MAGE: Multi-Agent Graphical Environment

Leen-Kiat Soh, Hüseyin Sevay, and Costas Tsatsoulis

Department of Electrical Engineering and Computer Science
University of Kansas, Lawrence, KS 66045
{lksoh, hsevay, tsatsoul}@ittc.ukans.edu
tel: (785) 864-7764 fax: (785) 864-7789

Abstract

This paper describes our continuing research effort towards building a graphical development environment for rapidly creating, visualizing, and testing multi-agent software applications. Our system, Multi-Agent Graphical Environment (MAGE), addresses the need for enabling existing programs to be incorporated into agent-based software frameworks with minimal programming and the need for creating new agents and linking them to others. Users can create new agents and convert legacy programs to agents using the graphical user interface in MAGE and a messaging API at the source code level. Since MAGE handles all the agent-related bookkeeping and communication in a manner transparent to the application layer, users need not build an entire agent infrastructure for their applications. The graphical environment in MAGE enables the users to view and edit the configuration of their multi-agent systems. MAGE facilitates debugging by animating the message-passing among agents and allowing the users to execute their systems in a single-stepping mode. At the application layer, MAGE agents use KQML as their communication language. Below the application layer, MAGE employs CORBA to enable each agent to exchange messages with other agents. Currently, we are in the process of deploying MAGE on a web browser besides utilizing it to solve problems in domains such as information retrieval and data mining.

Introduction

As the computing platform of today's software becomes more and more global, the research and development towards building tools for rapid creation and deployment of both single- and multi-agent systems (Sycara 1998) is gaining increasing significance. Such tools are critical because they alleviate the users from having to build an underlying infrastructure for providing the high-level support required in agent-based problem solving. Using these tools, users can create agents that provide new services to the current computing environment; they can create groups of agents that work in collaborative settings, or they can experiment with different agent architectures for their applications.

An agent can be thought of as an autonomous, goal-oriented, and temporally continuous entity (Franklin and Graesser 1996). On the other hand, a multi-agent system (MAS) is a loosely coupled network of agents that work together to solve problems that are beyond their individual capabilities (Durfee et al. 1989). This approach to problem solving is attractive because it allows problem solvers (i.e., agents) to be specialized and localized, and yet be collaborative in solving problems in a distributed manner.

The multi-agent system architectures that exist today have been applied to solve problems in domains such as information processing, information planning, telecommunications, control and monitoring, and design (O'Hare and Jennings 1996; Kuokka and Harada 1996; Sycara et al. 1996; Wilkins and Myers 1996; Lander 1997). The increasing demand for such systems demonstrates that as much of the creation of agent-based systems as possible needs to be automated to save time in development, testing, and deployment.

Building a MAS from scratch is a difficult task, because a developer has to first design and build an environment to support agent-based problem solving before designing any agents in that environment. Our system, Multi-Agent Graphical Environment (MAGE), provides such an environment to agent developers. Since it hides the lower-level implementation details from the users, it enables them to start developing solutions at the agent level.

The objectives of MAGE are threefold. First, MAGE aims to facilitate the integration of software applications that are potentially written in different languages. These applications may be homogeneous or heterogeneous, centralized or distributed. Applications are homogeneous if they are written in the same language on the same operating system platform, and they are heterogeneous otherwise. With this perspective in mind, the goal of MAGE is to provide interoperability, software reuse, and integration of legacy programs. MAGE is written in Java, and it currently generates proxies to enable smooth integration of agents written in C (if compiled with a C++ compiler) and C++. Second, MAGE aims to provide developers with a graphical environment for building a MAS. Its graphical user interface (GUI) allows developers to create an agent, attach an application to that agent, and establish connections with other agents. Meanwhile, the actual registration and deployment of each agent is activated automatically by MAGE as the user is creating or modifying a system. This approach helps developers concentrate more on their domain-specific modeling and analysis. Therefore MAGE can be a tool for rapid prototyping and testing of multi-agent systems. Third, MAGE aims to automate as much of the agent design, testing, and deployment process as possible through its GUI. Developers are able to view the messages as they are being exchanged between a sender and a receiver and a log of the actions performed by each agent. This mode is useful for debugging and analyzing the behavior of the system from different perspectives such as the role played by each agent in solving particular problems and the impact of the communication traffic on the final solution.

In this paper, we present MAGE as a graphical environment that requires minimal coding from developers to build multi-agent systems. Section 2 reviews agent-building tools that are closely related to MAGE. Section 3 describes MAGE in detail. Section 4 discusses the GUI. We conclude with a report on the current status of MAGE and a description of future directions.

Related Work

Current agent-building tools can be categorized into three groups. The first group of tools focuses on mobile agents. For example, in SodaBot (Coen 1994), a framework called the Basic Software Agent (BSA) provides an environment in which the user can build an agent and deploy it at remote sites where BSA has already been installed. Other mobile agent-building tools include Aglets (Lange and Oshima 1998), Mole (Straser et al. 1996), Odyssey (General Magic 1998), and Kafka (Nishigaya 1997).

The second group of tools provides infrastructure support such as agent architectures and high-level agent-oriented languages, and it also provides generic software libraries with support for capabilities such as communication and coordination, but without a concentration on mobility. For example, the Intelligent Agent Factory (*http://www.bitpix.com*), JATLite (*http://java.stanford.edu*), Agentx (Schneiderman 1998), and Voyager (ObjectSpace 1997) are tools that provide software libraries for building agents. LALO (*http://www.crim.ca/sbc/english/lalo*) is an Agent Oriented Programming (Shoham 1993) language. ARCHON (Wittig 1992; Jennings and Cockburn 1996) is a distributed AI programming framework and a general MAS architecture. COOL (Barbuceanu and Fox 1996) is a language for coordinating the activities of autonomous intelligent agents. The multi-agent system by Lejter and Dean (Lejter and Dean 1996) is another example that provides software libraries in addition to user interfaces for tracing and debugging.

The third group involves tools that provide a multi-agent computing environment. For example, the Java-based Agent Framework for Multi-Agent Systems (JAFMAS) provides a generic methodology for developing multi-agent systems based on speech acts (Chauhan 1997). It also provides communication ability, linguistic and coordination support through a number of Java software libraries. AgentBuilder from Reticular Systems, Inc. includes tools for managing the agent-based software development process, analyzing the domain of agent operations, designing and developing networks of communicating agents, defining behaviors of individual agents, and debugging and testing agent software (Reticular Systems 1998).

MAGE belongs to this third category. MAGE is a tool that requires minimal coding from developers. It provides a simple KQML-based send-receive messaging protocol to enable each agent to communicate with other agents. MAGE uses the information that the user enters through its GUI to automatically generate code for proxies that transparently link agents written in languages other than Java[1] to agents written natively in Java over a common agent communication layer. This capability sets MAGE apart from both AgentBuilder and JAFMAS, which support Java only at the time of this writing.

There are two systems that are very similar to MAGE: ZEUS (Collis et al. 1998) and CABLE (Wooldridge and Jennings 1995). ZEUS, developed at the British Telecommunications Laboratory, is an agent-building toolkit. It has a visual component that allows agents to be created using graphical development tools. It offers editors that enable the developer to specify various aspects of an agent application, from the attributes of individual agents and the tasks they perform, to how they will interact with each other. Then the Code Generator in ZEUS converts the agent specifications to Java source code that is ready to be compiled. This source code for a new agent needs to include the class definitions from the ZEUS agent library. The code produced by the Generator tool is created in the form of callback methods, which allows the developer to integrate the agents with application specific code.

CABLE is a system architecture developed by the GRACE Consortium (Wooldridge and Jennings 1995).

[1] Currently there is support for C and C++ programs.

CABLE provides the developer with an Agent Definition Language (ADL) for defining agents and a parser known as the Scribe for compiling agent definitions written in ADL into agent specifications. Agents are developed using ADL and C++. ADL allows users to define their agents at high level, so that they need not worry about the underlying details. Communication among agents takes place in a local area network using CORBA as in MAGE.

There are several significant differences between MAGE and the above two systems. Unlike ZEUS, MAGE neither assumes the application of each of its agents nor requires the specification of any attribute or aspect of an agent application. Unlike CABLE, MAGE does not require developers to code in another language such as ADL to create a MAS; instead, MAGE solicits agent creation through its GUI, requiring minimal coding from developers.

The Architecture of MAGE

Figure 1 shows the architecture of the MAGE system. The user interacts with MAGE through the GUI, which also acts as a system agent that monitors and collects statistics on the various activities of the MAS being designed such as communication traffic and message content. During an interactive session with MAGE, the user can create agents graphically. MAGE then spawns and registers each agent to the system, and this enables that agent to communicate with others in the system through a common agent communication layer, currently implemented with an ORB.

Each agent is an application that performs one or more tasks. Compatibility among heterogeneous applications is possible using a proxy-based encapsulation of the application within the agent. Figure 2 depicts the software layers in a MAGE agent. When an agent is created, the developer is prompted to supply three pieces of information: (1) the name of the agent, (2) the path to the application, and (3) the computer language in which the application was written. Since Java applications will use the underlying MAGE software library to build agents, they are ready to be run. However, if the application is not written in Java, MAGE automatically generates a proxy that links this application to the agent communication layer transparently through a socket-based connection. This setup process is handled automatically by the underlying language-specific KQML messaging API. Currently MAGE uses the CORBA implementation available in Sun JDK version 1.2.

We have so far described the basic architecture of MAGE without going into the implementation details. The reader may refer to *http://www.ittc.ukans.edu/mage* for more information on the system.

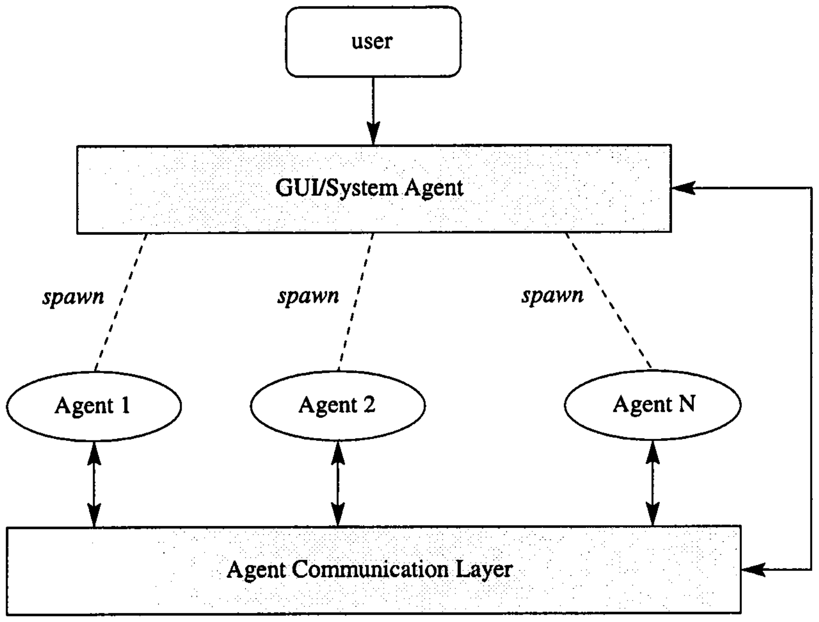


Figure 1: The MAGE architecture

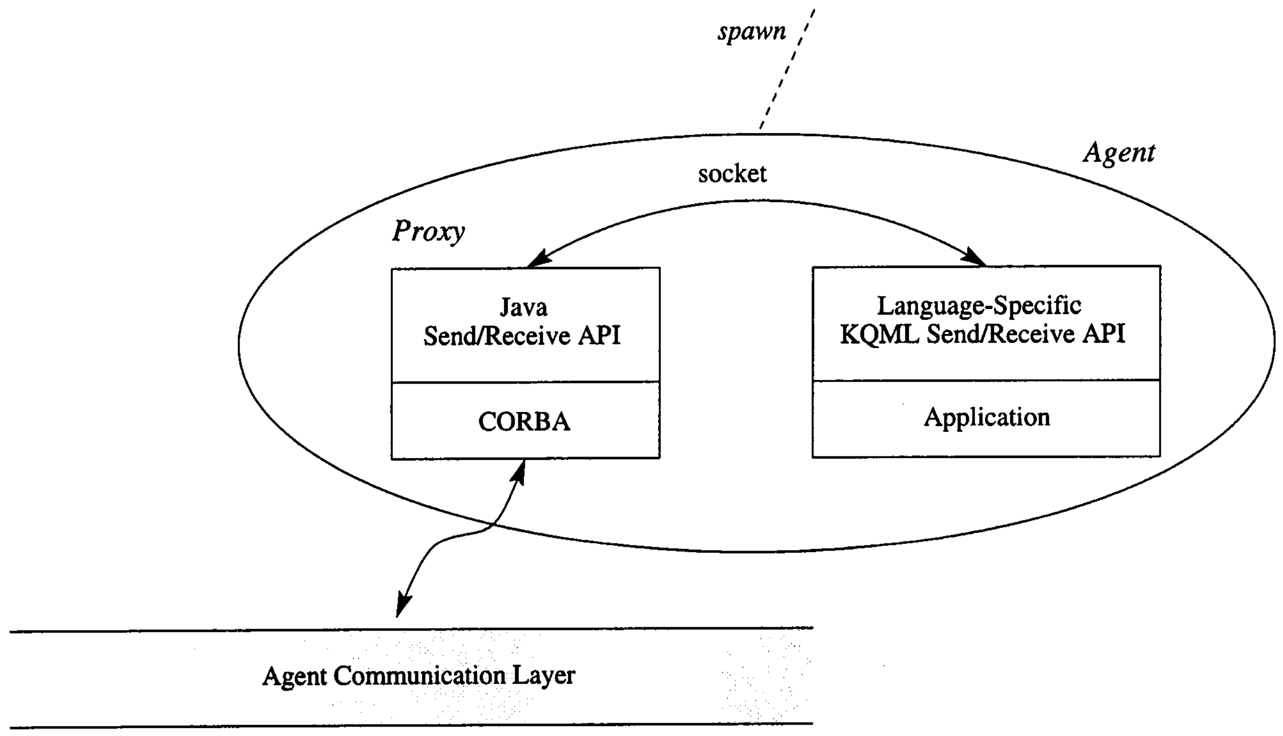


Figure 2: Software design layers of a MAGE agent

The GUI of MAGE

The GUI of MAGE handles all the agent-related bookkeeping in addition to interacting with the user for the specification of agents. As shown in Figure 3, it has three pull-down menus: (1) File, (2) Edit, and (3) Action. The *File* menu allows the developer to perform tasks such as creating a new MAS, opening a previously saved MAS, saving the current MAS working model, printing out the MAS, and exiting from the system. The *Edit* menu allows the developer to undo a process, redo a process, select all agents, toggle-display the message panel, and edit the properties of each agent. The *Action* menu allows the developer to connect two or more agents, disconnect two or more agents, remove an agent, create an agent, register it to the system, call an agent for explicit simulation and testing, and run the entire MAS.

Next we will present some screen captures of GUI to illustrate the primary features of MAGE. Figure 3 shows the main work window of MAGE and the function panel that pops up when the user clicks on the rightmost mouse button.

In the example shown in Figure 3, five agents have been created. The user can create an agent by clicking on the *Create Agent* option in the function panel. Then the user needs to enter the name of the agent, the complete path to the application with which the agent will be associated, and the language in which the application was written. After the creating a MAS, the user can save it in a file that can be retrieved later for modifications.

Figure 4 shows the window that allows the user to browse the file directory and load a previously designed MAS into MAGE.

In MAGE, each agent has a set of properties. The user can store relevant information regarding each agent by invoking the *Properties* option in the main function panel. Figure 5 shows the properties window of MAGE. In this window, the developer can navigate from one agent to another. For each agent, the user can change the name, information, labels, disconnect its links, connect to other agents, etc.

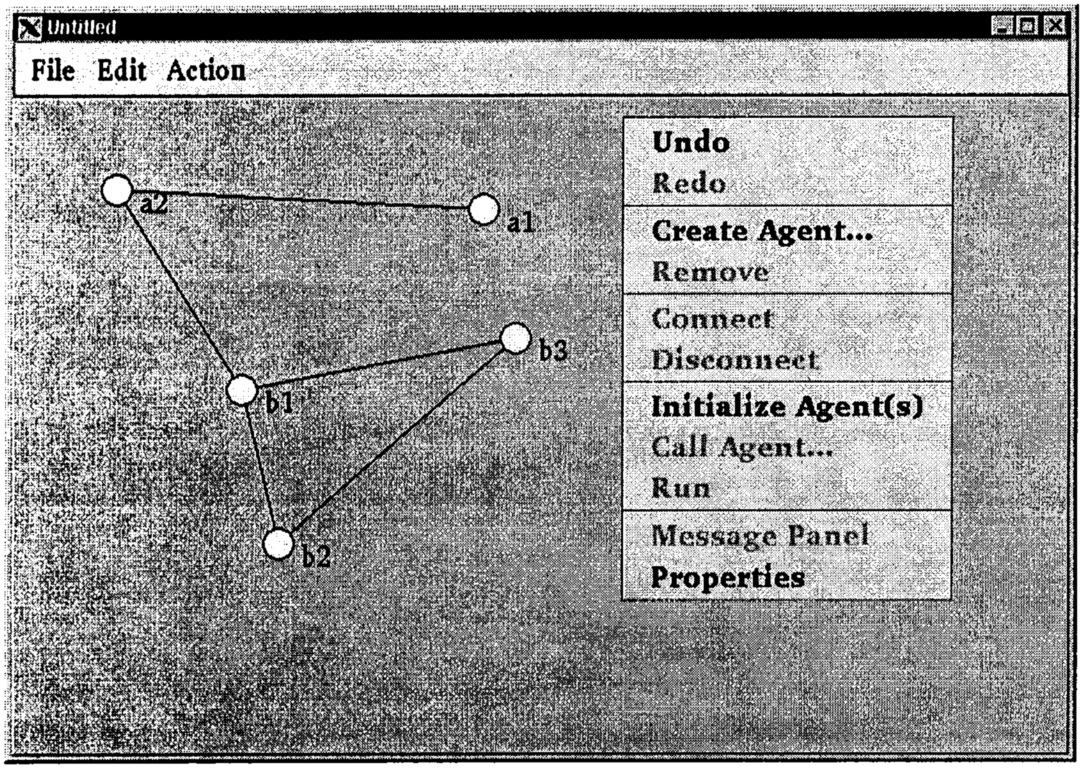


Figure 3: MAGE: work window and function panel

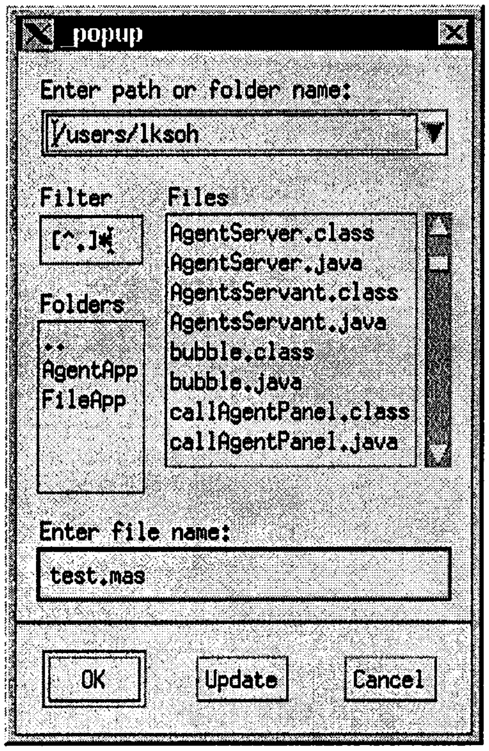
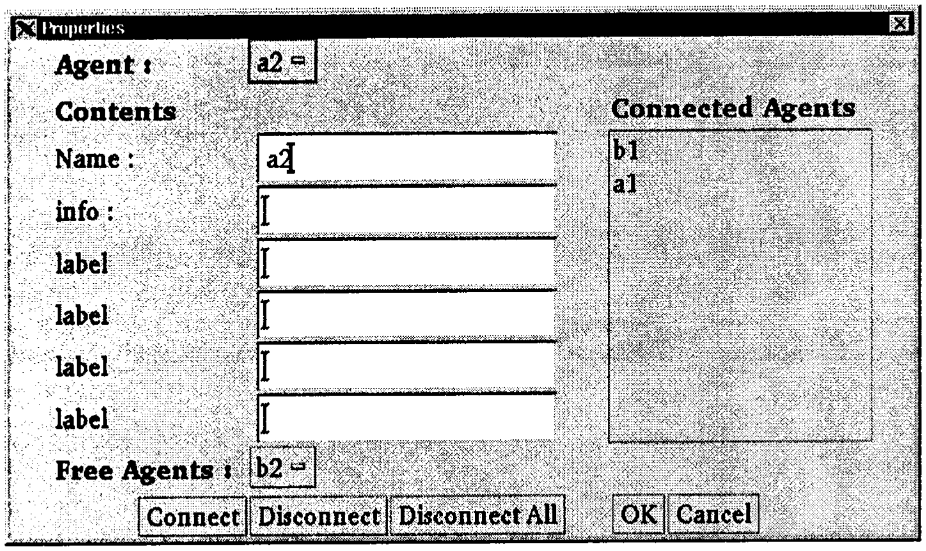


Figure 4: MAGE: file access window


Properties

Agent : a2

Contents

Name : a2

info :

label

label

label

label

Connected Agents

b1

a1

Free Agents : b2

Figure 5: MAGE: agent properties window

After registering the agents to the system, the user can test the communication links among agents by using the *Call Agents* option. This feature allows the developer to explicitly send a message from one agent to the other to verify that the communication links work properly. The actual message-passing is executed when the developer invokes *Run*, which will run all the agents in the system.

Figure 6 shows an example message-passing animation that MAGE offers. For each message sent, a (red) bubble will move from the agent that sends that message towards the agent that the message is intended for.

Current Status

MAGE has entered its second phase of development. Currently, a developer is able to create, connect, register, and activate agents in a MAS. The developer is required, however, to insert the message-passing operations (based on *send* and *receive* primitives) into the legacy programs to identify the receiver and the content of the message being sent. There are several lines of work that are currently taking place. First, the GUI is being refined to have more user-friendly features such as confirmation dialog boxes. Second, proxies are being written to encapsulate Lisp, CLIPS, and other programming languages with socket-based communication support. This task expands the generality of MAGE. Third, we are implementing built-in communication protocols (point-to-point, broadcast, notification, federation and matchmaking). The developer will be able to define the protocol by which a specific agent will communicate with others. For example, if an agent uses point-to-point, then the user will need to define inside the agent code when messages will be sent and to whom. If the agent uses broadcasting, then the messages will be sent to every agent in the MAS. If the agent chooses to be part of a matchmaking architecture (Kuokka and Harada 1996), MAGE will then add all of that agent's functions/methods as advertisements of capabilities to a matchmaker in the system.

Conclusions

We have described Multi-Agent Graphical Environment (MAGE), a tool that allows developers to build multi-agent systems graphically. This tool relieves developers from the implementation details of the underlying infrastructure support needed to build a MAS. MAGE uses a proxy-based encapsulation to link agents written in languages other than Java. This provides for interoperability among disparate applications. Thus, MAGE can be very useful in system integration and software reuse. The GUI facility of MAGE allows developers to build a MAS easily and quickly. Thus, we also see MAGE as a computer-aided tool that enables quick prototyping and testing of distributed software. Finally, MAGE facilitates debugging and analysis via its animation and message log during the execution of the MAS.

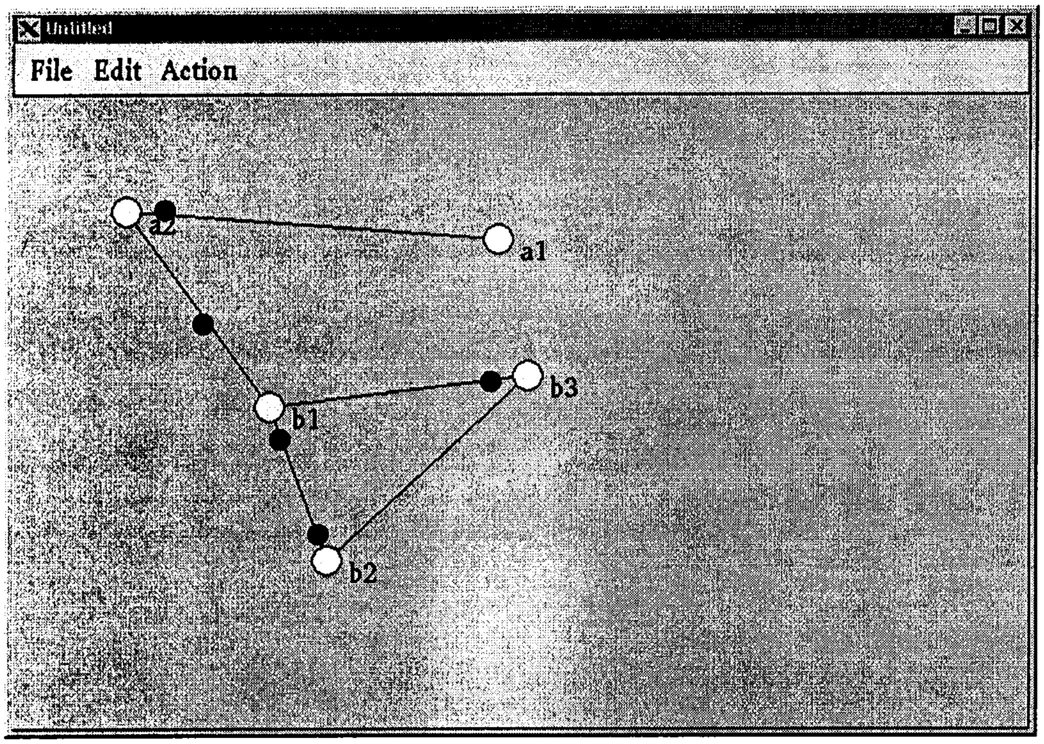


Figure 6: Animation of message-passing in MAGE. A (red) bubble moves from agent *a2* to *b1* when *a2* sends a message to *b1*

Acknowledgments

Part of this work was supported through a grant by the Kansas Technology Enterprise Corporation. The authors wish to thank Danico Lee, Steve Ganje, and Chien-Ming Chen for their help in programming.

References

Barbuceanu, M. and Fox, M. S. 1996. Capturing and Modeling Coordination Knowledge for Multi-Agent Systems. *International Journal on Cooperative Information Systems* 5(2/3):275-314.

Chauhan, D. 1997. JAFMAS: A Java-based Agent Framework for Multiagent Systems Development and Implementation. Master's Thesis, ECECS Dept., Univ. of Cincinnati.

Coen, M. H. 1994. SodaBot: A Software Agent Environment and Construction System. AI Technical Report 1493, AI Lab, MIT.

Collis, J.; Ndumu, D.; Nwana, H.; and Lee, L. 1998. The Zeus Agent Building Tool-Kit, *BT Technology Journal* 16(3):60-68.

Durfee, E. H.; Lesser, V. R.; and Corkill, D. D. 1989. Trends in Cooperative Distributed problem Solving, *IEEE Transactions on Knowledge and Data Engineering* 11(1):63-83.

Franklin, S., and Graesser, A. 1996. Is it An Agent, or Just a Program?: A Taxonomy for Autonomous Agents. In *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*, Springer-Verlag.

General Magic. 1998. Odyssey Information, http://www.genmagic.com/technology/odyssey.html.

Jennings, N. R.; and Cockburn, D. 1996. ARCHON: A Distributed Artificial Intelligence System for Industrial Applications. In *Foundations of Distributed Artificial Intelligence*, G. M. P. O'Hare and N. R. Jennings, eds., John Wiley & Sons.

King, D.; and O'Leary, D. 1996. Intelligent Executive Information Systems. *IEEE Expert* 11(6):30-35.

Kuokka, D.; and Harada, L. 1996. Matchmaking for Information Integration. *Journal of Intelligent Information Systems* 6(2/3):261-279.

Lander, S. E. 1997. Issues in Multiagent Design Systems. *IEEE Expert* 12(2):18-26.

Lange, D. B.; and Oshima, M. 1998. *Programming and Deploying Java Mobile Agents with Aglets*. Addison-Wesley.

Lejter, M., and Dean, T. 1996. A Framework for the Development of Multiagent Architectures. *IEEE Expert* 11(6):47-59.

O'Hare, G., and Jennings, M. eds. 1996. *Foundations of Distributed Artificial Intelligence*, John Wiley & Sons.

ObjectSpace, Inc. 1997. *ObjectSpace Voyager Core Package Technical Overview.* Version 1.0, http://www.objectspace.com/.

Nishigaya, T. 1997. Design of Multi-Agent Programming Libraries for Java. Fujitsu Laboratories Ltd. White paper, http://www.fujitsu.co.jp/hypertext/free/kafka/document.html.

Reticular Systems, Inc. 1998. AgentBuilder, Reticular Systems, Inc. White paper, http://www.agentbuilder.com/Documentation/WhitePaper.

Schneiderman, M. 1998. Agentx: Distributed Computing Technology for the Next Millennium, International Knowledge Systems. White paper, http://iks.com/agentx.htm.

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

Straser, M.; Baumann, J.; and Hohl, F. 1997. Mole—A Java Based Mobile Agent System. In *Special Issues in Object-Oriented Programming,* Mühlhäuser, M., ed., Springer-Verlag.

Sycara, K. 1998. Multiagent Systems. *AI Magazine* 19(2): Summer 1998: 79-92.

Sycara, K.; Pannu, A.; Williamson, M.; Zeng, D.; and Decker, K. 1996. Distributed Intelligent Agents. *IEEE Expert* 11(6):36-46.

Wilkins, D. E.; and Myers, K. L. 1996. Asynchronous Dynamic Replanning in a Multiagent Planning Architecture. In *Advanced Planning Technology: Technological Achievements of the ARPA/Rome Laboratory Planning Initiative*, A. Tate, ed., 267-274.

Wittig, T. ed. 1992. *ARCHON: An Architecture for Multi-Agent Systems*, West Sussex, UK: Ellis Horwood Limited.

Wooldridge, M. J. and Jennings, N. R. 1995. Intelligent Agents: Theory and Practice, *Knowledge Engineering Review* 10(2):115-152.

■