A HANDS-ON LOOK AT JAVA MOBILE AGENTS

What are the benefits and drawbacks of current Java mobile agent systems? The authors installed and evaluated three leading systems available for download from the Web.

JOSEPH KINIRY AND DANIEL ZIMMERMAN
California Institute of Technology

Several companies riding the current wave of popularity of Java want to let you play in the mobile agent sandbox. Each claims that mobile agent technologies are going to change the way we live and work, and each wants to be the company that provides the breakthrough system we all end up using. What do these agent systems actually do and, more importantly, what distinguishes one agent system from another?

To help answer these questions, we downloaded three of the leading commercial systems—General Magic's Odyssey, IBM's Aglets, and ObjectSpace's Voyager—and looked at issues such as ease of installation, feature set, documentation, and cost. We also discuss new capabilities of Java 1.1 that show promise as simple yet powerful means to create mobile agent systems. We conclude with a brief look at the ways in which mobile agents are currently being used and the limitations of today's technologies.

JAVA MOBILE AGENT SYSTEMS

The current explosion of interest in mobile agent systems is due almost entirely to the widespread adoption of Java. As recently as three years ago, only a few mobile agent systems were under development based primarily on research languages like Tcl, Scheme, Oblique, and Rosette. Only one system was commercially available, Telescript from General Magic.

Java has changed all that. During the past year over a dozen Java-based agent systems have been announced for developers to choose

1089-7801/97/$10.00©1997 IEEE

from. The Java Virtual Machine and Java's class loading model, coupled with several of Java 1.1's features—most importantly serialization, remote method invocation, multithreading, and reflection—have made building first-pass mobile agent systems a fairly simple task.

Our examination of the Java-based mobile agent systems now commercially available showed that all share certain characteristics arising from their use of Java as a development platform.

- *They provide an agent server of some kind.* The agent server is the point of contact on a given machine, the sandbox into which agents move and in which agents act.
- *Agents can migrate from server to server in some fashion, carrying their state with them.* Some systems automatically migrate the agents according to an itinerary of some kind, while others let the agents determine their own "destiny."
- *Agents can load their code from a variety of sources.* In general, since all the agent systems use a specialized version of the Java classloader, they can load Java class files from the local filesystem, Web, and ftp servers.
- *All are 100 percent pure Java and use features of JDK 1.1.* This means that they should run on any computer with a 1.1-compatible Java runtime. Unfortunately, because of Java implementation oddities, many of the currently available systems only work properly in specific virtual machines.

Because these qualities are common to all the Java mobile agent systems, we do not include them in the discussions of the individual systems. Instead, the survey highlights only those features that distinguish each system from the others.

Unfortunately, one other common trait was the lack of sophistication of the demo applications included in the system documentation. Although some of these packages did seem to have some impressive functionality, the demo applications overall failed to show this effectively. In addition, documentation was often incomplete; in many cases, the API documentation (automatically generated by Javadoc) was the most informative supporting material included.

Before we turn to our discussion of specific Java mobile agents, a word on the current confusion about terminology in the agent domain is warranted. It seems everyone has a different definition of "agent." (For a comprehensive list of the systems currently available, see the Current Commercial Agent Systems and Current Research-Based Agent Systems sidebars.) It's used currently to refer to everything from robotic systems to e-mail filters to mobile code. The term agent itself is only the beginning of the problem. Should you use *mobile* agents or *stationary* agents? What's the difference between an *intelligent* agent and an *autonomous* agent? Do agents *cooperate* or *collaborate*? Digging through the jargon is something of a challenge. To locate our topic in the general agent domain, think of the software agent space as a diamond with a property at each of the four corners: *stationary* opposite *mobile* and *cooperating* opposite *isolated*. In subsequent sections we will address the issue of when and why you might want to use a mobile agent system versus others in the software agent space thus conceptualized.

CURRENT COMMERCIAL ENVIRONMENT

Most of the dozen or so mobile agent systems currently under development are in the early stages. Several companies have made software available for evaluation on the Web with the implicit hope that developers will provide them with both feedback and sufficient industry buzz to spark interest in the product when it hits the market. Of the many systems we looked at, we selected three relatively mature Java-based mobile agent systems that can now be downloaded from the Web for evaluation: General Magic's Odyssey, IBM's Aglets, and ObjectSpace's Voyager.

Mitsubishi's Concordia is another apparently interesting system scheduled for release in October 1997. Although we were not able to obtain a copy of Concordia for hands-on evaluation, we did have the opportunity to talk in depth to the engineers developing the system. You can read our report based on these discussions in *ICOnline*.*

General Magic's Odyssey

General Magic's Telescript, a proprietary language for creating mobile agents, has been the industry's most widely adopted agent system for several years. Indeed, General Magic first coined the term mobile agents, and in a controversial move, applied for a patent on the concept in 1993. In 1997 they received US patent 5,603,031.

With the advent of Java and its cross-platform promises, General Magic began developing Odyssey,* a new agent system implemented solely in Java that incorporates some of the concepts previously developed for Telescript. The Odyssey development team has about a dozen members, including the recent addition of Danny Lange, formerly a member of IBM's Aglets project. So far, there have been two public beta releases of Odyssey—we evaluated beta 2, which was released in early June.

Odyssey is distributed as a Unix- or Windows-friendly archive, with a self-extracting installer available for the Windows platform. It requires JDK 1.1 or higher due to its extensive use of Java RMI. General Magic also provides extra software to support CORBA's Internet Inter-ORB Protocol (IIOP) and Microsoft's Distributed Common Object Model (DCOM) for agent transport, making Odyssey the only system we examined that currently contains explicit support for multiple agent transport mechanisms.

The archive decompresses into an Odyssey directory hierarchy, including API documentation (in HTML), text files which briefly document how to set up the environment for

CURRENT COMMERCIAL AGENT SYSTEMS

Note that "*" next to an entry means source code is available for download; "†" means you can download software .

Agentsoft†
http://www.agentsoft.com

AgentSoft's products, *LiveAgent* and *SearchAgent*, assist in browsing and searching the World Wide Web.

Autonomy†
http://www.agentware.com/index.htm

Agentware i3 products use neural networks to customize information retrieval and/or content delivery to individual user or enterprise requirements.

Crystaliz
http://www.crystaliz.com/logicware/logicware.html

LogicWare is a programmable server based on an open architecture and designed to enhance Web clients and servers, with emphasis on support for interactive, collaborative Web applications.

Firefly Network
http://www.firefly.com and http://www.firefly.net

Firefly's tools allow Web sites to present personalized content to individuals, and enable users to find others who share their interests. Products include a profile management tool (*Passport Office*) and collaborative filtering technologies (*Community Navigator*, *Catalog Navigator*).

FTP Software
http://www.ftp.com/product/whitepapers/4agent.htm,
http://www.ftp.com/product/whitepapers/vip_wp.htm

The *Virtual IP Network Agent Applications* are agents that can be deployed in networks to extend traditional network management functions to IP-connected devices. The agents enable management, monitoring, and troubleshooting of end-user systems and network devices in a bi-directional manner from a single location.

General Magic *†
http://www.genmagic.com/agents

Odyssey is General Magic's initial implementation of mobile agents in 100% pure Java. The Odyssey class libraries enable developers to create their own mobile agent applications. The current release of Odyssey supports remote access of CORBA objects and of relational databases via JDBC.

IBM *†
http://www.trl.ibm.co.jp/aglets,

The Aglet Workbench is a 100% Java mobile agent technology. The Java Aglet API (J-AAPI) contains methods for initializing an aglet, message handling, and dispatching, retracting, deactivating/activating, cloning, and disposing of the aglet. Application developers can write platform-independent aglets and expect them to run on any host that supports J-AAPI.

KYMA Software
http://www.info.unicaen.fr/~serge/kyma.html

KYMA Atlantis is a development platform for intelligent agents and personal assistants. Using a combination of rule-based inferencing, fuzzy logic. and neural network technology the product solves problems that require multiple reasoning methods.

Microelectronics & Computer Technology Corporation (MCC)
http://www.info.unicaen.fr/~serge/kyma.html

Carnot enables the development and use of distributed, knowledge-based, communicating agents. The agents interact by using Carnot's actor-based Extensible Services Switch (ESS) to manage their communications through TCP/IP and OSI.

Microsoft *
http://www.microsoft.com/intdev/agent

Microsoft Agent is a set of programmable software services that supports the presentation of interactive animated characters within the Microsoft Windows interface.

Mitsubishi
http://www.meitca.com/HSL/Projects/Concordia

Concordia is a full-featured framework for development and management of network-efficient mobile agent applications for accessing information on any device supporting Java. Concordia-enabled applications perform data access, decision execution, and notification.

ObjectSpace *
http://www.objectspace.com/Voyager

Voyager is a 100% Java agent development platform that allows Java programmers to create network applications using both traditional and agent-enhanced distributed programming techniques. Voyager was designed to use regular Java message syntax to construct remote objects, send them messages, and move them between applications.

Oracle
http://www.oracle.com/products/networking/mobile_agents/html

Oracle Mobile Agents is a misleadingly named mobile middleware product—an asynchronous, secure, store and forward messaging system for wireless, dialup, and local area networks.

Odyssey and how to run the demo applications, and source code for many of the Odyssey classes. A PostScript document, "Introduction to the Odyssey API," is also provided in the archive, and the top-level HTML document in the package contains a pointer to the Odyssey FAQ (available online from General Magic's Web site). Finally, there are pre-compiled class files for the entire Odyssey system, which makes it easy to quickly try the three demonstration applications.

Setting up the environment to use Odyssey requires a single addition to the CLASSPATH variable and, if IIOP is being used, the installation of Visigenic's VisiBroker for Java (download instructions are provided with the IIOP archive). The DCOM protocol can only be used on Win32 systems. In addition, an "rmiregistry" process (provided by Sun as part of the JDK) must be started on every machine on which an Odyssey agent server will run. Odyssey will attempt to start an rmiregistry process if one is not running when it is needed but, as of the beta 2 release, this functionality does not work all of the time. General Magic lists this in the Odyssey FAQ as a known problem.

One unique feature of the Odyssey system is its audit trail mechanism. Using a class provided in the Odyssey distribution, a given application can direct status and debugging output to either standard output, a window, or a file on the host on which it is running. This facility is a significant advantage for programmers debugging their agents. Especially when working with a system involving complex interactions between agents, it is helpful to have debugging output from each agent delineated in a straightforward way or even archived on disk, rather than merely dumped to a console window (as in all other systems we examined).

However, there are some quirks associated with the current implementation of the audit trail mechanism. For instance, if an agent sets up an audit trail directed to standard output, the agent's output is displayed on standard out of the virtual machine in which it is running at any given moment. However, if an agent sets up an audit trail directed to a window, no debugging output is displayed while the agent is not in the virtual machine at which the audit trail was started. Since Java has no built-in support for remote displays, this is understandable behavior; but a better solution would be to log the audit trail until the agent returns to its "home" virtual machine and has access to the windowing system again. A more serious problem with the audit trail mechanism in beta 2, which we are sure General Magic will address in a future release, is the fact that an agent whose audit trail is directed to a file cannot migrate at all, with attempted migrations causing exceptions due to the nonserializable nature of files.

Odyssey also provides a facility for agent collaboration, allowing agents to set up and disband "meetings" on particular machines of the network. However, there is no documentation on this facility (other than APIs) in the beta 2 release, and since collaborative agents were not used in any of General Magic's demo applications, we didn't get a chance to see an Odyssey meeting in action. Another facility provided (undocumented in the beta 2 package but used in one of the sample applications) is a mechanism for exchanging references to what General Magic calls "published" objects. An agent can publish arbitrary objects, making global references to them available to all other agents in the same Odyssey system. While this seems like it could be a useful capability, the lack of documentation makes taking advantage of it difficult.

Overall, Odyssey is a fairly generic system that implements the basic functionality necessary for creating mobile agents and little more. The lack of comprehensive documentation and the inclusion of only three example applications, however, make it a challenging system to learn. Odyssey is currently free for noncommercial use, with a full license agreement available online. General Magic has not yet determined whether Odyssey will become a commercial product—they have specifically prohibited commercial deployment of Odyssey-based applications, although they provide a contact address for potential licensees of the technology.

IBM's Aglets

Of the agent systems we reviewed, IBM's Aglets Workbench* has definitely received the most press coverage. There are certainly other legitimate contenders in the space, but we would guess that aglets (a pun on "agent" plus "applet") are so popular for three reasons:

- the package is easy to install,
- the example agent applications are "flashy" (they have GUIs), and
- the brand name is IBM. Regardless, the Aglet system shows promise as a straightforward mobile agent technology that fits nicely into the Java world.

Perhaps the best feature of aglets is that they make the world of agents accessible to the Java novice. Package installation is fairly simple: Unpack the standard zip file for Wintel and compressed archive for Unix, make an addition to the CLASSPATH and PATH, and an AGLET_HOME environmental variable. There are other optional environmental variables (for example, AGLET_EXPORT_PATH), which can be set for additional functionality.

The example aglets are simple, yet functional enough to show off the important aspects of the system. Most importantly for the novice, every component of the system comes with an easy-to-use GUI. This means that you don't have to see a single line of code to implement mobile agents on your system: Given 15 minutes, you can download this package and be using aglets.

Like the other agent systems we examined, the Aglets Workbench demands Java 1.1. Unlike the other systems, IBM does provide some recourse for systems that do not yet

CURRENT RESEARCH-BASED AGENT SYSTEMS

For a comprehensive listing of research-based agent systems, see The Agent Society home page at www.agents.org.

Carleton University's Perpetuum Mobile Procura Project
http://www.sce.carleton.ca/netmanage/perpetum.shtml

Investigation of mobile code for managing networks. The goal is a plug-and-play network, i.e., one that is self-configuring and immune to faults through self-recovery.

Caltech's Infospheres Project
http://www.infospheres.caltech.edu

Research into the theory and implementation of compositional systems (systems built from interacting components) that support peer-to-peer communication among persistent multithreaded distributed objects.

Carnegie-Mellon's Oz Project
http://www.cs.cmu.edu/afs/cs.cmu.edu/project/oz/web/oz.html

Development of technologies (including AI) to help artists create interactive dramas, involving the building of believable agents in dramatically interesting microworlds.

Cornell and University of Tromsø's Tacoma
http://www.cs.uit.no/DOS/Tacoma

Focus on operating system support for agents as well as how agents can be used to solve problems traditionally addressed by operating systems.

CRNI's Knowbot Information Service
http://info.cnri.reston.va.us/kis.html

Search of various Internet directory services to find a given street address, email address, and phone number.

Dartmouth's Agent Tcl
http://www.cs.dartmouth.edu/~agent

General agent framework written in Tcl to support applications that require the retrieval, organization, and presentation of distributed information in arbitrary networks.

European Computer-Industry Research Centre's Mobile Service Agents
http://www.ecrc.de/research/dc/msa

Research of mobile service agents to provide user access to information and tools regardless of system or location.

German Research Center for AI's InteRRaP Agent Architecture
http://www.dfki.uni-sb.de/~jpm/interrap.html

Layered architecture designed to model autonomously interacting agents.

IBM T.J. Watson Research Center's Itinerant Agents
http://www.research.ibm.com/massdist

Itinerant Agents "roam among a set of networked servers until they accomplish a given task." An Agent Meeting Point on each machine (1) accepts, authenticates, and executes incoming agents and (2) provides knowledge-based routing of services requests to the appropriate third-party agents.

Institut de Grenoble's Mobile Assistant Programming for Efficient Information Access
http://fidji.imag.fr/map.html

Programs in the interpreted language Scheme that can move between nodes, create clones, perform useful operations on WWW documents, and report results.

MIT Media Lab's Software Agents Group
http://agents.www.media.mit.edu/groups/agents

Development of techniques and algorithms in agent decision-making, learning, and collaboration.

Open Group's Mobile Objects and Agents and WAIBA Initiatives
http://www.opengroup.org/RI/java/moa

Generic framework for developing mobile agents in Java.

support Java 1.1. Aglets will also work on Java 1.0.2 with the RMI preBeta 2 release, and a supplementary package called ManualSerialization can be downloaded to provide an interim solution for systems that don't have RMI.

To use the Aglet system, you must first run an aglet server. The primary server provided with the Aglets Workbench is called Tahiti. Simply typing agletsd -port 9000 starts the Tahiti aglet server on port 9000 of your computer.

The first time the server is run, a registration panel pops up to let the user identify him/herself to the agent system. This information is used to tag the agents created by this server with their owner's identity. After registration, the primary agent workbench window appears, the user a multi-line display showing the current agents running in the server's address space, a set of buttons to assist in agent management, and a set of menus that complement the buttons' functionality and provide debugging support. With this interface users can create, deploy, retract, and kill aglets.

Additionally, the user can ask the server to display information on memory usage, thread state, and log messages. The user interface even provides for some first-pass preferences: general preferences support customizing the Tahiti

CURRENT RESEARCH-BASED AGENT SYSTEMS (continued)

Purdue's Coast
http://www.cs.purdue.edu/homes/mcrosbie/research/autonomous.html

An autonomous agents' approach to intrusion detection. The analogy is the human immune system—many small bodies (or agents) cooperate to perform a large, complex task.

Stanford University's Knowledge Sharing and KIF Projects
http://www-ksl.stanford.edu/knowledge-sharing;
http://cdr.stanford.edu/ABE

Investigations related to the ARPA knowledge-sharing effort and agent-based software applied in the engineering domain.

Swedish Institute of Computer Science's Agents Kernel Language (AKL)
http://www.sics.se/ps/agents.html

A language in which computation is performed by agents interacting through stores of constraints.

Technical University of Berlin's Magna Project
http://www.fokus.gmd.de/oks/research/magna_e.html

Part of the development of an agent-based electronic service market.

University of Kaiserslautern's Agents for Remote Action (ARA)
http://www.uni-kl.de/AG-Nehmer/Ara

Development of a portable and secure infrastructure for the execution of mobile agents in heterogeneous networks.

University of Maryland's AgentWeb (KQML and Tcl/Tk KQML Agents)
http://retriever.cs.umbc.edu/agents

Part of the ARPA knowledge-sharing effort to develop methodology and software for sharing and reuse of knowledge relevant for building agent-based systems.

University of Massachusetts' Multi-Agent Systems Laboratory
http://dis.cs.umass.edu/dis.html

Research into the development and analysis of sophisticated AI problem-solving and control architectures for both single-agent and multiple-agent systems.

University of Michigan's Distributed Intelligent Agents Group
http://ai.eecs.umich.edu/diag/homepage.html

Research into intelligent and multi-agent systems with planning and coordination aspects.

University of Ottawa's Intelligent Agents Group
http://deneb.genie.uottawa.ca/webdata/research/int-agent1.html

Investigation of personal agents for delivering of multimedia documents over heterogeneous networks to heterogeneous devices (including cellular phones, graphical workstations, and laptops).

University of Washington's SoftBots
http://www.cs.washington.edu/research/projects/softbots/www/softbots.html

Research into intelligent software agents that use software tools on a user's behalf on the Internet.

Vrije University's Software Agents
http://arti9.vub.ac.be/www/research.html

Intelligent software agents for flexible presentation of information, navigation through large amounts of distributed information, support for administrative procedures, and negotiation.

Xerox PARC's Dynamics of Computation Area
ftp://anonymous@parcftp.xerox.com/pub/dynamics/dynamics.html

Focus on the relation between local actions and global behaviors of large distributed systems (social and computational).

interface, network preferences help set up HTTP tunneling via Web proxies, and security preferences allow the owner to specify which aglets are to run as trusted, whether or not to show warning messages, and so on. (See the feature article in this issue, "A Security Model for Aglets," pp. 68-77, for more information on the directions IBM is taking with this feature). All in all, Tahiti is a surprisingly functional and simple interface for managing small numbers of aglets.

The Aglets Workbench comes with source code for the demonstration programs and the base aglet classes. This is a reasonable amount of code; if any more were made available, a developer might get bogged down in the details or depend on some undocumented features of the system (or IBM would lose their edge). The Aglet system also comes with a reasonable amount of documentation: installation guide, "quick start" document, release notes, and descriptions of the sample programs. Also available via the site are the list of known bugs and the Aglet's Cookbook.* (A version of the cookbook will be published later this year by Addison-Wesley under the title, *Java Agent API: Programming and Deploying Aglets Using Java*, with Danny Lange and Mitsuru Oshima as authors.)

What we liked best about the Aglet system is its clean design. IBM has gone to great lengths to make the system mirror several models in Java: applets, AWT callbacks, Java Beans, and a publish-subscribe messaging model that is sure to look something like the upcoming Java Messaging Service (JMS).

For example, seasoned applet programmers will appreciate the several ways in which the aglet model reflects the applet model. Applets have destroy, init, start, and stop methods; aglets have methods like clone, deactivate, dispatch, dispose, and run. Likewise, aglets have an AgletContext that is very similar to an AppletContext. Finally, the Aglet base class also provides a set of methods to help support standard agent capabilities. Some of these methods (and their applet twins) include: getAudioData (getAudioClip), getCodeBase (getCodeBase), and the getImage/getIconImage duo (getImage).

The aglet callback model is straightforward. When a primary action is performed on an aglet (like creation, destruction, migration, and retractions), a corresponding callback method is invoked to give the aglet a chance to have its say. For example, assume that some rogue aglet A calls the deactivate method on your electronic wallet aglet B, effectively turning it off for a while. Before your aglet's deactivate method is actually called, its onDeactivate callback will be invoked, giving your aglet the chance to "just say no."

Simple but complete property and messaging mechanisms are also provided as part of the Aglets Workbench. The property system lets an aglet tag itself with arbitrary property-value pairs (these pairs are one-to-one relationships between Strings only—a definite restriction) and query other aglets on their tags. The messaging infrastructure permits aglets to exchange messages containing arbitrary content in a synchronous or asynchronous fashion. Also, since an aglet should not necessarily wish to deal with every message it receives, the Aglet class provides a subscribe model so that the aglet can register its interest in particular message types with the system, optimizing message delivery and agent code.

The Aglets Workbench is promising, especially given IBM's commitment to Java and the Aglet team's involvement in the mobile agent work at the OMG. One noteworthy problem is the lack of sufficient documentation to write a new agent server, so that Tahiti must be used and accepted with its limitations. As this article was going to press we were told that the new alpha 5 release of the AWT does includes a new aglet client and server API for writing your own server, but we were not able to evaluate this capability.

The lead architect for the Aglet system, Danny Lange, has left IBM to join General Magic on the Odyssey project, leading to speculation about whether aglets will continue to grow and evolve under the auspices of IBM research in Japan. Will Odyssey now take off as aglets did?

ObjectSpace's Voyager

ObjectSpace is primarily a traditional object consulting and development firm but in recent months has moved into Java development. ObjectSpace's first product was JGL, the Java Generic Collection Library, a set of collection classes and algorithms that has now been licensed and adopted by most Java IDE vendors. Their second foray into the marketplace is Voyager,* a Java-based mobile agent system that exhibits several unique and innovative features.

The Voyager project started in mid-1996 and now includes six core team members. ObjectSpace's first press release on Voyager dates back to 2 April, 1997, and they shipped their first evaluation copies of the system just a few weeks later. We evaluated the third public release of Voyager, version 1.0 beta 2.1, which became available in early June. The Voyager system is freely available for internal commercial use, although ObjectSpace's licensing agreement prohibits the distribution of various specific types of applications developed using Voyager technology.

Voyager uses 100 percent pure Java and requires Java 1.1 since it uses serialization and reflection extensively. The full Voyager package is delivered as a Unix compressed archive or a Windows self-extracting archive. To install the system, you unpack the archive and set CLASSPATH and PATH environmental variables appropriately. The Voyager core classes are supplied in two compressed Java archive files named voyager.core.jar and voyager.tools.jar. Only the first archive is necessary for running Voyager agents; the second archive contains the tools to build agents. No source code is provided for the core classes of the Voyager package.

A great deal of documentation is available for Voyager via both the distribution and the ObjectSpace Web site. This documentation includes an agent comparison white paper (comparing Voyager to Odyssey and Aglets), an RMI comparison white paper, a 170-page User Guide, a technical overview (essentially the first chapter of the User Guide), and the full API documentation for the package. For a quick executive summary, the technical overview is the most appropriate document. In general, Voyager is the most well-documented mobile agent system available today.

Voyager comes with several dozen examples, most of which are trivial, but well-focused, short pieces of code that highlight a particular aspect of the system. The User Guide walks the reader through all aspects of the system, at each step explaining how to build the example code and run the demonstration in question. This is a good thing, because all of Voyager's example applications are provided only as Java source code—you have to build them. The reason for this decision becomes fairly obvious once you understand the Virtual Object, Voyager's key communication framework and tool to support inter-agent communication and control.

The Virtual Object is a kind of post-processed proxy to a remote object or agent. Other systems that use an RPC-like mechanism, like RMI, require the developer to go through a series of steps to describe first the interface and

then the implementation of an object. Voyager takes any existing Java class (source code or class file) and modifies it with the "vcc" tool, a Virtual Code Compiler, which creates the Virtual Object mirror of the source class.

For example, let's say a Java package provided by a vendor included a class file called Foobar.class. With vcc, you would process the Foobar class and create a VFoobar.class class file. Using this new "virtual Foobar," you could then instantiate, communicate with, and migrate instances of Foobar around a network. In short, the Virtual Object facility is the key to the Voyager system. Once any arbitrary object is processed with vcc, it exhibits some of the properties of an agent: It can be migrated from server to server and accessed remotely by other virtual objects in an RPC-like fashion; and it can have its own life-cycle. Note that such virtual objects do not necessarily have their own threads of control. Unless specifically designed to be otherwise, they are simply passive objects that can be moved and manipulated remotely.

The communication facility in Voyager is very flexible. It provides for asynchronous, synchronous, and future remote method calls. References to remote virtual objects can be passed as parameters to methods and can be serialized. Method calls across address spaces have the same semantics as local method calls; they are fully polymorphic, allowing access to local and remote objects through virtual object proxies using exactly the same syntax. Finally, virtual objects let you potentially treat *any* object as, in some sense, a distributed object: The objects can be created in remote address spaces and migrated from address space to address space; and they can have individual lifecycles.

Voyager's migration mechanism is also innovative. While Voyager provides an agent server (named "voyager," appropriately enough) like all the other Java mobile agent systems, it is not necessary to run such a server on all the nodes in your network. This is because a virtual object can migrate not only between agent servers, but also to the Java runtimes of other arbitrary virtual objects.

Voyager has many innovative features but, like its competitors, some limitations as well. First, the use of virtual objects changes the way most developers build software. A Java file can only be processed with the vcc tool if that file is syntactically correct. More often than not, however, that file will reference other virtual object classes, which in turn reference other virtual object classes. The resulting web of dependencies can be very frustrating, leading the developer to create empty Java classes that will just compile, but not necessarily function, so that the system will build.

Less importantly, we take issue with some of ObjectSpace's terminology. For example, ObjectSpace calls Voyager an agent-enabled ORB (Object Request Broker, a term from the CORBA world). While Voyager does provide at least as much ORB-like functionality as Java RMI, it is not an ORB in the CORBA sense of the term, the core functionality of which is to facilitate inter-object communication by shuttling messages to and from remote objects implemented in a variety of languages and instantiating persistent distributed objects. These features are not provided by the Voyager system as of the 2.1 beta release.

As we were going to press, ObjectSpace released beta 3.0 of the Voyager package, which addresses many of the concerns we had with beta 2.1. Although we did not have time to evaluate beta 3.0, we are told that it includes improvements to the "vcc" tool, as well as adding a distributed persistence system, a JavaBeans-compatible event service, a naming service, and support for multicast communication. In addition, ObjectSpace promises several future improvements to the Voyager system, and if their development pace and success with JGL are an indication of their track record, they will likely make good on their promises in short order. These improvements include support for communication using UDP, integration with CORBA, and a more robust and complete security model.

RELATED TECHNOLOGIES

While we can't classify them as mobile agent systems, several Java technologies show promise in supporting such systems in the near future.

Java Remote Method Invocation (RMI)* is used as the communication framework in many Java mobile agent systems. RMI provides an RPC-like mechanism for Java processes, permitting Java objects to communicate through method calls across address spaces. The primary motivations for using RMI today are that it

- supports multiple transport protocols (TCP and HTTP, with others in the works),
- provides a basic object registry/bootstrap mechanism, and
- moves code, not just data, between address spaces.

JavaSoft is making RMI the crux of its network communication offering for Java, and is planning on providing much more functionality in later releases. If RMI is extended to include an object instantiation mechanism to support persistent objects and, crucially, a more integrated security model, it will lend itself to the development of even more advanced mobile agent systems in Java.

Java Servlets,* Java applications/objects that run within JavaSoft's Java Server* on behalf of a user, fit into the mobile agent model as well. Once Java servers are able to instantiate arbitrary servlets in other servers (several issues, principally relating to security, make this model infeasible at present), simple Web servers serving client request for HTML pages will become mobile agent frameworks. This model would allow agents to run on Web servers and possibly in your Web browser with no need to install software on your

machine or change the way you work. The free Jigsaw Web server* from the World Wide Web Consortium supports the Servlet API.

APPLICATIONS OF MOBILE AGENTS TODAY

In what ways are mobile agents being used, and for what kinds of applications are they a good technical solution?

They have been characterized as "a solution in search of a problem," by leading researcher John Ousterhout, author of the tcl scripting language; and many people in the agents' field would certainly agree. Mobile agent proponents will begrudgingly admit that the problems for which their systems are appropriate can also be solved with normal distributed computing technologies like RPCs and distributed objects. While this is inherently true, are mobile agents nonetheless a more natural and simple model to help to solve old problems?

Some companies are clearly betting this is the case. FTP Software, for example, believes that agent technology is the most promising solution for the automation of many tasks in network configuration and management that today must be done manually. To that end they are integrating a suite of agent products into their Virtual IP network in 1997, including a tool to retrieve asset management data from networked devices and a security scanning agent. Crystaliz,* on the other hand, is designing an open mobile agent framework to help build a particular point solution, an advanced distributed version control system completely integrated with the World Wide Web.

Crystaliz's current product line focuses on collaborative applications over the Web. The company's new effort, A4 (Agents, Agencies, Artifacts, and Appliances), is a general mobile agent framework built to support their particular problem domain—distributed asynchronous collaboration with adaptive systems. The choice to use mobile agent technology was motivated by their belief (certainly not a new model) that objects are documents and documents are objects, and that mobile agents are a promising mechanism for supporting object/document migration, replications, and partitioning issues.

The components in their systems, the objects/documents, are simply agents that have state and can migrate. This fact allows them to deal with issues of naming and load-balancing in a simple fashion. Likewise, since the agents are arranged in partially ordered space (a directed acyclic graph), agents can be manipulated in contexts ("workspaces") with a transactional model. Finally, management and manipulation (search and retrieval) of this information space is facilitated by stationary intelligent agents.

Problems Mobile Agents May Solve

While there has been of yet no clear demonstration that mobile agents can make certain applications easier to develop, or improve reliability and efficiency when used in the proper contexts, some problems are frequently pointed to as being amenable to mobile agent technologies.

One such area is applications that depend on collaboration between large numbers of people who are not collocated. The agents used in such applications can act as "proxies" for the people involved, gathering at one central location to do whatever collaboration is necessary and then returning "home" to display their results at the users' convenience.

The idea of mobile agents collaborating in a central location may also be useful when applied to collaborations where not all the collaborators are trusted. Two competing companies could, for example, send mobile agents to a "neutral" server, where collaboration could take place without the risk of confidential information being stolen from either company's computer system by a hostile agent.

Another area in which mobile agents may be useful is in processing data over low-reliability networks. In such networks, which may include portable and/or wireless computers, the limited network connectivity can be used to transfer agents, rather than data, from place to place. In this fashion, an agent can travel to nodes on the network with sporadic connectivity, process the information on those nodes without the threat of a network disconnection, and then return "home" or go to the next stop on its itinerary when network connectivity is restored.

Limitations of Mobile Agents

There are many tough issues facing mobile agent systems before the applications just described become commercially viable. The most important issues are, in our opinion, knowledge representation and network security. How do agents represent their knowledge and communicate their needs to other agents? The "meeting" and "collaborate" mechanisms available in many of today's systems are extremely limited. In addition, the static interfaces supported by today's agent communication mechanisms limit the flexibility and possibilities for agent interaction. Some of the more promising work in this domain is the Knowledge Query and Manipulation Language (KQML),* and its related efforts, languages, and systems (KIF, Ontolingua, and others) from the Knowledge Sharing Effort Consortium. KQML is both a message format and message-handling protocol designed to support run-time knowledge sharing between agents. Only with the use of technologies like KQML can mobile agent systems hope to reach the complexity level that will be demanded by commercial applications in electronic commerce, collaboration, networking, and other such areas. However, the level of complexity involved in KQML implementations makes the integration of KQML and agent systems problematic.

Commercial agent systems have only recently begun to support fully autonomous multi-hop agents, and therefore little work has been done on issues of security in such systems.

While the Java sandbox and security models have enabled developers to make some progress toward solving the rogue agent problem, i.e., an agent damaging a system or network, very little work has been done to solve the inverse problem, that of an agent being attacked by the host computer. For example, if your shopping agent is traveling the network making purchases for you, carrying with it your credit card number, a "rogue agent server" could steal this sensitive information while your agent is migrating or quiescent.

Additionally, the standard computer security "weakest link problem" is exasperated by mobile agent technology. An obvious implementation for agent systems is inter-organizational migration—an agent from company A migrates to a server at company B. In this scenario the security of the corporate network is only as good as the security afforded by the weakest agent server against the strongest "rogue agent." Even a miniscule security hole in a mobile agent system can turn into a gaping flaw in a corporate network. Since agents represent real people, the standard computer security problems of authentication, trust, and culpability, especially in commercial environments, are serious problems agent systems will need to solve before they will be broadly adopted by the general public.

CONCLUSION

The mobile agent market is booming. There are more choices for developers today than ever before, and the competition is fierce. This can only be a good thing for the companies and users that hope to take advantage of the benefits of agent technology. While several efforts show promise, it is not clear from currently available systems that there will be one "winner." Several companies, including Crystaliz, GMD FOKUS, General Magic, and IBM, are working together under the Object Management Group umbrella to define a Mobile Agent Facility* for CORBA. This work will help to ensure that different agent systems will be able to work together, an important step toward making agent technologies relevant to the day-to-day problems of our business and personal lives.

ACKNOWLEDGMENTS

The authors would like to thank their advisor, K. Mani Chandy, for indulging their desire to write this article, and Mary Baxter for her assistance with editing.

Joseph Kiniry received an MS in computer science from the University of Massachusetts in 1994 and is pursuing a PhD in computer science at the California Institute of Technology, where he works in formal methods, practical models, and interoperability issues for distributed, object-oriented, and multithreaded applications. He is one of the designers and implementers of the Caltech Infospheres Infrastructure, a Java-based agent-capable object request broker and distributed computing platform. Prior to attending Caltech, he was a senior researcher at the Open Software Foundation Research Group. His home page is http://www.cs.caltech.edu/~kiniry/.

Daniel Zimmerman received a BS in engineering and applied science in 1996 from the California Institute of Technology. He is pursuing a PhD in computer science at the California Institute of Technology, where he works in formal methods, practical models, and interoperability issues for distributed, object-oriented, and multithreaded applications. He is one of the designers and implementers of the Caltech Infospheres Infrastructure. His research efforts have earned him a three-year graduate research fellowship from the National Science Foundation. His home page is http://babylon.caltech.edu/ dmz.html.

Readers may contact the authors at Caltech Mailstop 256-80, Pasadena, CA 91125; {kiniry, dmz}@cs.caltech.edu.

URLs FROM THIS ARTICLE

***The Agent Society** • www.agent.org
***Crystaliz, Inc** • www.crystaliz.com
***FTP Software's Virtual IP Network** • www.ftp.com/product/whitepapers/vip_wp.htm and
***Agent Applications** • www.ftp.com/product/whitepapers/4agent.htm.
***General Magic's Odyssey** • www.genmagic.com/agents
***IBM's Aglets** • www.trl.ibm.co.jp/aglets/index.html
***JavaSoft's Java Remote Method Invocation** • chatsubo.javasoft.com/current
***JavaSoft's Java Servlets** • http://jserv.javasoft.com
***Mitsubishi Electric ITA Horizon Systems Laboratory's Concordia** • www.meitca.com/HSL/Projects/Concordia
***Mobile Agent Facility Specification** • www.genmagic.com/agents/MAF
***ObjectSpace's Voyager** • www.objectspace.com/Voyager
***UMBC's KQML Web** • www.cs.umbc.edu/kqml
***World Wide Web Consortium's Jigsaw Web Server** • www.w3.org/Jigsaw