Mobile Agents as an Architectural Concept for Internet-based Distributed Applications

– The WASP Project Approach –

Stefan Fünfrocken, Friedemann Mattern

Department of Computer Science, Darmstadt University of Technology
Email: {fuenf,mattern}@informatik.tu-darmstadt.de

Abstract. After introducing the concept of mobile agents and potential application domains, we motivate why mobile agent technology is an interesting concept for large Internet-based system structures. We then describe the Java-based WASP agent environment which integrates agent execution platforms into WWW servers and thus promotes a world wide infrastructure for mobile agents. We sketch first prototype applications, and we mention some unique aspects of the WASP project such as fully transparent migration of Java objects. Finally we report on some experiences we gained when realizing our mobile agent system.

1 Mobile Agents

Mobile agents are software processes which can autonomously migrate from one host to another during their execution. While roaming the Internet or a proprietary intranet and visiting other machines, they do some useful work on behalf of their owners or originators.

By transmitting executable programs between (possibly heterogeneous) machines, agent-based computing introduces an important new paradigm for the implementation of distributed applications in an open and dynamically changing environment. This paradigm can even be understood as an *architectural concept* for the realization of distributed systems. It is particularly well-suited if adaptability and flexibility are among the main application requirements.

From the point of view of classical client-server computing, which nowadays is the prevalent architectural model for distributed systems, mobile agents can be seen as an extension or generalization of the well-known remote procedure call (RPC) principal. But whereas in the RPC case merely data is moved from the client to a procedure that already resides on the server (and the client usually remains idle while the remote procedure is executed), in an agent-based framework the client dispatches an agent which travels to the server and performs its task there by interacting locally with the server's resources.

Hence, mobile agents (which can be understood as an elaborated form of mobile code [4]) are able to emulate remote procedure calls, but more importantly, they also allow for much more flexible and dynamic structures than traditional systems based on the client-server paradigm. Compared to lower level mechanisms such as RPC or simple message passing, the use of mobile agents for distributed applications has several potential benefits:

– *Asynchronous task execution*: While the agent acts on behalf of the client on a remote site, the client may perform other tasks.

– *More dynamics*: It is not necessary to install a specific procedure at a server beforehand and to anticipate specific service request types; a client or a service provider may send different types of agents (e.g., realizing new service handlers) to a server without the need to reconfigure the server.
– *Reduced communication bandwidth*: If vast amounts of server data have to be processed (e.g., weather data) and if only a few relevant pieces of information have to be filtered out, it is more economical to transfer the computation (i.e., the agent) to the data than to ship the data to the computation.
– *Improved real time abilities*: Agents acting locally on a remote site may react faster to remote events than if these events and reactions to them have to be communicated between the remote machine and a central entity.
– *Higher degree of robustness*: A dispatched agent may be instructed how to deal with potential problems such as unavailable servers (e.g., go to alternate sources or retry at some later time). Although mobility introduces new failure cases, in general fault tolerance is promoted because a mobile agent has the potential to react dynamically to adverse situations.
– *Improved support of nomadic computing and intermittently connected devices*: Instead of being online for a longer period, a mobile user may develop an agent request while being disconnected, launch the agent during a brief connection session, and receive back the agent with the result at some later time.

Several academic research projects (e.g., [1, 8, 12]) explore the mobile agent paradigm, and several commercial systems (e.g., Aglets [9], Voyager [13], Concordia [14]) have been introduced recently. Most of these systems are based on Java for the programming of agents, but they largely differ in their migration and security models and most importantly in the support and services they provide for the agents. Some aspects of our own mobile agent project WASP ("Web Agent-based Service Providing") [5, 6] will be presented further down in Section 5.

2 Applications with Mobile Agents

Compared to traditional distributed computing schemes, mobile agents promise (at least in many cases) to cope more efficiently and elegantly with a dynamic, heterogeneous, and open environment which is characteristic for today's Internet. Hence, mobile agents can be useful in many applications.

Certainly, *electronic commerce* is one of the most attractive areas in that respect: a mobile agent may act (on behalf of a user or owner) as a seller, buyer, or trader of goods, services, and information. Accordingly, mobile agents may go on a shopping tour in the Internet: they may locate the best or cheapest offerings on WWW servers, and when equipped with a negotiation strategy, they may even do business transactions on behalf of their owners.

Another general application domain is *searching for information* in the Internet or information retrieval in large remote databases when queries cannot be anticipated: Agents may incorporate an implementation of a specific search query (i.e., a retrieval procedure) and thus allow for semantic information compression by remote filtering of data. In particular, collecting information spread across many sites and performing some kind of transactions when appropriate information is encountered, is a useful application for mobile agents.

Monitoring is also a typical application domain: Agents can be sent out to wait for certain events or certain kinds of information to become available and then react appropriately (e.g., by buying shares on a stock market host). Similarly, mobile agents may also be used for the automation of many tasks in *network configuration and management* (e.g., for remote diagnosis). Agents may install software on remote machines, or they may personalize remote devices and services.

Other uses of agent technology include *workflow management systems* and *groupware applications*: Active documents that contain semantic routines to process their content may be realized by agents which travel to appropriate places in an organization. One last example of a potential application area is *entertainment*: Mobile agents may enable distributed multi-user games, they may locate persons with a similar interest, and they may represent a player on a game host.

In general, mobile agents seem to be a promising technology for the emerging open Internet-based service market. They are well-suited for the personalization of services, and dynamic code installation by agents is an elegant means to extend the functionality of existing devices and systems. Agent technology therefore enables the rapid deployment of new and value-added services.

However, in order to become a widely accepted technology in practice, some problems remain to be resolved. The most important aspects are probably security concerns (protecting hosts from malicious agents, but more crucially also protecting agents and agent-based applications from malicious hosts) [3]; but interoperability with other systems, coordination and communication aspects, and the management of large societies of mobile agents also pose interesting challenges.

3 Mobile Agents for Internet-based System Structures

The emerging Internet-based electronic commerce infrastructure is several orders of magnitude larger than most traditional distributed systems [2]. Furthermore, these systems have to be highly flexible and have to cope with a number of challenging properties: Connectivity between nodes in the Internet is highly variable, mobility (which entails frequent and prolonged disconnections) plays an ever increasing role, and embedded systems and devices with limited resources and dynamic behavior (such as smartcards) are being integrated into these systems.

Conventional system architectures like remote procedure calls which were designed several years ago (i.e., before the WWW phenomenon) with a more static and reliable system structure in mind, may not be well suited for large Internet-based applications. Mobile agent technology, when combined with more traditional mechanisms in an appropriate way, enables architectural concepts (such as function shipping, ’call by visit’, or code on demand) that deal much better with these conditions. Furthermore, mobile agents are a higher-level abstraction than messages or procedure calls, their inherently distributed nature often provides a natural view of a distributed system, and they seem to enable structures in a networked environment that fit more naturally with the real world.

It should also be noted that software agents, which bring together the two concepts “process” and “object”, are interesting building blocks for flexible system architectures, even if they are not always mobile. In fact, *stationary* or permanently resident agents are probably as important as mobile agents: They encapsulate autonomous activities in a stronger way than classical objects, they communicate with other (mobile) agents via the same protocols and interfaces, and together with mobile agents they provide a uniform way to structure large distributed systems.

Since the dynamic creation of agents is a basic functionality of typical agent systems, agents are also an ideal mechanism to enable parallel processing. A typical example would be a search agent that sends out child agents to visit multiple machines in parallel. Of course, mechanisms to control the high degree of dynamism of such agent-enabled parallel computations then become a necessity.

In order to make use of existing distributed system functionality, it would be desirable to have interoperability mechanisms that connect agent platforms to middleware concepts like CORBA or emerging Internet infrastructures like Jini. In fact, the OMG recently proposed MASIF (Mobile Agent Systems Interoperability Facilities), a standard that deals with interoperability issues between different agent systems and CORBA services [11].

The ubiquitous availability of agent environments is a necessity for any successful usage of the mobile agent paradigm. The deployment of the mobile agent infrastructure, however, should not impose much overhead (such as the installation of a whole CORBA system). In our opinion, the only way to promote fast dissemination of mobile agent platforms is the usage of a well established and widely used technology: the World Wide Web.

However, in contrast to just using Web technology, a more interesting idea seems to be the integration of mobile agents and the WWW by enabling WWW servers to host mobile agents. WWW servers are ideal places for mobile agents, since most of the accessible data and electronic commerce shops in the Internet reside on WWW servers. Furthermore, almost every user has a WWW browser which can be used to communicate and control mobile agents running on WWW servers, and which could even serve as a home base for personal mobile agents. Our WASP project (see Section 5) builds upon this idea of using agent technology in conjunction with the WWW – by extending (and not just using) the WWW we aim at providing a ubiquitous mobile agent platform.

4 Infrastructure for Mobile Agents

In an agent-based computing scenario, hosts must provide a kind of a “docking station” for mobile agents which acts as a local environment or *agent platform*. Such a platform is responsible for launching, receiving, and providing residence to agents, and it has to provide the necessary services, resources, and runtime support. It may also act as a meeting point for agents or even provide a trusted computing base (e.g., a hardware-based secure execution environment). The main tasks of a local agent platform can be summarized as follows:

– *Mobility support*: Arriving agents have to be installed and registered, and the code together with the state of agents that want to migrate to other hosts has to be packed together and sent over the network.
– *Resource management*: Agents have to express their resource requirements (e.g., memory, cpu share, communication bandwidth) and the agent platform has to check authorizations, quotas, and also act as a firebreak against monopolization or excessive use of resources.
– *Execution support*: Agents must have access to runtime libraries and services. The agent platform should also support the creation of new agents.

- *Communication support*: Agents should be able to communicate with other locally residing agents, but also with remote agents and with their owner or creator. For that, the agent environment should support standard communication mechanisms and protocols.
- *Directory and information service*: Agents must be able to check the availability of services and they should also be able to learn about the local presence of other agents. They might also expect help in localizing remote agents.
- *Security support*: An agent platform must ensure the privacy and integrity of agents and its own infrastructure. For that, it needs means for encryption and decryption of agent code, and it must provide authentication, authorization, and access control mechanisms.
- *Event delivery service*: The information about pertinent events has to be conveyed to agents which have expressed an interest.
- *Support for fault tolerance*: Correct and reliable execution of agents should be guaranteed even when partial failures (e.g., missing resources, unreachable migration goal) occur.

Besides these mainly local tasks of an agent environment, there are tasks which require cooperation among several distributed agent platforms and hence necessitate standard protocols and interfaces. Examples of such generic global services are the localization of agents, forwarding of messages, and brokering facilities. Management of whole agent societies (e.g., finding lost agents or termination of agents that went astray) is also a global (and non-trivial) task of an agent infrastructure.

Certain application classes may require that agent platforms provide further, more specific services. An example of such services is support of application frameworks (e.g., for electronic cash). To support such frameworks, agent platforms should allow an easy integration of application specific resources and services.

One additional feature of a mobile agent platform not directly associated with its core functionality is the provision of agent programming functionality. Experienced agent programmers but also casual users might want support to tailor the behavior of an agent to their special needs or habits.

5 The WASP Mobile Agent Platform

To study the effects, benefits, and challenges of the mobile agent paradigm, and to experiment with some novel features, we designed and implemented our own mobile agent platform. The WASP platform (Web Agent-based Service Providing) provides most of the services and tasks described in the previous section, in particular support for resource management, mobility, agent execution, communication, and security. It is unique in the way it achieves theses tasks by relying on established Java distributed computing concepts and, more importantly, by integrating agent environments into WWW servers with the help of server extension modules. As an additional benefit of relying on the well-established WWW infrastructure, the WASP platform may easily be deployed in the Internet. In contrast to other mobile agent projects that make use of the WWW (e.g., [10]), we are not just using the HTTP protocol for agent transfer or control, but we integrate the new technology into the WWW by offering a module that can be combined with existing WWW severs, and by giving agents access to the local data of a host through a web-like interface.

Our primary intention was not to realize a complete general purpose agent platform, but to develop the platform, application scenarios, and concrete applications (starting with some modest prototypes) in parallel. Using this evolutionary approach we want to gain experience and learn about essential features of an agent environment. In contrast to other projects [1, 9, 12] which aim at providing a general mobile agent environment, we decided to develop a platform tailored to an application domain which in our opinion should offer the best chances for a wide range of mobile agent enhanced applications: electronic commerce on the WWW.

With respect to electronic commerce as our primary application domain, and in order to promote a seamless WWW integration, we emphasized the following points in the WASP project:

– Provision of identical access mechanisms to local WWW resources for users (e.g., via a WWW browser) and agents that act on behalf of a user.
– Support of payment mechanisms that allow electronic commerce transactions between agents as well as between an agent and a WWW based service.
– Easy interoperability of agent-based applications with existing services and legacy applications.
– Simple communication means for interactions between agents on the one hand and between users and agents on the other hand.
– Tools to support agent application developers (e.g., creation of new agents from patterns of similar agents, support of an adequate programming style).

5.1 WWW Integration

Serious applications based on mobile agents need ubiquitous availability of an agent environment. We think that the WWW is ideal in that respect since it can serve as a world wide platform for distributed, mobile agent-based applications. Therefore, we gave our platform the ability to get easily integrated into existing WWW servers. This is supported by using standard server extension interfaces (CGI and servlets) and by using the classical HTTP protocol for agent transfer.

Figure 1 shows the overall architecture of the WASP mobile agent platform: we developed a WWW server together with an agent-specific part called Server Agent Environment (SAE). We used Java as the implementation language since most WWW servers support Java's servlet interface (which we need to attach our SAE) and because of Java's ubiquitous availability – especially in Web browsers, which agents use to communicate with the user. The WWW server redirects all agent related request (e.g., agent start, agent migration) to its attached SAE (see [5] for further details). Agents may be started by an HTTP request to a URL designating a particular agent type on some server. The actual start of the agent is done by the server's SAE. After being loaded and initialized by the SAE, the agent may send its Java-based GUI to the user's browser. The transfer of migrating agents is realized with an HTTP post request to a SAE specific URL at the target WWW server.

5.2 Communication Concepts

Communication is a necessity for mobile agents. A mobile agent platform should not restrain the availability of classical network communication schemes. Instead it should provide a set of standard mechanisms and an open internal architecture which allows an easy

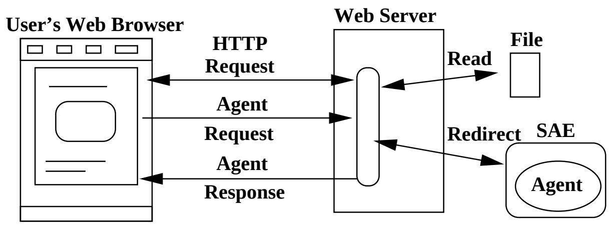

Fig. 1. The general architecture of the WASP platform

integration of new communication mechanisms. Therefore we designed our remote communication infrastructure such that it is easily extensible by using a modular approach. Our platform currently offers the following communication mechanisms to agents:

– Message-based communication, where agents can send messages to any other agent, whether the agent is local or resides in another SAE. The message can be sent asynchronously or synchronously.
– Stream-based communication, where agents can exchange Java streams. The streams are automatically reconnected when an agent migrates.
– Remote object communication, where agents can make use of CORBA, RMI (Remote Method Invocation), or other communication architectures that offer remote objects (e.g., DCOM). Agents can export such objects or can connect to remote objects that are exported by standard applications.
– Local object communication, where agents can export and import references to Java objects which are accessible at the local SAE.

5.3 Agent Localization

When realizing an agent platform, one would ideally like to import existing functionality from traditional middleware platforms. Unfortunately, this is not always possible, as the example of name services shows. A name service is basically used to find the location of an object. Unfortunately, a traditional name service such as DNS, NIS, or WINS is not designed to cope with mobile objects which move very dynamically. Any sensible name service for mobile agents has to use efficient mechanisms to keep track of the agents to reliably return the current location of an agent. Such a service could use, for example, mechanisms similar to those in mobile telecommunication systems as for example GSM.

Our platform does currently not include an agent name service to locate the agents, but uses URL-style naming conventions to name (and locate) agents and objects exported by agents. Since our agents move from WWW server to WWW server only, this is the most natural way for names in our system. We make use of the naming mechanism of the underlying system to locate a host offering a mobile agent enhanced WWW server. To find and communicate with a specific agent, the agent programmer currently has to make the agent to export a proxy object which is left behind on the original server and to which the agent has to connect regularly when underway in order to drop its new host address. This is not an ideal solution, of course, and should be replaced by a better mechanism in the future.

6 Agent Programming Support

As mentioned earlier, support for agent construction should be provided by the agent environment. Because of that, one part of the WASP project deals with support for agent programmers. On the one hand these might be agent users who want to program simple agents without requiring in-depth knowledge about agent programming and the agent platform. On the other hand service providers should be supported by a tool that enables the realization of more complex service agents. Language aspects and simplicity of migration at the language level are of course also an issue concerning usability.

6.1 Agent Construction Tool

To support programmers, we developed a tool that allows the graphical construction of agents from so-called agent templates. Basically, the tool is designed as a management tool for code templates and code fragments, and offers two kinds of functionality: high level construction and low level construction.

In high level construction mode, one can graphically insert so-called agent components into agent templates. Agent templates define and implement an agent's basic functionality. They require the presence of some particular subfunctionality, as for example a database query component, a payment component, a data carrier component, or a migration component. The agent programmer can then fill in the code for these components from a list of components that provide the required functionality but differ in their implementation.

We also considered to incorporate Java's component architecture Java Beans into our agent construction method. We found, however, that the design patterns, and in particular the asynchronous, event-based communication mechanism that is used by Java Beans, is not well suited for agent components. The reason is that from the viewpoint of an agent programmer the agent components should interact through method invocation, passing parameters over a known interface. Java Beans, however, do not rely on interface knowledge. Instead, they communicate over events in an anonymous way. Of course, one could simulate method invocation and parameter passing using the event model, but this is rather involved.

6.2 Transparent Migration

Migration is a key concept of mobile agents, which from the programmer's point of view comes in two different programming styles that reflect different capabilities of the underlying system:

– *non-transparent migration* which assumes that after migration an agent is restarted from the beginning or at a predefined code entry point, and
– *transparent migration* which assumes that an agent execution continues on the new target host directly after the instruction that initiated migration.

Transparent migration requires automatic capturing of the entire execution state of an agent, but poses much less burden on the agent programmer: there is no need to explicitly code the agent's suspend and restart procedures (which specify where to continue execution after migration and which variables have to be saved).

Unfortunately, Java does only support non-transparent migration. Because we consider transparent migration to be more convenient for the agent programmer, we developed a preprocessor which automatically converts Java code written in transparent migration programming style (i.e., including a "go to <target>" operation) into basic Java. To capture the necessary state information (i.e., the method call stack, the relevant variables, and the current value of the program counter), the preprocessor inserts code that saves (and later restores) this information. In this way we achieve, with modest overhead, transparent migration on the language level, without modifying the Java virtual machine ([7] describes the mechanism in detail).

7 WASP – Status and Current Work

Main parts and the basic functionality of the WASP platform have been implemented. Current work in the WASP project concentrates on prototype applications, support for security and electronic commerce services, and some general enhancements of the agent platform.

7.1 Prototype Applications

We have already realized several small applications with the WASP platform:

– A *WWW newspaper* as a simple electronic commerce application (see Figure 2). A user starts a newspaper agent and personalizes it. This user agent then creates one or more search agents that search the WWW for information corresponding to the profile of the user. This is done by migrating to information servers, communicating with the local information agent, and ordering the desired information. The local information agent then sends a data carrier agent to the user's machine. The data carrier agent has full control over the information it carries, and releases (i.e., decrypts) it under certain conditions only (e.g., when it is paid for).
– A *login trace* application, which traces user logins across a cluster of machines. A so-called walker agent is created at some machine and searches the system log to learn from which machines a specific user logged on to the current machine. The walker agent builds the transitive closure by recurrently migrating to machines from which remote logins originate. The collected login graph information is eventually carried back to the system administrator.
– An *application management* scenario. We are currently developing an agent-based system to manage a whole cluster of WASP-servers and their SAEs. This system is used to gain deeper insight in how mobile agents fit into traditional management tasks.

7.2 Security and the Java Card

Electronic commerce applications impose strong requirements on mobile agent security: since commerce agents act on behalf of users and may carry electronic money, agents have to be protected from malicious attacks, and users have to be securely identified before they can authorize agents to act on their behalf.

Besides the basic security mechanisms to protect a host from malicious agents [6], we are currently experimenting with the integration of the Java Card (a smartcard that

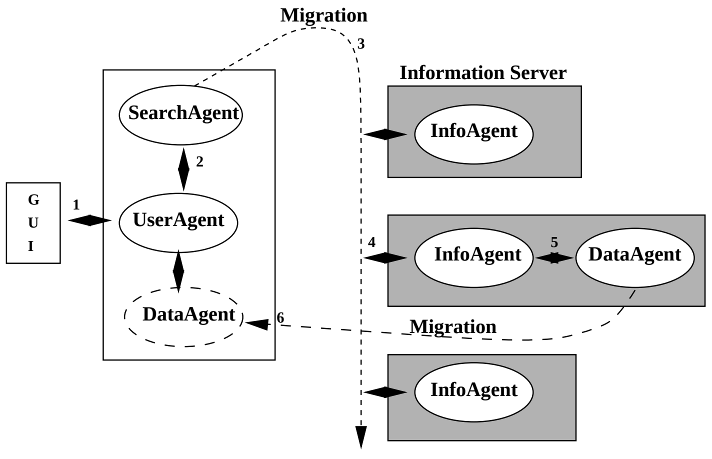

Fig. 2. WWW newspaper scenario

contains a Java bytecode interpreter) into our system. In a first step the Java Card is used to authorize mobile agents to act on behalf of the user starting the agent. Here the Java Card simply identifies the user who signs the rights he or she grants to the agent.

In a second step we want to integrate the Java Card in a more appropriate manner: Since the card is able to run Java code, we plan to use it as a trusted computing base [15] for mobile agents. For that, the Java Card is attached to the host running our SAE. All such cards own a private key. Then an agent can carry code which is encrypted with the public key of a smartcard. Only the card itself can decrypt and execute this code. An encrypted agent moves its code (or parts of its code) from the SAE to the Java Card and executes it while running in this highly secure environment.

7.3 Electronic Commerce Framework

As mentioned earlier, we think that electronic commerce is one of the major application domains for mobile agents. Sun is currently working on the development of an electronic commerce framework based on Java (JECF), which offers an open payment platform for Web commerce. We are currently integrating JECF into the WASP system. During this integration we will examine what kind of payment mechanisms are reasonable for mobile agents, and what implications the dynamic generation of agents imposes on electronic commerce scenarios.

8 Experiences

Realizing a mobile agent infrastructure and implementing prototype applications on top of it is a major task: Support of mobility and implementation of security mechanisms for mobile code are among the main challenges, and even when building on established Internet technology (such as Java or WWW) and when relying on functionality from existing middleware platforms, much remains to be done.

One example is Java's built-in possibility to save and restore an object's state (i.e., object serialization and deserialization). As we have seen, this does not capture the execution state (i.e., the run time stack and the program counter). Also, it is unclear what the canonical environment of a mobile agent realized by a Java object should be: Is the class code of dynamically instantiated objects of an agent to be loaded from the local platform or does it have to be fetched remotely from the original server that created the agent (as it is done in our system)?

Besides such conceptual questions there are also numerous technical problems. For example, the Java virtual machine currently does not allow to unload class code (only object instances can be garbage collected) after all objects of the class are destroyed. Because of this, the code of all agents that once run in a virtual machine wastes memory. These and many other problems have to be solved when realizing a usable mobile agent environment.

Another important point are support services. Some services such as directory services or communication services are mandatory, others such as message forwarding or agent control would greatly simplify application development, but could be left out in a first step. Unfortunately, object mobility adds a new flavor also to those issues for which solutions exist in traditional middleware systems. This often renders existing solutions inappropriate (as we explained for the name service or Java's component model) and necessitates the implementation of services adapted to the mobile agent paradigm. This is unfortunate since it increases the size and complexity of agent platforms and thus hampers their widespread deployment.

We do not yet have much experience with the realization of mobile agent-based applications. We found it rather easy to program sets of communicating and cooperating agents, but design rules are still missing – often the best way to structure an application and to decompose the functionality into several agents is not clear.

9 Conclusions

The WASP project started in 1996 in order to explore the use of mobile agents in electronic commerce scenarios. In the beginning we were somewhat skeptical about the general applicability of this paradigm. Although we have learned that the realization of a general and widely deployable agent platform is non-trivial, we are now convinced that mobile agent technology, when appropriately combined with traditional middleware mechanisms, is ideally suited for large Internet-based applications. Some issues, such as security, fault tolerance, and interoperability remain to be solved in a satisfactory way, however. We believe that integration into the WWW infrastructure, as it is done in the WASP project, is almost mandatory for successful deployment and use of mobile agent technology.

References

1. Baumann J., Hohl F., Rothermel K., Straßer M., *Mole - Concepts of a Mobile Agent System*, WWW Journal, Special Issue on Applications and Techniques of Web Agents 1 (3), 123-137, 1998
2. Berbers Y., De Decker B., Joosen W., *Infrastructure for Mobile Agents*, Proc. 7th ACM SIGOPS European Workshop, 1996, pp 173-180

3. Farmer W.M., Guttmann J.D., Swarup V., *Security for Mobile Agents: Issues and Requirements*, Proc. NISSC96, 1996
4. Fugetta A., Picco G.P., Vigna G., *Understanding Code Mobility*, IEEE Trans. Softw. Eng. 24(5), 342-361, 1998
5. Fünfrocken S., *How to Integrate Mobile Agents into Web Servers*, Proc. WETICE'97 Workshop on Collaborative Agents in Distributed Web Applications, Boston, MA, June 18-20, 1997, pp 94-99
6. Fünfrocken S., *Integrating Java-based Mobile Agents into Web Servers under Security Concerns*, Proc. 31st Hawaii International Conference on System Sciences (HICSS 31), Kona, Hawaii, January 6-9, 1998, pp 34-43
7. Fünfrocken S., *Migration of Java-based Mobile Agents – Capturing and Reestablishing the State of Java Programs*, in Rothermel K., Hohl F. (eds), Mobile Agents (Proc. 2nd Int. Workshop), Springer-Verlag, LNCS 1477, 1998, pp 26-37
8. Gray R.S., *Agent Tcl: A Flexible and Secure Mobile-Agent System*, Proc. 4th Annual Tcl/Tk Workshop, Monterey, CA, 1996, pp 9-23
9. Lange D., Chang D.T., *IBM Aglets Workbench – Programming Mobile Agents in Java*, white paper, IBM Corporation, Japan, August 1996
10. Lingnau A., Drobnik O., Dömel P., *An HTTP-based Infrastructure for Mobile Agents*, WWW Journal 1, pp 461-471, 4th Int. WWW Conference, MA, Dec 1995
11. MASIF: http://www.camb.opengroup.org/RI/MAF/ ;
http://www.osf.org/∼dejan/papers/ma2.fr.ps.gz
12. Peine H., Stolpmann T., *The Architecture of the Ara Platform for Mobile Agents*, in Rothermel K., Popescu-Zeletin R. (eds), Mobile Agents (Proc. 1st Int. Workshop), Springer-Verlag, LNCS 1219, 1997, pp 50-61
13. Voyager: http://www.objectspace.com/voyager/
14. Wong D., Paciorek N., Walsh T, *Concordia: An Infrastructure for Collaborating Mobile Agents*, in Rothermel K., Popescu-Zeletin R. (eds), Mobile Agents (Proc. 1st Int. Workshop), Springer-Verlag, LNCS 1219, 1997, pp 86-97
15. Yee B., *A Sanctuary for Mobile Agents*, Proc. DARPA Workshop on Foundations for Secure Mobile Code, Monterey, CA, 1997