Agents on the Web

Mobile Agents

Michael N. Huhns and Munindar P. Singh

May 1, 1997

There are beginning to be a lot of agents executing on the web, and some of them are starting to move around. While most agents are static, in that they exist as a single process or thread on one host computer, others can pick up and move their code and data to a new host in the web, where they then resume executing.

"Beam me up, Scotty!"

Are such agents mobile, itinerant, dynamic, wandering, roaming, or migrant? And, are they sent, beamed, teleported, transported, moved, relocated, or RPC'd? These are some of the questions swirling around web discussion groups these days. However, since anything that can be done with mobile agents can be done with conventional software techniques, the *key* questions are really

> *Are mobile agents a useful part of a distributed computing system?*
>
> *Are there applications that are easier to develop using mobile agents?*
>
> *Under what circumstances is it useful for an agent to be mobile?*

We find that there are very few such circumstances, in spite of all the effort being spent on developing techniques for mobility. And, there is a fundamental reason, historically called the **procedural-declarative controversy** [see sidebar], why this is so. Nevertheless, there are several appropriate uses for mobile agents

Appropriate Applications

In general, the best applications for mobility might be those that involve the dynamic installation of code to extend the functionality of an existing system. This would address a potential limitation of current static systems, which are not easily enhanced. However, new functionality can be installed without requiring a full-blown mobile agent. All you need is a standard message type "install(function_name, version, argument_types, code)". The receiving agent can autonomously decide—based among other things on its level of trust in the sender—whether to install the corresponding code; if it does, new functionality becomes available. And, you never need to ship state information around.

Disconnected Operation

A major consideration for personal digital assistants (PDAs) is battery capacity and, therefore, connect time. Because of this, PDAs are forced to spend most of their existence off-line. Now, suppose you have constructed an agent that knows your preferences and interests, and can filter information sent to you from multiple sources. Further, suppose your agent can provide real-time feedback to the sources that would enable them to improve the precision of their information. This agent can run on your PDA, where you can interact with it and instruct it. However, you do not want your agent to stop functioning when you turn off your PDA—when this happens, your agent should move to a host that is on-line.

Testing Distributed Network Hardware (a multihop application)

Graham Glass (gglass@objectspace.com), from ObjectSpace, Inc., has suggested the following application for mobile agents. A distributed telecommunication switch is built out of thousands of different cards, each containing different hardware. The rules for testing this hardware vary from board to board. There are routines for testing an individual board, groups of boards, and entire systems. The code for thorough testing can be quite large, and can improve over time. Since the bandwidth for performing system tests can be quite large also, such tests are often performed off-line.

A traditional approach to network diagnostics is to load the board-level testing code directly into the boards and have these boards self-test periodically, sending their results to a main testing controller. The system level tests do not fit into the boards and consume too much network bandwidth, so they are loaded remotely when the system is inactive.

A mobile agent approach is to launch testing agents into the active network These agents roam between boards, performing tests in a stochastic way. Larger system-level testing agents can displace smaller board-level tests when necessary. This allows boards to accommodate many testing strategies with a small amount of memory, since the agents can come and go over time. Testing agents carry with them both their previous testing history and the means to perform the test, a natural set of associated items. Testing agents can make local decisions, allowing them to repeat tests as necessary or test boards around them without having to report back to a central controller and consume precious bandwidth.

Customized Searches on Servers

The most frequently proposed use for mobile agents is to send them to execute on servers, particularly when the servers have more information than can be reasonably communicated back to a client for processing there *and* lack the necessary procedures to perform the desired processing themselves. This is a special set of circumstances that does not often hold. Even when it does, you should compare it to a declarative approach, which would be to implement a protocol of search primitives that could be invoked via messages between a user agent and the server agent. This approach would mitigate the security worries that the mobile agent would run amok, intentionally or otherwise.

This approach would also offer efficiency advantages. When a mobile agent runs remotely, the server gives up control of disk, memory, and processor resources to the agent. Instead, if the server accepted a sequence of declarative search primitives, it could schedule and carry them out in a way that is optimized to its current state. For example, a modern DBMS could use its own optimized techniques to compute a join much more efficiently than a remote user could program an agent to compute one.

Information Commerce

There are times when an information consumer would like to apply proprietary algorithms from one company to proprietary data from another company. A solution would be to find a trusted third party to whom both the data and the algorithms, encoded in a mobile agent, could be sent.

Mobile Agent Frameworks

There are a number of active efforts underway to develop systems, protocols, and frameworks for both the construction and use of mobile agents. Most of the following frameworks allow agents to be started, stopped, and moved, and a few allow them to be monitored.

- Odyssey from General Magic<ODY>—agents are programmed in Java
- ARA "Agents for Remote Action" from the University of Kaiserslautern<ARA>—agents are programmed in Tcl, C/C++, or Java
- MOA "Mobile Objects and Agents" from The OpenGroup<MOA>—uses OS process migration technology
- Concordia from Mitsubishi Electric Information Technology Center<CON>—agents are programmed in Java
- Aglets from IBM<AGL>—agents are programmed in Java
- TKQML from University of Maryland Baltimore County<TKQ>—migrating agents are programmed in Tcl and communicate in KQML
- Agent Tcl from Dartmouth<ATC>—transportable agents are programmed in Tcl

There is an effort by the Object Management Group (OMG) to establish industry standards for mobile agent technology and interoperability among agent systems, such as Odyssey, Aglets, and MOA. The OMG intends to define a Mobile Agent Facility (MAF) for CORBA. A draft of the MAF specification is available from General Magic at <http://www.genmagic.com/>.

Concerns

Security

There are two main aspects to security involving mobile agents. The first, and most commonly considered, is protection of the server against intentionally or accidentally malicious agents. The second is protection of a mobile agent against malicious servers. The former aspect has been dealt with extensively in the context of operating systems, which establish and maintain protection levels for process execution. Security in the latter aspect cannot be guaranteed, because in order for the mobile agent's code to be executed, the agent has to expose both its code and data to the server. A detection, but not prevention mechanism, is to have the agent return itself with its data, to verify that it has not been altered.

A prevention mechanism might hinge on a determination of legal responsibility: are you liable or not for your agent's deeds? And, who pays if, e.g., a malicious server causes you to buy something on another server?

Authentication, integrity, confidentiality, and nonrepudiation are other important aspects of security. Authentication validates the identity of the person or agent with whom you are interacting. Integrity ensures that what you see has not been tampered with, confidentiality ensures that what you intend to be private remains so, and nonrepudiation means you are liable and cannot change your mind.

Survivability—Too Short or Too Long!

Mobility can improve the survivability of an agent—it can move if its execution on a host is threatened—but it can also result in the agent continuing to exist long after its usefulness has ended. Once mobile agents are launched, it becomes difficult to monitor and manage them. The problem is compounded when agents are given the ability to replicate.

BEGIN SIDEBAR

"The Procedural-Declarative Controversy"

We view the mobility of agents as primarily an issue of infrastructure—a matter of how agent functionality might be realized. A client seeking information from a server can either send a procedure to execute on the server and find the desired information, or send a message requesting the server to find the information using its own procedure. Our objection to the usefulness of mobile agents lies in their being a low-level *procedural* means to achieve what communication techniques can support at a higher *declarative* level. Similar objections have arisen time and again throughout the history of computing, and higher level techniques have always won out. Examples include high-level programming languages vs. assembly languages, SQL vs. navigational queries, conceptual vs. physical data models, and formal grammars and compiler generators vs. hard-coded compilers.

Some of the trade-offs were debated in 1975 during what was called the “procedural-declarative controversy” <Winograd75>. In this controversy, which was focused on AI knowledge representation, declarative approaches were said to describe *what*; while procedural approaches describe *how*.

In a narrow sense, procedural approaches can be more efficient. However, when the flexibility of solutions and the productivity of programmers are taken into consideration, declarative approaches usually pay off. Declarative approaches offer advantages in

- Modularity—requirements can be captured independently from each other.
- Incremental change—it is much easier to add or remove components from a declarative specification than to rewrite procedural programs.
- Semantics—declarative notations can be given a formal semantics directly, whereas procedural languages must first be mapped to declarative structures. Formal semantics is crucial for validating tools for building agents and their interaction protocols. It assures predictable behavior, and enables efficiencies in implementation without jeopardizing soundness.
- User interfaces—declarative specifications are easier to generate than procedural code, leading to greater productivity for interface developers and, coupled with clean semantics, greater predictability for users.
- Inspectability—being explicit, declarative specifications can be examined to determine (a) the current constraints on an agent and its interactions, (b) how far the constraints have been satisfied, and (c) the rationales for different actions.
- Learnability—declarative specifications are easier to learn, enabling an agent to discover how other agents behave, and how to participate in an ongoing "discussion" among agents.

Mobility includes procedural encodings in two distinct respects. One, the behavior of a mobile agent is procedurally coded. This might be reasonable for some static agents as well. Two, the interactions of a mobile agent are implicit in the code that constitutes it. This is unnecessary when the agent is static. A static agent’s interactions can be explicitly specified in terms of protocols involving its communications (see our column in IC, 1(2):73-75). Static agents can then be supplied by different vendors and programmed in different languages as long as they communicate properly with each other.

Ultimately, there is no difference between a very complex request language and a very simple programming language. We are, in fact, really talking about a continuum of approaches.

<Winograd75> Terry Winograd, “Frame representations and the declarative/procedural controversy,” in D. Bobrow and A. Collins, eds., *Representation and Understanding*, Academic Press, New York, 1975. Reprinted as Chapter 20, pp. 358-370 in Ronald Brachman and Hector Levesque, editors, *Readings in Knowledge Representation*, Morgan Kaufmann, San Francisco, 1985.

END SIDEBAR

Challenges for Mobile Agent Technology

We believe that agent mobility has a useful, albeit limited role to play in distributed computing. For it to be successful in that role, agent languages are needed that

- can express useful remote computations
- do not violate the security of the sender or receiver
- are portable and extensible.

Furthermore, techniques for managing distributed computations are needed that can

- disseminate extensions to the programming language interpreter
- authenticate senders
- prevent deadlocks and live-locks
- control agent lifetimes
- prevent the flooding of communication or storage resources.

System of the Bimonth

To experiment with mobile agents, we suggest you try General Magic's brand new release of Odyssey Version 1.0 Beta. You can download it for free from <ODY>. Check it out!

References

<AGL> Aglets: <http://www.trl.ibm.co.jp/aglets>
<ARA> Agents for Remote Action: <http://www.uni-kl.de/AG-Nehmer/Ara/>
<ATC> Agent Tcl: <http://www.cs.dartmouth.edu/~agent/>
<CON> Concordia: <http://www.meitca.com/HSL/Projects/Concordia/>
<MOA> Mobile Objects and Agents: <http://www.osf.org/RI/DMO/dmo.htm>
<ODY> Version 1.0 Beta release of Odyssey: <http://www.genmagic.com/agents>
<TKQ> Agent Support for Tcl: <http://www.cs.umbc.edu/agents/kqml/papers/tkqml.ps>