1997

Mobile Agents

Michael N. Huhns
University of South Carolina - Columbia, huhns@sc.edu

Munindar P. Singh

Publication Info

Published in *IEEE Internet Computing*, Volume 1, Issue 3, 1997, pages 80-82.
http://ieeexplore.ieee.org/servlet/opac?punumber=4236

AGENTS ON THE WEB

Michael N. Huhns • *University of South Carolina* • *huhns@sc.edu*
Munindar P. Singh • *North Carolina State University* • *singh@ncsu.edu*

MOBILE AGENTS

A lot of agents are executing on the Web, and some of them are starting to move around. While most agents are static (existing as a single process or thread on one host), others can pick up and move their code and data to a new host where they resume executing.

"BEAM ME UP, SCOTTY!"

Are these 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 also be done with conventional software techniques, the key questions are really the following:

- 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?

Only rarely do circumstances call for an agent to be mobile, in spite of all the effort being spent on developing techniques for mobility. There is a fundamental reason for this, historically called the procedural-declarative controversy (see sidebar).

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, you can install new functionality without employing 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 install the code, new functionality becomes available—without your having to ship state information around.

Disconnected Operations

A major consideration for personal digital assistants (PDAs) is battery capacity and therefore connect time. Thus, PDAs must spend most of their existence offline.

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 those sources, helping them 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 you turn it off, your agent should move to a host that is online.

Testing Distributed Network Hardware (A Multihop Application)

Graham Glass of ObjectSpace* has suggested an application for mobile agents. A distributed telecommunication switch consists 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 required for performing system tests can also be quite large, the tests are often performed offline.

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. Because the system-level tests do not fit into the boards and consume too much network bandwidth, they are loaded remotely when the system is inactive.

The Procedural-Declarative Controversy

The mobility of agents is primarily an issue of infrastructure—a matter of how we might realize agent functionality. 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 asking 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. 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. In each situation, the higher level technique won.

Some of the trade-offs were debated in 1975 during what was called the "procedural-declarative controversy."[1] In this controversy, which was focused on artificial intelligence (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 several advantages:

- *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 that build agents and their interaction protocols. It ensures 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.
- *Adaptability*. 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 last column "Conversational Agents," Vol. 1, No. 2, pp. 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.

REFERENCE

1.T.Winograd, "Frame Representations and the Declarative/Procedural Controversy," in Representation and Understanding, D. Bobrow and A. Collins, eds., Academic Press, New York, 1975. Reprinted in Readings in Knowledge Representation, R. Brachman and H. Levesque, eds., Chap. 20, pp. 358–370. Morgan Kaufmann, San Francisco, 1985.

In a mobile agent approach, testing agents are launched 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 less memory, since the agents can come and go over time.

Testing agents carry with them both their 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, consuming 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 they can reasonably communicate to a client for processing and also lack the necessary procedures to perform the desired processing themselves.

But this special set of circumstances rarely holds. Even when it does, using mobile agents might not be as effective as using a declarative approach and implementing a protocol of search primitives. The protocol could then 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.

It would also offer efficiency advantages. When a mobile agent runs remotely, the server surrenders control of disk, memory, and processor resources to the agent. If, however, the server accepted a sequence of declarative search primitives, it could schedule and carry them out in a manner optimized to its current state. For example, a modern database management system could use its own optimized techniques to compute a join much more efficiently than a remote user could program an agent to do so.

Information Commerce

At times an information consumer may want to apply proprietary algorithms from one company to proprietary data from another company. To do so, the consumer will want 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

Several efforts are under way to develop systems, protocols, and frameworks for both the construction and use of mobile agents. Most of the frameworks included here allow agents to be started, stopped, and moved, and a few allow them to be monitored. The following frameworks use agents programmed in Java:

- Odyssey* (ODY) from General Magic
- Concordia* (CON) from Mitsubishi
- Aglets* from IBM

Other frameworks use different languages and processes in the construction of their agents:

- Agent Tcl* (ATC) from Dartmouth uses transportable agents programmed in Tcl.
- Agents for Remote Action* (ARA), from the University of Kaiserslautern, uses agents programmed in Tcl, C/C++, or Java.
- Mobile Objects and Agents,* (MOA) from The Open Group, uses OS process migration technology.
- TKQML, from the University of Maryland, Baltimore County, uses migrating agents programmed in Tcl and communicates in KQML.

The Object Management Group is working 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.*

URLs FOR THIS COLUMN

Agent Support for Tcl • www.cs.umbc.edu/agents/kqml/papers/tkqml.ps
Aglets • www.trl.ibm.co.jp/aglets
ARA • www.uni-kl.de/AG-Nehmer/Ara/
ATC • www.cs.dartmouth.edu/~agent/
CON • www.meitca.com/HSL/Projects/Concordia/Mobile
Graham Glass, Object Space • gglass@objectspace.com
MAF • www.genmagic.com/
MOA • www.opengroup.org/RI/java/moa/
ODY V. 1.0 Beta release • www.genmagic.com/agents.html