Adaptable Software Retrieval Service for Wireless Environments Based on Mobile Agents*

E. Mena, J.A. Royo†
IIS department
Univ. of Zaragoza
Spain
{emena, joalroyo}@posta.unizar.es

A. Illarramendi, A. Goñi
LSI department
Univ. of the Basque Country
Spain
{jipileca, alfredo}@si.ehu.es

Abstract

One of the most frequent tasks using any kind of computer is the retrieval of new software, that is commonly downloaded from websites that offer freeware, shareware and demos, such as Tucows. However, naive users must deal with too many technical details in order to take advantage of these facilities. Moreover, navigating those websites without exactly knowing what we look for results in high communication costs, specially for wireless device users.

In this paper we present a Software Retrieval Service that, first, allows users to obtain software by browsing a catalog (that contains a semantic description of software available at different repositories), and second, it minimizes the use of the network by adapting its behavior to the current network status. The service has been developed using mobile agent technology. In the paper we describe the main elements that take part of the service and stress how the system adapts its behavior to the current network status.

Keywords: *Mobile computing, Wireless and mobile applications, Mobile agents.*

1 INTRODUCTION

Working with any kind of computer (desktop, laptop, palmtop), one of the most frequent task for the users is to obtain new software in order to improve the capabilities of those computers or devices. For that, an often used approach is to visit some of the several websites that contain freeware, shareware and demos (such as Tucows [11] and CNET Download.com [1]). However, that approach can become cumbersome for naive users — they may not know: 1) the location of software websites and their navigation structure, 2) the programs that fulfil their needs, 3) the features of their devices, and 4) when new interesting releases are available — and can become annoying for many advanced users. Moreover, if those users use a wireless device, the time expended by them to look for the software, to retrieve and install it should be minimized as much as possible in order to reduce communication cost and power consumed.

Taking into account the previous scenario, we propose in this paper a Software Retrieval Service (SRS) that allows users to find, retrieve and install software. This service presents the following features: 1) *Semantic Search*: the service helps users in the task of obtaining software by allowing them to express their requirements at a semantic level, and guides them browsing a customized software catalog in order to select the adequate software. For that, the service makes use of a catalog that contains a semantic description of software available in different repositories, and so makes transparent for users the technical details related to that task. 2) *Quality of Service (QoS)*: the user specifies some features of what s/he wants and the system is on charge of dealing with technical details such as the features of the user device (Operating System, memory capacity, etc.) and the current network status; so it offers a customized and adaptable service to the users putting a special emphasis on optimizing communication time.

Concerning related work, to our knowledge, agents have not been widely used for software retrieval. In [5] 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 Ariadne project [6] they use agents, not for retrieving software but for accessing websites; they work on automatic wrapper construction tech-

*This work was supported by the CICYT project TIC2001-0660 and the DGA project P084/2001.

†Work supported by the grant B131/2002 of the Aragón Government and the European Social Fund.

niques and build an ontology on top of each website in a semi-automatic manner. OntoAgents [2] allows to annotate websites to perform a semantic search; data can be accessed using a web browser or performing a search that is managed by agents, which consider the different terms in the website.

In the rest of the paper, we briefly explain the features of the software retrieval service in Section 2. In Section 3 we describe how to measure the network status. In Section 4 we show how the network status is considered to obtain user software catalogs. In Section 5 we show how the network status is considered to access remote information following different techniques. We present some performance results in Section 6. Finally, Section 7 includes some conclusions.

2 The Software Retrieval Service (SRS)

The service proposed in this paper takes part of a more global system called ANTARCTICA [3] which main goal is to offer different wireless data services that enhance the capabilities of mobile devices[1]. The implementation of the SRS is based on the agent technology [4, 10]. So, four main agents participate in the service: *Alfred* (the user agent situated at the user device) who is an efficient majordomo that serves the user and is on charge of storing as much information about the user device and the user her/himself as possible; The *Software Manager* agent (situated at the GSN[2] that provides coverage to the user device) that obtains software catalogs customized to user requirements; The *Browser* agent (situated at the user device) that helps the user to navigate the customized catalog in order to select the wanted software (new information requested by the user and not available to the Browser will be retrieved by the *Catalog Updater* agent); and the *Salesman* agent that carries the selected program to the user device and installs it whenever possible.

Three main tasks take part when dealing with the service: 1) to specify user requirements, 2) to prune the global catalog in order to present to the user a software catalog containing only that part related to her/his requirements (to avoid confusing the user and overloading her/his device), and 3) to attend user refinements on such a catalog trying to predict future user requirements (to minimize network communication). The responsible for the first task is Alfred, for the second one the Software Manager agent, and for the third one the Browser agent.

In the rest of the section we describe these three main steps of the service, and in Sections 4 and 5 we detail how the Software Manager, Catalog Updater, and Salesman agents adapt their behavior according to the wireless network status and user preferences, which is the key of the success of the SRS.

[1] Some services, as the one described here, require that target mobile devices have some features like color hi-res screen and the capability of executing Java agents.

[2] *Gateway Support Nodes (GSNs)* are proxies in the fixed network that provide services to wireless device users.

2.1 Initialization

Alfred is an efficient majordomo agent that serves the user and is on charge of storing as much information about the user device, and the user her/himself, as possible. Let us start with the situation in which the user wants to retrieve some kind of software. For that, the user specifies a list of keywords <*feature, value*> describing the software s/he needs. The information provided can be imprecise (in Figure 1), the user does not know neither the web browser name nor the concrete Windows version of her/his computer). Moreover, the user can specify the level of detail (expressed as a percentage) that s/he wants in such a catalog, the more detail the bigger catalog.

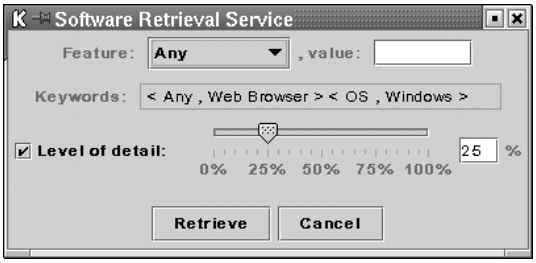


Figure 1. Alfred's GUI for the SRS

In addition to the user information, Alfred can add more keywords concerning the user device (OS, RAM memory, video and audio cards, etc.) and previous executions of the service. All the information provided by the user is stored by Alfred; a detailed description of the *knowledge* managed by Alfred can be found in [7].

2.2 Obtaining a Software Catalog

After the user specifies her/his software needs, Alfred sends a request to the Software Manager agent which obtains a catalog corresponding to the user request. For this task, we advocate using a global software catalog, called *SoftCat*, that describes *semantically* the software available in a set of websites storing pieces of software. So, instead of users having to deal directly with different software repositories, our system uses a common software catalog to help users to retrieve software.

However, the SoftCat catalog must be pruned in order to obtain a software catalog customized for the user. This pruning process is very important to avoid presenting the user categories and pieces of software that cannot be installed on the user device or those that could surely make

naive users spend more time reading the catalog, and consequently, finding the wanted software. Thus the system minimizes the communication cost by sending interesting information only.

Therefore, when the Software Manager is invoked to perform a prune of the SoftCat catalog, the following steps are followed: 1) *Pruning SoftCat using keywords* (if any was specified by the user) that nodes in the result must satisfy. Let us call Cat_{keywds} to the catalog after considering the keywords; 2) *Setting the level of detail*, by considering returning more information than what it was requested by the user (see Section 4.1); 3) *Pruning*[3] Cat_{keywds} according to the level of detail obtained before, let us say $n\%$, to obtain Cat_{pruned} as result of this task; 4) *Obtaining an incremental answer* to avoid sending data that are already on the user device (for that, the Software Manager stores the ids of the nodes previously sent to each user); 5) *Compressing the catalog*, if it is worth (see Section 4.2); and 6) *Creating a Browser agent* initialized with the catalog obtained (this specialized agent will travel to the user device and help the user to find the wanted software as explained in Section 2.3). It is important to stress that, although the Software Manager could have selected a level of detail higher than the specified by the user, the Browser will exactly show to the user what s/he asked for. The rest of the information can be used by the Browser as a buffer to perform future catalog updates, as explained in Section 2.3.

2.3 Catalog Browsing

Once on the user node, the Browser agent presents the catalog as a graph: inner nodes are categories and leaves are programs that can be downloaded. In order to help users, under each node in the catalog there is a bar that represents graphically: 1) how much information about that node is shown, and 2) how much information about that node is available at the user device; the rest of the bar, indicates how much new information about that node could be requested to the Software Manager.

The following are the different actions that a user can perform after studying the catalog presented: a) *To show the information about a node*, the Browser shows (on the right side of the GUI) all the features of such a node, including the list of programs under it when such a node is a category; b) *To open/close a node*, its immediate descendants are shown/hidden; c) *To request more or less information under some node*, the user can specify a new level of detail for that node or provide new keywords for that node and its descendants; and d) *To download a program*, when the user has (fortunately) found a piece of software that fulfils her/his needs. As a consequence of this last action the Browser remotely creates a Salesman agent on the GSN and the Browser agent simply ends its execution. The Salesman agent will visit the user device carrying the specified program [8].

The first and second actions in the previous paragraph can be performed by the Browser without remote access. The third task can also be performed locally only if the new level of detail is below the limit of the Browser buffer, as the Browser has pruning capabilities. In other case, the refinement cannot be performed using the information currently available to the Browser, then the new information must be requested remotely to the Software Manager. For that task, the Browser creates a *Catalog Updater* agent, whose goal is to retrieve from the GSN the needed information. In Section 5.1 we show the different alternatives considered by the Catalog Updater in run-time to perform this task.

Finally, as explained in Section 2.2, new catalogs are returned in an incremental way (only new information is sent to optimize communication costs). Thus, the Catalog Updater merges properly the previous user catalog with the new information, and then finishes its execution. In this way, notice that the Browser upgrades its knowledge with each user refinement, making less frequent the need for remote connections. So, future refinements can be attended faster and avoiding the use of the network.

3 Estimating the Network Status

Anytime an agent in our system wants to establish a wireless connection between the user device and the GSN, or vice versa, the real network speed must be estimated to calculate the time needed for any data transfer. Depending of the network status, several decisions are taken as explained in Sections 4 and 5.

The network speed is measured by simply ping-ing the GSN from the user device (or vice versa) right before the transfer, let us call it $netspeed_{ping}$. However, independently of the network speed, it is very interesting to estimate the reliability of the network link between the user device and the GSN when sending data during a certain time, i.e., the *real* network speed, $netspeed_{real}(t)$. Thus, we are interested in estimating the probability of maintaining a network connection open during a certain time t, $p(t)$. For that we consider past network connections, as we assume that the user device will always connect to the GSN using a similar kind of network. This probability is used to estimate in *run-time* the number of retries needed to completely send a concrete amount of data, $\#retry(t)$. Therefore:

$$p(t) = \begin{cases} \frac{F(t)}{T-S(t)} & T > 0 \\ 0 & T = 0 \end{cases} \qquad \#retry(t) = \left\lceil \frac{p(t)}{1-p(t)} \right\rceil$$

$$netspeed_{real}(t) = \frac{netspeed_{ping}}{\#retry(t)}$$

[3] Every immediate descendant is pruned proportionally: to reduce a node to its $n\%$, each immediate descendant is reduce to its $n\%$.

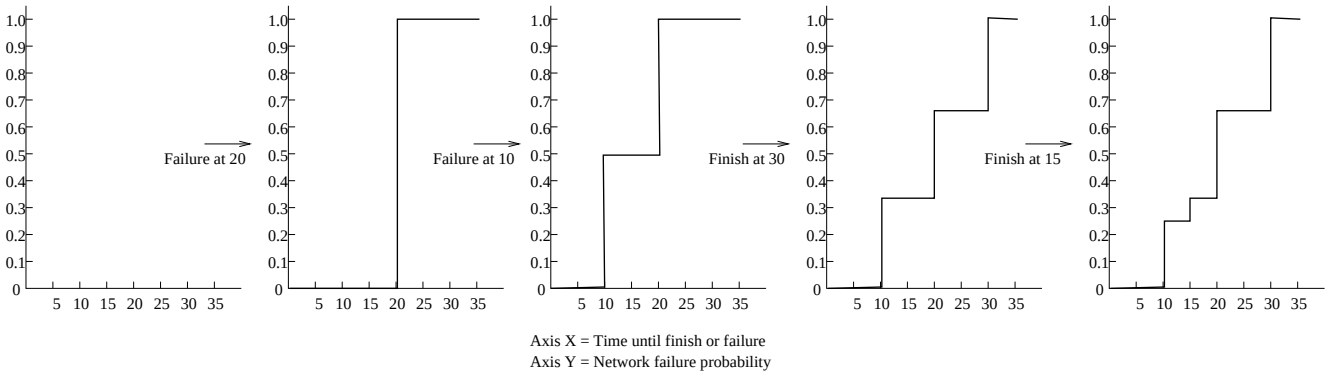


Figure 2. $p(t)$, the network failure probability

where $p(t)$ is the probability of network failure for connections that takes t seconds, $F(t)$ is the number of past connections that failed before or exactly at t, $S(t)$ is the number of past successful connections equal or shorter than t, and T is the total number of past connections (of any time length). In Figure 2 we can observe how the failure probability $p(t)$ varies as new connections are broken or finish without errors. Thus, after the four connections in the figure, probability of maintaining a connection open during, for example, 23 seconds is 0.66 ($p(23) = 0.66$).

Notice that when a new connection is broken at time t, the failure probability for times longer than t is increased (a connection that needed more than t seconds has just failed). However, when a new connection finishes without errors at time t, the probability for times longer than t still remains the same (finishing at t does not guarantee anything for times longer than t). In any case, a connection finishing or failing at time t decreases the probability of times shorter than t (as everything was fine until instant t).

We finally define $t_{retry}(t) = t \times \#retry(t)$, i.e., a function that, given a time t, returns the total time needed to maintain open a network connection during t seconds, after considering the number of retries estimated.

4 Adaptability when Obtaining a Software Catalog

In this section we explain how the network status influences the Software Manager behavior when obtaining a customized catalog for the user.

4.1 Selecting the most appropriate level of detail

The level of detail is a percentage that indicates the amount of data that should be included in the resulting catalog; for example, a level of detail of 30% indicates that only the 30% of SoftCat should be included in the result. If the user specified a level of detail (otherwise, the default value is 0%), the system must provide her/him with at least such a level of detail. However, as a new connection is needed, the system estimates if it is worth to retrieve more information than what it has been requested (to avoid future network connections), i.e., to consider a higher level of detail. This estimation takes into account parameters as the user profile and the network status. This approach also improves the efficiency when the user successively requests slightly higher level of detail about the same node. Thus, the following steps are followed:

a) To obtain the time needed for the information requested, by considering the size of Cat_{keywds}, the specified level of detail, and the nodes already sent to the user, $nodes_{user_k}$, as only incremental answers are sent to the user, as explained before.

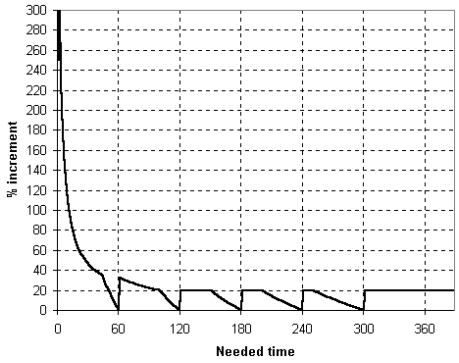
$$nodes_{user_k} = \{node_i \in SoftCat \mid node_i \text{ on device of } user_k\}$$

$$t_{needed} = t_{retry}\left(\frac{\left\lceil \frac{|Cat_{keywds}| \times level\ of\ detail}{100} \right\rceil - |nodes_{user_k}|}{netspeed_{ping}}\right)$$

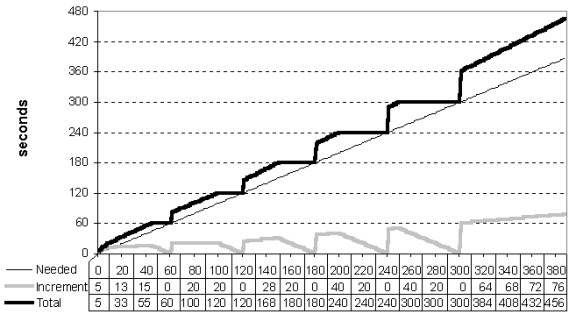
where $|x|$ is the size in bytes of set x, $netspeed_{ping}$ is the run-time measured network speed, and $t_{retry}(t)$ is the resulting time after considering the estimated number of retries when maintaining a connection open during t seconds (see Section 3).

b) To increase the answer time proportionally. In general, t_{needed} could be increased in a concrete percentage $\%_{incr}$ specified by Alfred (different users could have different increments depending on their expertise, device capabilities, network link, etc.). However, we must consider some special situations: 1) t_{needed} could be very short (if t_{needed}=20 seconds it is reasonable to make the user wait around 10 seconds more, i.e., an increment of 50%, however the higher t_{needed} the lower increment), and 2) communications could be billed using *charging units (CU)*[4], so

[4]By *charging units* we mean the time unit used to charge the user her/his phone bill; per seconds, minutes, etc.



(a)



(b)

Figure 3. (a) Percentage incremented for t_{needed} and (b) T_{incr} ($\%_{incr}$=20%, CU=60 sec)

the system should avoid using new CUs for just small increments.

In Figure 3.a, we show how the percentage incremented varies depending on the needed time, when $\%_{incr}$=20% and CU=60 seconds. Notice that the system selects increments higher than $\%_{incr}$ when t_{needed} is short, and increments lower than $\%_{incr}$ to avoid using a new CU, until the $\%_{incr}$ of the needed time is greater than one CU; in Figure 3.a, that happens when needed time is 300 seconds (5 minutes) or greater. In such a case the Software Manager chooses to spend one more CU rather than performing a too small increment. In Figure 3.b we show the function T_{incr} (labeled as "Increment") that returns the extra time corresponding to a given t_{needed}, therefore $t_{final} = t_{needed} + T_{incr}(t_{needed})$. When $t_{needed} = 0$ (the user did not specify a level of detail) the system considers 5 seconds as the minimum time to spent. Notice that if the user is not billed using CUs, T_{incr} will always increment the time in the percentage specified by Alfred ($\%_{incr}$).

c) To obtain the new level of detail, by considering the time used to return the catalog (t_{final}), the real netspeed, the averaged node size ($\overline{|node|}$), the number of nodes on the user device ($\#(nodes_{user_k})$) and the number of nodes in the keyword-pruned catalog ($\#(Cat_{keywds})$):

$$level\ of\ detail = \frac{\left(\frac{t_{final} \times netspeed_{real}(t_{final})}{\overline{|node|}} + \#(nodes_{user_k})\right) \times 100}{\#(Cat_{keywds})}$$

where $netspeed_{real}(t)$ is the run-time estimated network speed by considering the probability of disconnection during time t, as explained in Section 3.

4.2 Compressing The Catalog

The catalog obtained for the user can be returned compressed to reduce the use of the network when sending it to the user device. Next we show when it is worth to compress the catalog:

$$realNetSpeed = netspeed_{real}\left(\frac{n \times \overline{|node|}}{netspeed_{ping}}\right)$$

No compress: $t_{noZipped} = \frac{n \times \overline{|node|}}{realNetSpeed}$

Compress: $t_{zipped} = n \times t_{node} + t_{noZipped} \times r_{zipped}$

Therefore: $t_{zipped} < t_{noZipped}$

$$\iff realNetSpeed < \frac{\overline{|node|}}{t_{node}} \times \left(1 - r_{zipped}\right)$$

where n is the number of nodes to send (the number of nodes of the incremental answer), $\overline{|node|}$ is the averaged node size, $netspeed_{ping}$ is the measured network speed (see Section 3), $netspeed_{real}(t)$ is the run-time estimated network speed by considering the probability of disconnection during certain time (see Section 3), t_{node} is the time needed to compress one node, and r_{zipped} is the ratio of compression.

For example, if $\overline{|node|}$=850 bytes, t_{node}=0.15 miliseconds, and r_{zipped}=0.83, the Software Manager will measure the network speed and compress the catalog when the *real* network speed is lower than 941 Kb/sec.

5 Adaptability to Access Remote Information

In this section we explain how the network status influences the Catalog Updater behavior when updating a software catalog, and the Salesman agent behavior when downloading a piece of software into the user device.

5.1 Updating Software Catalogs

In order to update the user software catalog, the Catalog Updater agent can follow two alternatives: 1) a remote call from the user device to the Software Manager at the GSN, or 2) to travel to the GSN, communicate with the Software Manager locally, and travel back to the user device. To select a choice, the Catalog Updater considers the number of

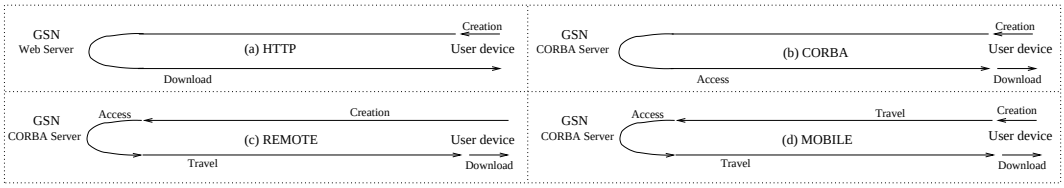


Figure 4. The (a) HTTP, (b) CORBA, (c) Remote, and (d) Mobile strategies

retries needed to maintain a network connection open during a certain time (see Section 3). Therefore:

$$\begin{aligned} t_{remoteCall} = \quad & t_{retry}(t_{call} + t_{newCat} + t_{return}) \\ t_{mobileAgent} = \quad & t_{createAgent} + t_{retry}(t_{goToGSN}) \\ & + t_{newCat} + t_{retry}(t_{goBack}) \end{aligned}$$

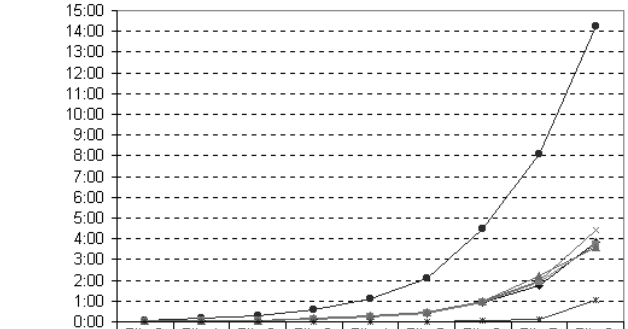
where t_{newCat} is the time spent by the Software Manager to obtain the requested software catalog, and $t_{retry}(t)$ is the resulting time after considering the estimated number of retries when maintaining a connection open during t seconds (see Section 3). We can see that, if a connection is broken, a remote call must be always reinitiated from the beginning. However, when using agents, network disconnections only affect the agent while it is traveling between the user device and the GSN. Therefore, in unstable networks (like wireless) the mobile agent approach is expected to be better. Anyway, the Catalog Updater chooses one of these two alternatives in *run-time*, as the network status is estimated right before a remote connection is needed, and the other parameters can be estimated by considering the network speed and the size of the data to transfer.

5.2 Software Downloading

To download the pieces of software selected by the user we have chosen a concrete approach: the software is carried to the user device by a mobile agent (the Salesman). We here compare the use of mobile agents for software downloading with other alternatives.

The different possibilities considered, from less to more "sophisticated", are: 1) *HTTP*: to request the software using an HTTP connection, a module is created on the user device which accesses a remote web server on the GSN. The whole process is a downloading by itself since after the access finishes, the requested piece of software is already saved on the user computer disk (see Figure 4.a). 2) *CORBA*: to request the software by invoking a remote service of a CORBA Server, a module is created locally which invokes a remote CORBA service of a CORBA server at the GSN. This service returns the piece of software requested as an array of bytes that is saved into a file on the user computer disk (see Figure 4.b). 3) *Remote*: a mobile agent is *remotely created at the GSN* and then it *locally* requests the software to the CORBA server, and travels back to the user computer to download the data (see Figure 4.c). 4) *Mobile*: a mobile agent is created on the user device; it travels to the GSN, accesses the software using a local CORBA connection, and travels back to the user computer to finally download the software (see Figure 4.d).

Tests are based on the retrieval of real pieces of software available on our GSN. We run 20 executions of the different strategies for each file using a wired connection[5]. We show in Figure 5 the comparison among the different methods. In HTTP, data is requested and saved into a file, byte by byte. In HTTP2, an array of bytes is requested and then saved into a file, which terribly speeds up the downloading but practically hangs the computer. We have also performed a test using an FTP connection and a non-Java program (concretely, ReachOut SuperFTP Client). In Figure 5 we can observe that *times corresponding to CORBA, Remote and Mobile methods are almost identical for all range of files*[6], so mobile agents do not improve but neither make worse downloading time. However, a partitioned downloading should be considered for big files, independently of the strategy used.



	File 0	File 1	File 2	File 3	File 4	File 5	File 6	File 7	File 8
HTTP	0:03	0:09	0:16	0:36	1:05	2:06	4:28	8:04	14:15
FTP	0:01	0:02	0:03	0:07	0:13	0:26	0:58	1:58	4:25
CORBA	0:01	0:02	0:03	0:07	0:14	0:26	0:56	1:45	3:51
REMOTE	0:01	0:02	0:03	0:07	0:13	0:26	0:57	1:56	3:42
MOBILE	0:02	0:03	0:04	0:09	0:16	0:28	1:01	2:14	3:37
HTTP2	0:00	0:00	0:00	0:00	0:00	0:01	0:02	0:08	1:04

Figure 5. Comparison among different download techniques

[5]For a GSM connection, differences in time are bigger, due to the slower net speed, but proportions are similar.

[6]$|\text{File } 0|$=100 KB, $|\text{File } 8|$=30 MB, $|\text{File } i+1| \simeq 2 \times |\text{File } i|$.

6 Evaluation of the System

A cost model for our software retrieval service and for Tucows can be found in [7], as well as a comparison between them. Moreover, an analytical comparison between the performance of a Tucows-like system and our SRS appears in [9]; both systems were modeled first in UML and then using Petri nets. As result of that analysis, it is concluded that the SRS behaves better than Tucows when at least the 40% of the refinements are managed locally. In Figure 6 we can see that the percentage of user refinements managed locally by the implemented prototype in the real tests performed is above 40%. If the system does not consider the network status, the number of refinements managed locally is decreased between 16% and 29% (25% as average) for the software pieces of the tests. Even in that case, the percentage of user refinements managed locally is above 40% as shown in Figure 6.

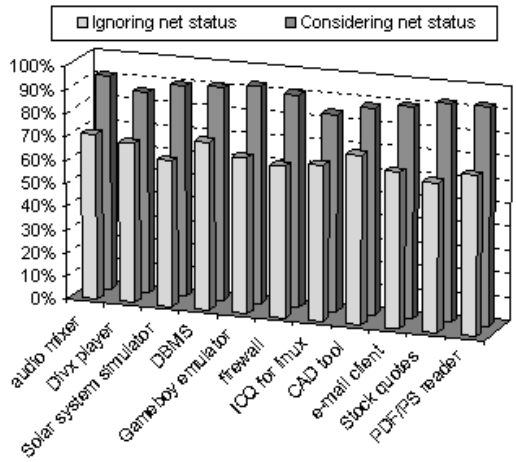


Figure 6. % of user requests handled locally

7 Conclusions

Dealing with wireless devices, one of the most expensive cost is that associated to the use of the wireless network. Therefore, there exists a great interest in defining techniques that optimize that use. In this paper we have presented one data service for wireless environments that makes a rationale use of a wireless network when users need to retrieve software. The service optimizes network use by: 1) providing the users with the possibility of retrieving software visiting only one software catalog instead of visiting several ones, 2) sending a customized catalog to the user device, and 3) analyzing in run-time the network status and deciding how much data should be transmitted through the wireless network to increase the number of software catalog refinements locally managed. Moreover, the use of the agent technology in the implementation of the service also contributes to the optimization goal by allowing to increase asynchronous communications and the recovery process in the case of network failure. Performance results corroborate this when comparing our approach with a Tucows-like approach.

Acknowledgements

We would like to thank V. Roche his priceless help in the implementation of the prototype, and J. Campos and F. Tricas for their help to model network disconnections.

References

[1] CNET Inc., 1999. http://www.download.com.

[2] G. W. et al. Ontoagents. http://WWW-DB.Stanford.EDU/OntoAgents/.

[3] A. Goñi, A. Illarramendi, E. Mena, Y. Villate, and J. Rodriguez. Antarctica: A multiagent system for internet data services in a wireless computing framework. In *NSF Workshop on an Infrastructure for Mobile and Wireless Systems, Scottsdale, Arizona (USA)*, October 2001.

[4] R. Gray, D. Rus, and D. Kotz. Agent TCL: Targeting the needs of mobile computers. *IEEE Internet Computing*, 1:58–68, August 1997.

[5] IBM Corporation. TME 10 Software Distribution - Mobile Agents SG24-4854-00, January 1997. http://www.redbooks.ibm.com/abstracts/sg244854.html.

[6] C. Knoblock, S. Minton, J. Ambite, N. Ashish, I. Muslea, A. Philpot, and S. Tejada. The ariadne approach to web-based information integration. *To appear in the International the Journal on Cooperative Information Systems (IJCIS) Special Issue on Intelligent Information Agents: Theory and Applications*, 10(1/2):145–169, 2001. http://www.isi.edu/info-agents/ariadne/.

[7] E. Mena, A. Illarramendi, and A. Goñi. A Software Retrieval Service based on Knowledge-driven Agents. In *Fith IFCIS International Conference on Cooperative Information Systems (CoopIS'2000), Springer series of Lecture Notes in Computer Science (LNCS), Eliat (Israel)*, September 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. Analysing internet software retrieval systems: Modeling and performance comparison. *Wireless Networks: The Journal of Mobile Communication, Computation and Information (WINET)*, 2002. To appear.

[10] S. Papastavrou, G. Samaras, and E. Pitoura. Mobile agents for WWW distributed database access. In *Proceedings of the International Conference on Data Engineering*, 1999.

[11] Tucows.Com Inc., 1999. http://www.tucows.com.