach, we have begun to integrate and extend selected CID contextual learning mechanisms into the KAoS architecture to assess their value in performance support applications (Mathé and Chen 1994).

The SRAR model provides a formal framework for integrating situational (problem statement situational patterns) and analytical (problem-solving resources) knowledge (figure 12). In the beginning, agents are "inexperienced" and must rely on broad analytic knowledge (e.g., nominal models of tasks and procedures that may be incomplete and incorrect). These analytic models may be acquired through automated knowledge acquisition (Bradshaw et al. 1993) or process- and task-modeling tools (Bradshaw et al. 1992). Learning mechanisms rely on the reinforcement of successful actions, the discovery of failure conditions, and the generation of recovery actions to improve performance. Elements of the analytical knowledge are transferred into situation patterns that embody refinements of how procedures are carried out in real fact by particular people in particular contexts. Over time, situational patterns multiply and become more complex, while analytical knowledge becomes more structured. The result over time is a set of agents that have learned by experience how to adapt to particular people and situation patterns.

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 Open-

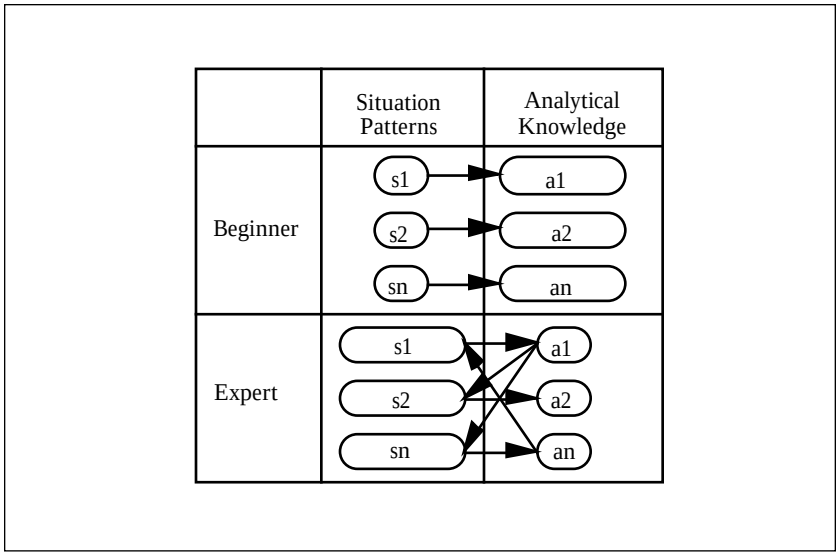

Figure 12. The SRAR model.

Doc, ActiveX, and Java and with distributed object services such as those provided by CORBA, DCOM, AND THE INTERNET

- It supports structured conversations that 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; and 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, permitting 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 (yellow pages), which are advertised to the Matchmaker by agents offering services
- It provides facilities for agent names (white pages), which allow a Domain Manager to 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 allows 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 (e.g., Aglets) and protocols (e.g., KQML) 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 all those whose contributions have made KAoS a reality: especially Jack Woolley (advisor), Steve Tockey (advisor), and members of the 1992-93 Seattle University KAoS team (Nick George, Rob Jasper, Monica Rosman LaFever, Katrina Morrison, Dan Rosenthal, Steve Tockey), the 1993-94 AgONy! team (David Atler, Mike Bingle, Tim Cooke, John Morgan, Dan Van Duine, Michael Zonczyk), 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, and the 1995-96 KAoS OpenDoc (Khalighi Dariush, Harold Edwards, Dennis O'Brien, James "Bat" Masterson, Feliks Shostak, Norm Thorsen) and KAoS OLE (David Chesnut, Phil Cooper, Mike Faulkner, Dan Klawitter, Phuoc Huu Nguyen, Jaimie Roeder) teams. Thanks also to Ian Angus, Larry Baum, Isabelle Bichindaritzz, John Boose, Guy Boy, Kathleen Bradshaw, Alberto Cañas, Bob Carpenter, Dick Chapman, Jim Chen, Phil Cohen, Dave Cooper, Stan Covington, Rob Cranfill, Al Erisman, Megan Eskey, Ken Ford, Peter Friedland, Brian Gaines, Ralph Giffin, Kirk Godtfredsen, Roger Guay, Pat Henderson, Pete Holm, Earl Hunt, Renia Jeffers, Dick Jones, Randy Kelley, Oscar and Sharon Kipersztok, Cathy Kitto, Vicki Lane, Henry Lum, David Madigan, Jack Martz, Cindy Mason, John McClees, Bill McDonald, Mark Miller, Nathalie Mathé, Peter Morton, Ken Neves, Janet Nims, Sue Pickett, Luis Poblete, Steve Poltrock, Judy Powell, Josh Rabinowitz, Alain Rappaport, Tyde Richards, Tom Robinson, Kurt Schmucker, John Schuitemaker, Doug Schuler, Dick Shanafelt, Kish Sharma, Mildred Shaw, Dave Shema, David C. Smith, Jim Spohrer, Keith Sullivan, Amy Sun, Steve Tanimoto, Rob Thompson, Jim Tomlinson, Sankar Virdhagriswaran, Helen Wilson, and Debra Zarley. The work described in this chapter was supported in part by grant R01HS09407 from the Agency for Health Care Policy and REsearch and by a collaborative research agreement with NASA-Ames.

We expect to make a version of KAoS publically available in late 1997. For information on the availability of KAoS, contact Jeff Bradshaw at jeffrey.m.bradshaw@boeing.com, (206) 865-6086.

Notes

1. For surveys illustrating the variety of software agents research, see, for example, Bradshaw (1997), Wooldridge and Jennings (1995), and Nwana (1996).

2. The OMG is the world's largest software development consortium with a membership of over 600 software vendors, developers and end users. Established in 1989, its goal is to provide a common architecture framework for distributed object-oriented applications based on widely available interface specifications.

3. Within the OMG, work is underway to provide standards for interoperability between Microsoft's Distributed Component Object Model (DCOM) and CORBA. Many ORB vendors already provide their own versions of this capability. Various approaches to providing object system interoperability are discussed by Foody (1995).

4. In the spirit of Hewitt's "open systems" (Hewitt 1991; Hewitt and Inman 1991), we do not believe that it is practical or desirable that future systems rely on a specific common agent architecture. What is important is that societies of different kinds of agents, regardless of internal structure, be able to coordinate their activities (Haddadi and Sundermeyer 1996), and that specialized agent architectures and languages optimized for particular domains can proliferate while still being able to interoperate with more general ones.

5. 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 the Mediation Agents subsubsection).

6. It is still too early to tell if agent-oriented programming will require fundamentally different models of software development (Raccoon 1995) and user-interface design (Gentner and Nielsen 1995; Erickson 1997).

7. Russell and Norvig (1995, 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 (Resnick and Martin 1990; Martin 1988) describe examples of how, in real life, people quite easily and naturally shift between the different kinds of descriptions of designed artifacts. See Erickson (1997) for an additional useful perspective on the advantages and disadvantages of encouraging users to think in terms of agents.

8. See Haddadi and Sundermeyer (1996) for a survey of belief-desire-intention (BDI) agent architectures. Note that the idea of "inheriting" knowledge is somewhat different than that of inheriting methods, or attributes.

9. Alternatively, we have considered whether facts should be defined as beliefs that are global (i.e., all agents pointed to the same set of facts). This definition would prevent the problem of two agents having contradicting "facts" (which is otherwise possible in our architecture). However this approach would impose the daunting requirement that all agents in a potentially unbounded and dynamic agent environment have continuous access to the current global set of facts.

10. There are many issues related to agent persistence that currently remain unsolved. For example, there are problems associated with deactivating an agent while it is involved in an ongoing conversation. If the agent is saved then restored to an environment that has significantly changed, much of its previous knowledge, desires, and intentions may no longer apply.

11. The basic message unit in KAoS is the message tag: {tag = value; }, which can be expanded by recursively replacing the value with another message tag or series of message tags. Our approach is similar in spirit to that of Sims, who has argued the benefits of "semantic data" in OMG forums and in various publications (e.g., Sims 1994, pp. 138–144).

12. The KAoS architecture neither requires interpretation of content by Matchmaker and proxy agents nor disallows it when it is possible and desirable to do so. By way of

comparison, Finin's description of KQML (Finin, Labrou, and Mayfield 1997) states that every implementation "ignores the content portion of the message," whereas Genesereth's (1997) 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.

13 An agent that comprises more than one agent is called a composite agent. It would necessarily contain more than one instance of the generic agent.

14 Strictly speaking, the bounds of the set of communicating ORBs would constitute the maximum bounds of a particular agent domain; the actual bounds of a particular agent domain could be much smaller according to what was most convenient for agent developers. The domain constitutes the logical unit of administration for some set of agents, so in principle several agent domains with different policies or application scope could coexist on a single ORB.

15. 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 (Brockschmidt 1994). 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 used here—namely, the composition of an entity (such as an agent) from several object instances.

16. As Finin et al. (1995) observe, proxies can be used to provide a number of services: firewall gateways, protocol gateways, message processing, filtering and annotating, and agent composition.

17. The ability to carry on socket-based communication is currently required of all proxy agents, who optimally may implement additional protocols as well.

18. The Matchmaker performs a similar rôle to a KQML "agent server" facilitator which uses the advertise and recommend performatives (Finin, Labrou, and Mayfield 1997). See Kuokka and Harada (1995) for a discussion of KQML and matchmaking; and Decker, Williamson, and Sycara (1996) for a comparison of matchmaking and brokering approaches. Future versions of KAoS may include brokering capability.

19. Notable exceptions are Barbuceanu and Fox's (1995) COOL, Kuwbara's (1995) AgenTalk, and the GOAL cooperation service framework (Cunningham 1995). Labrou (1996) and Labrou and Finin (1994) have suggested a scheme by which a future version of KQML could implement conversation policies.

20. For an excellent discussion on the role of convention in language use, see Deuchar (1990).

21. Nothing in the architecture precludes a more sophisticated approach, based on an emergent model of agent communication and notions such as joint intention and planning (see the Formalizing Semantics and Modeling Dialogue as a Joint Activity subsection).

22. A concept similar to our conversation policies is that of dialogue grammars (Cohen 1994). We discuss limitations of dialogue grammar models for agent communication in the Formalizing Semantics and Modeling Dialogue as a Joint Activity subsection.

23. An object-oriented design for a finite state machine is described by Ackroyd (1995).

24. The Query conversation policy is described in the Query subsubsection.

25. We are not, however, claiming that the conversation for action model is necessarily well suited for human-to-human conversation (Cohen 1994; DeMichelis and Grasso 1994; Suchman 1993; Cohen and Levesque 1991; Robinson 1991; Bowers and Churcher 1988).

26. See von Martial (1992) for a discussion of asynchronous conversation design techniques abased on finite state models.

27. There is an analog to Apple Event Suites, which group high-level interprocess events supporting a functional area (Apple 1993). Requirements for suite conformance in KAoS, however, are somewhat more stringent than in Apple Event Suites.

28. Replacement of the initial verb of a conversation policy in a specialized suite is permissible when the new initial verb is a strict specialization of the generic illocutionary act in the original conversation policy. For example, *query* is a more specialized *request,* and *decline* is a more specific kind of *inform.*

29. The reason that an advertisement uses the Offer conversation policy rather than the one for Inform is to give the Matchmaker 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). On the other hand, the Inform conversation policy is used for retire, since the agent providing the service should be able to control when its services are no longer available (i.e., the Matchmaker should not be able to refuse the agent's announcement that it is withdrawing its services).

30. The Domain Manager suite and the Proxy suite also reuse conversation policies from the Core suite.

31. 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 (Tarrago 1992). This monumental unfinished structure, on which construction still continues after more than a hundred years, symbolizes our desire to investigate architectures 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 (Brand 1994).

32. Standardization of agent transfer protocols is an increasing topic of concern. See for example, discussions by White (Gardner 1996; White 1996), Chang, and Lange (Lange 1996; Chang and Lange 1996).

33. A related concept is Clark's (1992, p. 258) notion of joint conversational or perceptual experiences. The idea is that two people "cannot talk successfully to each other without appealing to their common ground," i.e., "the sum of their mutual knowledge, mutual beliefs, and mutual suppositions" (Clark 1992, p. 3).

34. An implementation of the joint action model would require that KAoS allow for conversations between more than two agents. We have not yet implemented such a capability.

References

Ackroyd, M. 1995. Object-Oriented Design of a Finite-State Machine. *Journal of Object-Oriented Programming* VOLUME(ISSUE): 50–59.

Apple. 1993. *Inside Macintosh: Interapplication Communication*., Reading, Mass.: Addison-Wesley.

Atler, D.; Bingle, M.; Cooke, T.; Morgan, J.; Duine, D. V.; and Zonczyk, M. 1994. *AGONY! Battleship Documentation*, Department of Software Engineering, Seattle University.

Ball, G.; Ling, D.; Kurlander, D.; Miller, J.; Pugh, D.; Skelly, T.; Stankosky, A.; Thiel, D.; Dantzich, M. V; and Wax, T. 1996. Lifelike Computer Characters: The Persona Project at Microsoft Research. In *Software Agents*, ed J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.

Barbuceanu, M., and Fox, M. S. 1995. COOL: A Language for Describing Coordination in Multi-Agent Systems. In Proceedings of the First International Conference on Multi-Agent Systems, ed. V. Lessor, 17–24. Menlo Park, Calif.: American Association for Artificial Intelligence.

Barker, D.; Benoit, P.; Tomlinson, J.; and Landon, S. 1995. kaos corba Design Document, Department of Software Engineering, Seattle University.

Bos, M.; Boyer, R.; Dutfield, S.; and Jones, E. J. 1995. *TAINT OPEN JOB Design Document*, Department of Software Engineering, Seattle University.

Bowers, J., and Churcher, J. 1988. Local and Global Structuring of Computer-Mediated Representation: Developing Linguistic Perspectives on CSCW in COSMOS. In Proceedings of the Conference on Computer-Supported Cooperative Work. New York: Association of Computing Machinery.

Bowman, C. M.; Danzig, P. B.; Manber, U.; and Schwartz, M. F. 1994. Scalable Internet Resource Discovery: Research Problems and Approaches. *Communications of the ACM* 37(8): 98–107, 114.

Boy, G. 1992. Computer-Integrated Documentation. In *Sociomedia: Multimedia, Hypermedia, and the Social Construction of Knowledge*, ed. E. Barrett, 507–531. Cambridge, Mass.: MIT Press.

Boy, G. A. 1997. Software Agents for Cooperative Learning. In *Software Agents*, ed J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.

Boy, G. A. 1991a. Indexing Hypertext Documents in Context. In Proceedings of the Third ACM Conference on Hypertext, 1–11. New York: Association of Computing Machinery.

Boy, G. A. 1991b. *Intelligent Assistant Systems*. San Diego, Calif.: Academic.

Boy, G. A., and Mathé, N. 1993. Operator Assistant Systems: An Experimental Approach Using a Telerobotics Application. In *Knowledge Acquisition as Modeling*, eds. K. M. Ford and J. M. Bradshaw, 271–286. New York: Wiley.

Bradshaw, J. M.; Robinson, T.; and Jeffers, R. 1995. GAUDI: An Unfinished Architecture for Performance Support. Paper presented at the Fifth International Conference on Human-Machine Interaction and Artificial Intelligence in Aerospace (HMI-AI-AS '95), Toulouse, France.

Bradshaw, J. M. 1997. An Introduction to Software Agents. In *Software Agents*, ed J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.

Bradshaw, J. M. 1994. Adaptivity in KAOS, A Knowledgeable Agent-Oriented System. Paper presented at the International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU), Paris, France.

Bradshaw, J. M.; Boose, J. H.; Covington, S. P.; and 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, 4.1–4.20. Banff, Alberta, Canada: SRDG Publications.

Bradshaw, J. M.; Carpenter, R.; Cranfill, R.; Jeffers, R.; Poblete, L.; Robinson, T.; and Sun, A. 1997. The Many Roles of Agent Technology in Knowledge Management: Examples from Applications in Aerospace and Medicine. Presented at the AAAI Spring Symposium on Knowledge Management, Stanford University, Stanford, California, March.

Bradshaw, J. M.; Ford, K. M.; Adams-Webber, J. R.; and Boose, J. H. 1993. Beyond the Repertory Grid: New Approaches to Constructivist Knowledge-Acquisition Tool Development. In *Knowledge Acquisition as Modeling*, eds. K. M. Ford and J. M. Bradshaw, 287–333. New York: Wiley.

Bradshaw, J. M.; Holm, P.; Kipersztok, O.; and Nguyen, T. 1992. *EQUALITY:* An Application of *AXOTL II* to Process Management. In *Current Developments in Knowledge Acquisition: EKAW-92, eds.* T. Wetter, K.-D. Althoff, J. H. Boose, B. R. Gaines, M. Linster, and F. Schmalhofer, 425–444. Berlin: Springer-Verlag.

Bradshaw, J. M.; Holm, P.; Kipersztok, O.; Nguyen, T.; Russo P. J.; and Boose, J. H. 1991. Intelligent Interoperability in DDUCKS. Paper presented at the AAAI-91 Workshop on Cooperation among Heterogeneous Intelligent Systems, Anaheim, California.

Bradshaw, J. M.; Richards, T.; Fairweather, P.; Buchanan, C.; Guay, R.; Madigan, D.; and Boy, G. A. 1993. New Directions for Computer-Based Training and Performance Support in Aerospace. Paper presented at the Fourth International Conference on Human-Machine Interaction and Artificial Intelligence in Aerospace, 28–30 September, 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, Wash.: Microsoft.

Brown, C.; Gasser, L.; O'Leary, D. E.; and Sangster, A. 1995. AI on the WWW: Supply-and-Demand Agents. *IEEE Expert*, 10(4): 50–55.

Browne, D.; Totterdell, P.; and Norman, M., eds. 1990. *Adaptive User Interfaces.* San Diego, Calif.: Academic.

Campagnoni, F. R. 1994. IBM's System Object Model. *Dr. Dobb's* 24–28.

Carter, J. 1992. Managing Knowledge: The New Systems Agenda. *IEEE Expert*, 3–4.

Chang, D. T., and Lange, D. B. 1996. Mobile Agents: A New Paradigm for Distributed Object Computing on the WWW. In Proceedings of the OOPSLA 96 Workshop "Toward the Integration of WWW and Distributed Object Technology."

Chen, J. R., and Mathé, N. 1995. Learning Subjective Relevance to Facilitate Information Access. Submitted to CIKM-95.

Clark, H. H. 1992. *Arenas of Language Use*. Chicago, Ill.: University of Chicago Press.

Cohen, P. R. 1994. Models of Dialogue. In Cognitive Processing for Vision and Voice: Proceedings of the Fourth NEC Research Symposium, ed. T. Ishiguro, 181–203. Philadelphia: Society for Industrial and Applied Mathematics.

Cohen, P. R.; and Levesque, H. 1997. Communicative Actions for Artificial Agents. In *Software Agents*, ed J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.

Cohen, P. R., and Levesque, H. J. 1991. *Teamwork*, Technote 504, SRI International, Menlo Park, California.

Cohen, P. R., and Levesque, H. J. 1990. Intention Is Choice with Commitment. *Artificial Intelligence* 42(3).

Cunningham, J. 1995. GOAL Cooperation Service Framework. In Proceedings of the First International Conference on Multi-Agent Systems, ed. V. Lessor, addendum. Menlo Park, California: American Association for Artificial Intelligence.

Cypher, A., ed. 1993. *Watch What I Do: Programming by Demonstration.* Cambridge, Mass.: MIT Press.

Decker, K.; Williamson, M.; and Sycara, K. 1996. Matchmaking and Brokering. In *Proceedings of the Second International Conference on Multiagent Systems (ICMAS-96)* Menlo Park, Calif.: AAAI Press.

DeMichelis, G., and Grasso, M. A. 1994. Situating Conversations within the Language-

Action Perspective: The MILAN Conversation Model. In Proceedings of the Conference on Computer-Supported Cooperative Work, eds. R. Furuta and C. Neuwirth, 89–100. New York: Association of Computing Machinery.

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

Deuchar, M. 1990. Are the Signs of Language Arbitrary? In*Images and Understanding*, eds. H. Barlow, C. Blakemore, and M. Weston-Smith. Cambridge, U.K.: Cambridge University Press.

Durfee, E. H.; Gmytrasiewicz, P.; and Rosenschein, J. S. 1994. The Utility of Embedded Communications: Toward the Emergence of Protocols. In Proceedings of the Thirteenth International Distributed Artificial Intelligence Workshop, eds. M. Klein and K. Sharma, 85–93, Boeing Information and Support Services, SEattle, Washington.

Englemore, R., ed. 1988. *Blackboard Systems.* Reading, Mass.: Addison-Wesley.

Etzioni, O., and Weld, D. S. 1995. Intelligent Agents on the Internet: Fact, Fiction, and Forecast. *IEEE Expert* 10(4): 44–49.

Etzioni, O., and Weld, D. 1994. A Softbot-Based Interface to the Internet. *Communications of the ACM* 37(7): 72–76.

Finin, T., Labrou, Y., & Mayfield, J. 1997. KQML as an agent communication language. In *Software Agents*, ed J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.

Finin, T.; Potluri, A.; Thirunavukkarasu, C.; McKay, D.; and McEntire, R. 1995. On Agent Domains, Agent Names, and Proxy Agents. Paper presented at the CIKM Workshop on Intelligent Information Agents, Baltimore, Maryland.

Foody, M. 1995. Providing Object System Interoperability with Middleware. *Cross-Platform Strategies: Supplement to SIGS Publications* 11–21.

Gardner, E. 1996. Standards Hold Key to Unleashing Agents. *Web Week* 5.

Gasser, L. 1991. Social Conceptions of Knowledge and Action: DAI Foundations and Open Systems Semantics. *Artificial Intelligence* 47:107–138.

Genesereth, M. R. 1997. An Agent-based Framework for Interoperability. In *Software Agents*, ed. J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.

Gentner, D., and Nielsen, J. 1995). The Anti-Mac: Violating the MACINTOSH Human-Interface Guidelines. In Proceedings of CHI-95, 183–184. New York: Association of Computing Machinery.

George, N.; Jasper, R.; LaFever, M. R.; Morrison, K.; Rosenthal, D.; Tockey, S.; Woolley, J.; Bradshaw, J. M.; Boy, G.; and Holm, P. D. 1994. KAOS: A Knowledgeable–Agent-Oriented System. Paper presented at the AAAI Spring Symposium on Software Agents, 21–23 March, Stanford, California.

Grice, H. P. 1975. Logic and Conversation. In *Syntax and Semantics: Speech Acts*, ed. H. P. Grice. San Diego, Calif.: Academic.

Guay, R. L. 1995. Notebook Simulations as Electronic Performance Support Tools for Airline Maintenance. Paper presented at the Royal Aeronautical Society Flight Simulation Group Simulation in Aircraft Maintenance Training Conference, DATE, London, United Kingdom.

Haddadi, A., and Sundermeyer, K. 1996. Belief-Desire-Intention Agent Architectures. In *Foundations of Distributed Artificial Intelligence*, eds. G. M. P. O'Hare and N. R. Jennings, 169–185. New York: Wiley.

Hanks, S.; Pollack, M. E.; and Cohen, P. R. 1993. Benchmarks, Test Beds, Controlled Experimentation, and the Design of Agent Architectures. *AI Magazine* 14(4): 17–42.

Hewitt, C. 1991. Open Information Systems Semantics for Distributed Artificial Intelligence. *Artificial Intelligence* 47:79–106.

Hewitt, C., and 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.

Jennings, N. R. 1993. Commitments and Conventions: The Foundation of Coordination in Multi-Agent Systems. *Knowledge Engineering Review* 8(3): 223–250.

Kaehler, T., and Patterson, D. 1986. A Small Taste of Smalltalk. *BYTE* 145–159.

Knoblock, C. A., & Ambite, J.-L. 1996. Agents for Information Gathering. In *Software Agents*, ed. J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.

Kuokka, D., and Harada, L. 1995. On Using KQML for Matchmaking. In Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), ed. V. Lesser, 239–245. Menlo Park, Calif.: American Association for Artificial Intelligence.

Kuwabara, K. 1995. AGENTALK: Coordination Protocol Description for Multiagent Systems. In Proceedings of the First International Conference on Multi-Agent Systems, ed. V. Lesser, addendum. Menlo Park, Calif.: American Association for Artificial Intelligence.

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

Labrou, Y., and Finin, T. 1994. A Semantics Approach for KQML—A General-Purpose Communication Language for Software Agents. In Proceedings of the Third International Conference on Information and Knowledge Management, eds. N. R. Adam, B. K. Bhargava, and Y. Yesha, 447–455. New York: Association of Computing Machinery.

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

Maes, P. 1997. Agents that Reduce Work and Information Overload. In *Software Agents*, ed. J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.

Malcolm, K. C.; Poltrock, S. E.; and Schuler, D. 1991. Industrial-Strength Hypermedia: Requirements for a Large Engineering Enterprise. In Proceedings of the Third ACM Conference on Hypertext, PP–PP. New York: Association of Computing Machinery.

Malone, T. W.; Grant, K. R.; and Lai, K.-Y. 1996. Agents for Information Sharing and Coordination: A History and Some Reflections. In *Software Agents*, ed. J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.

Martin, F. 1988. Children, Cybernetics, and Programmable Turtles. Master's thesis, Media Laboratory, Massachusetts Institute of Technology.

Mathé, N. 1990. A Space Remote Control Application: Cognitive Modeling and Blackboard-Based Implementation. Paper presented at the Conference on Human-Machine Interaction in Aeronautics and Space, Toulouse-Blagnac, France.

Mathé, N., and Chen, J. 1994. A User-Centered Approach to Adaptive Hypertext Based on an Information Relevance Model. Paper presented at the Fourth International Conference on User Modeling (UM '94), Hyannis, Massachusetts.

Mathé, N., and Kedar, S. T. 1992. Increasingly Automated Procedure Acquisition in Dynamic Systems. Paper presented at the Seventh Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada.

Nelson, P. R., and Schuler, D. 1995. Managing Engineering Information with Hypermedia, Boeing Commercial Airplane Group, Seattle Washington.

Nelson, T. 1980a. Interactive Systems and the Design of Virtuality, Part 1. *Creative Computing* 56–62.

Nelson, T. 1980b. Interactive Systems and the Design of Virtuality, Part 2. *Creative Computing* 95–106.

Nwana, H. S. 1996. Software Agents: An Overview. *Knowledge Engineering Review* Forthcoming.

O'Hare, G. M. P.; and Jennings, N. R. ed. 1996. *Foundations of Distributed Artificial Intelligence.* New York: John Wiley and Sons.

Orfali, R.; Harkey, D.; and Edwards, J. 1995. Client-Server Components: CORBA meets OPENDOC. *Object Magazine* 55–59.

Raccoon, L. B. S. 1995. The CHAOS Model and the CHAOS Life Cycle. *ACM SIGSOFT Software Engineering Notes* 20(1): 55–66.

Reinhardt, A. 1994. The Network with Smarts. *BYTE* 10: 50–64.

Repenning, A. 1993. AGENT SHEETS: A Tool for Building Domain-Oriented Dynamic, Visual Environments. Ph.D. diss., University of Colorado.

Resnick, M., and Martin, F. 1990. Children and Artificial Life, E&L Memo, 10, Media Laboratory, Massachusetts Institute of Technology.

Rettig, M. 1991. Nobody Reads Documentation. *Communications of the ACM* 34(7): 19–24.

Richman, D. 1995. Let Your Agent Handle It. *InformationWeek* 44–56.

Riecken, D. 1997. The M System. In *Software Agents,* ed. J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.

Robinson, M. 1991. Computer-Supported Cooperative Work: Case and Concepts. In *Groupware 1991: The Potential of Team and Organisational Computing,* eds. P. R. Hendriks, 59–75. Utrecht, The Netherlands: SERC.

Russell, S., and Norvig, P. 1995. *Artificial Intelligence: A Modern Approach,* New York: Prentice-Hall.

Saffo, P. 1994. It's the Context, Stupid. *Wired* 3:74–75.

Shoham, Y. 1997. An Overview of Agent-oriented Programming. In *Software Agents,* ed J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.

Shoham, Y. 1993. Agent-Oriented Programming. *Artificial Intelligence,* 60(1): 51–92.

Shoham, Y., and Tennenholtz, M. 1992. On the Synthesis of Useful Social Laws for Artificial Agent Societies. In Proceedings of the Tenth National Conference on Artificial Intelligence, 276–281. Menlo Park, Calif.: American Association for Artificial Intelligence.

Siegel, J. 1996. *CORBA: Fundamentals and Programming.* New York: John Wiley.

Sims, O. 1994. *Business Objects: Delivering Cooperative Objects for Client-Server*. New York: McGraw-Hill.

Singh, M. P. 1994. *Multiagent Systems: A Theoretical Framework for Intentions, Know-How, and Communication*. Berlin: Springer-Verlag.

Smith, I. A., and Cohen, P. R. 1996. Toward a Semantics for an Agent Communications Language based on Speech-Acts. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence,* 24-31. Menlo Park, Calif.: AAAI Press.

Spohrer, J. C.; Vronay, D.; and Kleiman, R. 1991. Authoring Intelligent Multimedia Ap-

plications: Finding Familiar Representations for Expressing Knowledge. Paper presented at the 1991 IEEE International Conference on Systems, Man, and Cybernetics, Charlottesville, Virginia.

Suchman, L. 1993. Do Categories Have Politics? The Language-Action Perspective Reconsidered. In *Proceedings of the Third European Conference on Computer-Supported Cooperative Work*, 1–14. Dordrecht, The Netherlands: Kluwer Academic.

Tambe, M.; Johnson, W. L.; Jones, R. M.; Koss, F.; Laird, J. E.; Rosenbloom, P. S.; and Schwamb, K. 1995. Intelligent Agents for Interactive Simulation Environments. *AI Magazine* 16(1): 15–39.

Tarrago, S. 1992. *Gaudi*. Barcelona: Editorial Escudo de Oro, S.A.

Tockey, S.; Rosenthal, D.; Rosman LaFever, M.; Jasper, R.; George, N.; Woolley, J. D.; Bradshaw, J. M.; and Holm, P. D. 1995. Implementation of the KAOS Generic Agent-to-Agent Protocol. *Northwest AI Forum (NAIF) Journal*.

Van de Velde, W. 1995. Cognitive Architectures—From Knowledge Level to Structural Coupling. In *The Biology and Technology of Intelligent Autonomous Agents*, ed. L. Steels, 197–221. Berlin: Springer-Verlag.

Virdhagriswaran, S. 1994. Heterogeneous Information Systems Integration: An Agent-Messaging–Based Approach. Presented at the CIKM-94 Workshop on Intelligent Agents, Gaithersburg, Maryland.

Virdhagriswaran, S.; Osisek, D.; and O'Connor, P. 1995. Standardizing Agent Technology. *ACM Standards View*. Forthcoming.

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

Walker, A., and Wooldridge, M. 1995. Understanding the Emergence of Conventions in Multi-Agent Systems. In Proceedings of the First International Conference on Multi-Agent Systems, ed. V. Lessor, 384–389. Menlo Park, Calif.: American Association for Artificial Intelligence.

Wayner, P. 1995. Free Agents. *BYTE* 105–114.

White, J. 1996. A Common Agent Platform, General Magic, Inc. Sunnyvale, Calif.

Wiederhold, G. 1992. Mediators in the Architecture of Future Information Systems. *IEEE Computer* 38–49.

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

Woelk, D.; Huhns, M.; and Tomlinson, C. 1995. Uncovering the Next Generation of Active Objects. *Object* 4:33–40.

Wooldridge, M. J., and Jennings, N. R. 1995. Agent Theories, Architectures, and Languages: A Survey. In *Intelligent Agents: ECAI-94 Workshop on Agent Theories, Architectures, and Languages*, eds. M. J. Wooldridge and N. R. Jennings, 1–39. Berlin: Springer-Verlag.

Wooldridge, M. J., and Jennings, N. R. 1994. Formalizing the Cooperative Problem-Solving Process. In Proceedings of the Thirteenth International Distributed Artificial Intelligence Workshop, eds. M. Klein and K. Sharma, 403–417. Seattle, Wash.: Boeing Information and Support Services.

Zlotkin, G., and Rosenschein, J. 1994. *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers*., Cambridge, Mass.: MIT Press.