David Wong, Noemi Paciorek, and Dana Moore

Java-based Mobile Agents

To free your agents and yourself to get the best deals online, write them in Java. And hope everyone else does too.

The Internet and the World-Wide Web have become worldwide tools for e-commerce. However, despite their seemingly unlimited network bandwidth, ease of use, and secure means of online transactions, a shift in computing paradigm is still needed to fully exploit these features. Will mobile agents represent that shift, ultimately freeing up the Internet for its millions of netizens? The agent paradigm first needs to overcome several critical obstacles before this question can be answered. Here, we discuss why Java is such an effective implementation language for mobile agents in e-commerce.

The mobile agent concept grows out of three earlier technologies: process migration [5], remote evaluation [7], and mobile objects [3]—all developed to improve on remote procedure calling (RPC) for distributed programming. Early systems supporting process migration allowed an entire address space to be moved from one computer to another. One goal of this mechanism was to reduce network bandwidth (compared to RPC) when multiple RPC calls are needed to execute an application. While process migration allowed an entire process to be transferred to a remote host, this mechanism did not allow an easy way to return data back to the source node without the entire process returning as well (see Figure 1).

Next came remote evaluation programming, allowing one computer to send another computer a request in the form of a program (rather than an entire process address space). The remote computer receiving such a request executes the program referenced in the request within its own local address space and returns the results to the sending computer. Remote evaluation systems improved on

process migration by allowing remote programming to occur without having to transmit the process control data from the source to the destination host.

Despite their help reducing network bandwidth, remote evaluation systems lacked the ability to encapsulate more state information into the executable program at the remote host. Mobile objects (based on formal object-oriented programming techniques) extended remote evaluation by capturing more program behavior within the mobile object. Such objects can migrate from node to node while carrying executable code, data in the form of object-specific properties, and potentially other embedded executable objects. A number of mobile object systems were popular in the 1980s, but the one that could be said to have led most directly to mobile agents was the Emerald system developed at the University of Washington [3].

A number of mobile agent systems have evolved from this evolutionary process [1, 4, 8, 10], most notably the Telescript language and run-time environment from General Magic [9]. But mobile agents have since improved on mobile objects in a number of ways. For example, mobile agents further reduce network traffic for applications processing large quantities of data. Some of the earlier programming paradigms were based on the client/server model as a way to offload work to a remote server. The client/server model takes the view that it is more important to ship the data to the program source, whereas mobile agents give the developer enough flexibility to extend the model by shipping the program to the data source (see Figure 2).

Mobile agents also provide some autonomy, because they themselves can decide dynamically where and when to travel to a particular destination node based on some embedded mobility metadata to perform some required work. Mobile agents improve on all these earlier technologies for distributed programming by providing a way for executable code, program state information, and other data to be transferred to whichever host the agent deems necessary to carry out the actions specified in an application. Mobile agents readily adapt to changes in both the program state and the network environment (such as network partitioning and disconnected hosts) to modify their routing behavior.

Mobile agents also give the user a natural mode of asynchronous interaction. Applications that need access to legacy applications can be packaged within the mobile agent and subsequently shipped to the remote host where the legacy code is located—while the user disconnects from the network as desired. A store-and-forward mechanism is typically employed within the mobile agent framework to support such behavior.

The autonomous and asynchronous nature of mobile agents is also especially effective in protecting mission-critical applications from failure caused by unreliable networks; agents can reroute themselves and carry the codified business logic needed to continue their tasks, even when confronted by network partitioning. This aspect of mobile agent performance makes such agents especially attractive to traveling businesspeople who regularly disconnect their mobile devices from their home office servers.

Other features, such as fault tolerance and security, were available in some of the earlier distributed programming paradigms, although mobile agents make greater use of them.

As a Language for Mobile Agent Development

Java is the language of choice for mobile agent systems. Concordia, Odyssey, and Voyager are all Java-based (see Koblick's "Concordia" in this issue). Multiplatform support and the promise of write-once, run-anywhere operation make Java extremely well suited for mobile agent technology. Furthermore, the ubiquity of the Java virtual machine may someday facilitate dissemination of mobile agents throughout the Internet.

Java has several features not found in any other language that directly support implementation of mobile agents. For example, agent mobility requires facilities that convert an agent and its state into a form suitable for network transmission and, on the receiving end, allow the remote system to reconstruct the agent. Java's object serialization accomplishes this conversion and reconstruction almost transparently.

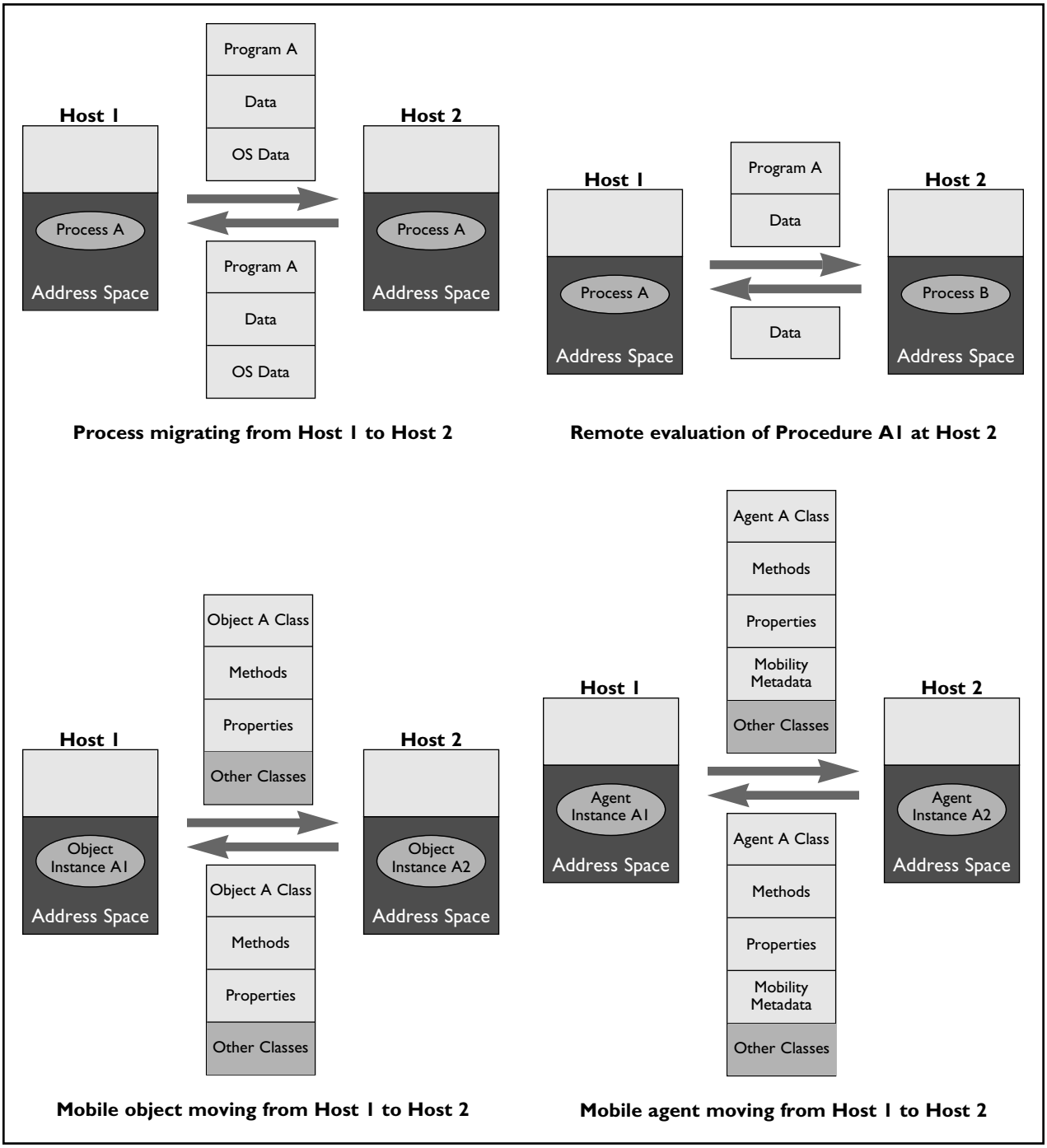

Figure 1. Evolution of distributed computing paradigms

Some Java-based and other mobile agent systems also provide persistent agent state information. Persistence is achieved by serializing an agent's state, writing it to persistent storage, and later retrieving that state and using it to reconstruct the agent. After a mobile agent has been serialized, it can be transmitted to another host and reconstituted upon arrival on the other side. Java's networking support includes sockets, URL communication, and a distributed object protocol called remote method invocation (RMI). In Java, programmatic access to distributed objects is achieved simply; RMIs are handled transparently by a local proxy or stub that interacts with the actual remote object.

Moreover, Java facilitates migration of code and state via its class-loading mechanism. Java's class loaders dynamically load the classes included in an application either locally from the Java `classpath` (list of directories) or through the network. For dynamically loading mobile agent code and

classes referenced by mobile agents, a specialized class loader provides several options:

- An agent's serialized form can include its classes as well as any classes they reference.
- An agent's classes can be loaded from a Web server or from another server.
- An agent's classes can be loaded through the `classpath`.

All code loaded by Java's class loaders is subject to security restrictions, which are very useful for mobile agent systems that have to protect their agents (and the hosts on which they execute) from unauthorized access. Java's security management supports development of fine-grain, highly configurable security policies. For example, agents launched by a particular user may be granted permission to write files, whereas another user's agents may be granted only read-access, and for a third user's agents, no file access at all.

Java also supports development of mobile agents that are tightly integrated with the Web. Applets may launch mobile agents from Web browsers and may also receive the agents they've launched after they complete their remote execution. Java also provides server technology akin to applets. So-called servlets function a lot like a Common Gateway Interface script and may launch and receive mobile agents. And Java's Naming and Directory Interface (JNDI) allows seamless connectivity to business information through unified access to multiple naming and directory services. Mobile agents may, for example, use JNDI service providers to locate the services they need, then connect to legacy systems.

Generic Mobile Agent Architecture

Generic Java-based mobile agent architecture consists of six major components: an agent manager; an interagent communications manager; a security manager; a reliability manager; an application gateway; and a directory manager. Each has to support development of robust, reliable, secure, real-world agent applications (see Figure 3).

For example, the agent manager sends agents to remote hosts and receives agents for execution on the local host. Prior to transport, the agent manager serializes the agent and its state. It then passes the serialized form to its counterpart on the destination host. In a highly reliable architecture, it actually passes the agent off to the reliability manager, which ensures that the agent is received by the agent manager on the remote host.

Upon receipt of an agent, the agent manager reconstructs the agent and the objects it references, then creates its execution context. The security manager authenticates the agent before it is allowed to execute. Thereafter, the mobile agent system (actually the Java virtual machine) automatically invokes the security manager to authorize any operations using system resources (such as reading a file). When the agent is ready to migrate to another host, it

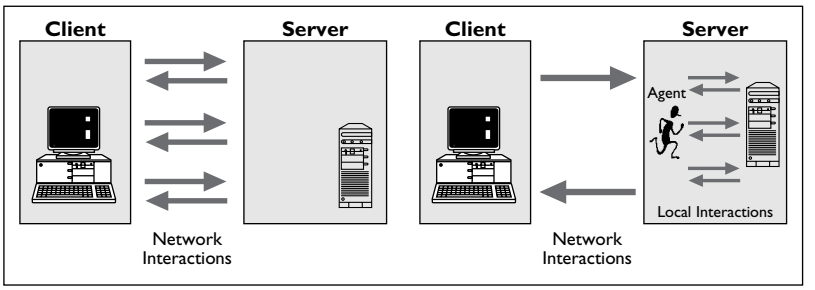

Figure 2. Client/server and agent computing models compared

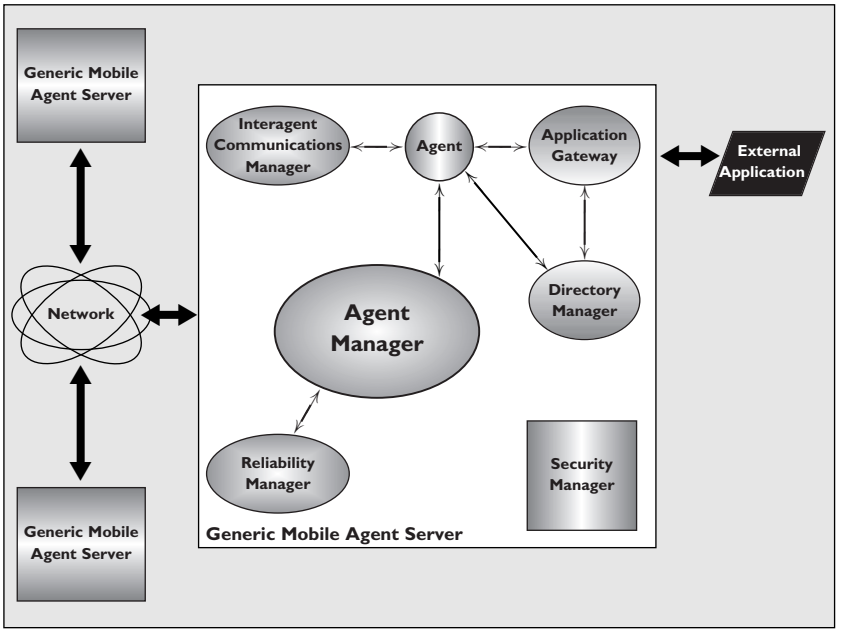

Figure 3. Generic mobile agent system architecture

Concordia

This Java mobile agent technology offers security and persistence while maintaining a record of an agent's travels.

Reuven Koblick

The scale of applications now being considered for network environments requires security and reliability previously available only in large transaction processing systems. To be viewed as a realistic development alternative for such applications, mobile agent systems have to provide highly secure and reliable environments. Mitsubishi's Concordia, introduced in 1998, is designed for such complex, secure, reliable, real-world enterprise applications.

Concordia offers features specially suited for these applications, including extensive security, reliable transmission of agents, access to legacy and native applications, remote administration, and agent debugging. It also includes several forms of interagent communication (for more detail, go to www.meitca.com/HSL/Products/Concordia).

Concordia provides a rich security model for protecting servers, agents, and Concordia itself from attack or unauthorized access. Agent protection is the process of protecting an agent's contents from tampering or inspection during transmission across a network connection or when stored on disk. Such protection ensures the privacy and integrity of the agent and the potentially sensitive information it carries.

Agent users need assurance that sensitive data carried by an agent cannot be compromised and that the agent cannot be redirected to perform unwanted actions. So, prior to transmission, Concordia encrypts an agent's bytecodes, member data, and state information through a combination of symmetric and public-key cryptography. Concordia servers also authenticate each other by exchanging digital certificates.

Concordia encrypts an agent's on-disk representation. For added reliability, it uses a persistent object store to periodically checkpoint an agent; in case of system failure and restart, the agent executes from its last checkpoint. Since the object store saves an agent and its state information, it could also be a potential security risk. So Concordia further secures this on-disk representation through encryption.

Agent authorization. Server resource protection ensures that an agent performs only the server tasks for which it is authorized and for no others. Concordia's server resource protection follows two design concepts: agent identification and resource permission. An agent's user identity uniquely represents the user who launched the agent. It consists of a user name identifying a particular individual, a group name identifying a group of individuals, and a password. Within the user identity, the password is always stored in a secure form and is never represented in clear text.

An agent roaming the network carries its own identity. At each stop in its travels, the agent's identity is verified against a list of the system's valid users. Each server includes a list of users as well as the corresponding resource-access permissions allowed for that user. Default permissions may also be configured and assigned to unknown users.

Resource permissions can be used to allow or deny fine-grain access to machine resources. For example, a resource can be constructed to allow read-access to a machine's file system. Another resource can deny such access. And a third can specify read-access only to a particular file on the machine. Concordia's resource permission mechanism is built atop the standard Java security classes, ensuring that agents use only the server resources to which they are granted access.

If a system's source code can be tampered with, no security policies can guarantee agent or server protection. Hence, class protection ensures that Concordia code is not compromised. Concordia's bytecodes are digitally signed. When an agent executes, Concordia guarantees the agent has not been altered in any way by verifying its digital signature.

Concordia's reliability features include transactional message queuing for guaranteed delivery of agents to remote systems, proxies to shield agents from the effects of system and network failures, and the object store.

Transmission across the network. The Concordia infrastructure provides reliable transmission of agents across the network by way of an underlying message-queuing subsystem. Concordia's queuing support is a natural fit for the disconnected operational mode of the mobile agent paradigm, providing a store-and-forward mechanism. Prior to transmission, an agent is stored in the local system's message queue and remains there until it has been received by the remote host. Agents can also be stored on the message queue of a local system while a remote host undergoes repair or is merely

being moved to a different physical location. When the remote server comes back online, the local server then forwards the agent to the server that was offline.

The message queuing subsystem provides additional reliability by maintaining a copy of the agent to be transmitted in an on-disk queue until the recipient of this agent transmission acknowledges receipt via the two-phase commit protocol.

Proxies increase Concordia reliability by shielding agents and other objects from the effects of server and system failures. Concordia provides proxies for components supporting potentially long-lived connections. Proxies transparently attempt to reestablish connections when they are unable to communicate with their original counterparts.

Concordia includes an extensive remote administration facility that starts up, shuts down, and configures Concordia nodes, or the places where agents execute. It also manages changes in the security profile of agents, as well as servers. The administration facility also monitors the progress of agents throughout the network and maintains agent and system statistics.

Concordia's "service bridge" component gives agents controlled access to native applications (such as legacy databases). It uses the system's security features to ensure that agents do not exceed the permissions granted them by the administration component. Instances of a service bridge can be located through lookup in Concordia's directory service.

A notable difficulty in agent development is tracking the progress of an agent through the network. Concordia's agent debugger monitors, controls, and modifies an agent as it travels and executes throughout the network. The agent debugger helps track an agent at all times. C

Reuven Koblick (reuven@meitca.com) is assistant laboratory director at the Mitsubishi Electric Information Technology Center America in Waltham, Mass.

requests the agent manager transport it to the correct location.

The security manager protects the host and the mobile agents against unauthorized access. All other mobile agent system components interact with it to authenticate and authorize mobile agents. The security manager also may protect agents by encrypting them before transmission and before they are saved to persistent storage. In highly secure systems, the security manager may digitally sign agents, and mobile agent systems may authenticate each other through an exchange of certificates. The security manager also allows authorized agents to pass through firewalls.

The reliability manager ensures the robustness of the mobile agent system. In highly reliable systems, it shields agents from the effects of server and system crashes. One of its main tasks is to guarantee the persistence of state associated with agents as well as with the mobile agent system. In addition, the reliability manager may use transactional queuing, possibly with a two-phase commit, to ensure agents reach their destination, even during system crashes.

The interagent communications manager in Java and other systems facilitates communication between mobile agents dispersed throughout a network. All but the simplest of applications use multiple agents to perform their computations, and the existence of multiple associated agents mandates interagent communication. Mobile agent systems typically offer messaging or distributed events. Some systems include more sophisticated forms of interagent communication, such as Concordia, which enables affiliated agents to cooperatively solve a complex problem that can be partitioned into smaller subtasks.

The application gateway serves as a secure entry point through which agents can interact with application servers (such as legacy databases). Agents may use the JNDI-based directory manager to identify the location of an application server and then migrate to the host on which the server is located. An arriving agent accesses resident servers through this gateway. The security manager has to authorize the agent's use of the gateway and the application server.

Although this generic architecture is sufficient for most application domains, certain extensions to it and improvements in basic distributed computing technology would make mobile agents more efficient and practical for e-commerce applications. The current generation of agent frameworks implements abstractions supporting

The "business rules" of agent and service interaction are still difficult to codify and debug.

mobility and delivering on the promise of scalable, flexible distributed object systems. However, rather like early operating systems, such frameworks are each essentially a closed universe. Current ones do not interoperate on some important levels; missing are layers of abstraction allowing agents created by different users to meet, converse, and share meanings and understandings.

Many users are concerned about the overall performance of systems that are unconstrained and highly distributed. As RMI and similar efforts aimed at making the network transparent continue to mature, unresolved technical issues as mundane as distributed garbage collection and error recovery become significant. System architectures that operate predictably in a single-machine context may not operate so well once their separate elements are distributed, due to the cost of object marshalling and unmarshalling [2]. Needed is better basic distributed object technology.

Find Me a Ski Vacation

While practically all mobile agent systems available today facilitate code mobility, the "business rules" of agent and service interaction are still difficult to codify and debug. Whereas Web-enabled and networked people can act as their own travel agents, they have trouble instructing an autonomous agent to: "Find me a ski vacation for the third week in January, where there is predictably good snow, at a ski lodge where rates are less than U.S.$129 per night." The human planner might be willing to trade off Colorado for Vermont to satisfy the constraints, but the idea is difficult to convey to an agent just about to head out into cyberspace to compete with hordes of other self-interested agents charged with the same mission. In part, the inability to convey knowledge has much to do with the need to define multiagent negotiation protocols [6].

Closely related to defining negotiation protocols, is the need for agents to encapsulate and understand commercial concepts, like place ("vacation" and "ski lodge"), calendar ("third week in January"), and price and currency ("rates less than U.S.$129"). The agent has to understand that a "ski vacation" is a variant of "vacation" that can occur only at certain places and only in the presence of "good snow." Encapsulation of such semantically rich data exchanged between agents and legacy applications via the application gateway, agents, and the user that launched them can be addressed by the extensible markup language (XML), a new Web markup language that allows users to specify arbitrarily structured data types.

Furthermore, to provide more reliable support for Internet-based online transaction processing, more extensible transactional agent support is needed in the generic architecture for Java-based and other mobile agent systems. For example, support for transactional workflow-like semantics is required to ensure that all the work an agent performs is atomic, that is, committed, as it travels through a series of hosts. This atomicity is needed in e-commerce applications—even in a simple travel reservation application using mobile agents in which agents might travel to a number of travel agency database servers to negotiate the lowest fare and in which database updates on the various servers have to be synchronized.

Solutions to address these limitations are beginning to emerge from such sources as CommerceNet with XML and the University of Maryland, Baltimore County, with KQML. Commercially available solutions are rather rudimentary but will certainly be upgraded over time as agents move from research labs into a market in which real money is at stake for users and service providers (see Table 1).

A Killer Opp

While there is (as yet) no single killer application to propel e-commerce, almost everyone developing e-commerce software recognizes that online commerce represents a killer opportunity. Initial beneficiaries of e-commerce development will be business-to-business interchanges. Forrester Research, an information consulting firm in Cambridge, Mass., estimates that by 2001, business

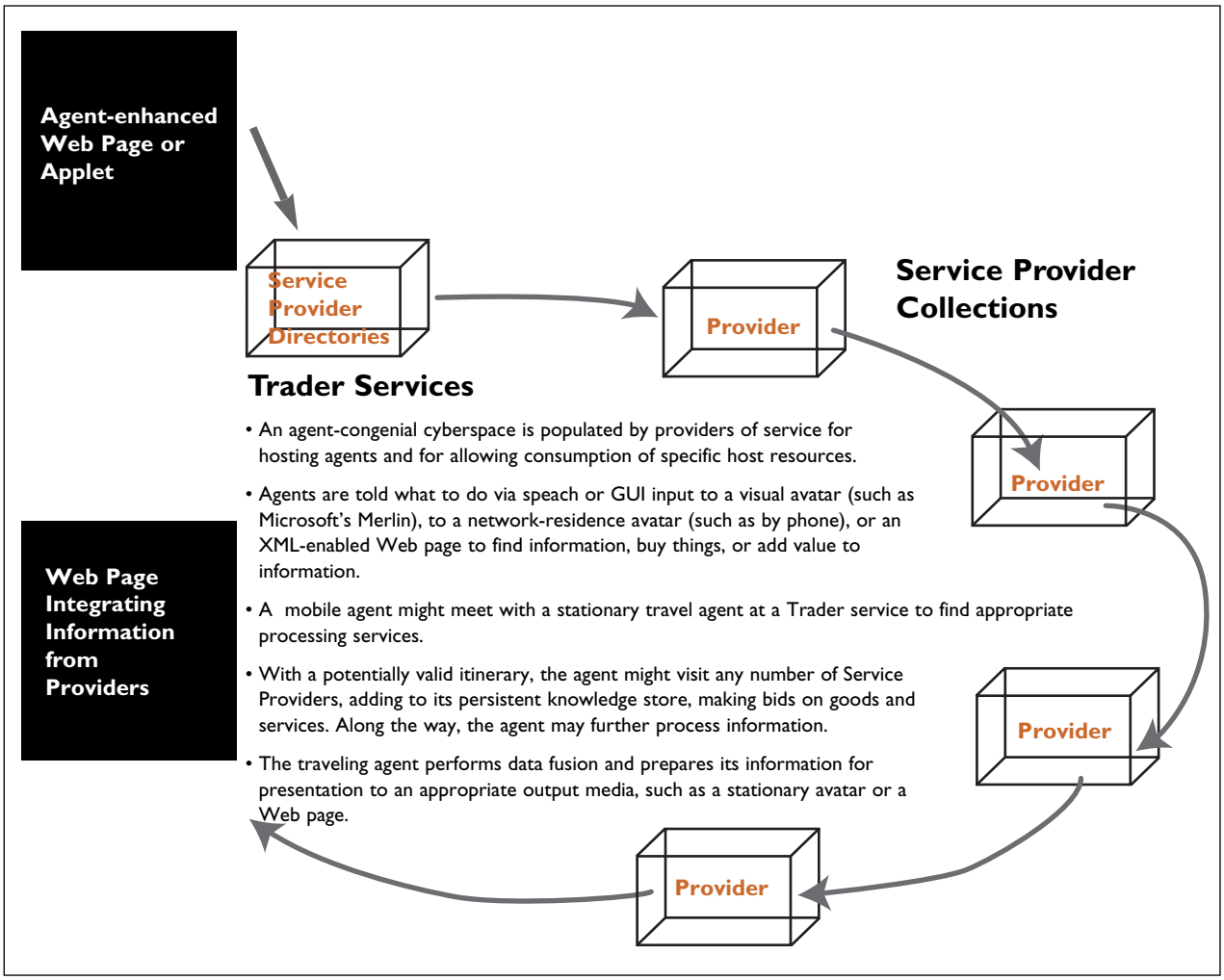

Figure 4. An e-commerce application

extranet and other electronic transaction strategies will be worth about U.S.$183 billion, while business-to-consumer transactions will be only around U.S.$17 billion. This disparity is due primarily to the fact that the vocabulary involved in business transactions is already codified for automated processing. For the foreseeable future, most interchanges will consist of dumb data transfers unaccompanied by a single line of code.

An electronic marketplace rich in killer opps—more than a shared space for simple electronic data interchange and where agents play a crucial role—will experience significant cultural, technological, and economic change. Figure 4 shows an example of an e-commerce application in which mobile agents negotiate with various providers for resources.

Social concerns are also significant, in the long run outweighing changes in technology. Mobile agents, by definition, are not tethered to specific applications and are not simply function calls but are goal-oriented and do not require user intervention. When they return to the place of their creation, users may not agree with the results or may be dissatisfied with the degree of precision in the answer. Meanwhile, the agent community has to resolve a number of technical issues:

- Deferred interaction. How frequently does the agent have to report back to the user?
- Control. How much control does the human user need? When does control begin to become costly?
- Understanding. How well can an agent really understand its human creator's goals, intentions, and negotiating style?
- Flexibility. What accommodation does the agent have to be able to make (or perhaps be forced to make) in a marketplace populated by other (possibly smarter) self-interested agents?

Technological Need	Potential Solutions
Structural. Cyberspace is poorly structured, difficult for an agent to traverse.	• Jini • Discovery and trader services • XML metadirectories
Syntactic. How do agents converse? What are the social rules for talking to strangers?	• Agent communication language • Java introspection • Mandatory agent query interfaces (COM and JavaBeans) • Agent meeting frameworks
Semantic. What do agents say to one another?	• Shared semantic bases • XML • KIF/KQML
Dynamic. Will self-interested agents, seeking to optimize their own utility curve, create a marketplace forever chaotic and unbalanced?	• Circuit breakers • Leveling strategies • Cyber equivalents of the Federal Reserve and the Securities and Exchange Commission

Table 1. Emerging strategies for developing e-commerce systems

- Robustness. How well does the agent have to be able to protect itself from malicious hosts and other agents?

We humans may ultimately realize we can't dictate the mechanism, or "how I want you to work," to an agent, but as long as we reach a reasonable agreement on interfaces, human needs may be served. Even so, given the dynamics of detached and deferred interaction, few of us anticipate designing or deploying systems without user interaction, especially when (as is likely to happen) there is a conflict in user goals. Most humans want to review and validate proposed transactions before their agents commit to them.

The Aglets Project

These Java mobile agents move comfortably throughout the Internet.

Hideki Tai and Kazuya Kosaka

When the Java programming language became available in 1995, IBM's Tokyo Research Laboratory decided to create a new system for mobile agents through a project called Aglets that would operate comfortably in the open system world of Java and the Internet. We designed Aglets by referring to Java's applet and event delegation model so Java programmers could learn and use it easily to take advantage of mobile agents. In 1996, we introduced the alpha version of Aglets, which we now call the Aglets Software Development Kit, as a freely downloadable software package (www.trl.ibm.co.jp/aglets). It was one of the first Java-based mobile agent systems.

An Aglet is a composite Java object that includes mobility and persistence and its own thread of execution and can communicate with other Aglets. It also has its own credentials to indicate who implemented and instantiated it. Based on these credentials, a receiving computer can limit the behavior of incoming Aglets. Using Aglets, a programmer can readily implement autonomous objects in a distributed computing environment.

The life cycle of an Aglet starts with its creation or its cloning. It might be dispatched to another computer or deactivated to or activated from a secondary storage medium before finally being disposed of. Each Aglet is rendered through a programming style involving a call-back model based on the Java event delegation model. During its life cycle, an Aglet receives various kinds of events in response to its actions; for example, if it moves to another computer, a mobility event occurs just before and after the move, and corresponding call-back methods, such as onDispatching and onArrival, are invoked. In this way, each event gives an Aglet the opportunity to determine how to react. A programmer implements an Aglet by filing its call-back methods as appropriate.

In business. To demonstrate Aglet effectiveness, we have to persuasively show its value in real applications, as well as their existence. So, in addition to Aglets, IBM's Tokyo Research Laboratory is developing e-commerce applications to show the business advantages of Aglets. One example is TabiCan, an Internet site hosted by IBM Japan offering several merchant agents for companies selling tickets online: www.tabican.ne.jp/ (in Japanese). When a user accesses TabiCan, a consumer agent is created and interacts with the merchant agents to find travel information, while living only 24 hours to monitor newly posted information.

When developing TabiCan, we assumed that independently developed agents would be able to interact with one another, because agents roam the Internet and meet unfamiliar agents at various sites. There-

What's Next?

We are already beginning to derive value from (admittedly rudimentary) Java-based and other systems of mobile agents. For example, we can create virtual assistants for ourselves that remain persistent in cyberspace and that we can access through graphical user interfaces or through voice commands. Some Web sites provide recommender agents that use simple persistence and a simple rules base to "remember" users' contacts and activities and notify them of events of interest in specific, limited domains.

Although e-commerce will initially largely follow existing social and commercial interaction models, in a more online context, the social and commercial models will begin to drift apart. For example, the online transaction environment will begin to resemble a huge bazaar in some domains; pricing will be determined through real-time evaluation. Online auctions will closely resemble their physical counterparts; many participants will be virtual delegates exercising selfish agent strategies. Selfish agents will themselves become a cottage industry. Market imperfections that heretofore favored the seller will begin to favor the buyer's tireless comparison-shopping agent. This buyer advantage will motivate many sellers in a given sector to hide price and feature information, but fear of lost opportunity will force sellers to raise the information content of their online offerings and express that information in a way that agents and metadata mappers (the next-generation search engines) can capture. Products will become services and vice versa.

Meanwhile, smarter companies will adopt new ways to disaggregate traditional value chains and augment the information component of their products, turning agents into welcome guests. Sellers will want agents to participate cooperatively and to succeed in the missions their creators entrust them with, codifying sufficient information to assure these agents fulfill their missions. And highly specialized

fore, we first developed an electronic marketplace framework, called e-Marketplace, on top of Aglets, to provide a meeting place for agents and define a high-level interaction protocol for interagent communication [1]. The framework also provides a swap-in and swap-out scheduling mechanism to accommodate thousands of consumer agents at the same site.

Mobile agents, like remote procedure calls, are among the fundamental technologies needed to build distributed applications. Today, a number of mobile agent systems have begun to emerge, though many have different application programming interfaces (APIs) and thus cannot interoperate with one another. Ultimately, we want to see one agent system accepting agents created by other agent systems so the systems interact, taking full advantage of mobility throughout the Internet. To achieve this vision, agent software vendors have to standardize the base agent API, which should be simple but extensible enough to allow agents to be mobile, persistent, active, secure, and interactive. The transfer and communication protocols should also be defined to preserve interoperability. The API may also define other service components, such as a directory service for finding agent systems and services and a tracking service for tracing the network locations of agents.

Middleware also has to be created by agent software vendors to reduce the gap between a bare mobile agent system and its applications. It has to be horizontal for multiple types of applications and vertical for specific types of applications to further accelerate the use of mobile agents. That's why we created Aglets and have made it publicly available. In view of feedback from users and our experience with TabiCan, we are quite confident that our dream can be realized.

However, a key question about Aglets, and about mobile agents in general, still needs to be answered: What is the killer application? Although many applications use mobile agents, the killer application has not been found yet. We remain optimistic, because if we create a standard agent API in Java, mobile agents will become more pervasive, spurring many researchers and developers to help create such an application. C

Reference

1. IBM Research. Aglets-based e-Marketplace: Concept, Architecture, and Applications. Research Report RT-0253, Tokyo Research Laboratory, Japan, 1997.

Hideki Tai (hidekit@jp.ibm.com) is an associate researcher in the Tokyo Research Laboratory of IBM Japan, Inc.

Kazuya Kosaka (kosaka@jp.ibm.com) is project manager for object technology in the Tokyo Research Laboratory of IBM Japan, Inc.

agents, knowledgeable in specific domains (available for lease, purchase, or even free) will enlighten us and our agent representatives.

We are using Java mobile agent technology to deliver mobile agents to the threshold of agent-enhanced e-commerce applications, seeking technology with the potential to inspire and support mass-market e-commerce applications. Even today's somewhat limited technology promises an e-commerce explosion in which mobile agents play a key role. C

References

1. Chang, D., and Lange, D. Mobile agents: A new paradigm for distributed object computing on the WWW. In *Proceedings of the OOPSLA96 Workshop: Toward the Integration of WWW and Distributed Object Technology* (San Jose, Calif., Oct. 6–10). ACM Press, N.Y., 1996, pp. 25–32.
2. Foster, S., Moore, D., and Nebesh, B. Autopilot: Experiences implementing a data-driven agent architecture. In *Proceedings of the 26th Technology of Object-Oriented Languages and Systems* (Santa Barbara, Calif., Aug. 3–7). IEEE Computer Society, Los Alamitos, Calif., 1998, pp. 155–162.
3. Jul, E., Levy, H., Hutchinson, N., and Black, A. Fine-grained mobility in the Emerald system. *ACM Trans. Comput. Sys. 6,* 1. (Feb. 1988), 109–133.
4. Odyssey white paper. General Magic Corp., Cupertino, Calif., 1998.
5. Powell, M., and Miller, B. Process Migration in DEMOS/MO. In *Proceedings of the Ninth ACM Symposium on Operating Systems Principles* (Bretton Woods, N.H., Oct. 11–13), ACM/SIGOPS, New York, 1983, pp. 110–119.
6. Sandholm, T. Agents in electronic commerce: Component technologies for automated negotiation and coalition formation. In *Proceedings of the Second Workshop on Cooperative Information Agents* (Paris, France, July 4–7). Springer-Verlag, Berlin, 1998, pp. 113-134.
7. Stamos, J., and Gifford, D. Remote evaluation. *ACM Trans. Comput. Sys. 12,* 4 (Oct. 1990), 537–565.
8. Voyager white paper. ObjectSpace Corp., Dallas, Tex., 1998.
9. White, J. Mobile agents. In *Software Agents*, J. Bradshaw, Ed. MIT Press, 1997, pp. 437–472.
10. Wong, D., Paciorek, N., Walsh, T., DiCelie, J., Young, M., and Peet, B. Concordia: An infrastructure for collaborating mobile agents. In *Proceedings of the First International Workshop on Mobile Agents* (Berlin, Germany, Apr. 7–8), Springer-Verlag, Berlin, 1997, pp. 86–97.

David Wong (wong@meitca.com) is a senior principal member of technical staff at Mitsubishi Electric Information Technology Center America in Waltham, Mass., and a principal architect and the technical lead of the Concordia mobile agent systems framework.

Noemi Paciorek (noemi@meitca.com) is a senior principal member of technical staff at Mitsubishi Electric ITA in Waltham, Mass., and a principal architect of the Concordia mobile agent systems framework.

Dana Moore (dana.moore@att.net) is a technical staff member at AT&T Research Laboratories in Columbia, Md., and a principal architect of the Autopilot agent workflow system for the U.S. Department of Defense.