# How to Integrate Mobile Agents into Web Servers*

Stefan Fünfrocken

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

## Abstract

*Mobile agents are a new paradigm for communication and cooperation in distributed computing. To combine the new paradigm with the promising World Wide Web platfrom, we integrated mobile agents into Web Servers. In this way, mobile agents may travel from Web server to Web server to access their local data. The paper describes how we integrated mobile agents support into a Web server. We present the state of implementation, and we give an outlook on our future work.*

## 1. Introduction

'Mobile Agents' are programs that can move from host to host 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 [17]. There are several interpreted languages which offer a so-called 'secure interpreter', like safe-tcl [4] or safe-python [13]. 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.

Application areas for mobile agent technology include tasks as simple as information retrieval, but is also used in mobile computing [15], telecomunication applications [12], electronic commerce [17], and other traditional areas of computer science. One special application area is the World Wide Web which is still growing at an exponential rate, 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 and data processing applications (e.g., databases, newspapers, financial portfolio applications) are adapted and integrated.

To combine mobile agent technology with the potentials and infrastructure offered by the World Wide Web, we developed a Web server extensions module which allows Web servers to host mobile agents, so-called Web agents.

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.

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 how we integrated mobile agents into a Web server, that is, the WASP infrastructure.

## 2. WASP Infrastructure

Integrating mobile agents into a Web server means to enable Web servers to start, receive, and execute mobile agents. Additionally, Web agents may access the Web data of the server, may want to communicate with a user, 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' servelets [8], or Jigsaw's resource interface [2], which is able to pass all relevant information to the SAE.

As an implementation language for our system and for agent-programming, we chose Sun's Web language Java [1], which already provides code shipping by the means 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 [7], Aglets [10], Java-to-go [16], and MOA [14]. 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.

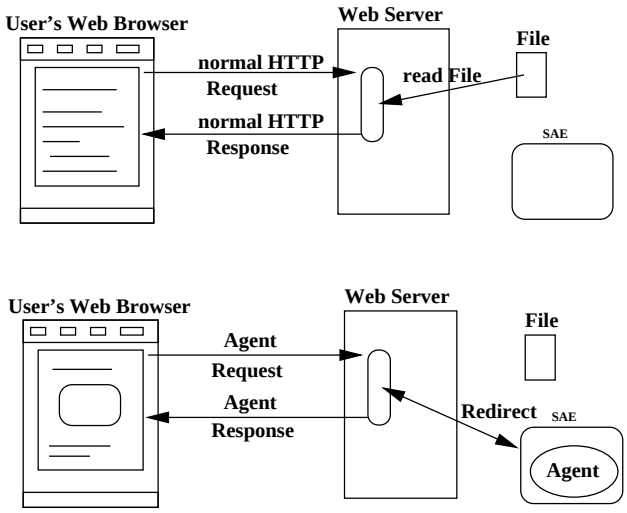


**Figure 1. General WASP Infrastructure**

Figure 1 shows the overall architecture of the WASP infrastructure which consists of a user's Web browser, our Web server, and the server's SAE. To traditional HTTP requests our Web server responds in the traditional way: it provides the user with the desired data through a HTTP response containing the data. But when it receives an agent related request, the server hands it to its SAE to fulfill the request. There is no need for the server to understand any SAE related request, it is sufficient that it identifies the request as a SAE request to forward it to its SAE. In this way, any changes in SAE requests do not affect the Web server implementation. So far, we only need two SAE related requests: one for agent startup and one for agent transfer. Both are normal HTTP requests, which implies that we do not need a special protocol such as ATP (Agent Transport Protocol), which is used by the aglets [10] system.

Generally speaking, our scenario is divided into two phases: the agent startup and configuration phase, and the agent service phase. In the first phase, a user wants to access an agent which is located at a Web server. The user uses its local Java-enhanced Web browser to connect to the server and requests the start of the Web agent. Since all agent related actions are executed by the SAE, the server redirects the user's request to it. The SAE loads and starts the agent. The first action of a Web agent is to get customized to the user's demands. 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 the configuration. After that, the second phase begins: depending on the configuration, the service is provided by executing 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.

## 2.1. WASP HTTP Server

To have full control over the whole architecture, we use a Java Web server that was developed earlier by us. This Web server was enabeled to accept Java scripts over the network and execute them at the server location. After start of the WASP project, we redesigned the server and used it to integrate our SAE. Figure 2 shows the architecture of the WASP Web server. Any incoming HTTP request, represented by a request object, is handed to the request manager, which is responsible to generate a HTTP response according to the request. The request manager has a pool of managers which it can instruct to process the request. Since HTTP requests may require different actions, there is a manager for each type of processing. So far, our server can handle requests which require to retrieve the content of a file, requests which require the execution of cgi-bin programs and scripts, and agent related requests which require to contact the SAE. The requests are processed by the content manager, the cgi manager, and the agent manager respectively. The content manager is assisted by a mime manager, which is responsible to set the correct mime content type of the response. The mime manager retrieves its knowledge at startup time from the system mime.types file. The content and cgi-bin manager, which both deal with files, use the document manager to access the files. The document manager, which is in fact an interface to our W3 Data Interface (see Section 2.5), is used to hide the way a document is stored, thus offering an abstract view to the content and cgi manager: a document may be read, written, or executed, independent of the way

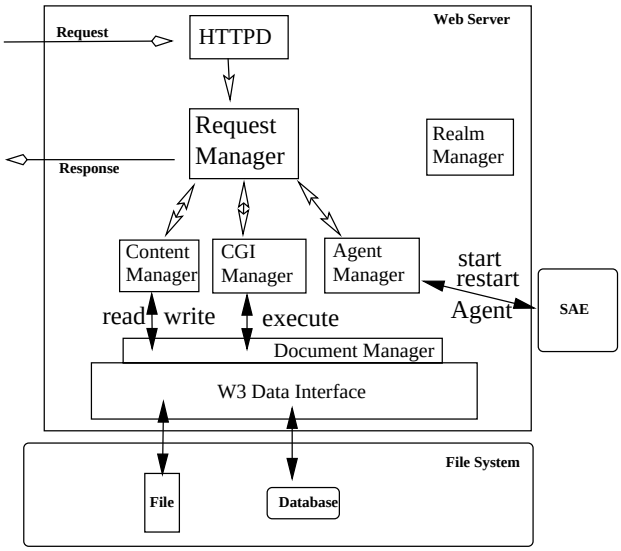


**Figure 2. WASP httpd Architecture**

the document is stored. Before any request is executed, the server checks whether the request may need an identification of the user making the request (see Section 2.6).

To allow our Web server to host mobile agents, we developed the SAE, which processes any agent related request on behalf of the server's agent manager.

## 2.2. Server Agent Environment

The SAE (see Figure 3) serves as an execution environment for Web 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. It has to execute the agent and guarantee access and security restrictions specified by the administrator of the Web site during agent execution. Unlike other agent environments, it provides Web agents with a uniform interface to the server's Web data (see Section 2.5) so that a Web agent doesn't have to care about the way Web data is stored locally. In addition to agent related tasks like agent

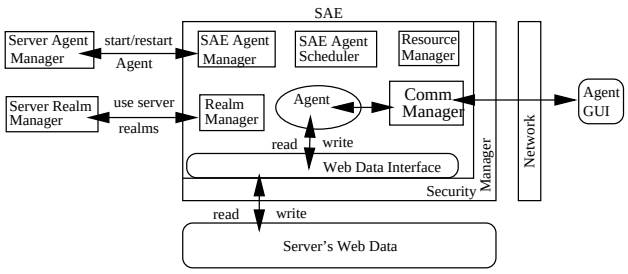


**Figure 3. SAE Architecture**

management, agent scheduling, security, and resource control, an execution environment has to address problems like configuration, administration, controlled startup and shutdown, persistency in case of crash and recovery, 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. Concerning persistency, Java's object serialization offers an appropriate way to implement object persistency in general.

## 2.3. Agent Startup

Web agents are physically divided into two parts: the service part which encapsulates the agent's functionality, and the agent's graphical user interface which the agent presents to the user at startup time. In this way, it is possible to offer more than one user interface to users. The agent may choose its graphical user interface depending on a startup parameter, thus presenting different views to the same functionality. This architecture makes it possible to update and add agent user interfaces without changing the code of the agent.

Figure 4 shows the agent startup scenario. All agents that are installed and may be started at a Web server are described on a normal HTML page, which contains a description of the service each agent offers and a link to the agent. When a user requests the start of an agent by a HTTP get request to the agent URL, 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 loads and 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. Technically, the communication between the agent GUI and

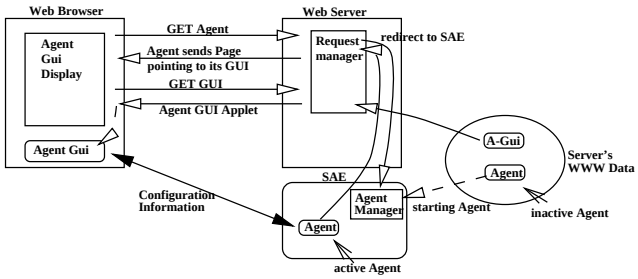


**Figure 4. Agent Startup Scenario**

the agent is done through the communication manager of the SAE (see Figure 3 and Section 2.6). 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, after configuration, respond to the user with a result URL. The user has to monitor this URL for the results of the service.

### 2.4. 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 [17] or ARA [15]: the flow of control cannot automatically 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, although this puts some burden on the programmer of the Web agent. As other systems that use this scheme of migration (Mole [7], Aglets [10], FFM [11], Tacoma [9]), 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 [3] 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 2.6).

### 2.5. 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. 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. To offer the advantage of our Web Data Interface not only to local Web agents, we use it as a document manager in our Web server implementation. This complements the view a remote user has of local Web data: he or she only requests documents (i.e., document contents), independent from the way the data is stored at the server. We enforce the usage of our Web Data Interface by using a security manager which does not allow agents to use any standard Java library I/O-method.

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 Figure 3 and Section 2.6). 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.

### 2.6. 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.

Data 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 or agent accessing the protected data has to identify itself to the system by the means of a password which, together with the user's name or deduced form the agent's ids, is stored with the configuration data of the realm. For every realm there is an owner (human or agent)

who can define access rights for the realm (read, write, execute).

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 unchangable parts of the Web agent with its manufacturer id when releasing it. Additionally, there is an agent id which in combination with the manufacturer's id has to be 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.

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 some 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.

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. In case of a human user, he or she is asked for its user id and password by his or her 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. 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.

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.

## 3. Using the SAE with Other Servers

We plan to offer our SAE as plug-in for Web servers, so that usage of it is not limited to our own Web server implementation. To make use of our SAE, a foreign Web server has to provide an interface which allows to hand the incoming request to the SAE and allows to hand back any response. Most existing Web servers offer the cgi-bin interface, which provides this. Java based Web servers as Jigsaw [2] or Jeeves [8] can integrate our SAE directly. We developed a Jigsaw resource object which connects our SAE to the Jigsaw resource tree and are working on a Jeeves SAE-servelet.

In addition to connecting the SAE to the server, there is another problem to solve: since Web agents executing in the SAE access local data which should be protected by the protection scheme a Web server is using, the SAE has to respect that scheme. Most existing Web servers offer simpler protection domains than our server is offering: they limit *any* access to the protected domain to the users registered in the realm. Our scheme has a finer granularity, by offering read, write and execute rights for priviledged and nonpriviledged accesses. Because of that, we have to incorporate the realm definitions of other servers in our realm system. The way we have to incorporate the realms differs with the way the SAE is connected to the server: when using the SAE as cgi-bin, we have no other choice then reading the servers realm definition file. This also means that any changes made to the realms have to be written to that configuration file. In contrast to that, Jigsaw offers the possibility to access the Jigsaw realm objects. In this way, we can make use of that objects. We are currenty investigating how we can incorporate the realms offered by the Jeeves server.

Summarizing, one can say that connecting our SAE to other Web servers is done very easily by using a cgi-bin approach. But because the SAE has to respect the access restrictions the server imposes onto its data, we have to provide a special SAE plug-in for each type of server.

## 4. State of Implementation and Future Work

Our current implementation consists of the Web server with its SAE, which provides agent and script 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 or agents. One can additionally specify rights to access the network for agents. Configuration of the server and its SAE is done by reading a configuration file; this will soon be replaced by an online utility. Resource control is done by offering a fixed amount of each resource. Our Web Data Interface is able to read HTML documents from the file system or data base. Reading and writing database-generated HTML documents is not fully implemented yet.

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 we plan to make use of.

## 5. Related Work

There are several research projects that deal with the implementation of general purpose agent systems [15] [9] [11] [6], some of them are using Java as an implementation language [7] [10] [14]. 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 [8] project developed a server-side-include interface called 'servelets' which enables servers to load and execute CGI-like Java programs directly into the Java virtual machine executing the server code. Serveletts can be bound to a URL and are loaded by the server, when a user requests the URL. The server can load the servelet from the local file system or over the network. In both cases it is the server requesting the servelet. Thus servelets cannot be compared to mobile agents. Our Web server will support the servelet interface. Additionally, servelets can also be sent to the server over the network.

Compared to Jigsaw [2], 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 developed a Jigsaw resource representing our SAE and investigate whether it is possible to provide it as servelet. For servers which are not Java-based, we plan to offer a CGI version of our SAE which may require a special interface module to the 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 integrated mobile agents into Web servers and proposes services on Web data implemented by Web agents.

## 6. Conclusion

In this paper we presented the architecture developed in the WASP project, which aims at the development of World Wide Web specific service applications implemented by using mobile agents.

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. By using the user's Web browser as GUI, 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.

## References


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

[2] Baird-Smith A., *Jigsaw Java HTTP Server*, by WWW Consortium, http://www.w3.org/pub/WWW/Jigsaw/

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

[4] 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

[5] Gensereth M.R, Ketchpel S.P., *Software Agents*, Comm. of the ACM, Vol.37, No.7, July 1994

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

[7] Hohl F., *Konzeption eines einfachen Agentensystems und Implementation eines Prototyps*, Diploma Thesis, Univ. Stuttgart, Dept. of CS , Nr. 1267 (1995)

[8] Jeeves Team, *Overview of the Java Http Server Architecture*, Jeeves Docu, Sun, 1996

[9] Johanson D., van Renesse R., Schneider F., *An Introduction to the TACOMA Distributed System*, Univ. of Tromso, Dept. of CS, CS Tech. Report 95-23, June 1995

[10] Lange D., Chang D.T., *IBM Aglets Workbench – Programming Mobile Agents in Java*, White Paper, IBM Corporation, Japan, August 1996,

[11] Lingnau A., Drobnik O., Dömel P., *An HTTP-based Infrastructure for Mobile Agents*, WWW Journal - Fourth Inter. WWW Conference Proc., Boston, MA, Dec 1995

[12] Magedanz T., Rothermel K., Krause S., *Intelligent Agents: An Emerging Technology for Next Generation Telecommunications?*, IEEE INFOCOM 1996, San Francisco, USA, March 1996

[13] Majewski S.D., *Distributed Programming: Agentware, Componentware, Distributed Objects*, Notes for the discussion on Safe-Python at the NIST Python Workshop, 1994,

[14] 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, COOTS

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

[16] Weiyi L., Messerschmitt D., *Java-To-Go, Itinerative Computing Using Java*, Univ. of California at Berkeley, Department of EE and CS,

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