Agents: A Triptychon of Problems

A short note for the ECOOP ‘95 Workshop
“Objects and Agents: Love at First Sight or a Shotgun-Wedding?”

Joachim Baumann
Joachim.Baumann@informatik.uni-stuttgart.de
Department for Distributed Systems
IPVR (Institute for Parallel and Distributed Computer Systems)
Breitwiesenstraße 20-22
D-70565 Stuttgart

1 Introduction

In this note I’d like to concentrate on three problems on different levels. The first is the conceptual problem of security in agent systems. The second, well, let’s call it marketing problem, is the speed of the system, and where it is important. The third, following from it, is an implementation problem: how can the transfer size of agents be minimized?

2 Agent related security

Security in agent systems consists of two parts, the security of the system against a possibly malicious and dangerous agent (something akin to the “classical” worm), and the security of an agent against a system that tries to spy it out, to steal certified information or an equivalent of electronic cash, or to change the agent’s code to something that works on the behalf of this system, but bills the agent’s original user.

System security is not that much of a conceptual problem, but a design decision. With enough effort, interpreters checking each statement to be executed, either statically and in advance, as Java [Java95a] does, or at runtime, as it is done e.g. in Safe-Tcl [Wayn95], authentication to ensure the correctness of the origin and owner given, encryption and/or signatures for communications, a system can be made effectively secure for all normal purposes. The quality of the security reached can be seen as proportional to the effort made.

A totally different (and much more complex) problem is the security of an agent against the system or other agents. While other agents can be held in check by using relatively simple measures (e.g. different address spaces), to guarantee that the system on which the agent is to be executed doesn’t try to tamper with it is a real problem, which has to be solved.

3 Speed

The first question here is, which parts of an agent system are time critical (at least from the viewpoint of a potential user)?

Normally, it isn’t very important, if an agent needs a second more or less to do its work. It works asynchronously, and simply comes back with the results (or sends them back) when finished. So it execution time won’t be a problem, which means an interpreted language could be used as an agent language. This brings, indeed, many advantages concerning security problems (for the underlying system). But if many agents, written in possibly different languages, have to be interpreted by different interpreters, communicate with one another over interpreter borders, then the slowdown could be considerable. We have, on one hand, these interpreted languages, slow but secure, and on the other hand, compiled languages, fast but insecure. We believe, that the solution lies in between, in the use of a virtual machine, to which all of the languages compile. This is definitely possible for Smalltalk (used in some dialects), in Prolog (e.g. the Warren Abstract Machine), Python, and Java [Java95b]. Imagine, being able to write programs in any of those languages (with comparable extensions for communication and migration), compiling them into the same Virtual Machine code, and executing them in the same environment. The speed increase would be definitely worth the additional work.

A crucial point is the duration (i.e. cost) of the migration of an agent. One major argument for the mobility of agents is to replace the cost-intensive networkwide communication of Client-/ Server connections by local communications. But this argument holds only true, if the time needed to transfer the agent doesn't exceed the time needed for the communication.

This divides our stock of agents into two classes: one, relatively small, roaming the net and collecting information for the other class, comparatively big, seldom migrating, with a lot of knowledge and decision-making powers. What follows is that it is necessary to optimize the migration of the first class, the small agents, and here the “local state information” (stack, local variables, PC etc.) transmitted can be a considerable part of the overall information (object classes, instance information). Which leads us to the next problem.

4 Minimizing the transfer size of agents

Principally, there exist enough algorithms for object migration, be it autonomous, or by the underlying system, for e.g. load balancing purposes in the field of distributed systems. But they have at least to be adapted to the different situation found in agent systems. But how much of the state information is really needed at the target end? It seems, that a migration step is a major break in the life of an agent. With a relatively high probability, the agent will do something different at the new place. An example (the flower buying agent again):

- Migrate to place that houses the first flower shop
- Find out how to connect to flower shop (local information service)
- Connect to flower shop and ask for offers
- Accept offer or migrate to next place
- Find out how to connect to flower shop ...

So the first thing this agent does at every site he migrates to, has nothing to do with its last action at the place before. This suggests, that most of the “local state information” will be useless at the target site (in most of the cases, anyway). Why transport it then? The solution is, not to transfer it, but have an entry point which is statically defined. Are there any limitations to this approach?

5 References

[Java95a] The Java Language Specification. http://java.sun.com/documentation.html, 4 1995.

[Java95b] The Java Virtual Machine. Specification. http://java.sun.com/documentation.html, 4 1995.

[Wayn95] Peter Wayner. *Agents Unleashed: A Public Domain Look at Agent Technology*. AP Professional, 1300 Boylston St., Chestnut Hill, MA 02167, 1995.