An Agent-based Approach for Helping Users of Hand-Held Devices to Browse Software Catalogs*

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

[1] IIS Depart., Univ. of Zaragoza, Maria de Luna 3, 50018 Zaragoza, Spain
{emena,joalroyo}@posta.unizar.es
[2] LSI Depart., Univ. of the Basque Country, Apdo. 649, 20080 San Sebastián, Spain
{jipileca,alfredo}@si.ehu.es
WWW home page: http://siul02.si.ehu.es/

Abstract. Considering the existing tendency toward the use of wireless devices we propose in this paper a new service that helps users of those devices in the (tedious, repetitive and many times costly in terms of communication cost) task of obtaining software from the Web.
The goal of the service is twofold: First, to allow users to obtain software by expressing their requirements at a semantic level. For that the service deals with an ontology, specifically created for the service, that contains a semantic description of the software available at different repositories. Second, to guide users in the task of browsing the ontology in order to select the adequate software.
The service has been developed using mobile agent technology. A software obtaining process based on adaptive 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. In this paper we describe the main elements that take part of the service and some performance results that prove the feasibility of the proposal.
Keywords: Mobile information agents, Ontologies, Adaptive information agents, Mobile computing.

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. For that, a well-known solution is visiting some of the several websites that contain freeware, shareware and demos (such as Tucows [12] or CNET Download.com [1]). However, that approach can become cumbersome for naive users —they may not know: 1) the different programs

* 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.

that fulfil their needs, 2) the features of their computers, and, 3) the websites where to find the software— and can become annoying for many advanced users. Moreover, if those users use a wireless device, the time expended to find, retrieve and install the software 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 that allows users to find, retrieve and install software. This service presents two main features: 1) *Semantic Search*: the service allows users to express their software requirements at a semantic level and helps them to browse customized software catalogs in order to select the adequate software. For that, the service makes use of an ontology (specifically created for the service) that contains a semantic description of software available in different repositories, and so it makes transparent for users most of the technical details related to the software retrieval task. 2) *Analysis of user behavior*: the system "observes" users information requests in order to anticipate their needs and even learns from its mistakes to improve its future behavior with those users. So it offers a customized and adaptable service to users putting a special emphasis on optimizing communication time.

The Software Retrieval Service takes part of ANTARCTICA [3], a system that provides users of wireless devices with a new environment that fulfils some of their data management needs. ANTARCTICA follows the widely accepted architecture for mobile computing [11] and so it deals with users of wireless devices and hosts situated at the fixed network that we call GSNs[1].

The implementation of the service is based on the agent technology [10]. Four main agents participate in the service (see Figure 1): *Alfred*, the user agent situated at the user computer who 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; the *Software Manager* agent (situated at the closest GSN[2]) that prunes the software ontology taking into account the user requirements; the *Browser* agent, which is created at the GSN and then it moves to the user computer to help her/him to navigate the pruned ontology and select the wanted software; new information requested by the user will be retrieved by the *Catalog Updater* agent, which is created by the Browser to update the user software catalog; and the *Salesman* agent which carries the selected program to the user computer and installs it whenever possible. The interactions among these agents are explained in detail in [6].

Two main tasks take part when dealing with the service: 1) to prune the ontology 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 computer), and 2) to attend user refinements on such a catalog trying to predict future user behavior (to minimize network communication). The re-

[1] The *Gateway Support Node (GSN)* is the proxy that provides wireless users with different services like the Software Retrieval Service.
[2] There exists one Software Manager agent on each GSN.

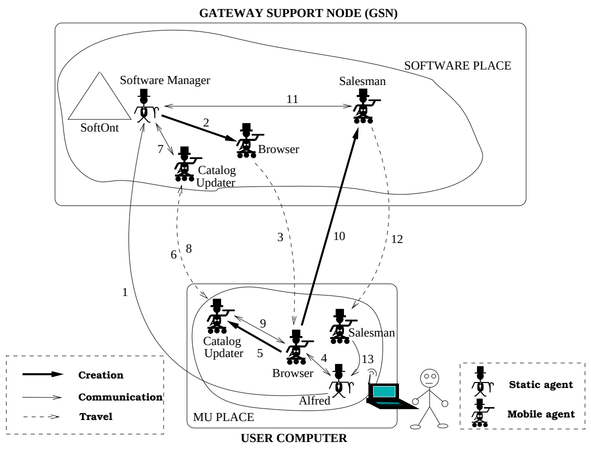


Fig. 1. Main architecture for the Software Retrieval Service

sponsible for those tasks are the Software Manager and the Browser agents, respectively.

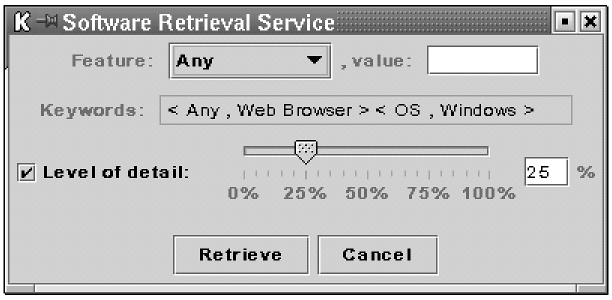
Concerning related work, to our knowledge, agents have not been widely used for software retrieval. In [4] they explain a mechanism to update several remote clients connected to a server taking advantage of mobile agents capability to deal with disconnections; however this work is more related to *push technology* than to services created to assist users in the task of updating the software on their computers. In the Ariadne project [5] they work on automatic wrapper construction techniques and build an ontology on top of each website in a semi-automatic manner. OntoAgents [2] allows the annotation of 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 these last two projects they use agents, not for retrieving software but accessing websites.

In the rest of the paper we detail the behavior, knowledge and adaptability of the Software Manager and Browser agents, which are the key of the success of the Software Retrieval Service. In Section 2 we explain how the initial interaction of the user with Alfred is performed. In Section 3 we describe how the first pruned ontology (the first catalog) is obtained. In Section 4 the catalog browsing is detailed including the analysis of the user behavior. We compare the software retrieval service with a Tucows-like approach in Section 5, and some conclusions appear in Section 6.

2 Initialization of the Software Retrieval Service (SRS)

Alfred is an efficient majordomo agent, situated at the user computer, that serves the user and is on charge of storing as much information about the user computer, and the user her/himself, as possible. Let us start with the situation in which the user wants to retrieve some kind of software. Two cases can arise:

1. The user exactly knows which program s/he needs, for example, JDK1.4.0 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, as a list of constraints *<feature, value>* describing the software they need. In the example the data entered would be *[<name, JDK1.4.0>, <OS, Win32>]*.
2. 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 in order to express their needs the best they can[3]. 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. 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.


Software Retrieval Service

Feature: Any , value:

Keywords: < Any , Web Browser > < OS , Windows >

[✓] Level of detail: 25 %

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

It is also possible that the user does not know any feature of the software. S/he could have seen it in the past but now is not able to remember the name, the exact purpose, etc. However, if s/he would see it again s/he could recognize it. Even in this case, the system will help the user as we explain in Section 3.1.

In addition to the constraints and the level of detail (if specified) Alfred can add more constraints concerning the user computer (e.g. OS, RAM memory, video card, etc.) and previous executions of the service (e.g. previous web browsers downloaded). Indeed, all the information provided by the user is stored by Alfred. Thus, with each user request, Alfred stores more information about

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

the user computer and about the user; a detailed description of the *knowledge* managed by Alfred can be found in [6].

After the user specifies her/his requirements, Alfred sends a software catalog request to the *Software Manager* agent residing at the GSN.

3 Obtaining a Software Catalog: The Software Ontology and the Software Manager

After receiving a request of Alfred (on behalf of the user), the Software Manager agent performs two main tasks: 1) To obtain a catalog corresponding to the user request, and 2) To create an agent that travels to the user computer, presents the catalog to the user, and helps her/him to find the wanted software.

For the first task, we advocate using an ontology, called *SoftOnt*, to describe *semantically* the content of a set of data sources storing pieces of software. This ontology will be stored in all the GSNs that belong to the ANTARCTICA system. The SoftOnt ontology, which stores detailed information concerning the available software accessible from the GSN, is managed by the Software Manager (one per GSN). So, instead of users having to deal directly with different software repositories, the system uses an ontology to help users to retrieve software. Structurally, SoftOnt is a rooted acyclic digraph whose inner nodes store information about software categories and whose leaves store information about programs.

In [7] we explained the process used for building the SoftOnt ontology. However, due to space limitations we summarize here the main steps involved in that process: the *translation* and the *integration* steps. In the translation step, specialized agents analyze HTML pages corresponding to several software repositories on the web, like Tucows [12]. Fortunately, those pages classify the different pieces of software in several categories and so, the specialized agents take advantage of those categories to create ontologies (one ontology per website) and they transform subcategories in websites into specializations in ontologies. In the integration step, the ontologies obtained from the different software repositories are integrated into only one ontology. Taking into account that in the considered context (software repositories) the number of data sources is low and the number of categories is not very high we advocate building only one ontology. Moreover, as the vocabulary heterogeneity problem on the context is limited, the process of integrating the ontologies can be automatized using a thesaurus for the automatic vocabulary problem resolution.

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.*

3.1 The Software Manager: Ontology Pruning

The SoftOnt ontology must be pruned in order to obtain a first software catalog to present to the user. This pruning process is very important due to three 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); 2) it avoids presenting very specialized categories and pieces of software that could surely make naive users spend more time reading the catalog, and consequently, finding the wanted software; and 3) it minimizes the communication cost by sending interesting information only.

The Software Manager is able to prune an ontology by considering different parameters:

- *Node to prune.* The Software Manager will only consider the subtree defined by the specified node and their underlying nodes. To prune the whole ontology, the node to prune should be the root node of the ontology.
- *Keywords.* They are a list of constraints <*feature, value*> that nodes in the result must satisfy.
- *Level of detail.* It is a percentage that indicates the amount of data that should be included in the result; for example, a level of detail of 30% indicates that only the 30% of the ontology should be included in the result. If keywords are specified, the level of detail is applied on the set of nodes satisfying the keywords.
- *Pruning strategy.* It indicates *which* nodes of the ontology will be selected, it is the selection criteria. Several pruning strategies have been implemented in our prototype:
    - *The most requested nodes.* The Software Manager updates global[4] statistics about the retrieval of each node in SoftOnt. Every time that a node is included in the catalog for some user, its count is increased. Thus, when this strategy is used, the most requested nodes are selected first.
    - *The most requested nodes by the user.* The Software Manager also stores which nodes are sent to each user (user statistics). Thus, by using this strategy the Software Manager selects first those nodes requested for the user in the past.
    - *The proportional strategy.* In this strategy, brother nodes[5] have the same priority. In other words, when a node must be pruned using a certain level of detail of $n\%$, then all the immediate descendants of such a node will be pruned using a level of detail of $n\%$. This strategy is very useful when the system has no idea of what the user is looking for, as it does not favour any concrete branch.
    - *The heaviest strategy.* In this strategy, the nodes with more underlying pieces of software are selected first. This strategy is based on the idea that the user could be looking for a software under very populated categories, like games.
- *Node type.* The pruning strategy could consider only nodes that represent software categories, nodes that represent pieces of software, or both. Thus, the *node type* parameter can be 'categories', 'programs', or 'nodes', respectively.

[4] Once a day, Software Managers on different GSNs can share and update their statistical information about the retrieval of each node in SoftOnt.

[5] Brother nodes are those with at least one common father node.

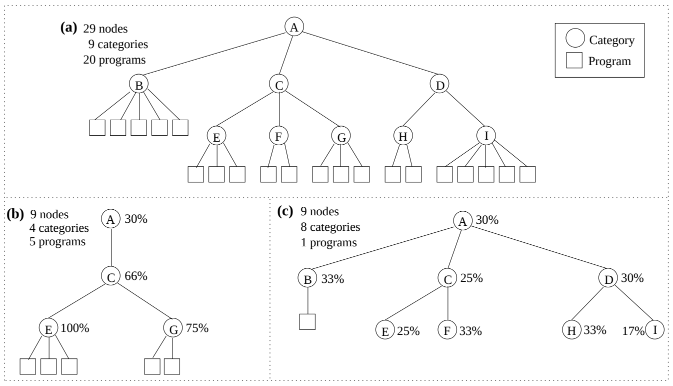


Fig. 3. Pruning an ontology: (b) with heaviest and (c) proportional strategies

Figure 3 shows the difference between two pruning strategies, when the node to prune is 'A', no keywords, level of detail is 30% and node type is 'nodes'. Notice that, independently of the pruning strategy, a level of detail of 30% indicates that only the 30% of the ontology must be obtained (9 nodes, in the example).

In the initialization of the service, the parameters sent by Alfred are: the level of detail for the whole ontology and some keywords; both parameters are optional for the first time. For future catalog refinements, the parameters sent by Alfred are: the node to prune, the level of detail for that node, and (optionally) new keywords. The rest of parameters are estimated by the Software Manager as explained in the following. Therefore, when the Software Manager is invoked to perform a prune of the SoftOnt ontology, the following steps are followed:

1. *Selecting the node to prune.* For the first catalog it will be the root node of SoftOnt, as the first time we consider the whole ontology. In future catalog updates, the user can request pruning a concrete node (see Section 4.1).
2. *Choosing the node type.* If no keyword and no level of detail was specified by the user, only categories will be included in the first catalog: the user has provided no information about what s/he wants, therefore the system will only include categories in the first catalog to help the user to choose first the kind of software wanted. In other case, the catalog will include both categories and program nodes.
3. *Pruning the SoftOnt ontology using keywords*, if any was specified by the user (pruning using keywords is very selective). Let us call $Ont_{keywords}$ to the ontology after considering the keywords:

$$Ont_{keywords} = \begin{cases} prune(SoftOnt, keywords) & \text{if any keyword} \\ SoftOnt & \text{otherwise} \end{cases}$$

   Only nodes of the kind selected in step 2 which fulfill all the keywords will be included in the result.

4. *Setting the level of detail.* If the user specified a level of detail, the system must provide her/him with at least such a level of detail. However, as a remote connection must be opened to return to the user computer the catalog requested, system estimates if it is worth to retrieve more information than what it has been requested, i.e., to consider a higher level of detail to avoid future network connections. This estimation is based on the current network status and a concrete percentage $\%_{incr}$ specified by Alfred (different users could have different increments depending on their expertise, computer, network connection, etc.). Technical details about how this increment is obtained in run-time can be found in [9]. This approach improves the efficiency of the system when the user successively requests lightly higher level of detail of the some node.
   If the user did not specify a level of detail (only possible for the first software catalog), the Software Manager will consider a level of detail of $\%_{incr}$.
5. *Applying a pruning strategy on $Ont_{keywords}$*, using the parameters obtained in the previous steps. For the first catalog, the proportional pruning strategy is selected, because it prunes brother nodes proportionally, which is a good idea for the first catalog. In future catalog updates, the system will automatically select the most suitable strategy (see Section 4.2). Let us call Ont_{pruned} to the result of this task.
6. *Obtaining an incremental answer.* The first catalog will be the complete Ont_{pruned}. Moreover, to avoid sending data that are already on the user computer, the Software Manager stores the ids of the nodes sent to each user[6] ($nodes_{user_i}$) and, in future catalog updates, it removes those nodes from the catalog obtained. Thus, only the *new* information is sent.
7. *Compressing the catalog obtained.* The information obtained in the previous step can be compressed to reduce the use of the network when sending a catalog to the user computer. In [9] we show when it is worth to compress the catalog by considering the catalog size, the current network status and other parameters measured in run-time.

Notice that, even when the user did not specify any keyword or level of detail, the system automatically sets during steps 1, 2, 4, and 5 the most appropriate values for the parameters needed to perform a prune (node to prune, node type, level of detail, and pruning strategy).

3.2 Creating the Browser agent

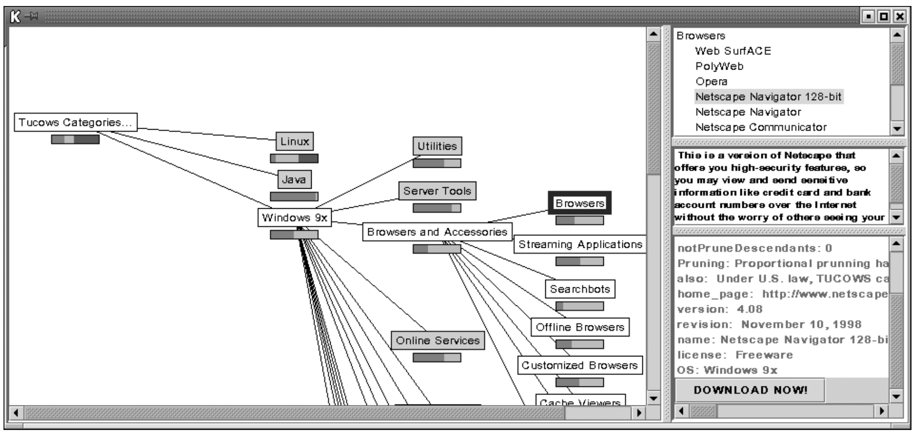
After the first catalog is obtained the Software Manager creates a Browser agent initialized with such a catalog. This specialized agent will travel to the user computer and help the user to find the wanted software as explained in the next section. It is important to stress that, although the Software Manager could have

[6] This information is also stored by the Browser agent at the user computer, thus when the user changes to another GSN (*handoff*) the Browser tells the new Software Manager which are the nodes already retrieved.

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 4.3.

4 Catalog Browsing: The Browser Agent

Once on the user computer, the Browser agent presents the catalog as a rooted acyclic digraph (see Figure 4) where nodes are software categories (shaded nodes represent nodes whose descendants are hidden). 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 (in middle grey); 2) how much information about that node is available at the user computer (in light grey); and 3) how much new information about that node could be requested to the Software Manager (in dark grey). For example, concerning the node 'Linux' in Figure 4, the Browser is showing the 10% of all the information available in the ontology, the 58% is available at the user computer, and the 42% remaining could be remotely requested to the Software Manager.



Browsers
Web SurfACE
PolyWeb
Opera
Netscape Navigator 128-bit
Netscape Navigator
Netscape Communicator

This is a version of Netscape that offers you high-security features, so you may view and send sensitive information like credit card and bank account numbers over the Internet without the worry of others seeing your

notPruneDescendants: 0
Pruning: Proportional prunning ha
also: Under U.S. law, TUCOWS ca
home_page: http://www.netscape
version: 4.08
revision: November 10, 1998
name: Netscape Navigator 128-bi
license: Freeware
OS: Windows 9x

Fig. 4. Browsing the catalog

In the following we explain the different actions that the user can perform on the catalog, how the Browser analizes the user behavior to anticipate future actions, and how the catalog refinements that request new information about some node are managed.

4.1 Navigating the Catalog: User Actions

The following are the different actions that a user can perform after studying the catalog presented:

- *To ask for information about a node.* Just by left-clicking on 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.
- *To open/close a node.* By double clicking on a node, its immediate descendants are shown/hidden.
- *To prune some node.* By right-clicking on a node, the user has the possibility to specify a new level of detail for that node or provide new constraints for that node and its descendants. Thus, different actions can be performed:
    - *To request less detail of a node*, when too many descendants below that node are shown, which makes the task of finding the wanted software too confusing for naive users.
    - *To request more detail of a node*, as the user could suspect that the wanted program could be under such a node. The Browser could have the requested information (no remote communication would be needed) or not (the Browser will have to remotely request those data to the Software Manager at the GSN). Sections 4.3 and 4.4 detail this task.
    - *To provide new constraints*, as the user could have remembered some feature of the wanted program therefore could want to provide a new constraint (a new pair <*feature, value*>). As the Browser has pruning capabilities, that task can be done locally[7], on the user node, without any remote connection.
- *To download a program*, when user has (fortunately) found a piece of software that fulfils her/his needs. As a consequence of this 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 computer carrying the specified program. See [8] for a more detailed description of the tasks performed by the Salesman.

4.2 Automatic Pruning Strategy Selection: Analyzing the User Behavior

We explained in Section 3 that catalogs can be pruned in different ways, what we call different pruning strategies. Thus the pruning strategy indicates *which* nodes of the ontology will be selected. The result of applying different pruning strategies is different although the same number of nodes is selected. Therefore, different pruning strategies select first certain kind of nodes (the most frequently requested, those with more programs, etc.).

In order to minimize the number of user refinements, i.e., the number of user actions needed to find the wanted software, the Browser tries to anticipate future user actions by analizing her/his past behavior. Thus, the Browser stores the nodes in which the user had some interest in the past. When the Browser detects that the user seems to follow a pattern that corresponds with the nodes that would have been selected by some of the available pruning strategies, then that pruning strategy will be used in the next prune.

[7] In our prototype, different constraints are joined by a logical AND operator.

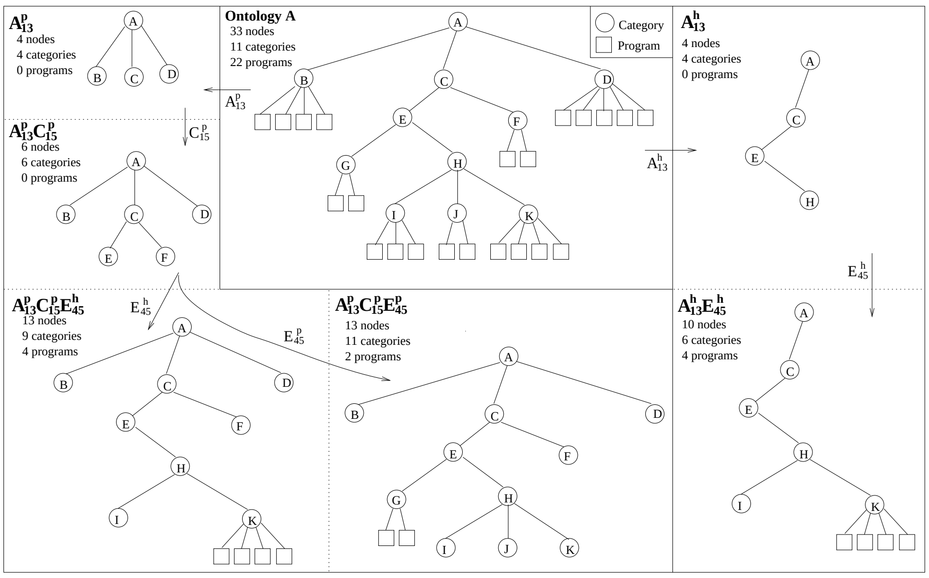


Fig. 5. Analyzing the user behavior and selecting new pruning strategies

In Figure 5 we show how the catalog changes with different pruning strategies. We use the following notation to indicate the different prunes: N^{str}_{lod} means that node N was pruned using the *str* pruning strategy (abbreviations: 'p' = proportional, 'h' = heaviest) and a level of detail of $lod\%$; for example, $< A^p_{13}C^p_{15}E^h_{45} >$ represents the result after 1) pruning node A proportionally and a level of detail of 13%, 2) pruning the resulting node C proportionally and a level detail of 15%, and finally 3) pruning node E using the heaviest strategy and a level of detail of 45%. In this example a threshold of three[8] is used to change the pruning strategy, and the proportional prune is used by default. 1) In $< A^p_{13} >$ we show the first catalog presented to the user (the whole is pruned for the first catalog). 2) The user requests a higher level of detail of node C; the pruning strategy remains the same as in the first prune, and the result is $< A^p_{13}C^p_{15} >$. And 3) the user requests a higher detail about the node E; it is the third time that the user selects the path that would have been shown if the root node would have been pruned using the heaviest strategy ($< A^h_{13} >$). Therefore the Browser selects the heaviest pruning strategy in order to make easier to the user the task of finding the wanted software, as s/he seems to be interested in nodes with many programs; the result is $< A^p_{13}C^p_{15}E^h_{45} >$. With the proportional strategy the result would be $< A^p_{13}C^p_{15}E^p_{45} >$. Notice that if the user was looking for programs under node K, the heaviest strategy would have selected those nodes just in two

[8] The default threshold should be a small value, three or five, and that threshold will increase each time the Browser detects an error in its estimation. The system stores a threshold for each user and pruning strategy.

user refinements ($< A^h_{13}E^h_{45} >$; however, selecting the heaviest strategy from the beginning is a very risky choice, there exist many chances to fail in helping a user for which the system has no information about what s/he is looking for.

Therefore, the Browser counts the nodes for which the user shows any interest and, whenever the user seems to follow a recognized pattern during a certain time, the corresponding strategy will be selected. Whenever the user does not follow the pattern of the current pruning strategy, the Browser will select the proportional strategy (which is the least risky) until a new pattern is followed. The Browser "remembers" previous mistakes, and the threshold of a rejected strategy is augmented anytime the user stops following its pattern; thus the Browser tries to improve its predictions.

4.3 Treating a New Refinement Locally

Some actions selected by the user can be performed by the Browser itself, without using network resources:

– *The user requests to open/close a node.* The Browser simply shows/hides the descendants of such a node, no new information is needed.
– *The user requests a lower level of detail of some node.* As no new information is needed and the Browser has pruning capabilities, it prunes the selected node by considering the level of detail indicated by the user. In this case, the bar corresponding to that node will indicate now that less information is shown; however, the indicator that represents the Browser buffer about that node will remain the same (because the amount of information locally available is still the same).
– *The user requests a higher level of detail below the buffer limit.* Again, the Browser has already all the needed information and can prune the catalog properly.

Notice that, by using the Browser pruning capabilities and the Browser buffer, many user refinements can be performed without using network connections. Nevertheless, if the requested refinement cannot be performed using the information currently available to the Browser, then the new information must be requested remotely to the Software Manager as explained in the following subsection.

4.4 Treating Refinements that Implies Using the Network

Some refinements requested by the user cannot be performed by the Browser itself: the user can request information that the Browser does not have, so a remote request to the Software Manager is necessary. For that task, the Browser creates a *Catalog Updater* agent[9], whose goal is to retrieve from the GSN the

[9] The Catalog Updater could be remotely created on the GSN; however if that remote creation fails due to network unstability, the Browser should retry such a task. Creating the agent locally permits the Browser to depute the Catalog Updater to manage network communications

needed information, by requesting the Software Manager to prune the node subject of the user refinement with the specified level of detail and keywords.

To achieve its goal, the Catalog Updater agent can follow two alternatives: 1) a remote call from the user computer to the Software Manager at the GSN, or 2) to travel to the GSN, to communicate with the Software Manager locally, and travel back to the user computer. To select a choice, the Catalog Updater considers the number of retries needed to maintain a network connection open during a certain time (see [9] for details about this estimate). 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.

As explained in Section 3, new catalogs are returned in an incremental way (only new information is retrieved to optimize communication costs). Thus, the Catalog Updater merges the previous user catalog with the new information properly, 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.

5 Performance Evaluation: SRS vs. Tucows

In this section we compare the use of the Software Retrieval Service (SRS) with the use of Tucows[10]. Figure 6 shows a real sample session with the two systems: (a) the network use and user refinements using Tucows to find a certain software (a CAD tool), and (b) the same situation when using SRS. Axis-x represents time in minutes; lines above axis-x represent access to the network and lines below axis-x represent user refinements (the longest line represents the moment in which the user found the wanted software).

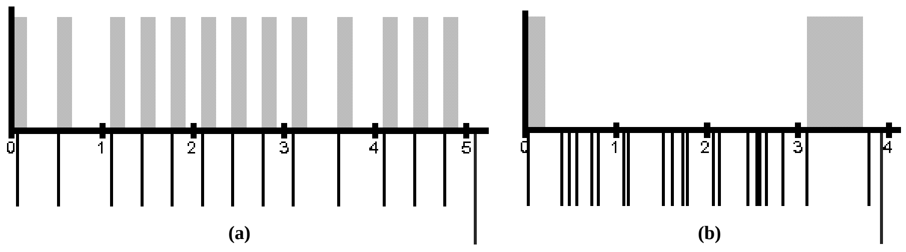


Fig. 6. Network use and user refinements using (a) Tucows and (b) SRS

We can observe that in Tucows (Figure 6.a) there is a network access for each user refinement which makes necessary a continuous connection to the net-

[10] Data was obtained after testing both software retrieval methods by different kinds of final users. Each user retrieved several pieces of software, first with the SRS and then with Tucows. Most of the users already knew Tucows.

work. However, in SRS (Figure 6.b), as the system is able to manage some user refinements without remote access, there exist long time gaps for which SRS did not access the network. This feature makes the system more robust to temporal network disconnections, and enables considering an automatic mechanism that decides to disconnect the user computer from the net, to reduce the cost of GSM wireless connections.

In Figure 7 we show how the total time was spent in different tasks (a) when using Tucows and (b) when using SRS. Axis-x shows the different software that users looked for, and axis-y shows the average time spent. Notice that, although using Tucows can sometimes be faster, using SRS reduces the network communication cost (for example in Figure 7, when looking for a DBMS).

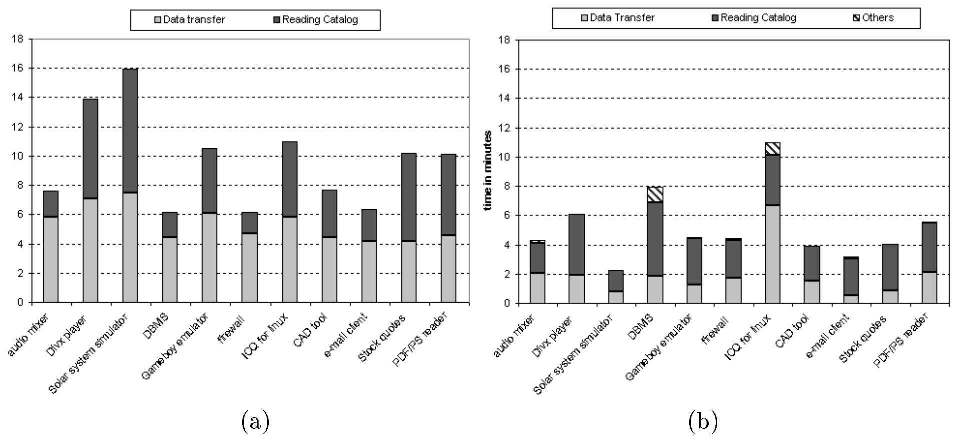


Fig. 7. Time-consuming tasks for different (a) Tucows and (b) SRS sessions

6 Conclusions

Taking into account the widespread use of mobile computers, we have presented in this paper a service that allows users of those computers to retrieve software from existing software repositories in an easy, guided and efficient way. Easy, because the service allows users to express their software requirements at semantic level, i.e., they express what they need but not how to obtain it. Guided, because the service, using specialist knowledge-driven agents, only presents to the user those software categories related to her/his requirements (a customized catalog) and helps her/him to browse those categories until the wanted software is found. Finally, it is efficient because, although the service can be used on any kind of computer, it puts on a special emphasis on mobile users, saving wireless communications. The reported performance results, obtained using the implemented prototype, corroborate this when comparing the service with a more classical way of obtaining software, such as accessing the Tucows website.

7 Acknowledgements

We would like to thank V. Pérez his priceless help in the implementation of the prototype. Special thanks to many anonymous users that tested our system for performance evaluation.

References

1. CNET Inc., 1999. http://www.download.com.
2. G. Wiederhold 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. IBM Corporation. TME 10 Software Distribution - Mobile Agents SG24-4854-00, January 1997. http://www.redbooks.ibm.com/abstracts/sg244854.html.
5. C.A. Knoblock, S. Minton, J.L. Ambite, N. Ashish, I. Muslea, A.G. Philpot, and S. Tejada. The ariadne approach to web-based information integration. *To appear in the International the Journal on Cooperative Information Systems (IJ-CIS) Special Issue on Intelligent Information Agents: Theory and Applications*, 10(1/2):145–169, 2001. http://www.isi.edu/info-agents/ariadne/.
6. 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.
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. E. Mena, J.A. Royo, A. Illarramendi, and A. Goñi. Adaptable software retrieval service for wireless environments based on mobile agents. In *2002 International Conference on Wireless Networks (ICWN'02), Las Vegas, USA*. CSREA Press, June 2002.
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. E. Pitoura and G. Samaras. *Data Management for Mobile Computing*. Kluwer Academic Publishers, 1998.
12. Tucows.Com Inc., 1999. http://www.tucows.com.