Roles for Agent Technology in Knowledge Management: Examples from Applications in Aerospace and Medicine

Jeffrey M. Bradshaw[1,3], Robert Carpenter[1], Robert Cranfill[1], Renia Jeffers[1], Luis Poblete[1], Tom Robinson[1], Amy Sun[1], Yuri Gawdiak[2], Isabelle Bichindaritz[3],Keith Sullivan[3]

1. Research and Technology
Boeing Information and Support Services
P.O. Box 3707, M/S 7L-44
Seattle, WA 98124
jeffrey.m.bradshaw@boeing.com

2. NASA Ames Research Center, Code IP
Moffett Field, CA 94035

3. Clinical Research Division
Long-Term Follow-Up, FB-600
Fred Hutchinson Cancer Research Center
1124 Columbia Street
Seattle, WA 98104

Abstract

This paper describes some of the roles of agents in knowledge management based our experience in aerospace and medicine. After an overview of agent technology and the KAoS agent architecture and applications, we show how agents can help address problems of 1) managing dynamic loosely-coupled information sources, 2) how to provide a unifying framework for distributed heterogeneous components, and 3) coordinating interaction at the knowledge-level.

1. The Place of Agents in Knowledge Management and Knowledge Sharing

Predicting the future is difficult business. A few short years ago, it seemed obvious to most of the knowledge-based community that an era of widespread knowledge sharing (Neches, Fikes, Finin, Gruber, Patil, Senator, & Swartout, 1991) was about to begin (Bradshaw, Ford, Adams-Webber, & Boose, 1993). Libraries of ontologies crafted by groups with common interests in particular knowledge domains would enable the development of computational environments in which explicitly represented knowledge would serve as a means of communication among people and their (Bradshaw, Holm, Boose, Skuce, & Lethbridge, 1992; Gruber, 1991.

How closely has reality approached our expectations? A few observations are instructive:

Observation 1. **Knowledge sharing as we originally envisioned it has not occurred on a widespread basis.** This is not meant to imply that efforts to promote knowledge sharing and reusability through methods like the use of ontologies have stopped—indeed, if the proceedings of the 1996 Banff Knowledge Acquisition Workshop are any indication, interest in the topic is healthier than ever. What we are saying is simply that knowledge sharing efforts have not yet had the widespread impact in applications we have been hoping for. Instead, the unforeseen explosion of Web technology and usage has led to a different form of knowledge sharing altogether. No longer does the bottleneck of knowledge acquisition command our attention as it once did—instead, we are scrambling to find ways to impose structure and meaning on the virtual firehose of mostly document-based "knowledge" that is available to us freely on the Web. The transition of Guha from a co-lead of the most ambitious hand-crafted ontology ever (Guha & Lenat, 1990) to a developer of methods for automatically structuring and navigating information on the Web (Guha, 1996) is a striking symbol of this very trend.

Observation 2. **The standard architecture for intelligent systems has been turned inside out.** Instead of one or a few large sophisticated systems that communicate in simple ways, there is an increasing demand for large groups of simple off-the-shelf components whose actions are coordinated in sophisticated ways (Orfali, Harkey, & Edwards, 1996. That the components will be heterogeneous, distributed, and highly interactive is now taken for granted, along with the expectation that the unifying framework in which they operate must not only successfully coordinate their use today but also allow for the introduction of new or replacement components and technologies in the future (Bradshaw, in preparation).

Observation 3. **Progress in standards for component-level interoperability has not**

obviated the requirement for knowledge-level interoperability. The wide adoption of distributed object (CORBA, DCOM, Java), data (HTML, QuickTime), communication (HTTP, TCP/IP, IIOP), and component integration standards (Netscape ONE, OpenDoc, ActiveX, Java Beans) has provided the means for us to package our technologies as interoperable components. However, as has been frequently argued (Bradshaw, 1996b; Gaines & Shaw, 1996; Genesereth, 1996; Gennari, Stein, & Musen, 1996; Kremer, 1996), there is still much work to be done on "knowledge-level" methodologies and standards that can ensure that the operational semantics of these components are explicitly represented.

Software agents have been proposed as one way to help resolve the problems raised by these three observations. While it is true that point solutions not requiring agents could be devised to address many if not all of the issues raised by above, the aggregate advantage of agent technology is that it can address all of them at (Harrison, Chess, & Kershenbaum, 1995.

Software agents can be generally defined as entities that function continuously and autonomously in a particular environment that is often inhabited by other agents and processes. Ideally, agents learn from their experiences, communicate and cooperate with people and with other agents, and, as required, move from place to place within private networks and on the public Internet.

Because the widespread use of agents is a fairly recent phenomenon, there is often confusion expressed about whether some particular software component is "really" an agent or "really" just a program (Franklin & Graesser, 1996). To some degree this is a debate that can never be fully resolved because agenthood is typically a matter of degree rather than kind (Bradshaw, 1996a). For example, while it is true that agents can certainly be implemented in Java, not all Java programs are equally "agent-like":

- Mobile agents tend to move around according to their own agenda ("multi-hop") whereas most garden variety applets are obtained from client pull ("single hop"). This aspect of relative autonomy is perhaps the most distinguishing characteristic of agents.
- Agents are capable of preserving their own state as they are activated, deactivated, and travel from machine to machine, whereas it is typical for applets to start up fresh each time. This gives agents the possibility of adapting and accumulating knowledge and experience over long periods of time.

What kind of roles do agents typically perform? At the user interface, agents can work in conjunction with compound document frameworks and document management tools to select the right data, assemble the needed components, and present the information in the most appropriate way for a specific user and situation (figure 1). Behind the scenes, agents can take advantage of distributed object management, database, workflow, messaging, transaction, searching, indexing, and networking capabilities to discover, link, and securely access the appropriate data and services.

In this paper we describe examples of some of the roles that agent technology can play in knowledge management. First we give an overview of the KAoS agent architecture and some of the aerospace and medical applications to which it is being applied (section 2). Then we show how agents can address problems related to the three observations above by 1) managing dynamic loosely-coupled information sources (section 3), 2) providing a unifying framework for distributed heterogeneous components (section 4), and 3) coordinating interaction at the knowledge-level (section 5).

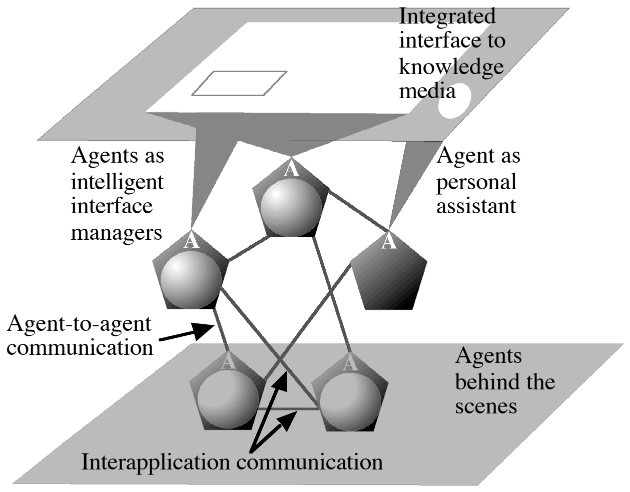

Integrated interface to knowledge media
Agents as intelligent interface managers
Agent as personal assistant
A
A
A
Agent-to-agent communication
A
A
Agents behind the scenes
Interapplication communication

Figure 1. An agent-enabled system architecture.

2. KAoS Overview and Applications

Overview. In 1992, we began a collaboration with the Seattle University (SU) Software Engineering program to develop the first version of the KAoS (Knowledgeable Agent-oriented System) generic agent architecture. We are currently enhancing two main versions: one written in portable Java code and the other written in C++ to take advantage of Microsoft's ActiveX and DCOM technologies. KAoS is described in more detail (Bradshaw, Dutfield, Benoit, & Woolley, 1996).

Basic characteristics of KAoS agents include a consistent structure providing mechanisms allowing the management of knowledge, commitments, choices, and capabilities. Agent dynamics are managed through a cycle that includes the equivalent of agent birth, life, cryogenic state, and death.

Each agent contains 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 entities, resources, or agent frameworks. *Proxy agents* extend the scope of the agent-to-agent protocol beyond a particular domain. The *Domain Manager*[1] controls the entry and exit of agents in a domain according to policies set by the domain administrator. The *Matchmaker*[2] can access information about the location of the generic agent instance for any agent that has advertised its services. The *Transport Agent*[3] facilitates *teleportation* (transfer of an entire agent from one agent domain to another) and *telesthesia* (transfer of the agent's extension to another host).

Messages are exchanged between agents in the context of *conversations*. *Verbs* name the type of illocutionary act (e.g., *request, promise*) represented by a message. Unlike most agent communication architectures, KAoS explicitly takes into account not only the individual message, but also the various sequences of messages in which it may occur. Shared knowledge about message sequencing conventions *(conversation policies)* enables agents to coordinate 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.

Our experience with the current KAoS architecture has shown it to be a powerful and flexible basis for diverse types of agent-oriented systems. The strength of the architecture derives from several sources:

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

[1] Also called the *CIA* (Central Intelligence Agent)

[2] Also called the *KGB Agent* (KAoS Generic Broker).

[3] Also called the *KOA Agent,* after the popular US campground chain that provides service hookups for the mobile homes of travelers.

Applications. The Boeing Company is exploring the use of portable airplane maintenance aids (PMA) and networked data access capabilities (Boeing OnLine Data—BOLD) to provide training and support to customers (Bradshaw, Richards, Fairweather, Buchanan, Guay, Madigan, & Boy, 1993; Guay, 1995. A new version of KAoS is being incorporated into one such prototype of an intelligent performance support system. The system, named *Gaudi,*[4] is designed around the processes, activities, and resources of the work environment. It is intended to directly and actively support necessary tasks, adapting the available information and services to the requirements of the user and situation.

KAoS agent technology is also a key component of a joint research and development collaboration sponsored by the National Aeronautics and Space Administration (NASA) (Gawdiak & Bradshaw, 1997). The intent of the

[4] 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 their designers and of providing suitable foundations 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).

collaboration is to develop interface standards and intelligent network technologies for a secure aviation extranet that can be commercially implemented and used by the aviation industry, the Federal Aviation Administration (FAA), and NASA. A similar use of KAoS agents will help support large-scale collaboration between medical staff at the Fred Hutchinson Cancer Research Center and primary-care physicians worldwide (Bradshaw, Chapman, Sullivan, Boose, Almond, Madigan, Zarley, Gavrin, Nims, et al., 1993; Chin, 1997).

While using a broad brush for technological details, we now describe some of the roles we are exploring for agent technology in these and other applications.

3. Agents for Managing Loosely-Coupled Information Sources

A major challenge of building and maintaining dynamic loosely-coupled distributed systems is finding the required information sources and computing services on-the-fly. We have been using agent technology in two different ways to address this problem.

Matchmaker 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.[5] In a distributed object environment, the agent domain could be implemented with a single object repository manipulated by a Matchmaker agent. In a CORBA environment, the OMG *trader facility* could be used in support of the Matchmaker function.

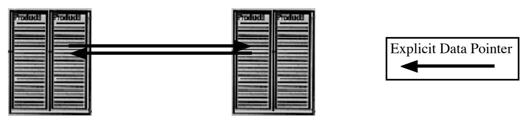
An agent *advertises* a service to the Matchmaker if it is prepared to respond to messages from other agents wishing to use that service. An advertise message may specify whether there are any restrictions on which agents may have access to and visibility of the advertised service. For example, certain 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 inference in matching potential service provider attributes to the requirements of the requesting agent.

Independent (external) hyperlinking. The SGML standard for document markup, which is in wide use in the aviation industry, was originally developed to solve problems of interchange between users of complex structured documents (Goldfarb, 1990). However as its usage has grown, the notation has been increasingly applied to other problems such as hyperlinking that are less well-suited to this general approach. The popular HTML format, on which Web-documents are based, is a simplified derivation of SGML. Figure 2 illustrates how markup-based linking works.

An increasing number of researchers are recognizing that linking information has special characteristics which place it outside the realm of document content (Tucker, 1994). They advocate the use of *independent links,* i.e., linking information that is encoded and stored separately from the content. Representing links independently rather than embedded as descriptive markup is not necessarily inconsistent with the use of SGML or HTML. As DeRose observes "SGML is distinct from descriptive markup, and descriptive markup is distinct from what we need to do to maintain a database of text" (DeRose & Raymond, 1993). He concludes that "[embedded] link storage as a sole linking mechanism is inadequate for managing large evolving multi-user hyperdocuments" (DeRose & Durand, 1994). Among the advantages of independent linking are (Davis, Hall, Heath, Hill, & Wilkins, 1992; Malcolm, Poltrock, & Schuler, 1991):

- New or updated links can be associated with read-only data, such as data available on CD-ROM.
- Different link sets can be added at runtime, or dynamically activated or deactivated for different organizations, users, and situations. This allows for extensive end-user customization of links, and minimizes the problem of too many irrelevant links for a given context (Boy, 1992; Boy, 1991.
- Through simple extensions, virtually any application can take part in two-way linking relationships (Davis, Knight, & Hall, 1994. Open hyperlinking is becoming an increasingly popular approach (Davis, Lewis, & Rizk, 1996; Østerbye & Wiil, 1996).
- Dynamic query links, whose linkends and anchors are calculated at runtime, can be used (DeRose & Durand, 1994).
- The general design is compatible with the emerging new generation of linking standards typified by the *ilink* approach in the HyTime extensions to SGML (DeRose & Durand, 1994).
- Configuration control, management of linkend location changes, and multi-user access can be off loaded from the document architecture onto the link server and its underlying database (Böhm & Aberer, 1994; Böhm, Müller, & Neuhold, 1994.
- Adaptive hyperlink architectures, such as those developed at NASA-Ames on the HyperMan project, can be more easily supported (Mathé, 1993; Mathé & Chen, 1994). (explained in more detail below).

[5] The Matchmaker performs a similar rôle to a KQML "agent server" facilitator which uses the advertise and recommend performatives (Finin, Labrou, & Mayfield, 1996). See (Kuokka & Harada, 1995) for a discussion of KQML and matchmaking; and (Decker, Williamson, & Sycara, 1996) for a comparison of matchmaking and brokering approaches.



• Links are stored as embedded hidden "markup" of the document

• Standard kind of linking for SGML and HTML (WWW) documents

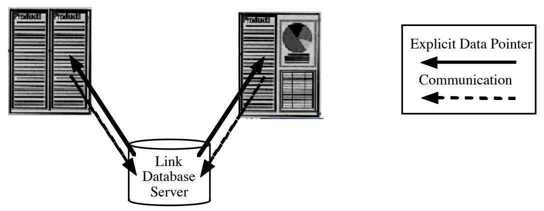
• Because of the use of markup, new links cannot be added without altering the source data

• HTML documents can sometimes compute rather than explicitly store the link destination, but the mechanism still relies on embedded markup

• Links are inherently one-way, and communication with non-SGML/HTML applications is typically in limited master-slave, viewer-only mode

Figure 2. Markup-based linking.

Figure 3 illustrates how independent linking works. Because Boeing has already invested heavily in developing robust and efficient approaches to linking within SGML documents, our goal has not been to completely replace the current embedded scheme with independent links. The two approaches can be straightforwardly combined and used simultaneously in the context where each makes sense.[6]



• Links are stored in one or more separate databases; link data points to places in the documents, but document data is "unaware" of links

• Standard kind of linking in modern hypermedia systems

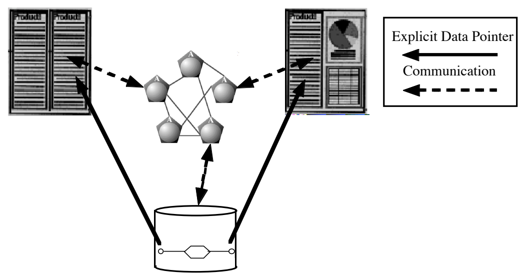
• Because links are stored externally, they can be added or modified without altering the source data

• Links are inherently two-way, unless otherwise specified

Figure 3. Independent linking.

Building on the foundation of independent linking, we have developed an agent-assisted approach (see figure 4). Unlike the typical "launch-and-forget" interaction between linked applications, each active application or software component is assigned one or more agents to be aware both of what is going on in the application and what is going on in the rest of the agent world. Thus, once links are established and traversed, back-channels of communication can be used to keep all active applications and documents "in synch" with the current context.

[6] It is important to note that many of the benefits of external linking can also be achieved by generating and interpreting SGML or HTML documents dynamically rather than storing them as static entities.



• Each active application or software component is assigned one or more agents to be aware both of what is going on in the application and what is going on with the rest of the agents

• Interapplication communication is peer-to-peer rather than master-slave; persistence of agents assures richer ongoing communication than what is possible in more typical "launch-and-forget" modes of interaction

Figure 4. An agent-assisted approach to independent linking.

4. Agents that Provide a Unifying Framework for Distributed Heterogeneous Components

Since airplanes and airports will last for several decades, and information systems become out-of-date on a much more rapid schedule, we need to be concerned about whether any software architecture has a possibility of outliving its designers and of providing a suitable foundation for unanticipated additions of significant new features. Current consensus on these issues is that the use of "objects" or "components" is a necessary but not sufficient enabler of reusability. Rather, it seems that the most robust unit of reusability is a "framework" (Grimes & Potel, 1995). We have prototyped various versions of a three-schema framework (Ford, Bradshaw, Adams-Webber, & Agnew, 1993), with agents providing dynamic coupling and interoperability between components using standard interfaces and data formats. The three-schema approach allows each level to be designed with specific purposes in mind: the external schema are optimized for human understanding and communication, the conceptual schema for semantic completeness, and the internal schema for performance. Because of this paper's focus on agents, we will limit our discussion to this topic.

We think of the agents as being sorted into functional layers: presentation services, application services, generic agent services, and data services (figure 5). Ovals contained within the large gray area represent various agents.

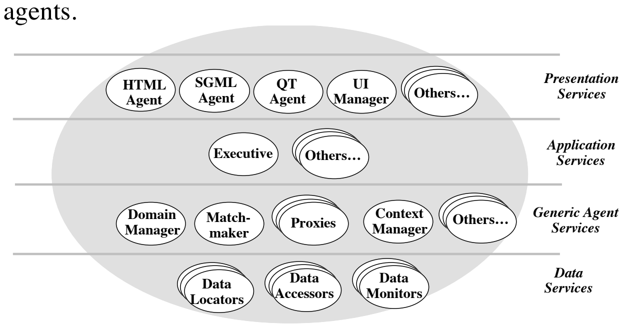


Figure 5. Agent-assisted component integration architecture

Agents providing presentation services are designed to hide the differences between viewers of different data types. From the point of view of the other agents, this means that there is a core viewer service protocol that is shared among all viewers. A viewer of a new kind of data need only implement an agent that converts this minimal viewer service protocol to the specific call formats that the viewer application expects. Once this is done, the new viewer is a full player in the architecture. The additional level of indirection provided by the agents allows components to be incrementally replaced with any other application, written in any programming language, without affecting the rest of the application. For this reason, we say that the architecture supports a "non-stick GUI."

The application services layer currently contains any agents supporting application specific services, in addition to the executive, which works with the context manager to deal with state information.

Generic agent services include the following:

- *Matchmakers (MM),* which were described previously.
- *Domain Managers (DM),* which keep track of a set of properties for some logical or administrative grouping of agents, provide "white pages" services for agents, manage the secure exit and entry of mobile agents into a domain, and otherwise facilitate agent interaction. This capability could make use of such things as underlying OMG Naming and Security Services and the Mobile Agent Facility, if available.
- *Context Managers (CM),* which interact with user-interface and task-specific agents to provide a global perspective on the user and situation that conditions the behavior of agents and tools.
- *Proxies,* which provide various connection and compatibility services to agents residing in different domains.

Data services include *data locators* which encapsulate search and indexing functions, *data accessors* which retrieve data from heterogeneous data sources, and *data monitors* which feed information to clients based on user-configurable "push" policies.

5. Agents for Coordinating Interaction at the Knowledge-Level

Figure 6 is a view of how agents fit into the overall client-server architecture. Specific client applications are built from various components that are integrated via an open presentation layer bus, such as Netscape's LiveConnect. The purpose of the bus is to allow HTML and client-side components (KAoS agents, Java, JavaScript, plug-ins and ActiveX components, ORBs) to share a common object and messaging model, enabling seamless integration of tools, services, and user-interface elements.

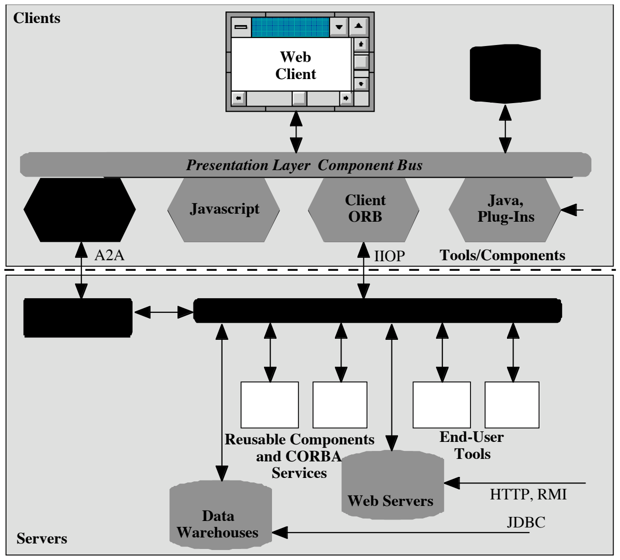


Figure 6. Conceptual view of the overall architecture.

In addition to standard client-server connection protocols such as HTTP, RMI, and JDBC, an IIOP connection to a server-side CORBA component bus will be provided. IIOP enables developers to selectively expose their interfaces, providing a standard way for system components to provide and access required services and data from each other. Interfaces are exposed to the ORB by compiling an interface specification written in IDL (Interface Definition Language). This not only ensures interoperability among our end-user tools and reusable components, but also allows us to take advantage of third-party CORBA services that can be used and customized as needed.

End-user tools and reusable components can be implemented as any combination of local Java applets, plug-ins, ActiveX components, and IIOP-enabled server-side components as desired. We look forward to taking advantage of forthcoming component integration frameworks such as JavaBeans will eventually allow the incorporation of OpenDoc LiveObjects and ActiveX components that can function as full peers to Java applets.

Though the lowest-common-denominator methods provided through the component bus will be adequate to enable interoperability between most system components, some specialized intelligent software modules may require a higher level of communication semantics than can be directly supported by IDL alone.

KAoS agents on the client and on the server can communicate using an agent-to-agent (A2A) protocol that runs on top of standard lower-level protocols such as IIOP and sockets. Generic agent services built on the foundation of existing distributed object services (such as described in section 4) will allow software components the option of using a common agent-to-agent interlingua (an “agent bus”) to communicate and coordinate their actions at the “knowledge-level” rather than relying on more primitive program-to-program protocols. Shared ontologies provide a common vocabulary for collaboration on problems that span different toolsets and information sources. For example, the diagram in Figure 7 shows two specialized components (SA1 and SA2) that are communicating and coordinating their actions by means of their respective mediation agents (MA1 and MA2), which live within the generic agent services domain.

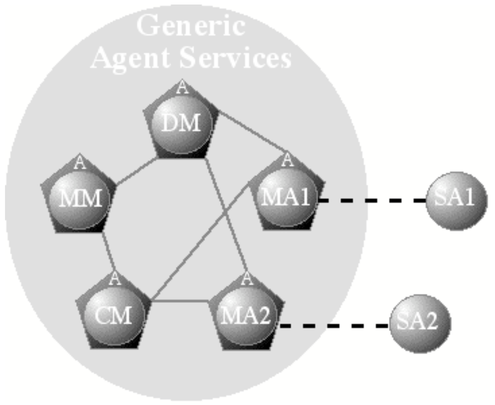


Figure 7. Example of two specialized components interacting through the “agent bus.”

The alternative to mediated communication between two components, is to rely on dedicated “full bridges” between each type of agent. While such an approach is possible for a small, fixed number of agent types, it quickly becomes impractical because the number of bridges increases as (n2 - n)/2 with the number of types of components. In the mediated approach, each agent type provides a “half bridge” to the common agent-to-agent protocol, greatly simplifying the work of the developers

Conclusion

Our experience to date has demonstrated some of the important roles that software agents can play in knowledge management. We expect to learn much more as we continue to apply them in a variety of settings.

Acknowledgements

The work described in this chapter was supported in part by grant R01HS09407 from the Agency on Health Care Policy and Research and by a collaborative research agreement with NASA Ames Research Center.

References

Apple_Computer (1993a). *AppleScript Language Guide.*, Reading, MA: Addison-Wesley.

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

Böhm, K., & Aberer, K. (1994). Storing HyTime documents in an object-oriented database. N. R. Adam, B. K. Bhargava, & Y. Yesha (Ed.), *Proceedings of the Third International Conference on Information and Knowledge Management (CIKM-94),* (pp. 26-33). Gaithersburg, MD, , New York: ACM ,

Böhm, K., Müller, A., & Neuhold, E. (1994). Structured document handling: A case for integrating databases and information retrieval. N. R. Adam, B. K. Bhargava, & Y. Yesha (Ed.), *Proceedings of the Third International Conference on Information and Knowledge Management (CIKM-94),* (pp. 147-154). Gaithersburg, MD, , New York: ACM ,

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

Boy, G. A. (1991). Indexing hypertext documents in context. *Proceedings of the Third ACM Conference on Hypertext,* . San Antonio, TX, , ,

Bradshaw, J. M. (1996a). An introduction to software agents. In J. M. Bradshaw (Ed.), *Software Agents*. (pp. in preparation). Cambridge, MA: AAAI/MIT Press.

Bradshaw, J. M. (1996b). KAoS: An open agent architecture supporting reuse, interoperability, and extensibility. B. R. Gaines & M. Musen (Ed.), *Proceedings of the Tenth Banff Knowledge Acquisition for Knowledge-Based Systems Workshop,* 2 (pp. 48:1-48:20). Banff, Alberta, Canada, , ,

Bradshaw, J. M. (in preparation). How computer-based training learns: What happens after it’s built. In A. J. Cañas, P. Feltovich, & S. Papert (Ed.), *Education and Smart Machines.*

Bradshaw, J. M., Chapman, C. R., Sullivan, K. M., Boose, J., Almond, R. G., Madigan, D., Zarley, D., Gavrin, J., Nims, J., & Bush, N. (1993). KS-3000: An application of DDUCKS to bone-marrow transplant patient support. *Proceedings of the Seventh European Knowledge Acquisition for Knowledge-Based Systems Workshop (EKAW-93) (complement),* (pp. 57-74). Toulouse and Caylus, France, , ,

Bradshaw, J. M., Dutfield, S., Benoit, P., & Woolley, J. D. (1996). KAoS: Toward an industrial-strength generic agent architecture. In J. M. Bradshaw (Ed.), *Software Agents*. (pp. 375-418). Cambridge, MA: AAAI/MIT Press.

Bradshaw, J. M., Ford, K. M., Adams-Webber, J. R., & Boose, J. H. (1993). Beyond the repertory grid: New approaches to constructivist knowledge acquisition tool development. In K. M. Ford & J. M. Bradshaw (Ed.), *Knowledge Acquisition as Modeling*. (pp. 287-333). New York: John Wiley.

Bradshaw, J. M., Holm, P. D., Boose, J. H., Skuce, D., & Lethbridge, T. C. (1992). Sharable ontologies as a basis for communication and collaboration in conceptual modeling. *Proceedings of the Seventh Knowledge Acquisition for Knowledge-Based Systems Workshop,* (pp. 3.1-3.25). Banff, Alberta, Canada, , ,

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

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

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

Chin, T. (1997). Federal grants will help develop web-based decision support tools. *Health Data Network News*, 20, January 20, 1, 8.

Davis, H., Hall, W., Heath, I., Hill, G., & Wilkins, R. (1992). Towards an integrated information environment with open hypermedia systems. *Proceedings of the ACM ECHT Conference,* . Milan, Italy, , ,

Davis, H., Lewis, A., & Rizk, A. (1996). OHP: A draft proposal for a standard Open Hypermedia Protocol. *Proceedings of the Second Workshop on Open Hypermedia Systems, Hypermedia-96,* . Washington, D.C., , New York: ACM Press ,

Davis, H. C., Knight, S., & Hall, W. (1994). Light hypermedia link services: A study of third party application integration. *Proceedings of the 1994 European Conference on Hypermedia Technology (ECHT '94),* (pp. 41-50). Edinburgh, Scotland, , ,

Decker, K., Williamson, M., & Sycara, K. (1996). Matchmaking and brokering. *Proceedings of the Second International Conference on Multiagent Systems (ICMAS 96),* (pp. in press). , , New York: ACM Press ,

DeRose, S. J., & Durand, D. G. (1994). *Making Hypermedia Work: A User's Guide to HyTime.*, Norwell, MA: Kluwer Academic Publishers.

DeRose, S. J., & Raymond, D. (1993). SGML for implementers (Tutorial Course #20). S. Poltrock (Ed.), *Hypertext '93,* . Seattle, WA, , ,

Finin, T., Labrou, Y., & Mayfield, J. (1996). KQML as an agent communication language. In J. M. Bradshaw (Ed.), *Software Agents*. (pp. in preparation). Cambridge, MA: AAAI/MIT Press.

Ford, K. M., Bradshaw, J. M., Adams-Webber, J. R., & Agnew, N. M. (1993). Knowledge acquisition as a constructive modeling activity. In K. M. Ford & J. M. Bradshaw (Ed.), *Knowledge Acquisition as Modeling*. (pp. 9-32). New York: John Wiley.

Franklin, S., & Graesser, A. (1996). Is it an agent or just a program?: A taxonomy for autonomous agents. *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages,* . , , Springer-Verlag ,

Gaines, B. R., & Shaw, M. L. G. (1996). A networked, open architecture knowledge management system. B. R. Gaines & M. Musen (Ed.), *Proceedings of the Tenth Banff Knowledge Acquisition for Knowledge-Based Systems Workshop,* 2 (pp. 45:1-45:22). Banff, Alberta, Canada, , ,

Gawdiak, Y., & Bradshaw, J. (1997). The NASA Aviation Extranet Collaboration. *Presentation to the Aviation Industry Computer-based Training Committee (AICC),* . La Jolla, CA, , ,

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

Gennari, J. H., Stein, A. R., & Musen, M. A. (1996). Reuse for knowledge-based systems and CORBA components. B. R. Gaines & M. Musen (Ed.), *Proceedings of the Tenth Banff Knowledge Acquisition for Knowledge-Based Systems Workshop,* 2 (pp. 46:1-46:16). Banff, Alberta, Canada, , ,

Goldfarb, C. F. (1990). *The SGML Handbook.*, Oxford: Oxford University Press.

Grimes, J., & Potel, M. (1995). Software is headed toward object-oriented components. *IEEE Computer*(August), 24-25.

Gruber, T. R. (1991). The role of common ontology in achieving sharable, reusable knowledge bases. In J. A. Allen, R. Fikes, & E. Sandewall (Ed.), *Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference*. (pp. 601-602). San Mateo, CA: Morgan Kaufmann.

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

Guha, R. V. (1996). *Meta Content Format.* Apple Computer,

Guha, R. V., & Lenat, D. B. (1990). Cyc: A midterm report. *AI Magazine*, Fall, 33-58.

Harrison, C. G., Chess, D. M., & Kershenbaum, A. (1995). *Mobile agents: Are they a good idea?* IBM T. J. Watson Research Center, March 28.

Kremer, R. (1996). Toward a multi-user, programmable web concept mapping "shell" to handle multiple formalisms. B. R. Gaines & M. Musen (Ed.), *Proceedings of the Tenth Banff Knowledge Acquisition for Knowledge-Based Systems Workshop,* 2 (pp. 48:1-48:20). Banff, Alberta, Canada, , ,

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

Malcolm, K. C., Poltrock, S. E., & Schuler, D. (1991). Industrial strength hypermedia: Requirements for a large engineering enterprise. *Proceedings of the Third ACM Conference on Hypertext,* . San Antonio, TX, , , Mathé, N. (1993). Facilitating access to information in large documents with an intelligent hypertext system. *Proceedings of the AIAA computing in Aerospace 9 Conference,* . San Diego, CA, , ,

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

Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T., & Swartout, W. R. (1991). Enabling technology for knowledge sharing. *AI Magazine*, 36-55.

Orfali, R., Harkey, D., & Edwards, J. (1996). *The Essential Distributed Objects Survival Guide.*, New York: John Wiley.

Østerbye, K., & Wiil, U. K. (1996). The flag taxonomy of open hypermedia systems. *Proceedings of the Seventh ACM Conference on Hypertext (Hypertext 96),* (pp. 129-139). Washington, D.C., , New York: ACM Press ,

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

Tucker, H. A. (1994). *Using HyTime for external references*. Hellerup, Denmark: DOCUMENTA ApS, July 25,1994.