Infrastructure for Mobile Agents: Requirements and Design*

Yariv Aridor and Mitsuru Oshima

IBM Research, Tokyo Research Laboratory
Yamato-shi, Kanagawa-ken, JAPAN
E-mail: tayariv|moshima@trl.ibm.co.jp

Abstract. Mobile agent technology makes it possible to reduce network traffic, overcome network latencies and enhance robustness and fault-tolerant capabilities of distributed applications. However, it is sometimes difficult or even impossible to take full advantage of these technical benefits because of the lack of an appropriate infrastructure for overcoming problems related to connectivity (e.g. access through firewalls), security, location transparency, and use of proprietary tools. This paper discusses these problems and introduces the requirements for various infrastructure components and their implementation with the aim of enhancing the practicality and accelerating the deployment of mobile agents.

1 Introduction

The ability of mobile agents to travel autonomously from one host in a network to another and to use distributed resources locally, makes it possible, among other things, to overcome network latencies, reduce network loads and even enhance the robustness and fault-tolerant capabilities of distributed applications. The ubiquity of the Internet along with the resolution of heterogeneity problems by Java [5] and script languages [3], facilitates the use of mobile agents in distributed systems.

However, it is sometimes difficult and even impossible to take full advantage of mobile agents due to lack of appropriate infrastructure which enables to overcome problems related to connectivity (e.g., access through firewalls), security, location transparency and usage of propriety tools. These problems are discussed ahead in this paper.

During their work on the deployment of Aglets [4], a Java-based mobile agent system developed in IBM Japan, the authors identified several required components of a comprehensive infrastructure to enhance the practicality and accelerate the deployment of Aglets and more generally, to make mobile agents a pervasive technology. For example, it should be possible to pull agents into hosts that are not directly reachable by agents (e.g., hosts behind firewalls), to locate

Appears in the Proceeding of the 2th international workshop on Mobile Agents, September 1998, LNCS #1477, Springer-Verlag.

agents transparently and to control agents from different types of client environments (e.g., hand-held devices, browsers and desktops). These infrastructure components are the subject of this paper.

In this paper, the term "infrastructure" refers to network components (e.g. servers) as well as the services and components used by the run-time layer of a mobile agent system (e.g., the implementation of its basic functionality). Thus, agent servers, naming services, and even security services are considered parts of the infrastructure of mobile agent systems. To the best of our knowledge, the infrastructure components discussed in this paper received so far very little, if any, attention in the literature.

The rest of the paper is organized as follows: section 2 identifies the requirements for infrastructure components which are described in the following sections 3-6. These also discuss related works. The last section concludes the paper.

2 Requirements for Infrastructure for Mobile Agents

A basic infrastructure of mobile agent systems is composed of stand-alone "agent servers" and "agent clients". The former are hosts with fixed network connections, which run special engines and daemons to create, run, and launch agents onto the network or to receive agents from the network and allow them to resume their execution. An agent engine uses a language interpreter and an agent system. The latter is composed of a security manager, shared resources (e.g., whiteboards) and run-time services for agents (e.g., dispatch and communication).

The "agent clients" are software environments that allow clients to create and control agents locally (in which case they run agent engines) or directly at remote hosts. The following describes some limitations of this infrastructure with respect to the practicality and widespread deployment of mobile agents, identifying motivation for new infrastructure components.

Communication. For inter-agent communication, agents should be located by means of various schemes. For example, the most convenient way for multiple agents, working in parallel on a specific task at different hosts, to locate and communicate with each other would be to use a shared naming server. Otherwise, communication with agents originated at a specific host can be most efficiently established from that particular host, by using local logging information which indicates their location.

Thus, mobile agent applications should benefit from naming services that (1) provide location transparency for agents and (2) use multiple locating schemes, subject to performance and security requirements. Such naming services are described in section 4.

Mobility. Mobility is limited if destination hosts are temporarily unavailable (e.g., shutdown), have dynamically assigned network addresses (e.g., when dialup network connections are used) which cannot be resolved or are protected by firewalls. For example, a search agent cannot return to its origin to report results if that origin is a mobile computer connected to the network only for short durations of time. An infrastructure component, providing a general and asynchronous way to pull agents, is introduced in section 3.

Management and Control. It should be possible to control agents from thin-client environments (e.g., hand-held devices) or web browsers (i.e., HTML-based interface) as well as to integrate support for mobile agents in familiar user interfaces. Thus, clients will not be restricted to use proprietary software or to become familiar with new user interfaces in order to take advantage of mobile agents, thus allowing mobile agent technology to become more ubiquitous. A desktop-like tool for agent management and an HTML-based user interface for mobile agents are described in sections 5 and 6, respectively.

3 Agent Box

While traveling, mobile agents may have problems in reaching their destination hosts. For example, these hosts may be temporarily disconnected from the network, be located inside firewalls, or have unknown addresses (e.g. if dialup network connections are used). In some cases, simple solutions in the form of polling (e.g., repeatedly trying to dispatch the agents) work although they consume a lot of network resources. In other cases, such as when a host is located behind a firewall, such solutions never succeed.

Another solution is to pull agents; that is to say, a host initiates a request to receive an agent from a remote host. This can be done by means of the `retract` operation, which is unique to Aglets [1]. However, (1) this operation can only pull a single known agent at a time, and (2) in practice, it would be necessary to write extra code allowing the target agent to suspend `retract` operations until it has completed its current task (including the release of local resources) and is ready to be pulled.

A general and asynchronous mechanism for overcoming problems of connectivity in distributed systems is queuing (e.g. e-mail and message-oriented middleware) [9]. In the context of mobile agents, hosts that may not be reachable by agents can use private queues (termed "agent boxes"), located at remote sites, where incoming agents can be queued and later be pulled by these hosts when they become available or reconnected to the network. In addition, hosts behind firewalls can use private queues outside their firewalls from which they can pull agents and host them. Agents inside agent boxes can be saved as streams of code and data and protected against unauthorized clients.

An agent box has also other benefits:

- It facilitates sharing of agents. In clusters of machines, agents can be pulled from a shared queue and run them simultaneously, for example, to improve load balancing.
- It provides a secured solution for accepting agents inside firewalls or security domains. Internal hosts will pull agents from agent boxes located outside these domains or firewalls. The agents themselves will be authenticated by

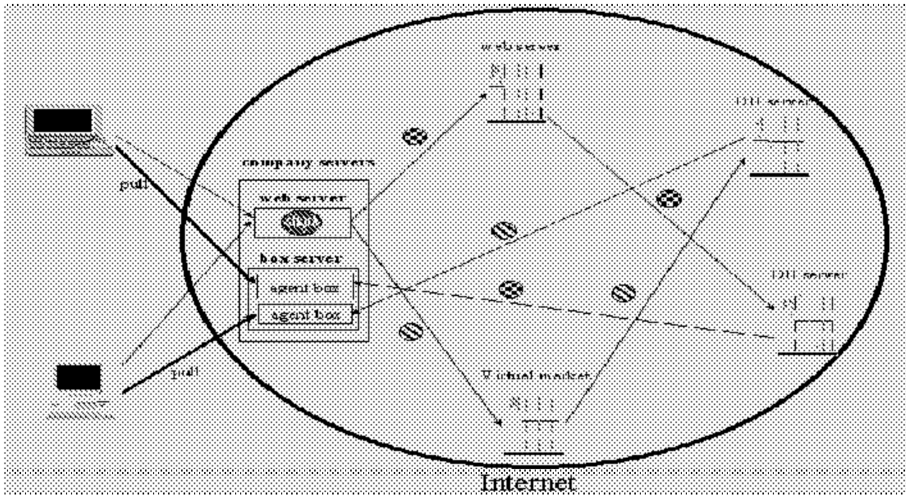

Fig. 1. A Usage Model for Mobile Agents Using Agent Boxes.

the corresponding agent box server just before they are queued. The main advantage over other solutions like external dedicated aglet server is that with an agent box server aglets are not active and so malicious aglets cannot harm the server. A similar idea was recently introduced in [2].

Figure 1 shows a usage model of agent boxes that incorporates some of the aforementioned benefits. Consider, for example, a company that provides mobile agent-based services (e.g., e-commerce agents and remote monitoring agents) via home pages (see section 6). Once a client downloads these pages and instantiates mobile agents (denoted by the circles), an agent box is created (with an expiry date) at the company's servers and the agents are initiated with the address of the box. The client is notified of its box's address via HTML pages created dynamically when its agents are created. The agents will provide the services, after which they will be queued in the client's box. [2]

The client, running an "agent client" configured to utilize its box, will pull the agents from the box after being reconnected to the Internet and becoming ready to host them. The key features of this model are that it (1) overcomes the problems of connectivity with clients' machines (2) keeps private the network addresses of clients' machines (which might be preferable for some clients).

As part of Aglets, we implemented a prototype of an aglet box server. The following design decisions were taken:

- Aglets are queued in agent boxes via the `dispatch` operation for transfer to

[2] While a mail box could be considered as an alternative to an agent box, in practice, clients might not want to expose their e-mail addresses or may not use e-mail readers capable of receiving and instantiating mobile agents.

remote hosts. [3]

- A `dispatch` operation to an aglet box returns successfully, immediately after the agent is queued.
- A proprietary message (i.e. not an aglet operation) is used to transfer aglets from the boxes to their final destination. At the destination, the aglets are instantiated as if they were all received directly from the network.
- Sending messages to aglets queued in boxes causes an exception which indicates that these messages cannot be handled as messages which are sent to aglets which have been serialized and saved on disks at some hosts.

We believe that the above integration of agent boxes in Aglets is also applicable in the context of other mobile agent systems, with minor modifications. The idea of an agent box is similar to that of a docking system used by Agent TCL [3]. The latter is a server, paired with a laptop, in which agents, originally sent to the laptop, can be queued while the laptop is disconnected. From the available documentation, we realized that a docking system is a system-level mechanism which is used to overcome connectivity problems only while laptops are disconnected, whereas an agent box is a more explicit mechanism which can be used at the application-level, to support all the other aforementioned benefits.

4 Naming Services

Locating agents in the network enables remote inter-agent communication and remote agent management. For example, the owner of a remote agent wants to send it a control message (e.g., dispose) or customize its behavior. Another example is a case of multiple agents dispatched to different hosts in order to perform tasks in parallel which later need to communicate in order to exchange temporary results or to synchronize. This section discusses several schemes for locating mobile agents and for the delivery of messages between them and presents their application in the context of Aglets.

4.1 Schemes for Locating Agents

Figure 2 shows three basic schemes for locating agents. Rectangles denote hosts while arrows denote massaging for locating agents. These schemes are:

Logging. An agent is located by following trail information, indicating its next destination, left in every agent server it already visited. Trail information for disposed agents can be garbage-collected according to, for example, the expired time or explicit notification by agents.
Agents can be located by trail information via two approaches, as shown in Figure 2. While redirection involves more communication than tracking, it is a simpler approach; intermediate hosts don't have to dedicate resources

[3] In this case, the `dispatch` operation uses the address of a specific aglet box, instead of an aglet server.

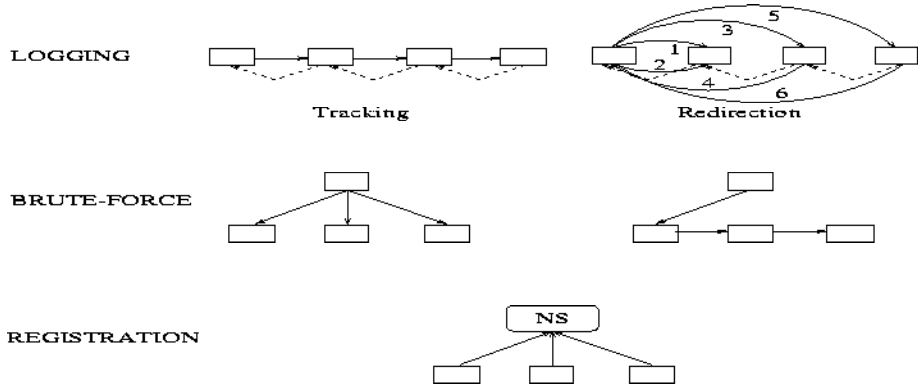

Fig. 2. Multiple Locating Schemes for Mobile Agents

to initiate requests and wait for their results. In both cases, one possible enhancement is, after an agent has been located, to update its current location in previously visited hosts (see dashed lines in figure 2) so that next time, it can be directly located from any of these hosts.

Brute Force. An agent is located by searching for it in multiple destinations. Searching can be done in parallel or in sequence.

Registration. An agent updates its location in a predefined directory server that allows agents to be registered, unregistered or located. Other agents use the directory to locate the agent.
In practice, communicating agents need to agree in advance upon a naming server. Such agreement can be simplified by adopting an architecture in which every agent server is associated with one available naming server. Then, if aglets share the same origin or arrive at the same host (see the aforementioned cases at the beginning of this section), they can simply share the default naming server associated with their origin instead of locating and exchanging explicit addresses of naming servers, in advance.

The selection of a specific location scheme is subject to security and performance requirements. The following are some examples:

- The Logging scheme needs to know the origin of the target agent or any other host it has previously visited. Otherwise, the Registration or the Brute-Force schemes might be preferable to searching for trail information itself. A typical case in which Registration is used is that of multiple agents, working in parallel on a specific task at different hosts, which try to locate and establish communication with each other. In this case, the trail information necessary to them for locating each other is not available.
- The Brute-Force scheme is simple to implement within an application; agents can be used to locate other agents. In addition, it is independent of external mechanisms like trail information or external naming servers. Thus, if agent hosts contacted for tail information or naming servers are overloaded or

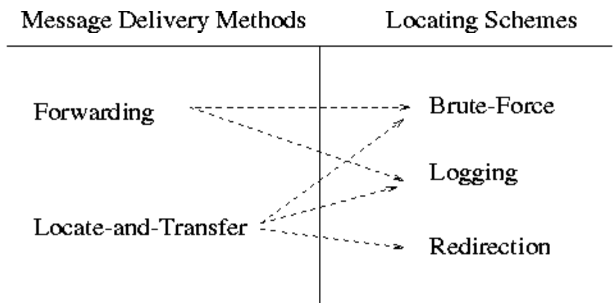

Fig. 3. Methods for Message Delivery Between Mobile Agents

temporarily unavailable, it it still possible to locate agents by the Brute-Force scheme.

4.2 Methods for Delivery of Messages between Aglets

There are two basic methods for delivery of messages, as indicated by figure 3:

Locate-and-Transfer. An agent is located after which the message is transferred directly to it; in this case, two separate phases are used.

Forwarding. Locating a receiver aglet and the delivery of a message to it are both done in a single phase (e.g. the message may be redirected by using trail information).

There are two main differences between these methods of message delivery:

- Locate-and-Transfer may not always give the locations of agents accurately, since they may be dispatched during the second phase of message transfer. With Forwarding, such cases can be eliminated, since agents are located "on-the-fly" during the delivery of messages to them[4].
- Forwarding may be more efficient than Locate-and-Transfer in the presence of small messages. Otherwise, it might be more efficient to locate an agent and then transfer a large message directly to it.

Each of the above methods can use multiple locating schemes, as shown by figure 3. For example, Forwarding can be done by redirection of messages (the Logging scheme) or by broadcasting messages to multiple hosts where agents may be located (the Brute-Force scheme)

[4] During forwarding, messages are temporarily queued while the receiver aglets are being transferred [11].

4.3 Naming Services for Aglets

As part of the Aglets, we prototyped naming services to provide aglets with location transparency and message delivery. These services have the following key features:

- Applications can benefit from multiple methods for message delivery based on multiple locating schemes.
- While a scheme for delivery of messages to an aglet is defined at the application level, the aglet system handles all the low-level details for communicating with that aglet even after it has been dispatched to other hosts.

Aglets communicate by message-passing; a sender aglet sends a message to a proxy object representing the receiver aglet, after which the proxy forwards the message to its aglet. The integration of naming services in Aglets is done by implicit use of different types of proxies, all of which are instances of subclasses of an **AgletProxy** base class, as shown in figure 4. Using these three types of proxies, messages can be delivered in any of the ways indicated by the arrows in figure 3. The specific type of proxy objects to be used is determined as follows:

- Forwarding based on the logging scheme (via tracking) is naturally the default method of message delivery, since it does not require any application specific information such as a specific naming server. The following is a segment of code for sending a message to a remote aglet:

  ```
  AgletProxy  p= <create an aglet>;  // Forwarder proxy is created
  ...
  if (p!=null) {
      p.dispatch(<destination>);
  }
  ...
  p.sendMessage(msg); // a message is forwarded.
  ```

  Once an aglet is created, it has a **Forwarder** proxy. Before being dispatched to a new destination, the aglet creates trail information at the current host, specifying its next location. The **Forwarder** proxy uses the trail information to later forward messages to this aglet.
- Registered aglets can be located by a **Locator** proxy. The following is a segment of code for sending a message to an aglet that is registered at the naming server associated with the current aglet server.

  ```
  AgletFinder finder = <an object which encapsulates access
                        to a naming server>;
  AgletProxy p = finder.lookupAglet("Aglet A");
  if (p!=null) {
     p.sendMessage(msg);  // a message is transferred
  }
  ...
  p.sendMessage(anotherMsg); // another message is transferred
  ```

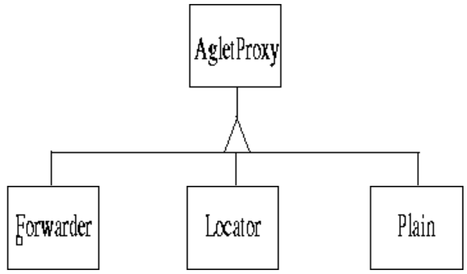

Fig. 4. Different Types of Aglet Proxies

The **lookupAglet()** returns a `Locator` proxy for aglet A, that maintains the address of the naming server. Then, whenever it is used to send a message to aglet A, it will find its current location via that naming server (assuming that aglet A updates its location at that naming server every time it is dispatched to a new destination) and transfer the message.

– If the location of remote aglets is known (e.g., if location information was previously made known to other aglets), then the `Plain` proxy is used to simply transfer messages to these aglets, as shown by the following code segment:

```
AgletProxy  p = this.getAgletContext().getProxy(url,id);
if (p!=null) {
  p.sendMessage(msg);  // a message msg is transferred
}
```

The location of the receiver aglet and its identifier are represented by the **url** and **id** objects. The **getProxy()** returns a `Plain` proxy object, which sends the message to that aglet.

Few other mobile agent systems support naming services for location transparency of agents based on limited schemes: Voyager [7] supports only forwarding of messages (based on the Logging scheme) while Odyssey [6] supports Registration via an interface to a naming server, which is configured with an agent system. In contrast, MOA [10] supports multiple schemes. In comparison with Aglets, MOA uses special location objects to encapsulate the current location of an agent, as well as the specific locating scheme to locate it. Before an agent is located, its location object needs to be located in naming servers. Finally, as mentioned earlier, MASIF standardizes the locating of agents via Registration for heterogenous type of agents [8].

5 Agent Viewers

Clients may want to use interactive tools to manage (e.g., to create, dispatch and dispose agents) local agents in their "agent clients". Figure 5 shows an innovative desktop aglet viewer in which aglets and locations are represented as

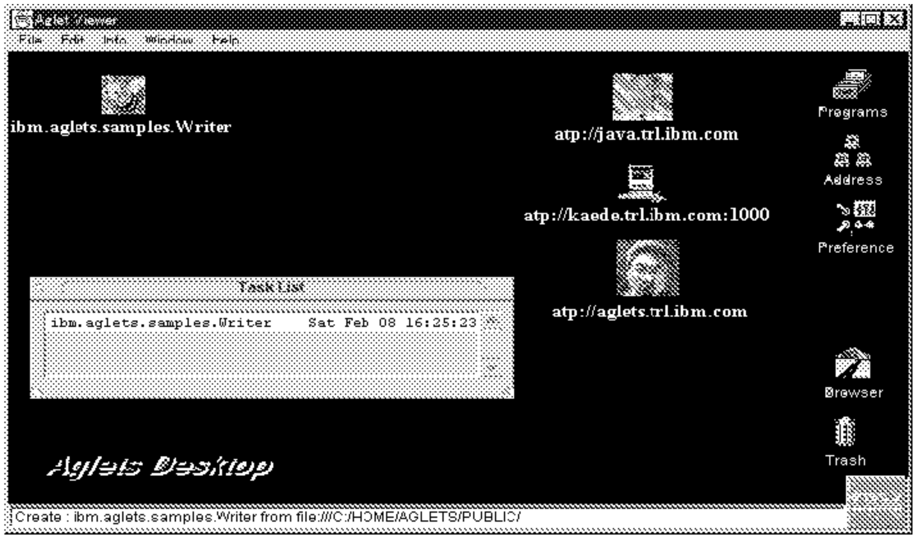

Fig. 5. A Desktop Aglet Viewer

icons. Operations such as **dispatch** and **dispose** are performed by dragging and dropping icons of aglets (e.g. the bee icon at the top-left corner) onto icons of locations (e.g., icons with title prefixes of "atp") or the Trash icon (bottom-right corner), respectively. Future versions will have icons of itinerary objects (i.e., objects encapsulating multiple locations and a scheme for routing between them) in order to be able to save hot "itineraries" (e.g., lists of database servers in which to repeatedly search for or monitor information) or even pop-up windows to allow remote communication with aglets. Although we do not expect all clients to use this tool as it is, it presents some initial ideas for embedding support for mobile agents in current desktop user interfaces.

6 Integration with the Web

The World Wide Web is a de facto ubiquitous infrastructure. Thus, new software technologies for operations such as audio/video streaming or pushing information are being integrated with the Web (e.g., using the HTTP protocol to deliver messages) to allow widespread deployment. The same motivation holds for mobile agents. For example, the ubiquity of Web browsers makes them perfect client tools for creating and communicating with remote agents. Clients are not required to install any proprietary software or familiarize themselves with new user interfaces. This section describes the advantages of the integration between mobile agents and the Web, in the context of Aglets.

6.1 Aglets inside Applets

Aglets, which are written in pure Java, can be created and launched by applets. These applets are downloaded with an aglet library to support manipulation of aglets. [5] Since an applet can establish communication only with its original host, aglets launched by it are transparently transferred to a special router server running at the original host, after which they are dispatched by that router to their destination hosts.

Compared with stand-alone "aglet clients", clients do not have to install any kind of software (not even Java) in order to run aglets inside applets. In addition, aglets that are downloaded together with their applets and run inside the browsers can be authenticated as part of the applet authentication and controlled by the applet security manager. They are likely to be more readily accepted by clients, at least in the near future, than stand-alone "aglet clients" with proprietary security mechanisms.

6.2 HTTP Messaging

The idea behind HTTP messaging is the usage of HTML as a user interface for remote agents. In Aglets, this is done via HTTP messaging: HTTP requests are used for the delivery of messages *directly* to remote aglets. These HTTP requests use a URL of the form:

```
http://mall.server:port/aglets/YourShopingAglet/report?type=brief
```

The path and arguments parts of the URL include the identity of a receiver aglet (e.g., YourShopingAglet), the message type (e.g., report) and parameters (e.g., type), respectively. The location of an aglet is specified by the host part of the URL.

As shown by figure 6, upon receiving such an HTTP request, an aglet server locates the receiver aglet and delivers the message directly to it. The aglet handles the message and sends a reply which is carried back by the corresponding HTTP response. The reply can be an HTML page created dynamically. Clients can create remote aglets by sending messages to known stationary aglets to create them. Moreover, by using naming services, the server can locate and forward messages to aglets that have already dispatched themselves to other hosts.

In practice, web sites and kiosks offering aglet-based services allow clients to control aglets within HTML pages; clients only need to click on HTML links that are actually HTTP messages to aglets in order to create remote aglets or send messages to remote aglets. In addition, (1) aglets can return HTML pages that are created dynamically to present actual results or to initiate communication with their users and (2) links that are HTTP messages to aglets can be saved as bookmarks so that clients can easily reconnect to their aglets whenever they want. No additional information needs to be saved. HTTP messaging is a very lightweight mechanism. As such, it is applicable to a broad range of clients devices.

[5] Alternatively, the aglet library can be installed as a plug-in, to improve performance.

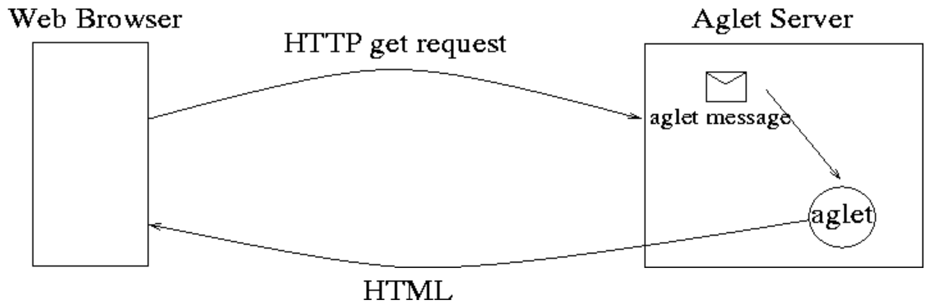

Fig. 6. Message-flow in HTTP Messaging

7 Conclusion

We have presented some requirements and designs of several infrastructure components in order to enhance the practicality of mobile agents and to accelerate their deployment on the Internet. Deployment is not considered simply in terms of how well mobile agents are integrated with the existing infrastructure (e.g. the Web), but also in terms of usage models (e.g. agent boxes) that are likely to be adopted by, for example, the Internet community.

Another aspect of deployment to be considered is the compatibility between software models of mobile agents and other models. For example, how compatible are the models of Java-based mobile agents with a component model like Java Beans. This issue influences the necessity for programmers to learn propriety models and the adequacy of available development environments (e.g., Beans editors) for development of mobile agent applications, thus indirectly influencing the deployment of mobile agents.

8 Acknowledgments

We would like to thank Danny B. Lange, the inventor of aglets, for pointing out the significance of the subject and for fruitful discussions. We are also grateful to Mike McDonald of IBM Japan for checking the wording of this paper.

References

1. Lange B. D. and Oshima M. *programming and deploying Java mobile agents with Aglets*. Forthcoming book, Addison-Wesley, 1998.
2. Ad Astra Eng. Jumping Beans. http://www.JumpingBeans.com.
3. Kotz D. et al. Mobile agents for mobile Internet computing. *IEEE Internet Computing*, 1(4):58–67, 1997.
4. IBM. Aglets Workbench. http://www.trl.ibm.co.jp/aglets.
5. Arnold K. and Gosling J. *The Java Programming Language*. Addison-Wesley, 1996.

6. General Magic. Odyssey. http://www.genmagic.com/agents/odyssay.
7. ObjectSpace. Voyager. http://www.objectspace.com/voyager.
8. OMG. Mobile Agent System Interoperability Facility (MASIF). OMG TC Document ORBOS/07-10-05, 1997.
9. Orfali R., Harkey D., and Edwards J. *The Essential Client/Server Survival Guide.* Wiley Computer Publishing, 1996.
10. Milojicic D. S., LaForge W., and Chauhan D. Mobile objects and agents (MOA). In *Processing of the USENIX Conference on Object-Oriented Technologies and Systems (COOTS)*, April 1998.
11. Kim W. and Agha G. Efficient support for location transparency in concurrent object-oriented programming languages. In *Proceeding of Supercomputing*, 1995.

This article was processed using the LaTeX macro package with LLNCS style