Technical Report TR-VS-97-01
An Infrastructure for Web-Agent-based Service Providing

Stefan Fünfrocken

Department of Computer Science, Darmstadt University of Technology,
Alexanderstr. 10, D 64283 Darmstadt, Germany
Email: fuenf@informatik.th-darmstadt.de

Abstract. Mobile agents are a new paradigm in distributed computing. In this paper we describe a platform, which supports Web agents — mobile agents that live in web servers and communicate with users through web browsers. Our belief is that this World Wide Web scenario identifies an application area for which the mobile agent abstraction which is well-suited. We present the state of implementation, and we give an outlook on our future work.

Keywords: mobile agents, service providing, World Wide Web, security, infrastructure, mobile agents application

1 Introduction

'Mobile Agents' are programs that can move from computer to computer to fulfill a task on behalf a user. To overcome the problem of heterogeneity, mobile agents are mostly programmed in an interpreted language for which an interpreter is available for a wide range of computer systems. Using an interpreted language also solves, to a certain extent, one of the most important problems for mobile agent systems: security. Since the interpreter which executes the mobile agent is local to a computer system, it can be modified to intercept all 'dangerous' commands of the interpreted language which interact with system resources. In this way, foreign agents interact with the local system through a trusted third party. Currently there is only one language which was designed for use as an agent language: Telescript by General Magic [Whi94]. There are several interpreted languages which offer a so-called 'secure interpreter', like safe-tcl [Bor93] or safe-python [Maj96]. Most of them, however, lack agent-related language constructs and concepts (e.g. a language-provided command to initiate agent transportation, or security concepts) and depend on external libraries for such functionality.

When the mobile agent concept first was promoted by General Magic [Whi94] with its Telescript Language, there was much hype about the new paradigm. Since General Magic had a very restrictive policy about implementation information of their system, several research groups began to build their own agent systems providing a general infrastructure for mobile-agent-enhanced applications. Although the paradigm seems to be very appealing at first glance, after a few years of research there is still no area of application for mobile agents which perfectly fits to the 'mobile agents' paradigm. This, together with security concerns (most systems implemented only weak security features in their first releases), is probably the reason why the agent paradigm hasn't met a wide acceptance so far. Without any reasonable dissemination of mobile agent systems, mobile agent technology will not have the chance of proving its power.

On the other hand, the World Wide Web is still growing at an exponential rate [Net96], and buzzwords like 'web centric computing' or 'Intranet' promote traditional Internet technology everywhere. There we have a widespread, well-accepted architecture, to which more and more existing, traditional data processing systems and applications (e.g. databases, newspapers, financial portfolio applications) are adapted and integrated.

To combine mobile-agent-technology with the infrastructure the World Wide Web offers, we developed an architecture which supports so-called Web agents.

2 The WASP Project

The infrastructure we present was developed as part of the WASP project. With the '**W**eb **A**gent-based **S**ervice **P**roviding (WASP)' project, we aim at providing services on Web data and using mobile agents to implement these services. The underlying hypothesis is that services for the World Wide Web is one application domain for which the mobile agent paradigm is a well-suited model.

Possible services on Web data include services which are as simple as link checking for a given web page, or as complex as for example a distributed work group service which maintains a set of web pages that are distributed between the different locations of the work group. In addition to managing different versions of a web page, the agent providing such a work group service collects and merges new versions of the pages (accessible to local work group members only) from the different web servers and publishes them for public access on a dedicated web server.

Both examples show that the mobile agent abstraction matches the user's expectation of such services: 'There is someone I can tell to care about our workgroup pages, and he or she will move to all the places to collect the information and bring it to the publication server.' Since the current WWW development shows that more and more existing data will be accessible through web servers we can offer services on that data, for which otherwise special solutions would be necessary. To provide services on such data, it is straightforward to integrate any service-supporting platform into web servers. With this in mind, the Web agent abstraction (i.e., mobile agents that live in web servers) is most apporpriate.

As an implementation language for our system and for agent-programming, we chose Sun's web language Java [Arn96], which already provides code shipping by the way of applets and persistent, movable objects by the means of object serialization. Because of this, we found that it is very easy to implement a system that provides rudimentary mobile agent functionality. This is also supported by the fact that there are several other research projects that deal with Java-based mobile agent systems like Mole [Hoh95], Aglets [Lan96], Java-to-go [Wei96], and MOA [Mil96]. To gain insight in Java's potential as an agent programming language, and because of announced Java API packages (i.e., security and electronic commerce) that offer valuable functionality in a standardized way, we decided to build a system of our own this also allows us to customize our environment to our special needs.

One major goal of the WASP project is to offer means to support web service *providing*. This support includes on the one hand an infrastructure for Web agents which are used to implement the services, and on the other hand tools to help a service provider to generate and manage those agents.

The remaining part of this paper will focus on the infrastructure we developed for Web agents in our project.

3 WASP Infrastructure

The necessary infrastructure for the proposed web services should enable web servers to start, receive, and execute Web agents. Additionally, web agents may access the Web data of the server, may want to communicate with a user[1], and have to be managed. Starting, executing, accessing Web data, and management of agents is done by a special extension module: the **S**erver **A**gent **E**nvironment (SAE). In this way, the code of web servers does not need to be changed. Of course this implies that the server has to provide a well-defined interface like CGI, Jeeves' serveletts [Jee96], or Jigsaw's filter interface [Bai96], which is able to pass all relevant information to the SAE[2]. Figure 1 shows the overall architecture of the WASP infrastructure which consists of the user's web browser, the web server, and the server's SAE.

Generally speaking, our scenario is divided into two phases: the agent startup and configuration phase, and the service phase. In the first phase, a user wants to access a service which is located at

[1] Communication between agents that are not belonging to the same service is yet not addressed by the project.

[2] We also assume that a server understands the http POST method.

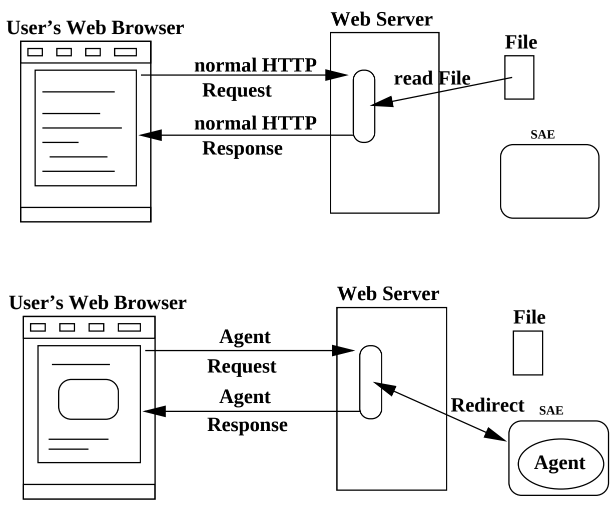

Fig. 1. General WASP Infrastructure

a web server. Since the service is provided through a web server, the user uses its local web browser to connect to the server and requests the start of the Web agent representing the service. Since all agent related actions are executed by the SAE, the server redirects the user's request to it, which then loads the agent and starts it. The first action of a Web agent is to get customized to the user's demands[3]. To do so, it sends its graphical user interface to the user's browser, which returns the required configuration-data to the agent after the user is done with his or her configuration. After that, the second phase begins: depending on the configuration, the service is executed by the Web agent. This normally requires that the Web agent accesses the local Web data of the server and may include migration to other web servers to access their Web data. After completing its task, the agent may return to the server by which it was started and possibly notify the user about results of its work[4].

3.1 Server Agent Environment

The SAE (see figure 2) serves as an execution environment for web agents and can be compared to any other mobile agent system, except that it is specialised to a certain kind of agents. Its primary task is to load Web agents either from the local disk when an agent gets started, or from the network when a Web agent migrates itself from some other SAE, and to guarantee access and security restrictions specified by the administrator of the web site. Unlike other agent environments, it provides Web agents with a uniform interface to the server's Web data (see Section 3.5) so that a Web agent doesn't have to care about the way web data is stored locally.

In addition to agent related tasks, an execution environment has to address problems like:

– configuration
– administration
– controlled startup and shutdown
– persistency in case of crash and recovery

[3] Of course there are services that don't need to interact with users. But this simply means to skip any user interaction.
[4] This scenario describes a service which requires that the Web agent returns to its starting server and presents results to the user. This is only one case of possible service scenarios. Other scenarios don't require a response to the user or even the return of the agent.

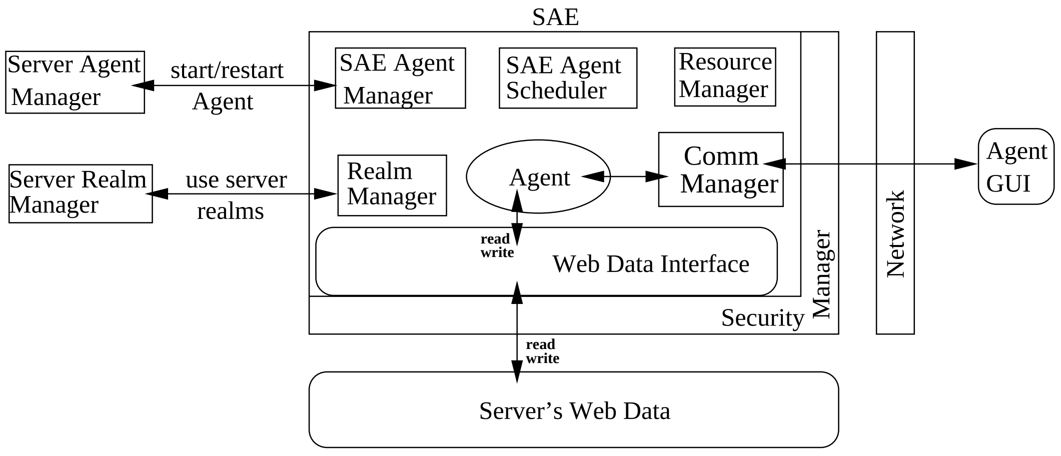

Fig. 2. SAE Architecture

which we also care about by offering a configuration and administration tool. A site administrator can use it by loading it into his web browser after identifying himself to the system[5]. Concerning persistency, Java's object serialization offers an appropriate way to implement object persistency in general.

3.2 Agent Startup

Figure 3 shows the agent startup scenario. When a user requests an agent to start by a http get request of a special URL which points into the servers SAE, the server hands this request to its SAE that identifies and loads the agent. After loading and starting, the agent sends its interface to the user's Web browser. This is done through an agent-generated HTML page, which points to the agent's GUI applet and serves as response to the user's http get request. The browser executes the Web agent's GUI applet, which requests all necessary information from the user. After the user is done, the configuration data is sent back to the Web agent.

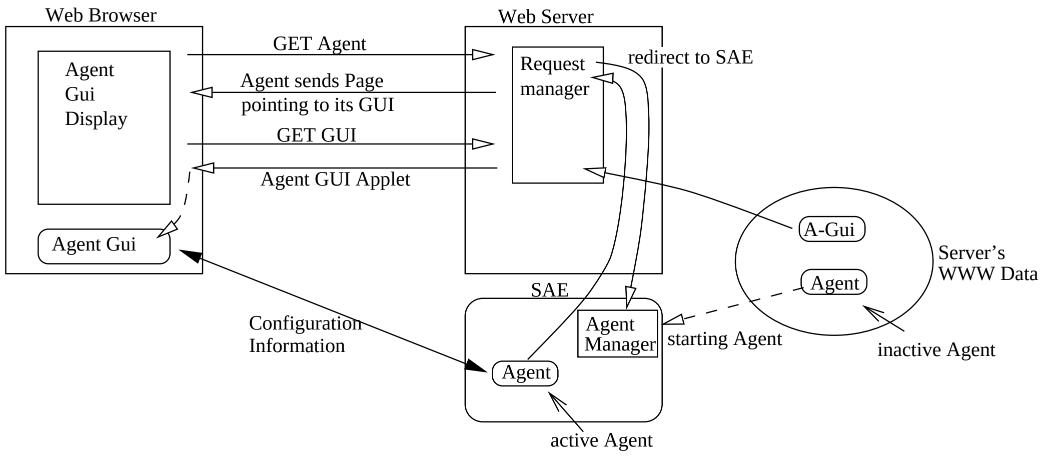

Fig. 3. Agent Startup Scenario

[5] Since this is a service on Web data, we provide this service through a special Web agent.

Technically, the communication between the agent GUI and the agent is done through the communication manager of the SAE (see Sections 2, 3.1, 3.6). After starting a web agent, the agent informs the communication manager that it awaits its GUI callback. In this way, the GUI can connect to the communication manager and request connection to its web agent to hand back the data.

By using a communication manager, it is possible that the web agent's GUI can wait for the return of its agent to pass back any result the user awaits. For this, the Web agent requests connection to its GUI from the communication manager. For services which require some time, the agent may respond to its GUI with a specific URL the user has to monitor for the results of the service.

3.3 Migration

Web agents have to conform to a certain interface in order to enable the SAE to control the agent. Since we use Java as an agent programming language, we can provide this interface by a class from which any Web agent implementation has to be subclassed. This, together with Java's possibility to declare methods final and therefore unchangeable by subclasses, allows us to fill this template with predefined functionality.

In our scenario, Web agents migrate themselves to other web servers by calling their go-method (inherited from the Web agent template) with the new destination. This method uses Java's object serialization functionality to dump the Web agent and initiates a http post request to the specified target web server. The target URL is a SAE-specific URL which the target web server identifies as a request of an agent in migration to enter the system and hands it to its SAE which handels the deserialization of the agent.

Since we use object serialization, the migration of Web agents is not transparent to the agent as for example in Telescript [Whi94] or ARA [Pei95]: the flow of control cannot directly be reestablished to that point in the code of the Web agent where it executed the go method. We consider this not to be a conceptual drawback[6], although this puts some burden on the programmer of the Web agent. As other systems that use this scheme of migration (Mole [Hoh95], Aglets [Lan96], FFM [Lin95], Tacoma [Joh95]), we provide a special method which is called by the target SAE when it restarts the flow of control for a migrated Web agent. The agent programmer has to fill this method in such a way that its agent will execute the correct code depending on its internal state.

Inside the post request, Web agents are transported as MIME [Bor93] message. If the server requires any authentication to perform the post request, the agent has to provide that information at the time it is sent (i.e., when the post request is constructed). For that purpose, the Web agent template defines (unchangeable) methods which yield the necessary security information, and provides the Web agent with means to fill in that information (see Section 3.6).

3.4 Dissemination

One concern for a web services is its dissemination. Depending on the kind of service, the service originator may decide to have its service present on a single web server, on a list of web servers (which may be predefined or dynamic, depending on server characteristics), or even on every web server the agent visits. To offer a variable dissemination scheme, our architecture provides Web agents with a special method which asks the server it is currently running on to install the agent locally. This call expresses the Web agent's wish to get installed, but does not install it automatically. Depending on the SAE configuration by the site administrator, the SAE possibly grants the Web agent's wish or simply ignores it.

[6] Obviously, every Java program using transparent migration can be transformed automatically to one using nontransparent migration and executing the original code statements in the same order and with the same variable values. Because of this, there is no conceptual difference between transparent and nontransparent migration.

We provide the site administrator with the possibility to specify classes of agents, for which can be specify how the SAE should react to an installation request. Possible reactions include: denial, allowance[7], and storing. If a Web agent is not installed, it may be stored in a place where someone has to inspect the Web agent and possibly run it in a separate environment and then decide whether to install it locally.

Using this technique yields a very flexible set of dissemination schemes which can be matched to any service requirement and respects site policies. Chaotic Web agents that repeatedly try to install themselves everywhere can be trapped by denying the post request of such an agent[8].

3.5 Web Data Interface

To provide Web agents with a uniform way to access any Web data, we aim at developing a Web data interface which hides the fact that the same type of Web data is stored and retrieved in different ways. Generally, a Web agent can read and write Web data such as a HTML page and it should not matter whether the page comes from a file, is generated by a program, or by a database[9]. Our current research in this part of the project deals with integrating database-generated Web pages into this interface. Writing to such a generated page is nontrivial because not only the database data used in the document has to be updated (which may include a schema modification) but possibly the generating program too.

Web agents will not only be able to read and modify existing data. To avoid restricting the service classes which can be implemented on top of our infrastructure, we allow agents to generate new Web data at a server. To prevent any denial of service attacks of malicious agents that write a huge amount of data, we provide our SAE with a resource control and security model (see Section 3.6). Technically, writing more data than allowed (either by resource negotiation or by server rule) will yield an exception to which the Web agent should react or will get terminated by the security manager on the next attempt to write more data.

3.6 Security

For mobile agent systems, security is one of the most important concerns. Site administrators allowing foreign agents to enter their system must be sure that the agent system prevents compromising their system. There have to be means to control agent access to the system, any resource usage by agents, and access rights to local data. Because of the importance of security, we planed our infrastructure from the beginning with a security model in mind.

Security is provided in our system by the means of 'protection domains', so-called realms. A realm consists of a set of data, specified by local URLs, to which access is restricted. Any user accessing the protected data has to identify to the system by the means of a password which, together with the user's name, is stored with the configuration data of the realm. In our system human users and Web agents are treated the same way when accessing Web data, except that for Web agents additional access rights exist to control network access or rights such as realm definitions. For every realm there is an owner who can define access rights for the realm (read, write, execute).

Data protection by realms is common for most of the existing Web servers, but we extended this concept by allowing any user and not just the server administrator, to specify realms. Our realm configuration facility allows easy set operations on existing realms to specify files and users of new realms. Furthermore we integrated the realm concept in our SAE to have a uniform way of specifying access to the server, access to the SAE, and access to Web data.

[7] An successful installation of a Web agent does not grant any access rights to local web data.

[8] Nevertheless this cannot completely prevent denial of service by overloading attacks. But this is a well-known problem in every client server environment, where a server has to accept a request before it can then react on the request origin.

[9] Java's UrlConnection class which hides different protocols from the user can be seen as a first step in this direction. However, it always includes a http request to a server. Our goal is to hide the way, Web data is stored or generated.

In addition to specify access rights, it is important to have a Web agent identifying scheme. For Web agents it is natural to have a manufacturer, who signs the Web agent with its manufacturer id[10] when releasing it. Additionally, there is an agent id which in combination with the manufacturer's id is unique. When installed on some Web server, the server can assign a server id to the agent. When started, an agent also gets assigned an incarnation id. An agent's server id together whith the incarnation id, can be used to identify Web agents that were started at a server, migrated and returned. We are currently examining which cryptographic technique and algorithms we will use to achieve this[11]. A Web agent's ids will be stored by predefined methods of the Web agent template, which will not allow a Web agent to change that data by itself[12].

When an agent wants to enter the system, it sends a http post request to the Web server it wants to migrate to. The URL of the post request points into the server's SAE. To allow only well known Web agents access to the SAE, the site administrator can configure a realm consisting of the SAE entry URL and add the allowed Web agents to that realm. In contrast to users, the password of Web agents is deduced from the Web agent's ids. In this way, the site administrator can grant access to all Web agents of a certain manufacturer by using the manufacturer id, specific Web agents of a manufacturer by using the manufacturer id and the agents id, or only to those Web agents that are installed and started locally by additionally using the agent's server and incarnation id.

Once the system is entered, access to any Web data may be restricted by realms. There is no difference between a user or a Web agent accessing local data. When an agent accesses restricted data, the Web data interface checks for the agent's presence in the list of allowed users (and agents) for that data and authenticates the agent by requesting its ids[13]. In case of a human user, he is asked for its user id and password by his Web browser on behalf of the Web server.

Every agent system offers critical system resources like CPU cycles, main memory, disk space, and network access to foreign agents. Because of this, agent systems have to take care about system resources and monitor their usage to avoid misuse. Because there is no way to distinguish an evil agent which tries to overload the system by spawning many threads that write huge amounts of data to the system's data storage, from one doing excessive parallel computation and writing many trace data, any agent system has to limit the usage of system resources for all agents. This can be done by negotiation with an agent when it enters the system, or by generally limiting those resources by some kind of quota.

In our system, all Web agents are granted a certain amount of each system resource. When trying to use more units than granted, the agent will receive an exception and is terminated when ignoring that. We are also working on a resource negotiation scheme which will ask each Web agent about its planned resource usage before running.

In addition to our agent system's inner security scheme, we secure all network communication, which occur in our architecture when an agent migrates or when a Web agent receives its configuration data by its GUI. In the first case, we encrypt the agent before transmitting it to the target SAE by using a session key. In the second case, we encrypt the data transferred. By doing this, we avoid that an agent in transfer is recorded, modified and then resubmitted to the target system, or is fooled by someone trying to submit changed or illegal configuration data to the agent.

[10] One can use a public key technique to achieve this.

[11] Because a Web agent's data changes during execution, one cannot use a signed message digest approach.

[12] One open problem of agent systems is that any server executing an agent has full control over the agent code and could easily extract and modify security related data stored in the agent. By building a net of trusted systems this danger could be reduced although not completely avoided.

[13] Since this requires the user to know an agent's ids and then adding this to the allowed users of a realm, we are currently developing a cookie mechanism which allows an agent to present the cookie it got from the data owner to the Web data interface in order to gain access permissions.

4 State of Implementation and Future Work

Our current implementation consists of the Web server with its SAE, which currently provides agent and script[14] submission and execution. At this time we are using a simple user/agent name and password protection scheme for specifying realms which can grant read, write, and execute rights to human users and additionally rights to access the network for agents. Configuration of the server and its SAE is done by reading a configuration file; this will be soon replaced by an online utility. The Web data interface gets currently extended to support writing database generated HTML pages. The resource control mechanisms, and the cryptographic support including the cookie mechanism will be fully available in two month.

After implementing the proposed infrastructure, our future work will concentrate on Web services. We will investigate how we can support a Web service manufacturer by tools that ease the implementation and organization of those services. This will also include investigations about suitable electronic payment systems for such services which we consider important for any commercial usage. Interfaces to emerging electronic cash systems will be provided through Java's electronic commerce API, which will be available soon.

5 Related Work

There are several research projects that deal with the implementation of general purpose agent systems [Pei95] [Joh95] [Lin95] [Gra96], some of them are using Java as an implementation language [Hoh95] [Lan96] [Mil96]. All these systems have a different focus: they aim at a platform supporting agents in general, which includes agent communication and agent control. So far, this is of minor interest to us, although it represents a general concern in our project. Our goal for the WASP project is to implement Web centric services, for which we developed the infrastructure that uses the server module SAE and a uniform access method to Web data. We are currently investigating what type of agents that come from other Java based agent systems could be integrated in our system.

Concerning our Web server implementation, there exist several other Web servers based on Java. The Jeeves [Jee96] project currently aims at the development of a server-side-include interface called 'serveletts' which enables servers to load and execute CGI-like Java programs directly into the Java virtual machine executing the server code. These serveletts must be loaded from the local file system of the server or can be downloaded from some other server on demand the local server. Our web server supports the servelett interface as well. Additionally, serveletts can also be sent to the server over the network. Compared to Jigsaw [Bai96], our server has a more fine granular access restriction scheme which originates from the needs for web-services and is enhanced by the Web agent server module.

In our efforts to offer our server module as a plug-in for other Java-based Web servers, we investigate whether it is possible to provide it as servelett and Jigsaw-Filter. For servers which are not Java-based, we plan to offer a CGI version of our SAE.

Summarizing, one can say that there is ongoing research which uses similar approaches but focuses on different aims. As far as we know, WASP is the first project that proposes services on Web data implemented by Web agents.

6 Conclusion

In this paper we presented the WASP project, which aims at the development of World Wide Web specific service applications. The proposed services are brought to users through a special kind of mobile agents which we call Web agents. While for traditional services there is a known service provider, this is not true for web services. Web services are provided by the means of Web agents,

[14] Scripts differ from agents by the submission format and provide no agent functionality at all. They can be seen as serveletts submitted through the network, and do not need but may conform to the servelett API.

which are the sole communication partner for any service user. The agent is the service, although the service provided by the agent may involve interaction with the manufacturer of the agent. This, however, is hidden from the user.

For the WASP project, we realised a Java-based Web server, a plug in server execution environment for Web agents, and a uniform access method to Web data. These components are the basis for our proposed infrastructure for Web agents. This infrastructure provides a general platform for service providers to implement all kind of services that deal with Web data. These services are brought to any service user by means of Web agents in a 'standard' way. By using as GUI the user's Web browser, to which one is familiar to, and the widespread network of web servers as a basis, we hope that there will be a good chance for a widespread dissemination of our platform, which provides easy but secure access to services and Web data.

Further extensions will provide tools that help service providers to generate and manage Web agents and prototypes for standard Web agents which can be configured in an easy way to a service provider's needs.

References

[Arn96] Arnold K., Gosling J., *The Java Programming Language*, Addison-Wesley, 1996 (ISBN 0-201-63455-4)

[Bai96] Baird-Smith A., *Jigsaw Java HTTP Server*, by World Wide Web Consortium, `http://www.w3.org/pub/WWW/Jigsaw/`

[Bor93] Borenstein N., Freed N., *MIME (Multipurpose Internet Mail Extensions)*, Network Working Group, RFC1521, 1993

[Bor93] Borenstein, N., *EMail with a Mind of Its Own: The Safe-Tcl Language for Enabled Mail*, `ftp://ftp.fv.com/pub/code/other/safe-tc.tar`

[Gen92] Gensereth M.R, Ketchpel S.P., *Software Agents*, Communications of the ACM, Vol.37, No.7, pp 48—53, July 1994

[Gra96] Gray R.S., *Agent Tcl: A flexible and secure mobile-agent system*, Proceedings of the Fourth Annual Tcl/Tk Workshop, Monteray CA, 1996,
`http://www.cs.dartmouth.edu/agent/papers.html`

[Hoh95] Hohl F., *Konzeption eines einfachen Agentensystems und Implementation eines Prototyps*, Diploma Thesis, Universität Stuttgart, Fakultät Informatik, Diplomarbeit Nr. 1267 (1995)

[Jee96] Jeeves Team, *Overview of the Java Http Server Architecture*, Part of the Jeeves Alpha2 distribution, Sun Microsystems, 1996

[Joh95] Johanson D., van Renesse R., Schneider F., *An Introduction to the TACOMA Distributed System*, University of Tromso, Institute of Mathematical and Physical Science, Department of Computer Science, CS Technical Report 95-23, June 1995

[Lan96] Lange D., Chang D.T., *IBM Aglets Workbench – Programming Mobile Agents in Java*, White Paper, IBM Corporation, Japan, August 1996, `http://www.trl.ibm.co.jp/aglets/`

[Lin95] Lingnau A., Drobnik O., Dömel P., *An HTTP-based Infrastructure for Mobile Agents*, World Wide Web Journal - Fourth International World Wide Web Conference Proceedings, Boston, MA, Dec 11-14, 1995

[Maj96] Majewski S.D., *Distributed Programming: Agentware, Componentware, Distributed Objects*, Notes for the discussion on Safe-Python at the NIST Python Workshop, 1994, `http://minsky.med.virginia.edu/sdm7g/Projects/Python/SafePython.html`

[Mil96] Milojicic D., Condict M., Reynolds, F., Bolinger D., Dale P., *Mobile Objects and Agents (MOA) Project*, OSF, position paper, "Distributed Object Computing on the Internet" – Advanced Topics Workshop, Second USENIX Conference on Object Oriented Technologies and Systems (COOTS), `http://www.osf.org/RI/DMO/dmo.htm`

[Net96] Network Wizards,
Internet Domaine Survey
`http://www.nw.com/zone/WWW/report.html`

[Pei95] Peine H., *The Ara Projekt*, University of Kaiserslautern,
`http://www.uni-kl.de/AG-Nehmer/Ara/ara.html`

[Sch94] Schneier B., *Applied Cryptograhpy*, Wiley & Sons, 1994

[Wei96] Weiyi L., Messerschmitt D., *Java-To-Go, Itinerative Computing Using Java*, University of California at Berkeley, Department of Electrical Engineering and Computer Sciences, `http://ptolemy.eecs.berkeley.edu/dgm/javatools/java-to-go/`

[Whi94] White J.E., *Telescript Technology: The Foundation for the Electronic Marketplace*, Whitepaper by General Magic, Inc, Sunnyvale, CA, USA