Dejan Milojicic
Hewlett Packard Laboratories
1501 Page Mill Rd.
Palo Alto, CA 94304
dejan@hpl.hp.com

Trend Wars

Mobile agent applications

This installment of "Trend Wars" addresses the area of mobile agents. Mobile agents are software abstractions that can migrate across the network (hence mobile) representing users in various tasks (hence agents). This is a contentious topic that attracts some researchers and repels others. Some dislike the "mobile" attribute; the others the "agent" noun. Mobile agent opponents believe that most problems addressed by mobility can be equally well, yet more easily and more securely, solved by static clients that exchange messages. Those who favor mobility justify its advantage over static alternatives with benefits, such as improved locality of reference, ability to represent disconnected users, flexibility, and customization.

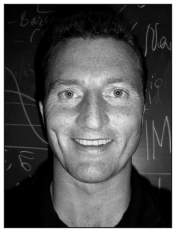
The ideas of mobile abstractions are probably as old as distributed systems. I first got interested in process migration (a predecessor of mobile agents) 10 years ago. It was a hard problem that attracted numerous PhD students; including me. It was also a time of promising distributed operating systems, such as MOSIX, V Kernel, Sprite, and Mach—each of which had a process-migration implementation. Unfortunately, none of them caught on. They were widely regarded as successful and interesting research, but no industrial adoption resulted. Mobile agents seem to be following a similar path.

The term "mobile agent" was introduced by Telescript, which supported mobility at the programming language level. In many ways, Telescript was ahead of time, including its support for mobile agents. Many mobile agent systems followed, most implemented in Java, which already supports mobile code, but also in scripting languages, such as Tcl/Tk or Python. Mobile agents seem suitable for applications, such as electronic commerce, system administration and management (especially network management), and information retrieval. However, few systems actually deployed them in an industrial setting.

In a brief summary, mobile agents have raised considerable interest in the research community (Agent Tcl, Tacoma, and Mole, for example) and in industry (Aglets, Concordia, Jumping Beans, and the like), but the promised deployment has not materialized. This installment of "Trend Wars" is an informal attempt to address this anomaly. In this installment, we have invited prominent proponents and opponents of mobile agents to address this topic. David Kotz codirects the Agent-Tcl project (nowadays called D'Agents) at Dartmouth; Danny Lange led the Aglets project at IBM and now deploys agent applications in General Magic. Charles Petrie, at Stanford's Center for Design Research, still expects a convincing story from the mobile-agent community. Dag Johansen (of University of Tromsø) codeveloped the Tacoma mobile-agent system as well as the StormCast distributed application that outlived different forms of mobility. Ad Astra's Chris Rygaard believes that a good industrial approach is the right way to deliver mobility, as he is trying to demonstrate with Jumping Beans.

These well-known researchers and developers will address the questions about mobile agents listed in the box.

—Dejan Milojicic



Dag Johansen

In what application domains do mobile agents have potential deployment?

I basically see three different domains. One is data-intensive applications where the data is remotely located, is owned by the remote service provider, and the user has specialized needs. Here, the user sends an agent to the server storing the data. The second domain is where agents are launched by an appliance—for example, shipping an agent from a cellular phone to a remote server. The third and maybe most important is for *extensible servers*, where a user can ship and install an agent representing him more permanently on a remote server. The agent is now a personalized, autonomous piece of code that runs remotely and only contacts the user whenever events of interest to the user occur.

Can you describe some of your applications in any of these domains?

Mobile agent (MA) questions

1. What are the application domains in which MAs have potential deployment? Is the "killer application" an absolute requirement for MAs or, for that matter, for adoption of any other mechanism?
2. How do MAs relate to other environments/infrastructures that MA could either contribute to or benefit from: OS process migration; middleware environments (Jini, DCOM, CORBA) networking (active networks, for example); mobile computing (PDAs and laptops, for example)?
3. Currently, there seems to be a separation between multiagent systems and the intelligent agents community on one side and the mobile agent community on the other side. Is there a potential synergy that could be exploited among these groups?
4. There are theoretical and practical studies on the justification of mobile entities with respect to performance, scalability, composability, manageability, fault isolation, and so forth. In your opinion, what should be the main motivation for deploying MAs? The same for not using them?
5. There seems to be an interesting inversion of the order of development of MA. Typically, universities start an investigation of a promising area, after which it gets transferred to industry. With MAs, there are a number of leading companies that started development (Telescript, Aglets, Concordia, Voyager, and Jumping Beans, for example) either before or in parallel to universities. Is this just a coincidence? Where can research institutions help industry?
6. Are there some hard problems that prevented wider MA deployment (security or availability of agent platforms for example); are there problems that are not worth pursuing further (thread state transfer, so-called weak versus strong mobility); are there some problems that have not been addressed sufficiently (such as versioning or composing)?
7. What is your best guess for the future? Will MAs ever be widely deployed or is it just another hard mechanism that community will abandon interest in? Is mobility intrinsically a hard problem to solve, or is it just too early to attack these problems?

We've been building agent-based applications over the last six years. We started out with multihop agents as the default design paradigm. But as we built actual applications, we noticed that single-hop agents dominated. One application where agents performed well involved satellite image data. We had access to terabyte storage of raw data for satellite images, and we shipped agents—in the size of 5 to 6 Kbytes—to the repository storing this data in order to filter through it. We built the same application using a traditional client-server design and compared their performance, but the use of agents produced a tremendous bandwidth-savings. We published this work in the February issue of Springer-Verlag's *Journal of Personal Technologies*. Another application implemented weather alarms. Here we ship an agent from a cellular phone and install it at a remote, extensible weather server continuously updated with current weather data; that agent warns you by phoning whenever a prespecified weather condition has occurred. This work was published in the *ACM Symposium on Applied Computing* in February.

Some opponents of mobile agents claim that there aren't any applications for multihop agents, but you're stating that even single-hop mobile agents are mobile agents?

I consider anything that involves migrating code and associated data to be an agent. And mobile agents are a useful structuring technique whether they are used for single-hop or multihops. However, I do not advocate mobile agents as a default structuring technique and I'm not so concerned as to whether we need any clear applications for mobile agents. We will see the need for this structuring technique in a number of environments—for instance, in managing a large-scale intranet, where you must install, modify, and customize software for many different users without bringing the servers down.

How do mobile agents relate to other environments to which they could contribute or from which they could benefit—for instance, OS process migration?

Mobile agents is not a new concept. It borrows from the Xerox Worm done 20 years ago, from OS process migration work done in the '80s, from remote evaluations done more than 10 years ago at MIT, and so on. The lesson from OS process migration is to avoid forcing a process to move while it is in the midst of execution. Migration is much simpler if done when the process is ready. One should give the process, in this case the agent, much more autonomy in determining when and where to move. Also, we learned that a much richer middleware environment is needed. For instance, brokers, which locate and schedule resources, can be a key to the success of agents. Nowadays, Jini is an example of what you'll need if you want to deploy agents in a real setting. So a more complete agent computing environment with brokers and mediators, electronic cash, and so on, is really needed.

There seems to be a separation between multiagent systems and the intelligent agents community on one side, and the mobile agents community on the other side. Is there a potential synergy that could be exploited among these groups?

Mobile agent builders tend to concentrate on the subsystems for shipping any piece of code around. For instance, we built the Tacoma systems to support more or less any piece of code moving about, whether or not it has some aspect that is called intelligence. The intelligent agent community, however, tends to focus on application specific problems. Intelli-

gent application builders could be using some of the many mobile agent systems available these days as an agent deployment mechanism. Mobile agent builders could then learn what is needed to support in a mobile agent system. We already use this iterative development process in Tacoma, where we always build applications to evaluate a new version of our mobile agent system. Through real application experience, we have learned to structure a typical mobile agent application as a troop, or group, of agents, where not all members of the troop are mobile.

Groups and group communication are hard to achieve even for stationary agents in a client-server model. So, I would imagine that the mobile agent community's lack of coverage of groups and group communication is because these things are even harder to achieve in the presence of mobility.

Actually, the Tacoma project has worked in this area. Our NAP (Norwegian Army Protocol) for implementing fault-tolerance is an example of an agent group. This work was just published in *IEEE ICDCS'99*.

What should be the main motivation for deploying mobile agents?

Of course system performance is important. Perhaps a more important motivation is the ability to customize for the Internet. You deploy some software representing you as a user, and it contacts you when some prespecified condition holds. Instead of having a "pull" model (where you contact the servers), some remote representative or a set of representatives are interacting on your behalf, doing an up-call when something happens in the network that's important to you.

Today it's easy for a system person to build an agent and deploy it. For agents to become widespread, you need a better way to create agents for novice users.

However, I have a modest attitude; I don't advocate mobile agents as the most important structuring thing. They're important to have in your tool set if you want to build a large-scale application. But trying to come up with, or trying to build a large-scale application where mobility is the most important thing—I'm not sure we'll ever find such an application. We should have a very balanced attitude, and just see that it's beneficial for the design and the design's outcome before we deploy mobile agents.

There seems to be an inversion of the order of development of mobile agents. Typically, universities start an investigation of a promising area, which then transfers to industry. In mobile agents, a number of leading companies started development either before or in parallel with universities. Is this just a coincidence?

You might say it was a coincidence that industry and universities got interested in mobile agents at the same time. General Magic and Jim White deserve a lot of credit for bringing forward many of the concepts and ideas that you see in agent systems today, even if they also borrow ideas from the Xerox Worm, process migration, and remote evaluations.

But several academic institutions were starting mobile agent work in '93. The things that motivated us—we started Tacoma in August '93—were Mosaic's emergence and the sudden growth of and hype surrounding the Internet. We conjectured that we needed a new structuring mechanism for this setting, but it shouldn't be at the process-migration level, where you force a process to move to another site. An autonomous piece of code should determine by itself when to move to, for instance, a server. A half year later we learned about General Magic and we started to use the word "agents."

Will the need for mobile agents or mobile objects increase?

Yes. The dominant client will be a smart phone containing a set of preprogrammed agents—or software to let you compose agents yourself—that you can send into the network. Agents won't necessarily run at the client—that is, the smart phone.

What problems have prevented wider deployment of mobile agents?

The main problem is still security, though not necessarily host integrity. Denial-of-service attacks are still hard to prevent, but the hardest problem is agent integrity. Another problem is how to compose agents. Today it's easy for a system person to build an agent and deploy it. For agents to become widespread, you need a better way to create agents for novice users.

What's your vision?

We probably shouldn't expect purely mobile applications to replace other structuring techniques in the near future. But in a large-scale intranet with applications spanning multiple administrative domains, different security domains, and so on, you need code that can be dynamically relocated at run-time. Agents have great potential here. Maybe they would account for 10% of the solution, and you structure 90% of the code the traditional way.

Dag Johansen is a professor in and the chair of the University of Tromsø's Department of Computer Science. This last year he has been a visiting professor at Cornell University. His research interest is distributed computer systems, currently focusing on mobile code architectures and implementations. Since 1993, he has been one of the architects on the Norwegian-US TACOMA (*T*romsø *A*nd *CO*rnell *M*oving *A*gents) project. He also recently cofounded Distributed Architecture Genesis Labs Inc. Johansen received his MSc and Doctorandus degrees from the University of Tromsø in computer science. Contact him at the Univ. of Tromsø, Dept. of Computer Science, N-9037 Tromsø, Norway; dag@cs.uit.no; http://www.cs.uit.no/~dag/.

Dave Kotz

What are the application domains in which mobile agents have potential deployment? Is the "killer application" an absolute requirement for MAs?

MAs will most likely be useful in three general areas. One is disconnected computing, such as laptops and PDAs—they frequently disconnect from the network or use a wireless network that might become disconnected on short notice. The second is information retrieval situations—applications where the agent can be sent to the large data source and filter through the data locally. The third category is dynamic deployment of software. A large organization has hundreds of, say, PDAs in its workforce, and they all need to be reconfigured with a new software version or some data set. An MA can convey that new software to all the PDAs. If it's just a matter of copying a new file to everybody's PDA, then that's not a big deal. But, if it involves some reconfiguration, the code represented by the MA is useful.

I don't think a killer application is an absolute requirement. In fact, I don't think there is a killer application for MAs. Almost everything you can do with MAs could be done with some other, more traditional technology. We tend to look at MAs as a technology that can solve a lot of problems in a uniform way rather than a technology that enables completely new things that weren't possible another way.

How do MAs relate to other environments or infrastructures?

I certainly think there's been a lot of work in OS process migration over the last decade or two, so we could learn a lot from that work. But a lot of the problems involved with MAs are very different. We tend to use interpreted languages and try to support heterogeneous architectures, operating systems, and even heterogeneous administrative domains, such as the Internet, while the traditional process migration world tends to be much more homogenous. We also tend to have different reasons to migrate. Process migration usually supported load balancing, and MAs are generally moving for other purposes—to become closer to some resource rather than just to balance CPU load.

MAs are good for higher-level things, but they're too heavyweight for low-level things such as an active network, which tends to be at the packet level. As a result, the issues tend to differ also.

Mobile computing, on the other hand, is an area where I think MAs have a lot to contribute. I think that PDAs, laptops, and other mobile devices will be a field where MAs will be launched, leaving the mobile platform and roaming through the wired network to accomplish the user's task or representing the users while they're disconnected. I think there are many ways that MA research can help that community.

Currently there seems to be a separation between the multiagent systems and intelligent agents community on one side and the MA community on the other. Is there a potential synergy that could be exploited among these groups?

This is interesting. The association is almost an accident because we both use the term "agent," although for slightly different reasons. You can say that there's a separation between two communities that should be the same, but that's only because we use the same word: agent.

So you think there is separation?

I think there are people who distinctly do multiagent systems or intelligent agents, and then there are people who distinctly do MAs; the groups don't necessarily talk with each other. Part of that is because the intelligent agents and multiagent people come from the AI community, and a lot of the MA community, including myself, come from the systems or even OS community. Thus, we have different languages, different goals, and different ways of looking at these problems.

Do you think there are any potential synergies that can be exploited? Is there potential for these two groups to benefit each other?

I believe that it's possible for MAs, for example, to have a lot of applications and be useful without them necessarily being intelligent. But, on the other hand, I think that there are a lot of applications for which people propose MAs—especially when the MAs are supposed to be autonomous—where the intelligent agent community could really help the MA community. They know how to make agents intelligent or autonomous, and that's something we don't know well. There's some synergy there that we might find useful.

And yet the intelligent agent community generally doesn't think much about mobility, so they don't even think about the particular kinds of problems that an MA might have in being autonomous. They don't necessarily derive their solutions to meet our needs, and vice versa.

One source of synergy might come from the DARPA/Control of Agent-Based Systems Project, or CoABS. It's a very large project with 20 to 25 groups from both industry and academia. One of the goals is to produce an agent grid, sort of a distributed system into which agents can enter, register their services, and look for other agents that provide the services they need to accomplish their tasks. The idea is that the Department of Defense could put together a team of agents and other resources on very short notice to support some sudden defense operation. The intelligent agent people need to find ways to put together these smart agents and get them to communicate. The MA people offer the ability to support this vision on a worldwide distributed network with mobile and wireless devices and so forth. The CoABS Project is pushing these two communities to work together to build a system that benefits both types of agents.

It seems this MA community can provide services beyond mobility and modular mobility, and even system support. If we abstract away

this notion of agents, the environment that you mentioned sounds much like the Jini environment.

Actually, CoABS is building it on top of Jini, using Jini to provide a low-level registry feature. The goal is to support much higher level sorts of services, and, in fact, most people in CoABS don't want to mess with the really low-level services, so I think that's why they chose Jini.

There exist theoretical and practical studies on the justification of mobile entities with respect to performance, scalability, composability, manageability, fault isolation, and so on. In your opinion, what should be the main motivation for deploying MAs? The same for not using them?

It's hard to choose the main motivation, but I would say performance and scalability tend to be my motivation for deploying MAs. I think the reason it's hard to say is that in situations like wireless PDAs, there are times when you're disconnected and just having your agent out there working for you is a higher performance than not being able to do anything at all. But I also believe that there are substantial improvements possible in terms of the turnaround time, even on a wired network. We also expect MAs to be more scalable, in terms of the number of supported users.

In terms of the motivation to not use them, at least from your list, I would say manageability is a concern. The owner of a system that accepts MAs is going to encounter a much wider variety of system activity than someone who just puts up a very dull server like an HTTP server. He will have a much harder time managing his system. I don't think enough research has been done to understand that challenge. Flexibility is another benefit that I would consider a strong motivating factor.

There are substantial improvements possible in terms of the turnaround time, even on a wired network.

What about security? I haven't mentioned it, but it's kind of obligatory.

That would certainly be one motivation. Although there are times when security might actually be an advantage, mostly it's viewed as a potential disadvantage. I think for most problems it's relatively solved. Certainly, the biggest security problem is protecting agents from malicious hosts, but for a lot of classes of problems that's not an issue. If you can avoid that issue, then I don't see security as a big limiting factor.

Typically, universities start an investigation of a promising area and then it gets transferred to industry. In the case of MAs, there are a number of leading companies that started development either before or parallel to universities. Is this just a coincidence? Where can research institutions help industry?

Curiously, many companies got involved with MAs very early. And I think—I'm really just speculating—that the reason was because they saw the potential of MAs very early, before a lot of the hard problems had been solved. That's why I think several companies actually gave up on MAs fairly early.

Where can research institutions help industry? Well there are still several very hard problems to solve, like the security of MAs from malicious machines and a lot of performance and scalability problems. Most MA systems are based on relatively slow interpreters and need a lot of support from the system that doesn't exist yet. I also think agent programmability still needs some work.

Do you think research institutions could help in terms of deployment by coming up with a solution?

In fact, at Dartmouth we tried to start a cooperative network of machines contributed from organizations around the world where everybody would be granted an account on everybody else's machine, as long as they agreed to cooperate. Once you get in, you can install your MA systems on everybody else's machine and run your MAs around the cooperative cluster. Thus, you get access to a much broader range of machines—in fact, a worldwide network. We have five nodes from here to Switzerland and California, but so far only five people have been willing to get involved.

Are there some hard problems that prevented wider MA deployment (such as security or availability of agent platforms); are there problems that are not worth pursuing further (such as thread state transfer or the so-called weak versus strong mobility); are there some problems that have not been addressed sufficiently, versioning or composing?

Well, I certainly think people are very concerned about security these days, especially with all the hacking going on. Certainly, a lot of people view MAs as just another bunch of viruses, or, at the very least, allowing MAs to come into their system would also allow malicious agents—viruses or worms—to enter their system. While I think that most of the security problems of that sort have been solved, at least in theory, I'm not sure that there's any one MA system out there that's got all the problems solved strongly or robustly enough so that a lot of people will be willing to install it.

The other reason is, "Why should I bother?" You know, I put up a Web server on my computer because I want people to see my content. I'm willing to allow them to use my CPU cycles and my disk access cycles, because I want them to see my content. There isn't a good reason for me to allow them to send their MAs to me so they can compute digits of pi or do some computation that's of no interest to me. I think we must either come up with a cooperative world where everybody runs an agent server for everyone's benefit, fitting the old Internet style but not today's Internet, or we come up with a kind of market-based resource-sharing mechanism that allows me to sell my resources to people in return for money or some monetary equivalent (allowing me to use that accumulated wealth to buy resources from others).

The other possible motivation for me to make my machine available to other MAs is, again, to provide access to my con-

tent. If I'm a vendor of some information, it's possible that it will benefit my customers—whether they be paying customers or not—to be able to send their agents to access my information. It might be worth the risk that somebody will send an agent that does something else, so that I can provide the flexibility of agent-based access to my information.

So I think one of the big problems with MA deployment is finding a motivation for people to install these servers.

What is your best guess for the future? Will MAs ever be widely developed? Is mobility an intrinsically hard problem to solve, or is it just too early to attack these problems?

I don't think it's an intrinsically hard problem to solve. I think that most of the problems are solvable in the not-too-distant future. Whether it will ever be widely developed I think will depend more on nontechnical issues. I think that there will be perceptions of security or insecurity, and I think there will be motivational issues that have little to do with the technology and feasibility that will drive whether people actually deploy this stuff widely or not. Isn't that always the way it is with technology? Sometimes it's the nontechnical reasons that make something succeed or fail.

David Kotz is an associate professor in the Computer Science Department at Dartmouth. His research interests are in parallel and distributed operating systems, mobile agents, and wireless networks. He received an AB from Darmouth and a PhD from Duke. He is general chair of the upcoming 17th ACM Symposium on Operating Systems Principles. Contact him at the Dept. of Computer Science, Dartmouth College, 6211 Sudikoff Laboratory, Hanover, NH 03755-3510; dfk@cs.dartmouth.edu; http://www.cs.dartmouth/~dfk.

Danny Lange

Where can we potentially deploy mobile agents?

Primarily in distributed systems because mobile agents have a number of key features that allow them to reduce the network load and overcome network latency. They can encapsulate protocols, and they can work remotely, even asynchronously and disconnected from a network.

Is the "killer application" an absolute requirement for MAs?

I don't think there really is a killer application for mobile agents. In many cases, you can achieve the same results by using more traditional technologies. What we see right now is that for certain applications, mobile agents offer a superior solution, but I don't think you should look for a killer application, because you might not find it.

How do MAs relate to other environments or infrastructures?

The mobile agent idea originated in process migration, but I think it has moved up to the middleware level. We don't see any particular operating system support for mobile agents right now, but we are beginning to see mobile agents in middleware. They relate well with other technologies such as CORBA and Jini. On the other hand, I don't see mobile agents playing a role in mobile computing, PDAs, or laptops any time soon.

There seems to be a separation between the multiagent systems and intelligent agents communities on one side and the MA community on the other. Is there a potential synergy that could be exploited among these groups?

Yes, I think there's a huge potential for synergy. Firstly, many mobile agent systems clearly exhibit multiagent behavior. Agent-based network management is a good example. Collaboration between these agents is essential for successful deployment. I think we'll see more work in that area, when mobile agents and multiagent systems mature because they'll explore each other's fields of expertise. With respect to the intelligent agent community, I see less synergy. But maybe the mobile agent community will serve as role model for the intelligent agent community and inspire those folks to start implementing industrial-strength systems.

In your opinion, what should motivate MA deployment?

Many systems today are becoming increasingly flexible. You can update your Windows operating system on the fly just by connecting to Microsoft's Web servers. You can even figure out what updates you've missed. Mobile agent systems offer you this degree of flexibility in your distributed applications. For example, you have a much greater degree of flexibility because the agents are not bound to reside on specific machines. They can move around, you can update them on the fly, and so on.

But do you need MAs for that?

Admittedly, almost everything you can do with mobile agents, you can do with the client-server architecture.

So what makes the case for using one or the other?

Mobile agents offer a more uniform approach to handling code and data in a distributed system. The client-server architecture is much more static. Client-server based systems are hard to reconfigure, update, and so on. The mobile agent system is inherently much more flexible. One machine can be a server and another machine can dynamically take over that role by simply having the application move to that other machine.

What do you see as a limitation for MAs?

I think mobile agents are not yet ready for high performance applications.

Typically, universities start an investigation of a promising area and then the new technology gets transferred to industry. With MAs, a number of leading companies started the development either before or parallel to universities. Is this just a coincidence?

I noticed the mobile agent revolution happened with Java. Many hackers and programmers got into it, but not researchers. Even today there are many conferences that still prefer papers about SmallTalk rather than Java. I feel it's taken

quite a while for research institutions and academics to accept Java technology. Industry got a lead there, because industry research teams were quicker to embrace Java. But I think research institutions are catching up fast.

Are there problems that prevent wider MA deployment?

Yes, many hard problems haven't been dealt with yet. Security is one of them. So is strong migration and the programming model for agent-based applications. It would be wonderful to see more research on these subjects in the academic field, instead of seeing academics competing with industrial-strength mobile agent systems.

What is your best guess for the future?

I think what we see right now is how the concept of mobile agents is really spreading. It might spread far outside the mobile agent community, and you might see it in many disguises. Look at what is happening to commercial software distribution. You download code, it executes, it says you need to upgrade to the latest version of this and that, or there's a new patch available for this program. You see applets, servlets, and other examples of code mobility happening right now.

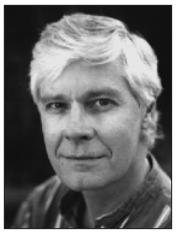
How does this relate to what's happened over the past 10 years?

I can draw a parallel to the whole research area of hypertext. Both academic and industry research were deeply engaged in developing so-called hypertext databases and engines. But when the World Wide Web came out, it was really "just" an HTTP protocol to access files on a server, which was a very scaled down, very simplistic solution. Researchers had been deeply engaged in building dedicated hypertext engines with complex and proprietary solutions to data storage, versioning, security, and sharing. The Web came from a totally different angle, and it just ignored all the hard problems and came up with a much more pragmatic solution. I would not be surprised to see mobile agents in the near future that are not based on mobile agent systems as we know them today. Maybe they'll come up disguised in operating systems or Web applications.

If you could start over, would you again develop Aglets today?

Working on Aglets was extremely exciting, and I think, in a way, more successful than I ever imagined it would be. But you are asking whether I would redo it—let's just say I'm very happy that I did it, I did it at the right time in the right place, but today I would not redo it. There are so many mobile agent systems out there. I would move on. Do something else.

Danny B. Lange is the director of agent technology at General Magic Inc. in Sunnyvale, California, where he heads the research and development of voice agents. He invented Java Aglets, lightweight mobile agents for the Java programming environment. His technical interests include hypertext, object-oriented database modeling, and program visualization. He has MS and PhD degrees in computer science from the Technical University of Denmark. He is a member of the ACM and the IEEE Computer Society. Contact him at danny@acm.org.



Charles Petrie

What are the application domains in which mobile agents have potential deployment? Is a "killer application" an absolute requirement for mobile agents, or for that matter, adoption of any other mechanism?

The answer to the first question is, "I don't know." My answer to the second is that a killer app usually is not only necessary for a new mechanism, but it's what people think of when they think of whatever the mechanism is. They don't think of the Web as HTTP because it's this great protocol. The Web got started because it was an easy way to share files. Same thing with e-mail. E-mail was a cool way to send messages fast and easily, and the mechanism was simply the protocols. There might have been really cool, nifty protocols that the people working on the technology thought were really cool, but that's not the reason that these mechanisms exist. It's because of the killer app—that's what people think of. So, for mobile agents to be adopted, a lot of machinery must be put into place—indeed for any one particular mechanism to win. The question is, what will make people adopt it?

(See http://www.cs.umbc.edu/agentslist/archive/1996b/date.htm, message subject "Practical Mobile Agents," for discussions on this topic.)

How do MAs relate to other environments or infrastructures?

Asked another way, the question might be "What value did these various mobile agent schemes add to the existing mechanisms such as Web-based search engines and crawlers?" The key difference seems to be the idea of local access to data, possibly to private data as opposed to local processing. Local access to data and access to private data are orthogonal dimensions of mobile agents. Imagine that along with hosting Web pages, Web servers also hosted a docking mechanism that let a spider download itself and visit all the public pages, performing its computations in some kind of safe sandbox. That visiting spider would then be able to go to the next Web server down the line, carrying its data to qualify as a multihop agent with persistent state.

That might be useful if you could show that the bandwidth requirements would be lower than for a remote spider exchanging page requests and answers. But that has only been done for small, carefully contrived simulations for particular situations. With regard to private data, imagine that a visiting spider could access your private data. Whoops, never mind. You'd want to set spider traps in that case. So that's not really an issue.

There were similar comments on the use of process migration. People were paranoid about allowing remote programs to come to their nodes, similar to agents here. If you have a well-defined interface between these mobile entities and the local servers that would prevent these local servers, in that case these servers are really shared—they don't belong to anyone. So, for example, in your company, you will have a few servers that will allow access only to certain well-defined and exported data through well-defined and

exported interfaces to some visiting entities. This might be an answer. Again, I'm not saying that this justifies the whole area of mobile agents.

That's what we have now with Web servers. You define what's your public information, you put a server out, and people can access it. So, what mobile agents would add would be that either there's some way for them to get data that you haven't decided should be publicly available.

David Chess made an application suggestion along such security lines. Suppose that you have data you would like for someone else to use, but not see. They could send you a mobile agent who would do local processing and then send back an answer that you would filter or censor. Notice though that this would not be a real mobile agent. There would be no multihops and no persistent state that would define an agent identity. In fact, this functionality could be accomplished by FTPing any random program for local processing and sending back a result. MAs don't contribute anything along the private/public data dimension and the case for bandwidth versus local processing has not been made persuasively. David Chess finally made the most telling statement on the agents list: "At the moment, I find I have a hard time turning *anything* into an example of why we want multihop mobile agents, except that it'd be a way cool thing to play with."

Currently, there seems to be a separation between the multiagent systems (MAS) and intelligent agents community on one side and the mobile agent community on the other. Is there a potential synergy that could be exploited among these groups?

You talk about the gap between the multiagent systems and the mobile agent community. I don't know that there is a gap. Pattie Maes told me in an interview published in the July/August '97 issue of *IEEE Internet Computing* that she is very critical of mobile agents, but that if there ever was a need for mobile agents, she would use them (see http://computer.org/internet/vln4/maes.htm). So far, static multiagent systems seem to be sufficient for the type of applications that interest multiagent systems people.

As a stereotype, we MAS folk concentrate on distributed computations for the communication medium—just messages across a WAN, like the Internet. The mobile-agents folks tend to look more at single agents visiting sites, so there's a difference in emphasis and in technology requirements. You notice that the multiagent-systems folk are solving conceptual problems subject to simulation and analysis that had to do with the distributed computation, whereas the mobile-agent folk tend to be looking for the right implementation with the right security and performance features.

The mobile agents crowd is also more systems-oriented, whereas they are lacking on the applications side. They're coming up with mechanisms, when the multiagents/intelligent agents communities want the applications domain and—I don't want to be insulting here—a kind of hacking of system support.

I don't think that's an insult. That's exactly consistent with what I just said. The MA people are better at and typically more focused on systems and implementation. So that's where the emphasis is. A good place to see several multiagent systems and mobile-agent people talk is in a roundtable discussion on agents in that same 1997 issue of *Internet Computing* as the Pattie Maes discussion. There's a nice quote there by John Ousterhout, developer of Tcl at Sun, who says "Mobile agents are a solution in search of a problem" (see http://computer.org/internet/v1n4/round.htm). I don't think you'd characterize him as a multiagent systems person, so it's not just those people who might have a problem with mobile agents.

There exist theoretical and practical studies on the justification of mobile entities with respect to performance, scalability, composability, manageability, fault isolation, and so on. In your opinion, what should be the main motivation for deploying MAs? The same for not using them?

You're asking about the motivation for deploying mobile agents. Again, to push on this—and maybe this isn't what the mobile-agent people want to push on these days—but if you could really show that multihop agents gave you a performance advantage over static agents exchanging messages, that would be a major motivation for at least the application in which you showed that. But you need experiments or extensive, unbiased analysis to compare them to.

Back when they were doing mobile agents, General Magic published a white paper on the Web that made the tactical argument claiming that bandwidth is too valuable to waste on exchanging messages. A lot of people in mobile agents have made that particular argument: it is flawed in three ways. First, usually these people don't do the work to show that exchanging messages is actually cheaper than exchanging agents. Certainly General Magic didn't do it in their white paper.

Second, arguing that bandwidth is just too valuable to waste assumes that message exchange is more costly. In 1997, Bob Metcalf and George Gilder argued about this in the first two issues of *Internet Computing*. Gilder claimed that bandwidth was abundant and growing, and he was proved right. Metcalf eventually had to eat his column.

Third, General Magic assumed that these mobile agents would be especially useful in home appliances—networked via telephone line dedicated to voice traffic. Of course, that's increasingly not the major Internet connection in homes. The fact that they weren't more prescient in this regard is really a special case of a bad assumption about the cost of bandwidth.

Notice now that the mobile-agent proponents are shifting their argument to say that mobile agents would be particularly good for the lower-bandwidth systems on the edge of the net-

Notice now that the mobile-agent proponents are shifting their argument to say that mobile agents would be particularly good for the lower-bandwidth systems on the edge of the network.

work. Even if mobile agents were shown to be more efficient in such situations, this argument keeps moving mobile agents further toward the edges, as the edge gets faster, rather than being in the mainstream of, say, electronic commerce.

To push on this a little bit harder, there was a recent article by Bill Venner in *Java World* that Danny Lange recently pointed to as a source of real application examples (see http://www.javaworld.com/javaworld/jw-05-1997/jw-05-hood.html). But if you read that article, there are no real applications mentioned. In fact, that article's justification for mobile agents are surprisingly vague and weak. Lange's pointer was the only answer to a recent repeated requests for examples of mobile-agent applications on the agent's list.

Having pointed to these examples of flawed arguments that characterize many of those posed by mobile-agents proponents, let me say that there has been some good performance evaluation work done.

Harrison, Chess, and Kershenbaum wrote "Mobile Agents: Are They a Good Idea?" (*IBM Research Report RC 19887*, 1995 http://www.research.ibm.com/massive/mobag.ps). They concluded—and they are proponents of mobile agents—that the only case in which mobile agents showed promise, at least in their experiment, was in real-time controls and remote instruments.

Let me mention a few others. I believe Manolis Marazakis has done some work showing some advantage in local query processing (http://www.cs.umbc.edu/agentslist/archive/1996b/ 0810.html).

The PhD thesis of Robert Gray (developer of AgentTcl) also included a good performance evaluation study http://actcomm.dartmouth.edu/~rgray/) that seemed inconclusive.

Theilmann and Rothermel published an IEEE paper this month that describes a possible communication cost savings with MAs. However, their abstract states at the beginning that "there is hardly any scenario in which this advantage has been proven or quantified on Internet scale." (http://www.informatik.uni-stuttgart.de/ipvr/vs/Publications/Publications. html#1999-theilmannEA-01).

All of these results depend upon setting up artificial situations in which the savings can occur in simulation in a restricted set of cases. None of them persuade me that the performance advantage of MAs is so big that that we should not try to get the same functionality with static agents and conventional web technology and rather go to a great deal of trouble to overcome the MA technical problems and pain of deployment.

Typically, universities start an investigation of a promising area and then it gets transferred to industry. In the case of MAs, there are a number of leading companies that started development either before or parallel to universities. Is this just a coincidence? Where can research institutions help industry?

Many mobile-agent problems are at least perceived as mostly software system implementation in nature and can largely be done by industrial software engineers. The more conceptual, less system-oriented work of the academic PhDs in multiagent systems is simply viewed by them as perhaps not as interesting. The idea of mobile code is easy to understand, but it presents all sorts of interesting technical challenges. Distributed computations are somewhat more mysterious. That, by the way, is criticism of multiagent systems work. But it is clear that there are a number of ways for academics to contribute, I think, especially in performance evaluation.

Are there some hard problems that prevented wider MA deployment (such as security or availability of agent platforms); are there problems that are not worth pursuing further (such as thread state transfer or the so called weak versus strong mobility); are there some problems that have not been addressed sufficiently, versioning or composing?

There are of course a lot of security issues waiting to be solved. One of the major ones is simply the fact that mobile agents can reproduce themselves exponentially, overwhelming the entire set of computers that have mobile-agent docking mechanisms. John Ousterhout mentioned this denial of service attack in that 1997 roundtable. That's a well-known problem.

But I am guilty like most of not having pushed on cool applications, because (like most geeks and academics) I'm interested in complicated, hard-to-understand topics.

But I think deployment is really waiting for the killer app and for easy installation. Security is certainly an issue, but it's not quite as overwhelming as we might all think it is. Microsoft has demonstrated that security is less of a concern if the application—such as Word attachments with macros—is sufficiently useful. So the question mobile-agent proponents need to answer is, "Why would I allow what is essentially a virus on my computer? What's in it for me?" That question hasn't been answered yet.

What is your best guess for the future? Will MAs ever be widely developed? Is mobility an intrinsically hard problem to solve, or is it just too early to attack these problems?

Mobility with satisfactory security is intrinsically difficult, but what will really turn mobile agents into the CB radio of the Internet is the lack of applications to motivate the work necessary to deploy the mobile agents and overcome the security issues.

To be fair, I criticize many—but not all—in the MAS community for also having failed to deploy commercially useful agents and not being driven by applications. Our agent infrastructures and languages tend to be driven more by notions of computational elegance than need. Here I have to plug our own JATlite (http://java.stanford.edu) agent-message router because that feature was driven by real needs in our distributed engineering research.

But I am guilty like most of not having pushed on cool applications, because (like most geeks and academics) I'm interested in complicated hard-to-understand topics. In particular, with

respect to electronic commerce, I said at ICMAS in Paris last summer that if the agent community (MA and MAS) did not deploy useful EC apps within a year, those technologies would be supplanted by Web-based technologies such as Java servlets, XML, and Jini and other infrastructures for portable Web services. If there are directory services and message protocols all implemented in Java classes for appliance services and the exchange of business models, why should we not use them for agents? The existing clunky agent infrastructures will fall by the wayside. My timing might have been off, but only because some of the Web technologies were optimistic in their deployment schedules.

That said, let me say that the future for agents has never been brighter because we'll be able to take advantage of a widely deployed Web infrastructure that will replace much of the conceptually uninteresting directory services and message delivery work that we have to do now. We'll have lots of real data and machines to play with. So something good is bound to happen.

Charles Petrie is a Senior Research Scientist at the Center for Design Research at Stanford University. His current research interest is agent-based distributed process coordination and project management, enabling concurrent planning and design, using agents with shared models for change propagation. He is the founder and EIC Emeritus of *IEEE Internet Computing*. Contact him at the Center for Design Research, Stanford Univ., Bldg. 560, Panama Mall., Stanford, CA 94305-2232; petrie@stanford.edu; http://cdr.stanford.edu/~petrie/bio.html

Chris Rygaard

Is a "killer application" an absolute requirement for mobile agents or, for that matter, adoption of any other mechanism?

Rapid adoption does require a killer application. But even though object-oriented programming doesn't have a killer application, it was adopted anyway. Mobility will be adopted just like OO was adopted.

What is the application domain for potential MA deployment?

They're powerful in the embedded world because those are memory-constrained environments. Mobility lets you dynamically swap apps. The most powerful application is in system management—extending an MA out to a remote device or node does the work for me, and saves me the effort of walking over there. And the device might not even have a UI.

So how does that differ from just servlets? People who criticize mobile agents typically claim that if you have to upload or download something to a device, it isn't a mobile agent.

We're getting into the definition of mobility; I'm trying to stay away from the phrase agents. My definition says it's got to at least include the state of the application as well as the code, and an applet does not fit that definition. It isn't necessarily a mobile agent if you download it. Just because you download something doesn't make it a mobile app.

These people who criticize mobility: they claim that almost anything you can think of can be handled by traditional methods. But they forget that mobility is great for all of those things that you didn't think of till it's too late.

How do MAs relate to other environments or infrastructures such as Jini and CORBA?

MAs are absolutely, totally, 100% unrelated to Jini. Everybody gets confused between Jini and mobile agents because they happen to have some code mobility, but other than that, they're completely unrelated to each other. Jini is a mechanism for discovering devices on the network. It's a mechanism for devices to announce themselves on the network. Mobile apps, on the other hand, are a mechanism for interaction on an established network.

Mobility can enhance CORBA. Mobile CORBA technology enhances CORBA to make server-side implementations mobile.

Is there a potential synergy between intelligent agents and mobile agents?

In the academic world, absolutely. Out in the real world though, AI is still targeted at just a few vertical applications. We have seen very few places where MAs and AI work together.

So you don't think that in the future, some programs, whether we call them intelligent or not, will represent users?

I think MAs help programs represent users. I described how I could send a mobile application out to a remote network node to do work for me so that I wouldn't have to walk over there. That is doing work on behalf of the user.

So basically you're saying it's just a program that does something...

That's correct.

Where can research institutions help industry?

So far, my experience has been that research institutions are not helping industry. For example, a lot of research institutions get big bucks out of the DoD to build systems that are unusable, and they end up competing with me. They're harming our marketing. Our marketing has been to address real-world issues. Research institutions don't, but they have bigger budgets.

So most of the work in research and development should be short term and very focused? But there are some areas, especially the new ones, where you need some basic research or investigation.

In general, sure, that's true. In many areas, some basic research is needed. I don't think mobility is one of them. Without a lot of basic research, we were able to build Jumping Beans. The guys who invented Java did the basic research. Mobility is just a technology that utilizes the capabilities already built into Java.

Are there some problems that prevent wider MA deployment?

Security is a difficult problem. Availability of agent platforms I don't think has been a problem.

ALSO SEE THE AGENT SYSTEMS AND APPLICATIONS / MOBILE AGENTS (ASA / MA) SYMPOSIUM ANNOUNCEMENT ON P. 96

(HTTP://WWW.GENMAGIC.COM/ASA/)

But specifically related to availability, environments that could foster Jumping Beans servers are not widely available around the world. You can't send your Jumping Beans just anywhere. So there needs to be some host where you can send it. Wouldn't that be considered a problem?

Look at it differently. I don't think that my customers need to be able to send Jumping Beans just anywhere, because of security. Consider the guy receiving it. He doesn't want to receive a mobile application from just anywhere. And a Jumping Beans deployment will be within one corporation. Because of security perceptions, there will be, at least in the beginning, only limited business-to-business mobile apps being used. An MA solution—all MA solutions initially—will be a vertical solution. There won't be a generic utility such as business-to-business E-commerce or something like that. It will be a single vertical solution within a corporation.

CORBA is popular, but I can't just make an IIOP connection to any place on the Internet.

What about thread-state transfer?

Thread-state transfer is not real. It cannot be made real. If you were doing a theoretical, mathematical calculation, thread-state transfer would be wonderful. If you were doing some simple calculations, thread-state transfer would be wonderful. But let's look at the real world. The real world has file access and interaction going on with the user. The real world has TCP/ IP connections going on with the network. That simply cannot be transferred. You can't do that with a user interface. The files don't exist on the target machine. Thread-state transfer is not real world.

The real world has TCP/IP connections going on with the network. That simply cannot be transferred. The files don't exist on the target machine. Thread-state transfer is not real world.

Some people did, at least in the process migration area, transfer many things over open channels. But that assumed that you would have some additional system support or distributed fault system.

There are problems with this. How do you handle file access? How do you take care of a file handle that's open, when you've moved to a machine where the file doesn't even exist?

Well, you would need to have some distributed file system support, especially if you were talking about the Internet.

Mobile apps do away with this. A distributed file system is not mobility, its just traditional distributed computing. It's got parts of mobility, but it's not true mobility. It messes up autonomy. A real-world requirement—not a true requirement, but a practical requirement of mobility—is an autonomous system. You send a mobile app over to another machine. That machine, once it has the mobile app, should be able to disconnect. You can't do that with file sharing.

So if two agents, for example, communicate and one of them moves away, and if they still want to collaborate on some task, they would still have to have some form of reconnecting their communication channels after migration.

Oh, sure. That's unrelated to thread-state transfer, though. The underlying mobility infrastructure should maintain those communication channels. The underlying infrastructure provides a layer of abstraction on top of the underlying communication mechanism to make that work. That's the idea behind Mobile CORBA.

At least with Java right now, natural file access depends on the file being on the local file system. And if you're doing file access and you try to transfer to thread state, you're going to have all kinds of problems.

Are there problems that have not been addressed sufficiently, such as versioning or composing?

Versioning is a tough one. I know that it hasn't been addressed completely. I think there are some extreme cases that will choke in Jumping Beans. But in most cases, different versions of an MA can easily coexist on the same Jumping Beans client.

Composing has not been addressed sufficiently in Jumping Beans. Our PlaceMaker product is available as a demo right now, not as a product. With PlaceMaker, you can graphically build a mobile application, graphically build an itinerary for it, and graphically send it off. We provide a graphical bean container and a palette of ordinary, off-the-shelf, third-party Java beans. You can drag and drop these beans into the container. The container is aware of mobility. It will mobilize its contents, even if the contents are not aware of mobility. So we can very easily graphically compose mobile applications with our stuff. However, it's not a full-blown development environment. Think of a mobile bean box, and you'll have an idea of PlaceMaker.

Chris Rygaard cofounded Ad Astra Engineering Inc. in 1996. He works on research and development of mobile systems and their security at the company, and is the chief architect of Jumping Beans. He has an MS in electrical engineering from Santa Clara University, and is a member of the IEEE and the Software Development Forum. Contact him at Ad Astra Engineering Inc., 961 The Dalles, Sunnyvale, CA 94087; crygaard@AdAstraEng.com.