COMPUTATIONAL MEDIA FOR MOBILE AGENTS

FOR MAS737: SOFTWARE AGENTS SEMINAR

Nelson Minar <nelson@media.mit.edu>

http://nelson.www.media.mit.edu/people/nelson/

December 31, 1996

Abstract

Exploration of issues of using mobile agents to access computational resources distributed on the Internet. The design of mobile agent systems in which no central controller orchestrates computations is considered with an emphasis on practical deployment and safety. Java and aglets are examined as potential technologies for enabling mobile agent environments; Java's capabilities are analyzed and briefly compared with other mobile agent systems.

1 Computational Resources Distributed around the Internet

The growth of the Internet has made having real-time access to computers all over the world commonplace. A considerable amount of research has been put into making it possible to use the resources of computers on networks, but the actual deployment of these research ideas is lagging. Computer resources come in many forms, from the raw material of CPU time, memory, disk storage and bandwidth to value-added capabilities like specialized databases and services. For the purposes of this paper, "distributed computation" will mean the use of any of these resources on other machines on the Internet.

File servers and batch job scheduling systems allow some distributed computation. These technologies are typically limited to local domains of machines that entirely trust each other. With the advent of the World Wide Web access to resources on a global scale has finally become convenient, in particular access to data and the occasional specialized service. But these capabilities are limited to a small set of pre-considered applications; open ended access is largely unavailable. There is a vast potential out there for greater use of underutilized computer resources, especially CPU time, but this potential is being stymied by the difficulty of performing operations with remote computers.

What is needed is a new computational medium, a network-based environment in which processing requirements can effectively be expressed and applied across networks of computers. Mobile agents are a compelling paradigm for global distributed computing. Mobile agents allow a large degree of flexibility in creating computations; they also are natural for doing distributed processing. An important aspect of the Internet-wide computational medium is that it will have no central designer, no benevolent monarch who controls and ordains the action of every aspect of the system. The computer network will be full of agents with conflicting goals, servers that cannot trust agents, agents that cannot trust servers. In the absence of central control the networked computer environment will need to be carefully designed to be robust in the face of adversity.

1.1 Economic Control

Economic incentives can be a means to motivate and control distributed computations. The motivational aspect is simple: if people can make money by selling access to their resources, then they will have an incentive to make those resources available. The Internet today is largely based on a gift economy; whether that economy is stable and whether it is a good idea to perturb it by introducing resources-for-sale are questions outside the scope of this paper.

Without a central coordinator how can a computer network be made to function effectively? There is a growing literature that suggests that economic incentive can be useful in controlling and optimizing complex distributed systems [DM88] [Cle96]. In the particular realm of efficient allocation of CPU time, the Enterprise [MFGH88] and Challenger [CMM97] systems make compelling cases for using a market based approach to distribute batch-level computations. Agoric computation [MD88] suggests a more radical vision of distributed computation, one in which individual parts of one task can be distributed across multiple machines with the exchange of money controlling and optimizing the computation.

Economic exchange should be considered as a basis for the design of a system to support mobile agent based distributed computation. If agents (or their human owners) are charged for their resource consumption then the resulting economy could provide compensation for the service providers and encourage agents to remain efficient in their use of resources.

1.2 Existing Distributed Computation

There are several existing examples of global use of distributed computer resources. CPU time from around the net has successfully been used to perform large computations, particularly in the specific domain of number theory and cryptosystems. The factoring of RSA-129 was carried out almost entirely by volunteers on the Internet who processed parts of the work and emailed back results [AGLL94]. RSA-130 was also factored in part by contributions that came in via Web submissions [RSA96]. The largest currently known prime number was found via a Web-based Mersenne prime checker [Mer96], and an effort is underway to use CPU time from all over the Internet to finally demonstrate the insecurity of the small key size of the DES cryptosystem [Tre96]. All of these computations are simple in that the tasks are easily distributable and only data needs to be transmitted during the computation, not code. Furthermore, deployment is relatively easy as the software is special-purpose and installed by volunteers personally motivated to see the computation succeed.

Web search engines provide other examples of distributed resource usage on the Internet. Anyone can make use of AltaVista's service by sending their server a query; a search uses AltaVista's CPU time, memory, and specialized database. Second-level search engines such as the Metacrawler [SLE96] are closer to a mobile agent paradigm. When Metacrawler receives a search query it takes that request, sends it off to other servers, collects the replies and returns them. Metacrawler essentially parasitizes the resources of other search engines to provide its services.

The current uses of resources over the net are limited. Distributed efforts such as the RSA factoring do not transport programs; volunteers set up specialized compute servers and then data is shipped to perform the computation. The computations allowed by Web search engines are restricted to search expressions. While these examples fall short of full mobile agent systems, they point to the potential of access to the resources of the entire Internet.

1.3 Mobile Agents as Distributed Computation

Mobile software agents are programs that live in computer networks, carrying on their computation and moving from host to host as necessary to fulfill their goals. Software agents in general have gained acceptance as a way to perform computations. Mobility for software agents is more controversial, in large part because of the complexity and security risks mobility implies. But there are many different reasons mobile software agents can be useful, enough that the approach has merit [HCK95].

Mobile agents provide a fresh, compelling way to organize the use of distributed resources on the Internet [CL95]. Software agents are a sort of "super-object-oriented" paradigm. Traditional object-oriented programming collect data and code in one (typically small) object. Software agents wrap up a large amount of data and code into one package; in this sense, a software agent is like a large object. Agents not only encapsulate state and rules; they also have purpose and autonomy. An agent is a way to encapsulate a computational task. A mobile agent can be sent over the network to use needed resources on a remote host. The agent metaphor makes it easier to understand the organization of distributed computations.

As an example of a distributed computation performed by agents, consider the construction of a computer animation where several different resources distributed across the Internet are needed. An agent could be programmed to first visit the character store at Disney to purchase model data. The agent then takes the data to a render farm where it would spawn many agents on the rendering machines to create the frames. Finally, the agent collects all the frames and takes them over to a host with specialized MPEG creation capability to package the final movie. By describing this computation in terms of mobile agent intentionality it is easy to understand how the various resources are coordinated to perform the task.

2 Requirements for Mobile Agent Systems

While mobile agents are a useful approach to distributed computation, in practice they have seldom been used. There are many technical challenges to implementing mobile agent systems. Most of these problems are in the structure of the computational medium, the environment the agents operate in. Servers must be designed, implemented, and deployed that not only allow mobile agents to run, but allow them to run safely.

2.1 Portability

Mobile agent code itself must be portable; when an agent arrives at a server the server needs to be able to execute that agent. Commonly used computer languages such as C and C++ are not very portable. Compiled C code only works on the machine it was compiled for and the source form is notoriously unportable. Portability can be achieved by running computer programs inside virtual machines interpreters, but overhead has limited the use of interpreted languages. Most mobile agent systems under development now rely at least in part on virtual machines to standardize the execution environment.

2.2 Ubiquity

In order for mobile agents to be successful they need access to many different computer resources. Servers for agents must be commonplace; there needs to be a widely accepted framework for executing mobile agents deployed on many machines across the Internet. In practice the requirement of ubiquity means that the execution environment needs to have market acceptability, be freely available, and be unencumbered by restrictive intellectual property requirements.

2.3 Network Communication

Mobile agents that live in the network need to be written in a language that makes network access simple. It must be easy to transfer objects across the network and to invoke methods of remote objects. Traditional computer languages treat networking structures as an afterthought, usually providing only a minimal socket library. Languages that better support network access have typically not been widely used. This situation is improving with the current development of language-neutral distributed object frameworks such as CORBA and OLE.

2.4 Server Security

A major concern specific to mobile agents is the protection of the servers running the agents. Running arbitrary programs on a machine is dangerous: a hostile program could destroy the hard drive, steal data, or do all sorts of other undesirable things. This risk must be thoroughly addressed if mobile agent environments are to succeed. Two types of security are possible to protect servers from malfunctioning and hostile agents: physical and social.

Physical security refers to building servers for agents in such a way that the agents cannot harm the server. The "laws of physics" of the server execution environment can be designed to make dangerous operations difficult or impossible. Common approaches involve creating a "sandbox" for visiting agents, restricting access to resources (preventing disk writes, for instance) and ensuring the agent cannot escape those restrictions. This approach to security is attractive; when it works, it is entirely effective. But the viability of physical security in the face of design complexity and server implementation bugs is unclear. In addition, physical security is typically focussed on protecting some underlying aspect of the server from the sandbox the agent is trapped in. But if multiple agents are put in the same sandbox how can the server guarantee that one agent cannot harm another? As we put more trust in the computations that take place inside sandboxes, the security of those sandboxes themselves becomes important.

A second approach to server security is using social enforcement mechanisms to punish the creators of harmful agents. If a server administrator can find out who is responsible for a malicious agent, then that person can be held accountable via social mechanisms (such as lawsuits). Digital signature technology makes identifying the authors of agents possible. But there are limitations to a purely social approach to security. It may not be clear which agent is responsible for damage, nor will it be easy to determine ahead of time which agent authors are trustable. In practice some combination of social and physical enforcement of server security will be useful.

2.5 Agent Security

The complement of server security is agent security: whether the agent can trust the server on which it is executing. A mobile agent might contain secret information such as proprietary data and algorithms. Worse, servers might have an incentive to subvert the computation of a visiting agent. In the Internet-based DES cracking effort currently under design a major concern is protecting the computation from sites that pretend to do pieces of the problem but return false answers [Tre96]. Physical security answers to this problem are difficult. Secure, trusted hardware on the server could guarantee agent safety but is unlikely to be widely deployed. Agent programmers can protect their agents by obfuscating their code and verifying the results of the remotely-performed computation but the general applicability of these techniques are unknown. Social solutions may be possible in the form of reputation systems for servers. This area of security has largely been unexamined.

2.6 Resource Accounting

If economic control and incentive are going to be factors in net-wide resource use some mechanism to account for the resources that an agent uses and a way for receiving payment for those resources is necessary. In theory these requirements are not difficult to meet. Servers can keep track of the resource usage of agents, explicitly accounting CPU, memory, bandwidth and disk usage. Digital cash systems can be used to pay for services. In practice, these technologies are not widely deployed and the overhead they impose presents an engineering challenge.

2.7 Conclusions

Creating an effective infrastructure for distributed computation is not easy. Mobile agents are a viable paradigm for thinking about access to computational resources but in order for mobile agents to succeed many technologies need to be put into place. The shape of most of these pieces is fairly clear; however, the engineering and social barriers to their deployment are formidable. With the growth of the Internet there has been much interest in developing the necessary pieces.

3 Java as Technology Base for Mobile Agents

In order for mobile agents to be prevalent a suitable technology base needs to be installed on a large number of machines on the net. Selecting the technology is not a small task: it needs to fulfill all the characteristics discussed above and it needs to be real, deployable. No system perfectly meets all the requirements, but the Java system from Sun Microsystems [AG96] is the best available system today. Java is a combination language and operating system that has working implementations and a large degree of market acceptance. At the time this paper was written (December 18 1996) Java 1.0 is delivered in final form and Java 1.1 has just entered beta.

3.1 Java Features

3.1.1 Portability

Java takes a virtual machine approach to portability. Java programs are shipped as bytecode; the bytecodes are a simple stack language that is interpreted by a virtual machine on the compute server. Some of the overhead of a Java interpreter can be reduced by “just in time” compiling: having the Java machine to compile the bytecodes down to native object code for execution. Even with a portable bytecode representation the exact semantics of every possible Java instruction needs to be specified in order for Java programs to be portable between different implementations. In practice, Java code has not been completely portable. Small tweaks to programs are usually required when moving between Java platforms. The incompatibilities are largely because of differences in the correctness and completeness of different virtual machine implementations.

3.1.2 Ubiquity

Java has enjoyed an enormous amount of market acceptance in a very short amount of time. A large part of Java’s quick spread has been because of aggressive marketing. Sun swiftly delivered a Java implementation to Netscape, thereby giving the average Internet user access to Java and creating a cultural phenomenon. More importantly, Sun has been mostly open about the Java technology, publishing specifications and APIs and giving away development tools. The Java design is unhampered by restrictions on distribution; free Java implementations are even underway [Jol96].

The actual use of Java so far has largely been in the very limited environment of applets for Web browsers. Applet environments are unsuitable for distributed computation; most browsers do not stay running very long and the applet security model does not allow enough resource access. There are a few Web pages that use embedded applets to distribute computation out to browsers but these are largely proofs of concept. Java must be deployed outside the browser in order to effectively create a computational medium.

3.1.3 Network Communication

One of Java’s main strengths is that it gives simple access to Internet communication. Source code for software can be transparently downloaded from anywhere on the Internet. Custom socket level transmission of byte streams is easy with existing Java class libraries. However, Java 1.0 does not have any standard facility for sending objects over the net, nor is there a standard way to invoke a method on a remote object. Ad hoc solutions are currently possible, but Java 1.1 adds two new standard facilities to make distributed objects easy. Object serialization [Jav96b] allows objects to be sent via byte streams for network transmission. Remote method invocation [Jav96a] allows object methods to be called over the network. These packages are already available in the Java 1.1 beta release.

3.1.4 Server Security

Java 1.0 takes an entirely physical-security approach to protecting machines from hostile programs [FM96]. A special object, a security manager, defines which resources a Java program is allowed to access. In the familiar case of Java applets the applet security manager creates a sandbox that prevents access to many resources including writing on the local hard drive and network communication to anywhere other than the host the applet came from. The Java virtual machine also contains a bytecode verifier that does static checks to prevent forbidden code sequences from being loaded, thereby ensuring the unbreachability of the sandbox surrounding the incoming code.

Java 1.0 does not have any implemented mechanisms for social enforcement of security policy. There are no guarantees about where a given piece of Java code comes from. Java 1.1 explicitly addresses this limitation with a Security API that includes digital signatures for Java code.

Currently used Java security managers are fairly inflexible: in practice, Java programs either have full access or run under strict applet restrictions. However, the security manager is a replaceable module. With the addition of class signatures it will be possible to selectively give access to certain resources depending on how much the server trusts the particular agent it is executing. A third party implementation is already available in E, a Java extension [E96]. Java 1.1 includes access control lists [Jav96c] that could form the basis of a system to define specific security policy for agents depending on trust level.

3.1.5 Agent Security

Java does not address the issue of security guarantees for agent code running on a Java virtual machine. Secure hardware implementations of Java should be possible but are unlikely to be commonly used. Several obfuscation programs exist to hide the structure of transported agents, but obfuscation does not stop determined attackers. It is not clear that it is possible to protect agents from servers that deliberately misexecute their code in any system.

3.1.6 Resource Accounting

Java also does not address the question of resource accounting. There do not appear to be any implementations of Java virtual machines that track the resources used by programs. Counting CPU, bandwidth and memory usage in a Java interpreter should be fairly simple, although the problem becomes more difficult in compiled code. The thread scheduler is probably the right place to add in CPU accounting, the garbage collector can determine memory usage, and the socket classes can track bandwidth usage. Implementing these systems efficiently will be challenging.

Java also currently has no explicit support for digital cash payments. The cryptographic basics of these systems are possible in any language; preliminary Java implementations do exist. A secondary issue is the actual distribution of digital cash: is there some easy (and secure) way to give an agent digital money to carry along with it as it executes? Efficiency of handling payments will also be a concern.

3.2 Existing Java Mobile Agent Platforms

Several systems exist today to implement mobile agent facilities in Java. One limited application is found in "servlets," Java code that runs on Web servers (analogous to applets running on a browser). JavaSoft's Jeeves [Jee96] is the most active servlet platform under development now. Jeeves is an HTTP server written in Java; it can be extended by loading in new Java classes from either the local disk or the network. One possible application is to allow clients to upload a servlet into a Jeeves server to perform computation on the server. Servlets run in their own restricted environment but are allowed to open a network socket back to their home to perform arbitrary communication. Jeeves is essentially a simple mobile agent system but is restricted in that servlets are intended to run in only limited ways inside the HTTP server. Servlets are not commonly expected to travel multiple times.

Several preliminary efforts are under way to create full mobile agents in Java. Implemented packages include Mole from the University of Stuttgart [Mol96] and Java2Go from Berkeley [LM96]. The SAIC also has published an API for mobile agents [SAI96]. All of these systems are based on Java 1.0: they use some sort of ad hoc implementation of object serialization and remote method invocation to create mobile agents. These systems are of interest in their different design approaches but probably will not be contenders for serious world-wide deployment.

Finally, a promising mobile agents project called "aglets" is under development at IBM [LC96]. Aglets relies on two basic specifications: an API for aglets (J-AAPI) and the Agent Transport Protocol (ATP). IBM has explicitly stated their intention to make aglets ubiquitous. ATP and J-AAPI have been put forward as standards. The specifications and implementations are available for free (with a restricted prerelease evaluation license).

Aglets implement mobile agents in a straightforward way. Resource providers run servers that accept agents; anyone can upload an aglet into a server. Java and the J-AAPI gives aglets portability. Aglets rely on the object serialization and remote method of invocation of Java 1.1 for effective network communication. Server security is handled by special aglet security managers. The security system is configurable and should be flexible enough to allow a variety of options, including policies based on the upcoming code signature specification. Aglets do not address agent security or resource accounting. A usable alpha version of aglets is currently being shipped; however, it requires Java 1.1 which is itself only available in beta for a few operating systems. As the software becomes more mature aglets looks to be the most likely platform for future mobile agent research.

4 Other Technologies for Mobile Agents

4.1 Telescript

Java is not unique in supporting mobile agents. General Magic's Telescript [Tel96] as one of the first mobile agent platforms. Many of the more complicated issues of mobile agents have been carefully considered in the design of Telescript. The Telescript language handles portability and network communication. Server security is implemented both physically (restricted execution environments) and socially (signed code). Telescript contains an explicit model of resource consumption, at least for CPU time. However, the crucial limitation of Telescript is its lack of deployment. General Magic has chosen a proprietary approach, giving out very few specifications and making access to their implementations difficult. Even if Telescript is as good as their papers indicate, it will not matter if no one ever uses it.

4.2 Inferno

Lucent Technologies has recently entered the distributed computation arena with Inferno, a network operating system based on Plan 9 [Inf96]. Beta versions of Inferno are now shipping for several operating systems under a restricted test license. Limbo, the language developed for Inferno, resembles a version of C augmented with abstract data types. Limbo itself seems unnecessarily limited in not supporting inheritance. However, the Inferno system is capable of running other languages; a version of Java in Inferno is underway. Inferno presents itself foremost as an operating system, thereby directly addressing questions of security. The underlying abstractions of Plan 9 should nicely support sophisticated resource access control for individual agents. Running Java on top of Inferno could combine the security features of Inferno with the acceptance of the Java programming language; whether such a union is necessary or desirable remains to be seen. In general, Inferno is too new to know what impact it will have on mobile agents.

4.3 ActiveX

Microsoft's ActiveX could also be useful technology for building mobile agents [Act96]. ActiveX is essentially a version of the OLE distributed object system with some enhancements to allow "controls" (ActiveX objects) loaded over the Internet. The major limitation of ActiveX is that it contains absolutely no physical security provisions: ActiveX controls are pure object code with no restrictions. Microsoft has opted to rely entirely on social enforcement to protect computers from malicious software. Furthermore, because ActiveX controls are object code they are not portable. While Microsoft has promised ActiveX support for the Macintosh, portability between Windows and Macintosh controls (or to Unix systems) seems unlikely. ActiveX has the virtue of running today and being reasonably open. It should be considered seriously because of the market power of Microsoft and the relative ease of developing ActiveX controls. But ActiveX lacks several of the necessary components to be an effective mobile agent environment.

5 Next Steps for Advancing Mobile Agents

Implementing mobile agents is possible with available technology; implementations will help researchers work on the many open questions of mobile agents. An obvious first concern is determining exactly what applications mobile agents are good for, what agents will be used for once the medium necessary to support them exists. A simple, useful application would be to provide customized access to online databases. For example, a Web search engine could be augmented by allowing users to send over an agent to perform more complex searches. It is likely that a system such as this will soon be developed, probably under the name of servlets.

In anticipation of a wider deployment of agents it is important to study the emergent behavior of mobile in complex networks of resource servers. Towards this end, it would be useful to have an actual distributed computer server based on mobile agents. The batch-level approach of the Enterprise system should be simple to implement in Java with aglets; the major open issue is resource accounting. In addition to being a first step towards mobile agents research, an Enterprise-like network could have practical use as a distributed CPU server. With the coarse grained resource distribution of Enterprise in place explorations could then be made towards the finer grained distribution of agoric computation systems.

While Java and aglets fulfill many of the needed technologies for mobile agents, several pieces are missing. Java 1.1 adds code signatures, the fundamental mechanism needed to establish trust level, but the details of controlling the resources an agent has access to will not be fully implemented. A good design for an access control system would be of general benefit in providing configurable sandboxes. The larger question of the longterm viability of sandbox security needs further exploration. Do sandboxes only protect the underlying computer from dangerous agents or is there some way to enforce security at a more detailed level so individual agents are protected from each other?

The question of how to guarantee agent safety from malicious servers is unanswered. A complete solution will very likely require rethinking the way we perform computation. One partial approach might be in inventing a way to do computations so that pieces of the work are independently verifiable for less cost than performing the computation itself. Reputation systems for servers could also be developed to allow some social enforcement of agent security.

Finally, issues of resource accounting have not been well addressed in mobile agent work to date. Resource accounting is not conceptually difficult but efficient implementations take work. Digital cash systems need to be created that allow agents to easily pay for the resources they use. And the merit of using economic systems needs much further study; the effectiveness of markets to control distributed computations will only be determined after much experimentation.

Mobile agents are a useful paradigm for enabling the use of distributed computational resources. A proper implementation of a network computational medium involves many complicated technical and social issues, but the solutions to many of these problems are at hand. It is time to implement and experiment with mobile agent systems; Java and aglets look to be the best technologies for doing that today.

References

[Act96] ActiveX home page. http://www.microsoft.com/intdev/, December 1996.

[AG96] Ken Arnold and James Gosling. *The Java Programming Language*. Addison-Wesley, 1996. ISBN: 0–201–63455–4.

[AGLL94] Derek Atkins, Michael Graff, Arjen Lenstra, and Paul Leyland. The magic words are squeamish ossifrage. In *AsiaCrypt '94*, 1994.

[CL95] Daniel Chang and Danny Lange. Mobile agents: A new paradigm for distributed object computing on the WWW. http://www.trl.ibm.co.jp/aglets/ma.html, December 1995. Submitted to OOPSLA '96.

[Cle96] Scott H. Clearwater. *Market-Based Control, A Paradigm for Distributed Resource Allocation*. World Scientific Publishing Co., 1996. ISBN: 9810222548.

[CMM97] Anthony Chavez, Alexandros Moukas, and Pattie Maes. Challenger: A multi-agent system for distributed resource allocation. In *Proceedings of the International Conference on Autonomous Agents*. ACM, 1997. Also available from http://moux.www.media.mit.edu/people/moux/papers/challenger.ps.

[DM88] K. Eric Drexler and Mark S. Miller. Incentive engineering for computational resource management. In B. A. Huberman, editor, *The Ecology of Computation*, pages 231–266. Elsevier Science Publishers, 1988. Also available from http://www.webcom.com/~agorics/agorpapers.html.

[E96] E: Extensions to Java. http://www.communities.com/e/index.html, September 1996.

[Fla96] David Flanagan. *Java in a Nutshell*. O'Reilly & Associates, 1996. ISBN: 1–56592–183–6.

[FM96] J. Steven Fritzinger and Marianne Mueller. Java security. Technical report, Sun Microsystems, Inc., 1996. http://java.sun.com/security/whitepaper.ps.

[HCK95] Colin G. Harrison, David M. Chess, and Aaron Kershenbaum. Mobile agents: Are they a good idea? Technical report, IBM T.J. Watson, 1995. http://www.research.ibm.com/massive/mobag.ps.

[Inf96] Inferno: la Commedia Interattiva. http://inferno.lucent.com/inferno/infernosum.html, December 1996.

[Jav96a] RMI — Remote Method Invocation. http://java.sun.com/products/JDK/1.1/docs/guide/rmi/, December 1996.

[Jav96b] Object serialization. http://java.sun.com/products/JDK/1.1/docs/guide/serialization/, December 1996.

[Jav96c] Security in JDK 1.1.
http://java.sun.com/products/JDK/1.1/docs/guide/security/, December 1996.

[Jee96] Jeeves home page. http://www.javasoft.com/products/java-server/, December 1996.

[Jol96] Java open language toolkit. http://www.redhat.com/linux-info/jolt/, December 1996.

[LC96] Danny B. Lange and Daniel T. Chang. IBM Aglets workbench: Programming mobile agents in Java. http://www.trl.ibm.co.jp/aglets/whitepaper.htm, September 1996.

[LM96] Weiyi Li and David G. Messerschmitt. Java-To-Go.
http://ptolemy.eecs.berkeley.edu/dgm/javatools/java-to-go/, December 1996.

[MD88] Mark S. Miller and K. Eric Drexler. Markets and computation: Agoric open systems. In B. A. Huberman, editor, *The Ecology of Computation*, pages 133–176. Elsevier Science Publishers, 1988. Also available from
http://www.webcom.com/˜agorics/agorpapers.html.

[Mer96] The great Internet Mersenne prime search.
http://ourworld.compuserve.com/homepages/justforfun/prime.htm, December 1996.

[MFGH88] Thomas W. Malone, Richard E. Fikes, Kenneth R. Grant, and Michael T. Howard. Enterprise: A market-like task scheduler for distributed computing environments. In B. A. Huberman, editor, *The Ecology of Computation*, pages 177–206. Elsevier Science Publishers, 1988.

[Mol96] Project Mole — mobile agents.
http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole.html, March 1996.

[RSA96] RSA factoring by Web. http://www.npac.syr.edu/factoring/, December 1996.

[SAI96] Autonomous agent programming using Java. http://riz.saic.com/AIT/agentref.html, December 1996.

[SLE96] Erik Selber, Greg Lauckhart, and Oren Etzioni. Metacrawler.
http://metacrawler.cs.washington.edu:8080/, December 1996.

[Tel96] Telescript technology: The foundation for the electronic marketplace.
http://www.genmagic.com/Telescript/Whitepapers/wp1/whitepaper-1.html, 1996.

[Tre96] Peter Trei. Communications on coderpunks@toad.com,
news://nntp.hks.net/hks.lists.coderpunks, December 1996.

[Way95] Peter Wayner. *Agents Unleashed, A Public Domain Look at Agent Technology*. Academic Press, 1995. ISBN: 0–12–738765–X.