Customisable Off-line Web Browsing with Mobile Software Agents

A. Yew, G. Pavlou
Centre for Communication Systems Research
School of Electronic Engineering and Information Technology
University of Surrey, Guildford, Surrey, GU2 7XH, UK

Abstract-**This paper presents a server-side multi-agent architecture aimed at increasing the efficiency of Internet browsing. The agent system uses the user profile to monitor the user's favourite web sites, to construct customised views of those sites, and to inform the user through a web interface. The proposed agent architecture allows customisation of the user's preferences, providing the user with a virtual home environment for the web. It therefore optimises the user's regular browsing pattern in a location transparent manner. After describing a prototype implementation, the paper discusses architectural alternatives aimed at improving its scalability.**

I. Introduction

There are significant efforts within the information technology industry towards developing applications that allow users to customise the Internet to their needs. As users usually have a regular browsing pattern, it is beneficial to develop an application that optimises a user's time based on this pattern. One of the first of such applications allowed the user to bookmark websites that they often visit through a web browser. Users no longer had to remember the exact uniform resource locator (URL) of a website, and accessing their favourite sites became as easy as clicking on an icon located on the web browser. Microsoft Corporation then tried to extend the limits of customisability adding a new feature called 'synchronisation' to their browsers that allowed all user-specified book-marked websites to be downloaded to the computer's cache at a single click of a mouse button. Internet connection time and costs were thus reduced, and cache copies of relevant web pages could be accessed without connecting to the Internet. More importantly, 'synchronization' allowed users to specify if they wanted the links of the book-marked website to be downloaded as well.

However, certain flaws in using 'synchronize' reflected the limitations of client-side software in accomplishing user customisation of Internet content. Firstly, synchronisation does not take into account whether the web pages it downloads are actually of relevance to the user. For example, a user interested only in shoes from an on-line shopping website uses the 'synchronise' function to check for any sales of shoes; Internet explorer would download the web page whether or not it contains any information on sales. The second flaw with the 'synchronise' method is that, until the download is complete, the browser has absolutely no idea if the page is different from the one it downloaded previously, thus incurring connection charges for executing a superfluous task.

Portals represent another effort that allows users to customize their browsing. Portals compile popular topics on the Internet, such as news, into a single web page, providing users with a one-click solution to finding information. However, portals such as America On-Line (AOL) and the Microsoft Network (MSN) only give users limited amount of customisability. Information provided by portals usually originates from sources that are defined by the provider of the portal. Therefore, users have no way of influencing the source from which information they want is provided.

There exists a need to merge the concepts of a portal and bookmarks together in order to provide a higher level of customisation for the Internet user. The latter should be provided "within" the network, i.e. as a Web server offering the proposed customisation service, and not at the user's terminal so that the service can be accessed from any terminal with Internet connectivity. This paper aims to propose, through the use of software agents, the concept of providing the Internet user a customisation service similar to that of a "virtual home environment".

The virtual home environment introduced above is similar to the same concept of the third generation mobile communication systems as specified by the 3rd Generation Partnership Project (3GPP). 3GPP specifies that end users must be able to access and customise their services independently of their terminal capabilities and location [TS-3GPP]. In this paper, the service provided is similar to that of a portal in which the user has control over the source of information. Furthermore, users are provided with additional capabilities to customise the content and delivery of relevant information from the web, such as keyword filtering and email updates. The service is location independent since it is located on a Web server, and can therefore be accessed from any location with Internet access. Any terminal can be used to access the service if it can communicate using HTTP [Bren-Lee], and understand

HTML. This implies that the service is “browser independent”.

II. Software Agents And Information Retrieval

Software agents are autonomous programs that can perform tasks and operations on behalf of a user. Agents can either be stationary or mobile. A stationary agent performs its tasks in the same physical environment in which it was created; a mobile agent travels to different locations in a network to perform its tasks. An agent will have a repertoire of actions enabling it to react to changes in its environment; these actions represent the agent’s ability to modify its environment. An agent and the environment in which it is executed are closely related to each other: a change in the environment might cause an agent to execute an action in response, which in turn result in repercussions on the environment.

With regards to object-oriented design, the concept of an agent can be easily confused with that of an object. There are three key differences between the two [Weiss]:

1) The locus of control, i.e. the decision whether or not to execute an action, lies with the agent.
2) Agents have flexible behaviour.
3) Agents are considered to have their own threads of control.

For a server to accommodate mobile agents, an agent platform must be present to receive the mobile agents. Within the agent platform, there is a hierarchy of generic components that provide certain services to the agents. A server must contain an *agency*/*engine*. An *agency*/*engine* is a virtual machine that allows, within a confined location on the server, agents to execute its code and *places* to exist. *Places w*ithin each agency represent the logical grouping of agents. The concept of a *place* allows agents to meet and if necessary, communicate with each other.

The issue of web reconnaissance has been investigated by the Letizia agent system developed by Massachusetts Institute of Technology [Liber]. When browsing a page that has many links, the user usually goes to each one to check its relevance to him/her, and this is where the Letizia agent architecture becomes really effective. The static agent will spawn and send an agent for each available link to check, on the user’s behalf, the relevance of the page. The advantage of spawning agents is to delegate a task to many agents whenever a task is deemed too tedious to be performed by a single agent.

Ideally, the architecture would be implemented by using mobile agents for monitoring a web site in the following manner: A mobile agent would travel to the server hosting the web page and be notified of updates by the server itself. The primary benefits of making the agent mobile in this case are the significant reduction of traffic in the network incurred by regularly polling a web site to check for new content, and the immediate notification of any updates by the mobile agent to the user. The latter is important as it determines the reactivity of the system to changes in a dynamic environment, which is the Internet: a higher reactivity ensures a better service for its users. In reality, however, the increasing number of security infringements into current commercial web sites implies that owners of these web sites are less likely to allow foreign mobile agents on their top-of-the line web servers. Therefore, designing mobile agents to travel to remote commercial websites to monitor web pages would be impractical. However it is possible to send agents to remote proxy servers located close to the commercial website so as to reduce the time needed to poll these sites. This approach was considered for the proposed approach in this paper.

III. System Implementation

A. *The proposed multi-agent architecture*

The components and types of agents proposed in this paper, and their respective responsibilities are presented below. Fig. 1 shows the relationships between the components, and the various agents belonging to various *places*, in the system.

1) Java servlets *(the amount varies per system)*: They collect relevant information from the user and pass them to the task agent. They present the results of the task agent on a single web page. They ask for user feedback to determine if a page is relevant to the user. They enable the user to define and customise his/her preferences

2) Task agents *(1 per user)*: They receive news of updates and new links from the Info agents. They spawn a new information agent for every new link in a monitored web site. They delete information agents that are no longer required. They filter the copy of the web page for user-defined keywords. They alert the user to updated websites via the Email agent. They send results consisting of updated websites back to interface agents. They suspend an Information agent at its request.

3) Info agents *(1 per web page)*: They monitor and retrieve contents of a web page from a remote host. They note the URL for all the links on the page. They are able to adapt the rate in which it retrieves web pages. They inform a task agent if the website has been updated. They inform a task agent about the new, and the redundant links, contained in the website. They inform a task agent about the location (filename) of the local copy of the web page.

4) Email agent *(1 per system)*: This agent sends information passed on by the task agent to the user via e-mail.

The Task agent acts as a mediator for the Java servlets, the Email and the Info agents, decoupling the various agents from one another. Its purpose is to liase with both the Info and the Email agents to present the results of the system to the user through the Java servlets. It also holds all aspects of the user's preferences for the service, which it uses to control its interactions with the other agents. Using the mediator design allows the architecture to be easily adaptable, because changes in code for an agent's class need to be reflected only in the Task agent and the affected agent class [Gamma]. The task agent also utilises a separate class *(Filter)* to perform the filtering of the website for keywords Figure 1 shows that the Info agent utilises a separate class (*Weblinks)* to check for new and old links of the website. The reason for using a separate class in both cases is to facilitate the modification of the respective behavioural aspects of the Info and Task agents.

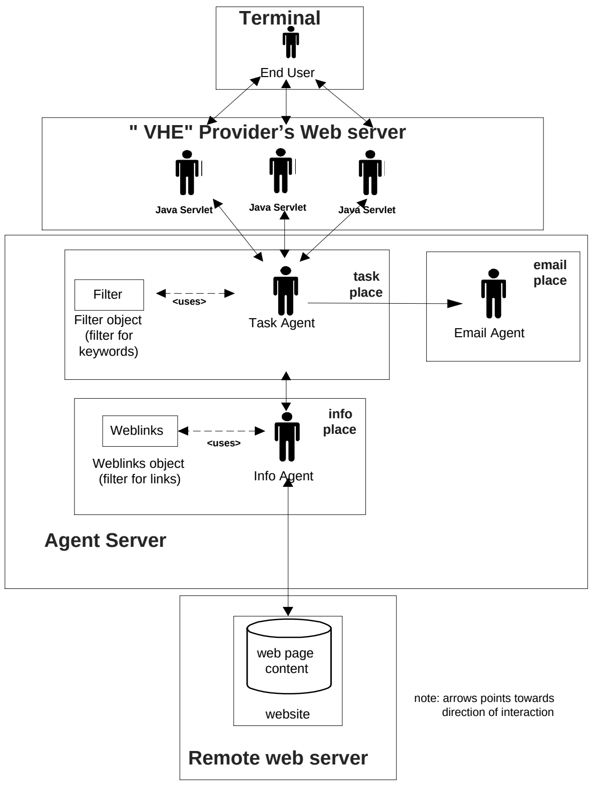


Fig. 1. Relationships between components and agents in the system

Fig. 1 also presents the logical separation of the various agents into different places within an agency on a single server with the exception of the Java servlets located on a dedicated web server. Placing the servlets on a separate web server increases the scalability of the overall system as the system is able to support more HTTP requests from users without affecting the performance of the agent server.

B. *Efficiency of the multi-agent architecture*

Efficiency is a very important design consideration for the proposed multi-agent architecture. The overheads in the proposed system can be split into two categories: cost of maintaining the system; and the amount of processing required for the system to work. The amount of processing incurred by the system is related to the number of software agents in the agent system, and affects the scalability of the system. Therefore, it is imperative that the design uses a minimal number of agents needed to accomplish the specified aims. The cost of maintaining the system is affected by the amount of network traffic generated by it. This implicitly refers to the polling of remote websites by the system to check for updates. This section will explain the design decisions that improve the efficiency of the architecture.

1. *Reducing network traffic*

When a web page is checked for an update, a copy of the web page must be downloaded and checked against a previous copy. The downloaded web page also allows the filtering of keywords to be performed. The writing of a website's content to a temporary local file, and the filtering for keywords, require large amounts of processor time. As remote websites have to be polled periodically by the Info agent for updates, websites that are updated rarely should be polled less frequently and vice versa. Therefore, a method for optimising the frequency of polling is required to reduce the connection costs and the processing incurred by the system.

Controlling the frequency of polling a web page typically requires the Info agent to implement a state machine internally. This state machine allows the Info agent to keep track of the time between updates by using an internal counter. However, the Info agent needs an input or an event to change its state. In this case, this event would be a sign indicating the lapse of a certain period of time. For example, consider an agent that is optimised to poll a website every 2 days, and receives an input every 2 hours. Therefore, it can logically deduce that it must perform an update after receiving 24 consecutive inputs, and uses its internal counter to remind itself of the number of consecutive inputs it had received. The need to maintain a state machine for each web page also implies that one Info agent monitors only one designated web page.

The rate of change in the optimum polling interval of a web site is as follows: in the case of an update, it increases by the specified minimum polling interval until it reaches 24 hours, after which it doubles until the specified maximum polling interval is attained; in the case of the web page being unchanged, it halves until it reaches 24 hours, after which it decreases by the specified minimum polling interval until the specified minimum polling interval is attained. This algorithm allows the system to be more responsive to web pages that are updated very often. The change in the optimal polling interval as determined by the algorithm is shown graphically in Fig. 2.

2. Reducing the amount of processing required

There are two ways of reducing the amount of processing generated by the proposed system. The most obvious method is through optimising the algorithms used in the system, as exemplified by varying the frequency of polling websites for updates. Another approach is to minimise the number of agents present in the system at all times. This implies that any Info agent monitoring a redundant web page should be removed from the system. There are two ways in which a web page can be redundant: the user does not want the system to monitor the web page anymore; the web page ceases to exist (eg. a dead link). Therefore, all Info agents maintains a list of Task agents that wish to be informed of updates in the web page that it is monitoring. Task agents can request to be taken off this list through user intervention (eg. users can change their preferences of web pages to be monitored). When an Info agent realises that the list containing interested Task agents is empty, it deletes the copies of the web page from the server's hard disk, and then removes itself from the system. This ensures that the number of Info agents in the system is controlled, and keeps the amount of processing required by the agent system minimal.

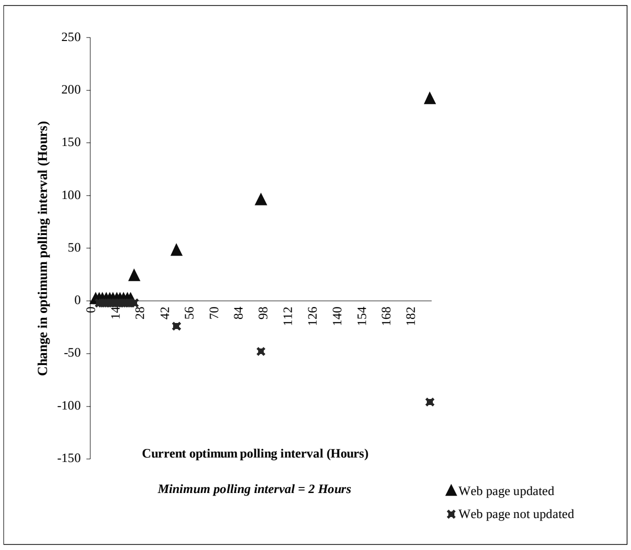
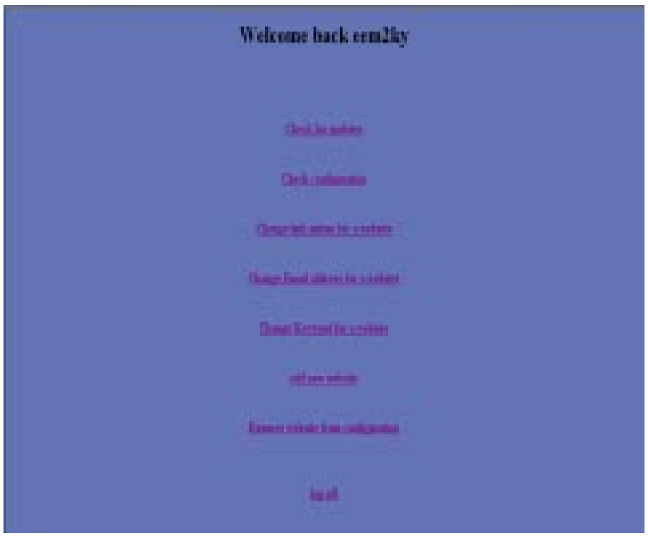


Fig. 2. Change in the optimum polling interval as determined by the algorithm

C. Overview of operations performed by the system

This section presents the operations performed by the agent system for a given scenario. Users have to go through an authentication process before they can access the service. Once authenticated, users are presented with a 'welcome' page and can then proceed to customise the service according to their needs. This requires the URL of the website that they want monitored to be typed in, with the added options of entering a key word for each website, and of being updated of a website's changes through e-mail. Fig. 3 gives a snapshot of the 'welcome' page, while Fig. 4 shows the graphical interface in which users can enter their preferences.

A Java servlet notes all these parameters and sends them to the user's task agent. If, as in the case of a new user, the latter has not been assigned, a Java servlet will then create a task agent for the new user. If there are no Info agents monitoring the user-specified website, the task agent will then create and initiate a new Info agent. Otherwise, it will request that the Info agent includes in its 'interested Task agents' list. The Info agent will from then on communicate directly with the task agent when the website is updated.


Welcome back eem2ky

Check for updates

Check configuration

[illegible]

Change Email address for a website

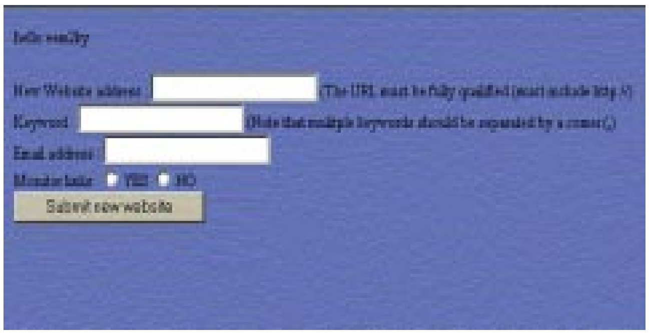
Change Keyword for a website

add new website

Remove website from configuration

log off

Fig. 3. The 'Welcome' page


hello eem2ky

New Website address: (The URL must be fully qualified (must include http://)

Keyword: (Note that multiple keywords should be separated by a comma(,))

Email address:

Monitor links: YES NO

Submit new website

Fig. 4. The 'add new website' customisation page

The Info agent monitors the website for new information, and notes the URLs of both new and redundant links contained therin. The Info agent will also adapt the rate at which it checks the web site according to the frequency with which the website is updated. In the event of an update, the Info agent saves the website's contents into a file, informs the task agent of the new and old links, and the filename of the copy of the website. The Task agent will note the URL of the website and the location of the file when it interacts with the information agent. The next paragraph describes the sequence of events when the user specifies that links of the website need to be monitored for new content as well.

If the Task agent finds a new linked web page, it checks if this web page is being monitored by another Info agent; if necessary, it will spawn another Info agent to download the contents of the linked website. Otherwise, it contacts the linked web page's Info agent asking to be included in its list of task agents to be notified of an update. The task agent also informs all information agents representing the redundant links that it wishes to leave their notification list. If filtering for keywords is required, the Task agent accesses the file containing the copy of the website and checks it for occurrences of the user-specified keywords. If the web page does not include the keywords, the Task agent will assign a negative result to this web page and will not inform the user about it as it is deemed irrelevant.

If an email alert is needed, the Task agent sends the relevant alert message to the user via the Email agent. Upon completion of all the above tasks and interactions, the task agent compiles, based on the user's preference, all the results, and presents them to the user on demand through a Java servlet.

IV. SCALABILITY

A. *Estimated scalability of the system*

Measurements were taken to estimate the scalability of the implemented system in terms of the number of web pages that it can monitor. The system was implemented using the Grasshopper2 agent platform [Grass] with the Java 1.2 standard development kit on Microsoft Windows 98. Grasshopper2 was chosen because of its compliance with the MASIF specifications [OMG]. The agent server used was an Intel Pentium III 450 Megahertz processor on an Intel 440BX platform with 256 Megabytes of PC-100 physical memory. It was assumed that the processing time available for Info agents to determine if a web page has been updated is only 30% of the specified minimum interval between polling. The other 70% of processing time is allocated to other tasks which includes the creation of new Info agents, the accessing of Info agents by Task agents, and the process undertaken by Info agents to update their state machine. By measuring the average time taken for an Info agent to determine whether a web page is updated and considering the above assumptions, an estimate of the maximum number of web pages that can be monitored is shown in Fig. 5. The monitored web page was located on a local server, and the web page remained unchanged for each measurement. The latter condition required more processing by the Info agent as each character in the web page had to be checked against the local copy. However, the theoretical number of possible monitored web pages shown in Fig. 5 represents a best-case scenario as it assumes that the optimum polling intervals of all Info agents are equal to the minimum interval between polling specified by the system administrator. In reality, there will be a mixture of optimum-polling intervals from various Info agents present in the system. An important point that can be deduced from the calculations is that the maximum number of web pages that the system can monitor is related to the minimum interval between polling specified by the system administrator. Therefore, increasing the responsiveness of the system will result in a reduced number of web pages the system can monitor.

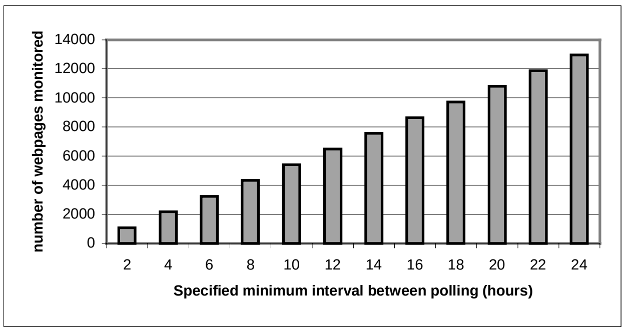


Fig. 5. Estimated scalability of the system

B. *Improving scalability using mobile agents*

It is possible to increase the scalability of the system by increasing the amount of processing time made available to Info agents to check for updates. This would require the reduction of the number of other tasks, such as the creation of new Info agents, at the agent server. Fig. 6 shows an example configuration of the proposed system where the respective agents and their various *places* are distributed over different servers. Allocating a server to each place allows the server to dedicate all of its resources to processing a few specific operations only.

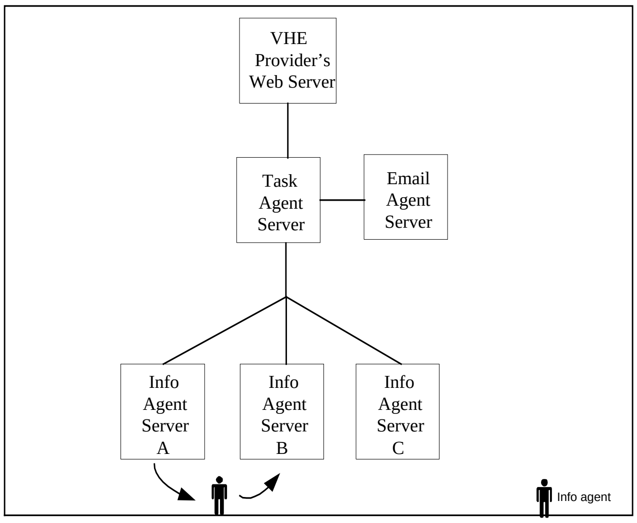


Fig. 6. A distributed view of the system

There are three different Info agent servers (A, B, and C) shown in the diagram. In this example, each of these servers has a specific purpose: Server A hosts Info agents whose optimum polling interval is less than a day; Server B hosts Info agents whose optimum polling interval is equal to, or more than a day; Server C allows the initialisation of new Info agents. After the new Info agents are initialised, they will move to the server that matches their optimum polling interval (which is always equal to the minimum polling interval). By specifying such constraints on the system, each server can provide more processing power to the agents it hosts as the variety of polling intervals used by Info agents present in any given Info agent server is reduced. The dedication of a separate server for initialising new Info agents serves to alleviate server A from performing such duties. The delegation of this task to server C is extremely important to the server hosting the group of Info agents that poll websites most frequently (server A). This is because the latter hosts the smallest number of Info agents (as was deduced from Fig. 5).

In order to support such a distributed system, the Info agents have to be mobile as their optimum polling interval can vary. Fig. 5 illustrates this concept where an agent from server A travels to server B when its optimum polling interval has been adjusted to the value of more than one day.

V. Summary

This paper introduced the concept of a customisable Web-browsing service realised through mobile software agents. This service provides the Internet user with added control over the way relevant information is conveniently presented by combining the functionality of browser bookmarks and Web portals. The use of a multi-agent architecture was explored as a possibility of providing that service. The architecture was implemented on the server side so as to prevent dependency on any particular browser technology, and to allow users to access their customised "home environment" from any location. As such, the service relates to some extent to the 3GPP concept of virtual home environment.

Software Info agents were used to perform off-line browsing on behalf of their users and implemented a polling-based algorithm to improved efficiency. The mobile characteristics of these agents allowed them to move to agent servers located close to the source of the information, thus reducing the time needed to download the web page. The mobile nature of the Info agents was also exploited to improve scalability within the same location. This meant that Info agents had to move within different servers on the same location as their polling frequency changes in order that a higher number of Info agents can be hosted.

While this service concentrated on Web-browsing, similar services could be provided for getting on or off-line access to users' home files and e-mail messages. In the case of on-line access, agent-based intelligent filtering of information at the home site and caching at the visited site, in combination with knowledge of the state of the network, can provide a true virtual home environment for the Internet.

Acknowledgment

A. Yew thanks G. Pavlou, A. Liotta, C. Bohoris, and S.C. Tan for their comments and support during the writing of the paper.

References

[Bren-Lee] Fielding, R., Gettys, J., Mogul, J.C., Frystyk, H., Masinter, L., Leach, P., and Breners-Lee, T.: 'Hypertext Transfer Protocol – HTTP/1.1', The Internet Society and The Internet Engineering Task Force, RFC 2616.

[Gamma] Gamma, E., Helm, R., Johnson, R., and Vlissides, J.: 'Design Patterns: Elements of Reusable Object-Oriented Software', Addison Wesley, 1995, pp. 273 – 282.

[Grass] The Grasshopper Agent Platform, web page: http://www.ikv.de/index.html.

[Liber] Lieberman, H.: 'Letizia: an agent that assists web browsing', Proceedings of the 14th International Joint Conference on Artificial Intelligence, Montreal, Canada, August 1995.

[OMG] Object Management Group (OMG): ‘Mobile Agent System Interoperability Facilities Specification’, November 1997, OMG TC Document orbos/97-10-05.

[TS-3GPP] Technical Specification Group Services and System Aspects: 'Virtual Home Environment (Release 4)', 3rd Generation Partnership Project, Technical Specification 3GPP TS 23.127 V4.0.0 (2000-11).

[Wiess] Weiss, G.: ‘Multiagent systems: a modern approach to distributed artificial intelligence’, (MIT press, 1999), pp42.