A Software Retrieval Service based on Knowledge-Driven Agents*

E. Mena[1], A. Illarramendi[2], and A. Goñi[2]

[1] IIS depart., Univ. de Zaragoza. Spain. http://www.cps.unizar.es/~mena/
[2] LSI depart., UPV. San Sebastián. Spain. http://siul02.si.ehu.es/~jirgbdat/

Abstract. The ability of retrieving and installing software in an easy and efficient way confers competitive advantage on computer users in general, and even more especially on mobile computer users. In this paper we present a software retrieval service that allows mobile users to select, retrieve and install software anywhere and at any time. This service makes use of agents that allow 1) a browsing of a software ontology automatically customized to different kind of users and user computers; 2) an efficient retrieval of the selected software; and 3) an automatic update of the available software ontology. A software obtaining process based on agents, that manage semantic descriptions of available software, presents a qualitative advance with respect to existing solutions where users must know the location and access method of various remote software repositories.

1 Introduction

One of the most frequent tasks of computer users is to obtain new software, in order to improve the capabilities of their computers. Different kinds of users need different kinds of software. Nowadays a common procedure to obtain software is to visit some of the several websites that contain freeware, shareware and demos (such as Tucows [11], CNET Shareware.com [2], CNET Download.com [3]), games (such as Games Domain [1] and CNET Gamescenter.com[4]), java-related software (like Gamelan [5]) or many others. However, that procedure presents problems for many users because they must:

1. Know the different programs that fulfil their needs. Not only their names but also where to find them in the vast Web space. Web sites are moved and re-designed which makes that links and bookmarks become obsolete very frequently
2. Know the features of their computers, in order to select the most appropriate version for their computers. This task implies having technical knowledge, among other things, about her/his system (CPU, OS and version, free disk space available, RAM memory, etc.) and the software installed previously (to request a full version or just an update).

* This work has been supported by *CICYT (Comisión Interministerial de Ciencia y Tecnología*, Spain [TIC97-0962]), MoviStar (a spanish cellular phone company) and the University of the Basque Country.

3. Be aware of new software and/or new releases of software of interest. Although few commercial programs currently alert about new releases, users need to keep an eye on the Web, or on other repositories, if they want to be informed about new software that could be of their interest.

Previous problems become even more important when users work with mobile computers using a wireless network media. Time expended in looking for the software, retrieving and installing it should be minimized as much as possible, in order to reduce communication cost and power consumed.

In this paper we present an alternative procedure: a Software Retrieval Service, based on the use of an ontology and the agent technology, that allows users to find, retrieve and install software in an easy and efficient way. Easy, because with the help of intelligent agents, users can browse the ontology that describes semantically the content of a set of data sources containing pieces of software, and so they can select from it the software (the service makes transparent for the users the location and access method of various remote software repositories); and efficient, because agents take care of reducing the wireless communication cost.

Concerning related work, to our knowledge, agents have not been widely used for software retrieval. In [6] they explain a mechanism to update several remote clients connected to a server taking advantage of mobile agents capability to deal with disconnections. However this work is more related to *push technology* than to services created to assist users in the task of updating the software on their computers.

In the rest of the paper, we present in Sect. 2 the cost model of software retrieval services based on webpages navigation. In Sect. 3 we explain the cost model of our proposal and present a comparison between the two approaches. A description of the different agents involved in the Software Retrieval Service is included in Sect. 4. Two of them, *Alfred* and the *Browser*, are described in detail in Sect. 5 and 6, respectively. Finally, conclusions can be found in Sect. 7.

2 Software Retrieval Based on HTML Pages Navigation

We said before that the most common way to obtain new software is by navigating HTML pages in (public or pay-per-download) software repositories. These repositories classify the different software in categories, in order to make easier the selection of the wanted piece of software. The user selects a category, browses the information, clicks on a link which involves a remote access, browses the information obtained, clicks again on another link and repeat this process until s/he requests a piece of software.

In the following we describe the cost model associated to this way of retrieving software. $C_{HTMLnav}$ is the cost to retrieve the software by interacting directly with the Web server that contains the software programs.

$$C_{HTMLnav} = [n \times (T_{newPage} + T_{readPage}) + T_{download}] \times C_{perSec}$$

where n is the number of links that the user has to navigate before finding the wanted program; $T_{newPage}$ is the time passed since the petition of a URL until the content of the new page arrives at the user computer; $T_{readPage}$ is the time spent by the user to check out a web page in order to find the link for the next software category or for the program to download; $T_{download}$ is the time passed since the petition of a URL that contains a software program until it is downloaded; and C_{perSec} is the cost of communication per second.

Notice that the above behaviour corresponds to the classical Client/Server (C/S) approach. $T_{newPage}$ and $T_{download}$ depend on the network transmission speed. Moreover, we cannot minimize $T_{readPage}$ as it depends on the kind of user (naive or expert). Thus, a system like this depends completely on the network and on the number of iterations needed to select the piece of software (i.e., n). The first problem is out of the scope of software system developers, and the second parameter is minimized by most common software repositories by classifying the enormous amount of available software in a hierarchy of categories.

A very important point is that users need several "clicks", more than five even when they perfectly know the software repository as well as what they are searching for[1]. Naive users that do not know neither the category of the software nor even the software itself could click a big number of times before finding the wanted software; some could even get lost in the taxonomy of categories. The system response to each click is fast when the network allows it. But the big disadvantage is that users need their computers to be connected continuously. This is an issue that we can improve by the use of mobile agents.

3 Our Proposal: the Software Retrieval Service

The procedure that our service supports for the software retrieval process is the following: first, the user receives the visit of an agent (the *Browser*) which helps the user to select the most appropriate software by browsing a catalog customized to that concrete user. The user can request more detailed information until s/he finally selects a piece of software. Then, a new agent arrives to the user computer (the *Salesman*) with the selected piece of software.
In the following we describe C_{SRS}, the cost model associated to the Software Retrieval Service:

$$C_{SRS} = [T_{BrowserCat} + T_{readCat} + m \times (T_{refine} + T_{readCat}) + T_{SalesmanCat}] \times C_{perSec}$$

where $T_{BrowserCat}$ is the time passed since the petition of the software service until the Browser agent arrives to the user computer with the catalog; $T_{readCat}$ is the time spent by the user to check out a catalog in order to request a new refinement or select the program to download; m is the number of catalog refinements requested by the user; T_{refine} is the time passed since the petition of a new catalog refinement until the Browser presents the new catalog; and

[1] We are not considering the use of search engines available in some software repositories.

$T_{SalesmanCat}$ is the time passed since the selection of the wanted software is made until the Salesman agent arrives to the user computer with such a piece of program.
We refine the above expression considering the next properties:

1. $T_{BrowserCat} = T_{Browser} + T_{catalog}$, i.e., the time needed for the Browser to travel to the user computer and the time needed to carry a catalog to the user computer, respectively.
2. Similarly, $T_{SalesmanCat} = T_{Salesman} + T_{download}$, i.e., the time needed for the Salesman agent to travel to the user computer and the time needed to carry the selected piece of software to the user computer, respectively.
3. Concerning refinements requested by the user, we consider $m = k + l$, where k is the number of catalog refinements that can be answered by the Browser itself ($T_{localRefine}$), and l is the number of catalog refinements that imply that the Browser connects remotely to obtain such an information ($T_{remoteRefine}$).

Therefore, the cost expression for the Software Retrieval Service results as follows:

$$\begin{aligned} C_{SRS} = [&T_{Browser} + T_{catalog} + T_{readCat} + k \times (T_{localRefine} + T_{readCat}) \\ &+ l \times (T_{remoteRefine} + T_{readCat}) + T_{Salesman} + T_{download}] \times C_{perSec} \end{aligned}$$

Comparison of Costs of Both Procedures. We would like to know now when it is better each one of both possibilities, HTML navigation or our proposed Software Retrieval Service. We consider the following properties between the SRS and the HTML approach, and the resulting comparison:

- $T_{readCat}$ equivalent to $T_{readPage}$ (time spent by the user reading a catalog is similar to the one spent reading a web page with categories).
- $n = 1 + m$ (the first catalog retrieval and the m refinements in SRS are equivalent to n web pages in the HTML navigation approach). Thus, $n = 1 + k + l$.
- $T_{remoteRefine} = T_{newPage}$, since the information needed by the Browser to perform a new refinement is similar in size to a web page.
- $T_{localRefine} = 0$ (such an activity is performed by the Browser on the user computer without the need of being connected to the net).

$$\begin{gathered} C_{SRS} < C_{HTMLnav} \iff \\ \iff T_{Browser} + T_{catalog} + T_{Salesman} < (k+1) \times T_{newPage} \iff \\ \iff |Browser| + |catalog| + |Salesman| < (k+1) \times |newPage| \end{gathered}$$

Notice that all the times are affected by the network speed, so we can rewrite the expressions in terms of size. We can observe here that, if we forget the size of the two agents which is small (in our prototype, the size of each agent is less

than 10K), the keys of the success of our approach are the size of the catalog initially carried to the user computer and the number of refinements that the Browser is capable to perform without external help (k), which also depends directly on the size of the initial catalog as well as on the "intelligence" of the agents. The estimation of the relationship between the catalog size and k will be the goal of future papers.

In [8] we show an example of the different behaviour, from the point of view of network remote access, of a Client/Server (C/S) approach, like HTML navigation, and a mobile agent approach, like our proposed service.

4 Agents Involved in the System

In this section we present briefly all the agents that take part of the Software Retrieval Service [8]. This service is situated in a concrete server that we call GSN[2] and is one of the services provided by the ANTARCTICA system [12]. Agents are executed in contexts denominated *places* [10]. Mobile agents can travel from one place to another. The proposed service incorporates one place on the user computer called the *User place*, and other three places situated on the GSN, called the *Software Acquisition place*, the *Software place* and the *Broadcast place*, respectively (see Fig. 1).

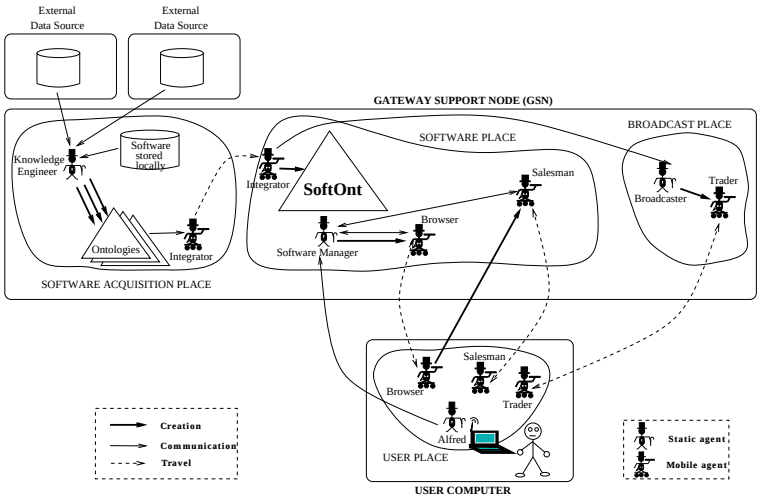

Fig. 1. Main architecture for the Software Retrieval Service

In the following we briefly describe such agents, grouped in three categories:

[2] The *Gateway Support Node (GSN)* is the proxy that provides services to computer users.

1. *The user agent.* **Alfred** is an efficient majordomo that serves the user and is on charge of storing as much information about the user computer, and the user her/himself, as possible. Although it helps the user in all the services provided by the GSN, we will stress its role in the Software Retrieval Service.
2. *SoftOnt ontology creation.* **The Knowledge Engineer** agent mines (local or remote) software repositories, with the help of specialized wrappers, in order to obtain a semantic description (an ontology) for each repository. Then, **the Integrator** agent performs the integration of all the ontologies obtained by the Knowledge Engineer with the goal of obtaining the ontology SoftOnt. During the integration process it uses a thesaurus for the automatic vocabulary problems resolution [7].
3. *SoftOnt ontology exploitation.* **The Software Manager** agent creates and provides **the Browser** agent with a catalog of the available software, according to the needs expressed by Alfred (on behalf of the user), i.e., it is capable to obtain customized metadata about the underlying software. For this task, the Software Manager consults the SoftOnt ontology. The software itself can be either stored locally on the GSN or accessible through the Web in external data sources. Thus, the GSN can have access to a great number of distinct software for different systems, with different availability, purpose etc. The goal of the Browser agent is to interact with the user in order to refine a catalog of software until the user finally chooses a concrete piece of software. When this is done, **the Salesman** agent carries the program selected by the user to her/his computer, performs any electronic commerce interaction needed (which depends on the concrete piece of software), and installs the program, whenever possible.

   The Broadcaster agent sends information of different nature to the users of any of the services provided by the GSN. Depending on the information, it can create **Trader** agents that carry non trivial information (demos, new releases, etc.) to users that manifested their interest in such an information.

Notice that we have only provided a description of the behaviour of the agents in the system, but we do not specify how the structure or state of such agents changes. In other words, how they *learn*, how its knowledge is structured. The reason is that the state of intelligent agents like these is too complex (ontologies, user preferences about many situations, etc.) to provide a brief description. A more detailed description of the *knowledge* managed by Alfred and the Browser, which are the core of the system, is included in the following sections.

5 Alfred: the User Majordomo

Alfred is an efficient majordomo that serves the user and is on charge of storing as much information about the user computer, and the user her/himself, as possible. Initially, Alfred is the only agent on the user computer that is able to interact directly with the user. When another agent wants to show/retrieve

data to/from the user it has to communicate with Alfred who[3] will create the appropriate user interface for each case (other agents do not know how the user wants to be interacted). In the opposite way, anytime the user wants to perform any action or request some information, s/he communicates with Alfred who will perform such tasks by himself or by communicating with other agents in the system to achieve the requested task.

5.1 Alfred's Goal

We now explain the role of Alfred in the Software Retrieval Service. Let us start with the situation in which the user wants to retrieve some kind of software. Two cases can arise:

1. The user only knows some feature of the program, for example, its purpose. In this case, the user needs some kind of catalog concerning the software available, in order to navigate it and find the wanted program. With the help of a GUI, users can write a list of constraints, expressed as a list of pairs <*feature, value*>, in order to express their needs the best they can[4]. The system will try to match such a query with the semantic description of available software (i.e., the SoftOnt ontology). We propose the use of a (specialized) thesaurus in order to deal with the vocabulary problem. Moreover, the user can specify the level of detail (expressed as a percentage) that s/he wants in such a catalog, and select some pruning strategy. Advanced users could be interested in many features of software while naive users could only be interested in just a few, such as a brief description, the name of the program and the OS needed to install it.

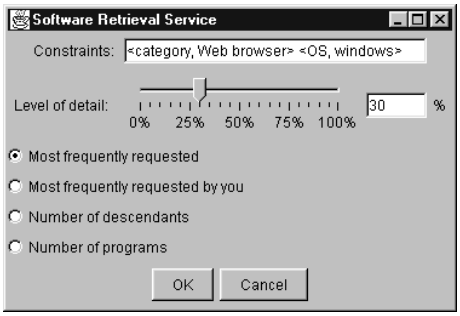

Fig. 2. Alfred's GUI for the Software Retrieval Service

[3] We denote Alfred as a person in an attempt of reflecting his role similar to a major-domo in the real world.

[4] The information provided can be imprecise. In Fig. 2, the user does not know neither the web browser name nor the concrete Windows version of her/his computer.

2. The user exactly knows which program s/he needs, for example, JDK1.2.2 for Win32, and also knows how to ask for it. This can happen because s/he is a usual client of this service. Thus, expert users could directly pose the request, with the help of a GUI, describing the software they need. The data entered is also represented as a (list of) pair(s), for example *[<name, JDK1.2.2>, <OS, Win32>]*.

All the information provided by the user is received and stored by Alfred. Thus, Alfred can add more useful information to users requests, taking previous executions as basis. This information is stored in the MU place and updated automatically by Alfred. Thus, with each user request, Alfred stores more information about the user computer and about the user. The information added by Alfred to the user request is also expressed as a list of pairs.

After the query is built, Alfred sends a request to the GSN, where the request is attended by a specialized agent, the *Software Manager* agent, which obtains a catalog with the kind of software that satisfies the user needs, according to the level of detail specified and the pruning strategy selected. Then, the Software Manager creates a specialized agent, the *Browser* agent, which travels to the User place in order to help the user to select the piece of software that fulfils her/his needs.

5.2 Alfred's Knowledge

As we said before, Alfred acts as a mediator between the user and all the services offered by the ANTARCTICA system. In the following we describe the kind of knowledge managed by Alfred, with a special stress in the knowledge involved in the Software Retrieval Service.

Basically we make a general distinction between the general knowledge of Alfred and the specialized knowledge related to concrete services:

1. Knowledge independent of the service: it is composed by technical knowledge related to the system, such as:
   - Hardware components available: RAM memory, sound and video card, CPU, type of network connection, and other peripherals installed.
   - Resource availability: free disk space, free memory, etc.
   - Software currently available: applications and software needed to install other software, such as Java, DirectX, etc.
   - General user preferences: common to all the services like GUI features, logging level, etc.

   Alfred obtains this information by consulting the OS. If the OS does not allow to automatically retrieve some data, Alfred would request such data to the user (only when needed).
2. Knowledge related to each service: the different kind of information that Alfred needs to deal with, when a service is requested by the user. The examples provided are related to the Software Retrieval Service:

- Technical knowledge concerning the service: information about other agents and places needed to achieve the goal of the service, such as residing nodes, names, services, etc. For example, Alfred needs to know how to communicate with the Software Manager agent residing on the GSN and any other coming agents like the Browser or the Salesman.
- Historical traces, with previous execution traces for such a service. This is a very important part of Alfred's "memory". The most important information of these traces are the interactions with the user, since we want to minimize them: all the information provided by the user is stored. In the same way, Alfred also saves the requests formulated to the user by other agents and the answers provided by her/him. This catalog of past interactions is updated every time that this concrete service is invoked.
- User preferences related with the concrete service, that have been extracted from the historical traces. For example, Alfred can detect that whenever the user needs a web navigator, s/he always selects the last version of Netscape. However, it is important to stress that these preferences can be time-changing, the user could like a different web navigator because of a new user interface or any other reason.

For the traces and the user preferences, Alfred manages probabilistic information of user's answers, in order to face the problem of different answers provided to the same question in different situations. For example, Alfred could store that the user selected Netscape as web browser the 85% of times. As we said before, the user can change her/his mind due to whatever reason. Thus, whenever Alfred can suggest different choices to the user, the most probable one will be selected taking into account the number of times that it was provided by the user as well as that recent answers could match better with current user preferences. Anyway, Alfred always reports to the user the reason of its proposal and the final decision depends on the user.

During the execution of the Software Retrieval Service, Alfred explains to the user some special situations that unable the system to satisfy the user, for example, "free disk space is not enough", "the requested software is not available for the current OS", etc. Hence, any refinement made on user requests, is communicated to the user for information purpose.

6 The Browser: a Specialist in Browsing Catalogs

We divide the explanation about the Browser agent into two subsections: 1) its creation, performed by the Software Manager agent; and 2) its main activity, when it helps the user to browse software catalogs.

6.1 The Software Manager and the SoftOnt Ontology

After receiving a request of Alfred (on behalf of the user), the Software Manager agent creates a new Browser agent which will be in charge of helping the user

to select the most appropriate piece of software. The Browser agent manages an ontology (a subset of a bigger ontology that describes all the software available at the GSN) provided by the Software Manager right after the Browser is created.

We advocate using an ontology, called *SoftOnt*, to describe *semantically* the content of a set of data sources storing pieces of software. This ontology, which stores detailed information concerning the available software accessible from the GSN, is managed by the Software Manager. So, instead of users having to deal directly with different software repositories, the goal is that a system uses an ontology to help users to retrieve software.

In [7] we explain the translation step that permits obtaining ontologies from software repositories and the integration step that generates a global ontology by integrating the previously obtained ones, along with the explanation of how the previous process can be performed automatically and the justification about the use of only one ontology.

The SoftOnt ontology is pruned by considering the constraints, the level of detail and a pruning strategy, which have been provided by Alfred. The level of detail (which is a percentage) indicates the amount of data that should be included in the catalog; for example, a level of detail of 30% indicates that only the 30% of SoftOnt should be shown to the user. The pruning strategy indicates *which* nodes of the ontology will be selected, it is the selection criteria; for example, the most frequently requested or the most frequently requested by that user. Thus, we could obtain, for example, the most frequently requested nodes until completing the 30% of SoftOnt. Notice that, independently of the pruning strategy, a level of detail of 30% indicates that the user only wants to see the 30% of the ontology obtained after considering the constraints indicated by Alfred.

This pruning process is very important due to two reasons: 1) it avoids presenting the user categories and pieces of software that cannot be installed on the user computer (different OS, or other restrictions); and 2) it avoids presenting very specialized categories and pieces of software that could surely make naive users spend more time reading the catalog.

Therefore, *the SoftOnt ontology constitutes the main knowledge managed by the Software Manager agent and the pruned ontology (customized to each user) constitutes the main knowledge managed by the Browser agent.*

6.2 Catalog Browsing

Once on the user node, the Browser presents the pruned ontology as a graph (inner nodes are categories and leaves are programs), and the user can navigate it and ask for a bigger level of detail of the terms s/he is interested in. In that case, the Browser will request more information to the Software Manager, which resides on the GSN. In order to help users, each node in the ontology has associated a percentage (calculated when a view of SoftOnt is built) that indicates how much of the total information has been retrieved until then.

The following are the different refinements that a user can request by selecting a node of the current catalog:

1. Request of more detail for some node. Sometimes, the information can be on the GSN, so the Browser communicates with the Software Manager Agent or travel to the GSN, depending on the amount of information requested. However, the Browser could have such an information and no remote communication is needed.
2. Refinement by providing new constraints. The user could want to provide a new constraint (a new pair <*feature, value*>) instead of navigating the catalog. Then the catalog must be re-built again, taking into consideration the new constraint. If the Browser would have pruning capabilities, it could be done locally, on the user node without any remote connection. In other case, the Browser will request the new catalog to the Software Manager Agent or travel to the GSN if the resulting catalog may be very big.

Notice that the Browser can decide whether requesting remotely a certain information to the Software Manager or travelling to the GSN to request the information locally. In order to decide between those two strategies, the Browser evaluates the impact of new user refinements. If they only require a small amount of bytes, such an information will be requested remotely; in the case of refinements that affect to many categories, the Browser can request to the Software Manager the size in bytes of the needed information; for example: which is the size of all the information under the term 'HTML tools' ?.

In the case of travelling to the GSN, the Browser does not travel with the previous catalog: it is stored temporally on the user node and updated when the Browser comes back with the new information. Moreover, it can decide taking advantage of the trip and requesting to the Software Manager more information than what it was requested by the user (without compromising the efficiency). In this way, notice that the Browser upgrade its knowledge with each user refinement, making less frequent the need for remote connections. So, future refinements can be attended faster and avoiding remote connections.

The process of catalog browsing ends when the last catalog presented to the user cannot be refined as it is composed by a list of names of software that fit the user needs. The user can then click on one of the different choices which correspond to programs that fulfil her/his needs. As a consequence of that action, the Browser agent remotely creates a Salesman agent on the GSN. This agent will visit the user computer carrying the specified program. After this, the Browser Agent simply ends its execution. See [8] for a more detailed description of the task performed by the Salesman.

7 Conclusions and Future Work

Retrieving software is an activity that requires a particular effort to users and that must be done with a certain frequency. Therefore, it is widely accepted the interest of systems that help users with that task of software retrieval.

In this paper we have presented a Software Retrieval Service based on knowledge-driven agents that allows users to browse a catalog that contains semantic

descriptions of available software, and to select and retrieve the wanted software in an efficient way. Although the service can be used by any kind of computer user it puts on a special emphasis on mobile users. Empirical results obtained through the implemented prototype and analytical tests [9] show the feasibility of the service.

As future work, we are studying the possibility of providing the Browser agent with the pruning capabilities of the Software Manager in order to make it more independent of the GSN and reduce dramatically the need of having an open network connection.

Acknowledgements

We would like to thank Daniel Fanjul and Ignacio García his valuable help in the implementation of the prototype.

References

1. Attitude Network Ltd., 1999. http://www.gamesdomain.com.
2. CNET Inc., 1999. http://www.shareware.com.
3. CNET Inc., 1999. http://www.download.com.
4. CNET Inc., 1999. http://www.gamecenter.com.
5. Earthweb & Sun Microsystems, 1999. http://www.gamelan.com.
6. IBM Corporation. TME 10 Software Distribution - Mobile Agents SG24-4854-00, January 1997. http://www.redbooks.ibm.com/abstracts/sg244854.html.
7. E. Mena, A. Illarramendi, and A. Goñi. Automatic Ontology Construction for a Multiagent-based Software Gathering Service. In *proceedings of the Fourth International ICMAS'2000 Workshop on Cooperative Information Agents (CIA'2000), Springer series of Lecture Notes on Artificial Intelligence (LNAI), Boston (USA)*, July 2000.
8. E. Mena, A. Illarramendi, and A. Goñi. Customizable Software Retrieval Facility for Mobile Computers using Agents. In *proceedings of the Seventh International Conference on Parallel and Distributed Systems (ICPADS'2000), workshop International Flexible Networking and Cooperative Distributed Agents (FNCDA'2000), IEEE Computer Society, Iwate (Japan)*, July 2000.
9. J. Merseguer, J. Campos, and E. Mena. Performance Evaluation for the Design of Agent-Based Systems: A Petri Net Approach. In *proceedings of the Software Engineering and Petri Nets (SEPN'2000) workshop within the 21st International Conference on Application and Theory of Petri Nets, Aarhus (Denmark)*, June 2000.
10. D. Milojicic, M. Breugst, I. Busse, J. Campbell, S. Covaci, B. Friedman, K. Kosaka, D. Lange, K. Ono, M. Oshima, C. Tham, S. Virdhagriswaran, and J. White. MASIF, the OMG mobile agent system interoperability facility. In *Proceedings of Mobile Agents '98*, September 1998.
11. Tucows.Com Inc., 1999. http://www.tucows.com.
12. Y. Villate, D. Gil, A. Goñi, and A. Illarramendi. Mobile agents for providing mobile computers with data services. In *Proceedings of the Ninth IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 98)*, 1998.