Web-based versus mobile agent-based software retrieval systems: Performance comparison[⋆]

José Merseguer, Javier Campos, and Eduardo Mena

Dpto. de Informática e Ingeniería de Sistemas, University of Zaragoza, Spain
{jmerse,jcampos,emena}@posta.unizar.es

Abstract. Formal methods are the most suitable way to model and analyze several kinds of software systems. However, conventional methods are gaining placed in Software Engineering field because they can be easily applied in all the stages of the software life cycle. The combination of formal and conventional methods could be an interesting approach to describe and analyze some software aspects, as for example, the performance of the system. In this way, we make use of a software performance process enriched with formal techniques. The process has as important features that it uses UML as a design notation and it uses stochastic Petri nets as formal model. Petri nets provide a formal semantics for the system and a performance model, while UML supplies the framework and tools to document the system.
The process is used in this article to present a performance comparison between software retrieval systems. Nowadays, there exist web sites that allow users to retrieve and install software in an easy way. The performance of these sites may be poor if they are used in wireless networks; the reason is the inadequate use of the net resources they need. In this article, we show that if this kind of systems are designed using mobile agent technology the previous problem might be avoided.

Keywords: Software performance engineering, stochastic Petri nets, Internet, UML, wireless networks, mobile agent technology.

1 Introduction

The tasks of retrieving and installing software using Internet have become common in the last years. There are sites (e.g., Tucows.com [3], Download.com [1], and GameCenter.com [2]) which permit to perform these tasks in an easy and friendly way. But the process of selecting the software could become costly and sometimes slow: the user must select a great number of categories until s/he finds the desired software, specially if s/he is a naive user. Every time the user selects a category, the web site sends a new HTML page with the contents of the category. This technique makes an intensive use of the network connection, which can result in a poor performance and expensive communication cost (in a wireless environment). Thus, it would be interesting to get results about the performance of this kind of systems to exactly know how costly is this process.

[⋆] This work has been developed within the project UZ00-TEC-03 of the University of Zaragoza.

A new technology, *mobile agents* [14], can aid to reduce the network connection time. We recall in this paper a software retrieval service belonging to the ANTARCTICA system [9] which has been designed using mobile agents. The aim of this service is to propose an alternative method to the current web-based software retrieval systems, called in this article "Tucows-like" systems. The ANTARCTICA system has been though to be used in wireless communication environvents where net speed is a problem, say 800 bytes/sec. is real in GSM networks. Therefore, it promotes a better use of the net resources, thus supplying better performance results. In the following, we refer to the ANTARCTICA software retrieval service as the "ANTARCTICA SRS".

Software performance engineering [16] proposes evaluating performance of software systems in the early stages of the development process. Thus, if performance problems are detected, it will be easier and less expensive to take the appropriate design decisions to solve them. The *Unified Modelling Language* (UML) [4] is the design notation that we will use to model software retrieval systems. The selection of UML is motivated because it is widely accepted by the software engineering community and it fits very well in the software life cycle. Since we are interested in performance aspects, we use UML extended with performance annotations. In general, software systems are complex systems even more if they are distributed, so we advocate for the use of formal models to obtain performance indices. But UML lacks of the necessary formal semantics to apply analytical techniques. Thus, we propose to semi-automatically obtain *Petri nets* (PNs) [13] from UML diagrams and to use them to obtain performance indices. PNs are a widely used formal model for concurrent systems and provided with a stochastic time interpretation, they are suitable for performance evaluation. Concretely, we use stochastic well-formed coloured nets (SWNs) [5].

Our goal in this work is to compare the performance of Tucows-like systems with the performance of the ANTARCTICA SRS using analytical software performance techniques. The performance index under study for both systems is the *network time*, i.e., how much time the network connection needs to be open in order to retrieve a software product. Also, it will be studied the impact of the intelligent agents in the ANTARCTICA SRS.

The rest of the paper is organized as follows. In section 2, the process followed to evaluate software performance is described. In section 3, a Tucows-like software retrieval system is modeled using UML and this model is translated into SWNs. In section 4, the main characteristics of the ANTARCTICA SRS are described. In section 5, we present a comparison between performance figures for both approaches. Finally, in section 6, some concluding remarks and related work are stressed.

2 The software performance process

"The software performance engineering (SPE) is a method for constructing software systems to meet performance objectives. SPE augments others software engineering methodologies; it does not replace them" [16]. In this section, we

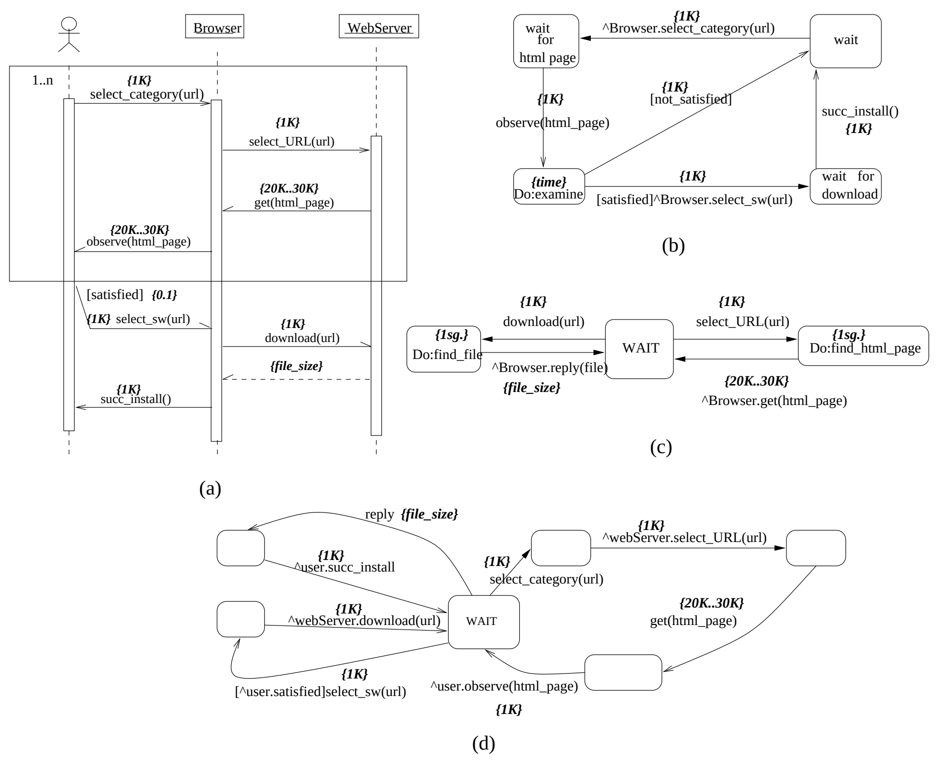


Fig. 1. (a) Annotated sequence diagram for the Tucows-like system. Annotated STDs for the Tucows-like system: (b) user, (c) the web server, and (d) browser

briefly present a software performance process, introduced in [11, 10], that will be applied in sections 3 and 4 to the software retrieval systems proposed in the introduction. An interesting characteristic of this process is that it uses a formal model (SWNs) to obtain performance indices. This formal model can be obtained semi-automatically inside the software development process. In this way, without much effort, the process allows to obtain a performance model as a by-product and it preserves the benefits of the software design methodologies.

2.1 Modeling using performance annotated UML diagrams

The process begins by modeling the system dynamics in a conventional way. The UML dynamic diagrams [4] will guide the process. *Sequence diagrams* and *state transition diagrams* (STDs) will be used in this paper. *Activity diagrams* are not used because we are not interested in modeling the actions performed by the systems that we model.

Figure 1.a depicts an example of sequence diagram, that will be explained later on. A sequence diagram represents messages sent among objects. Messages sent among objects on the same computer are considered as no time consuming in the scope of the modeled system (for instance, the messages sent between the user and the browser in Figure 1.a). Messages sent among objects on different computers, those which travel through the net, will consume time as a function of the message size and the net speed (for instance, in Figure 1.a the messages sent between the browser and the web server). Each message size is annotated inside braces. For instance, the select_URL message is labelled with {1 Kbyte} in Figure 1.a. It is also possible to annotate the size with a range in the UML common way (like the get message with label {20K..30K} in Figure 1.a). If the message size is unknown, the annotation is a label representing a performance parameter (e.g., the return of the download message is annotated with the label {file_size}, also in Figure 1.a).

In a sequence diagram, conditions represent the possibility for the associated message to be dispatched. Annotations, also between braces, express the event probability success associated with each condition (for instance, see probability {0.1} associated with the condition [satisfied] in Figure 1.a). A range or a probability (unknown) parameter are accepted too as event probability success.

In order to get a complete view of the system dynamics, a STD for each class with relevant dynamic behaviour must be developed, as those depicted in Figures 1.b, 1.c and 1.d. To study performance aspects on a STD, two elements are meaningful, the *activities* and the *guards*. In the following, we briefly comment the annotations used in these kind of diagrams.

Activities represent tasks performed by the object, therefore, they consume computation time that must be annotated. The annotation will be done between braces. For example, see in Figure 1.c the bold label {1 sec.} close to the activity find_file. If it is necessary, minimum and maximum values will be annotated.

Guards show conditions in a transition. They must hold in order to fire the event that they label. The probability associated to them, already annotated in the sequence diagrams, are also indicated in STDs to gain readability. In the same way, the size of the messages may be annotated or omitted.

2.2 From performance annotated UML diagrams to SWNs

From the performance annotated UML diagrams it will be interesting to obtain performance indices for the system. But, as we said, UML lacks of the necessary formal semantics to apply quantitative analysis techniques to obtain them. Even more, it is not suitable to specify some system aspects, for example concurrency, which is fundamental in performance evaluation. Therefore, the UML diagrams will be translated into SWNs, which allow to obtain performance indices, taking the proper decisions for the unspecified system requirements.

The strategy to obtain the SWNs from the performance annotated UML diagrams is as follows.

1. *Derivation of a SWN from each STD.* These SWNs will be called *component nets.* They represent the behaviour for each class with the underlying SWN formal semantics. To obtain a component net from a STD several rules must be applied, details can be found in [11], the following are the most important:

  - Each state of the STD is represented by a place, with the same name, in the SWN.
  - For each transition in the STD, there will be in the SWN:
    (a) A transition with the same as the event which labels the transition in the STD.
    (b) An arc from the place (which represents the initial state in the STD) to the transition in the SWN (which represent the transition in the STD).
    (c) An arc from the transition in the SWN (which represent the transition in the STD) to the place (which represents the final state in the STD).
  - Guards in the STD will become immediate transitions with the associated corresponding probabilities for the resolution of conflicts.
  - Activities inside a state of a STD are considering as time consuming, so in the SWN they will be considered as timed transitions; the rate of the exponentially distributed service time of the timed transitions are obtained automatically from the time annotation of the STD.

  As an example, see in Figure 2 the component nets obtained from the STDs in Figure 1.
2. *Obtaining a SWN for the system.* From the component nets, and guided by the sequence diagram/s, a *complete* SWN for the system is obtained. Transitions in the component nets that represent the same message are synchronized if the message has wait semantics; on the other hand, if the message has no wait semantics the transitions are connected with an extra place, modeling a communication buffer.

The outcome SWN models the behaviour of the whole system. Figure 3 depicts an example of a complete net obtained from the component nets in Figure 2 and from the sequence diagram in Figure 1.a. The performance figures for the system are obtained by analyzing the complete SWN for the system.

3 Modeling the Tucows-like software retrieval system

There are a variety of software retrieval systems, as the popular web sites Tucows.com [3], Download.com [1] or Gamecenter.com [2], that provide Internet users with facilities to retrieve and install software. These systems allow users to find software in two different ways, by using a keyword-based search engine or by navigating through categories especially designed to make the task easier.

The software architecture of these kind of systems for the navigation facility is basically the same, therefore, it is possible to model how these kind of systems work, making a number of assumptions, without loosing reality with respect to performance aspects. We will refer to these kind of systems as *Tucows-like* systems. Our intend is to evaluate performance indices for a Tucows-like system in order to compare them with those obtained for the ANTARCTICA SRS (modeled and evaluated in [10]).

The keyword-based search engine offers help to those users that know some features of the wanted software. This search facility will not be considered in this paper since it does not exist its counterpart in the ANTARCTICA SRS, therefore no comparison can be made. The navigation facility consists of several web pages residing in a server and organized as categories linked between them in a way that guides the user to find the software. For instance, a number of these systems present an initial web page where the categories correspond with different operating systems, say Windows 2000, Windows 95/98, Linux or Unix Themes. The ANTARCTICA SRS offers a mechanism to retrieve software similar to the navigation facility, but it makes use of intelligent agents to perform the task, therefore a performance comparison can be made between the two systems.

In this section, we apply the software performance process to the navigation facility of the Tucows-like systems.

3.1 System description and modeling assumptions

In a Tucows-like system, the *user* navigates, with the help of a *browser*, through different HTML pages (representing software categories and descriptions of concrete pieces of software) until s/he finds a piece of software that satisfies her/his needs. Then, that piece of software is downloaded.

It must be assumed that the user spends some *time reading* the information presented by the system. An exponentially distributed random variable with parameter $\lambda_{examine}$ ($\lambda_{examine}$ is obtained as the inverse of the time in seconds) will be used to model several kinds of users.

The number of HTML pages that the user must navigate until s/he finds the software is difficult to estimate (it depends on her/his experience). The probability that the user finds the software by selecting n categories models different kinds of users, from naive users, those who need to visit many categories to find the software, to expert users, those who find the software visiting very few categories.

Whenever the user requests an HTML page or a concrete piece of software, the web server must perform the corresponding activities to find the page or the piece. The time consumed by these activities will be modeled as exponentially distributed random variables with parameters $\lambda_{findHTML}$ and $\lambda_{findFile}$.

The browser, in the client machine, sends messages through the net to the web server in the server machine and vice versa. A exponentially distributed random variable with parameter λ_{m_i} will model the time spent by the message i navigating through the net. Notice that the messages sent between the user and the browser do not consume net resources.

3.2 UML diagrams with performance annotations

In this section we model the dynamic view of the Tucows-like system using UML notation as proposed in [11]. The *sequence diagram* in Figure 1.a shows the messages sent among the objects in the system with the purpose to retrieve the piece of software that the user needs. Two different kinds of messages can be distinguished, those that travel through the net (sent between the browser and the web server) and those that do not (sent between the user and the browser). This feature will be relevant in the SWN model in order to associate time to transitions that represent messages send through the net, taking into account the assumptions made in the previous section.

The sequence diagram in Figure 1.a begins with a select_category(url) message, its size is {1 Kbyte}, sent by the user to the browser. It represents the "click" performed by the user in the browser to select a category in a HTML page. The rest of the diagram describes in the same way the steps explained in the previous section for selecting software.

In order to get a complete description of the Tucows-like system dynamics and its load, we are going to develop the STD for each class with relevant dynamic behaviour.

User state transition diagram. In Figure 1.b, the behaviour of a user is represented. The user is in the wait state until s/he activates the select_category event. This event sets the user in the wait for HTML page state. The observe event, sent by the browser, allows the user to perform the examine activity that has associated the label {time}. This label models the time that the user spends reading the HTML page. This activity will be translated in the SWN net in an exponentially distributed transition, and by modifying its rate different kinds of users can be modeled, as we pointed out in the previous section. Once the activity is performed two situations can arise:

– If the requested software is not present in the current HTML page the user returns to the wait state.
– In other case, the user sends the select_sw(url) message to the browser, where url means the direction where the software is located in the server, and enters in the wait for download state. When the browser fulfills the necessary activities to complete the download, it sends to the user the succ_install() message and the user returns to the wait state.

Browser state transition diagram. Figure 1.d shows the browser's STD. The browser behaves as a server object: it is waiting for user's requests, represented by select_category and select_sw events.

When a select_category event arrives requesting a url, the browser sends to the web server the select_URL message and waits for a new HTML page. When the web server obtains results, it triggers the get event attaching the new HTML page, whose estimated size is {20K..30K}. Since this message is sent through the net, it will be translated in the SWN as an exponentially distributed transition with rate $\lambda_{m_{get}}$, as we pointed out in the previous section. After that, the HTML page is shown to the user.

When a select_sw event arrives requesting a url that contains a piece of software a download message with the url is sent to the web server. The browser waits for the reply message that contains the requested file with size file_size, it will be translated in the SWN in a transition with rate $\lambda_{m_{reply}}$. Finally, the file is installed (succ_install).

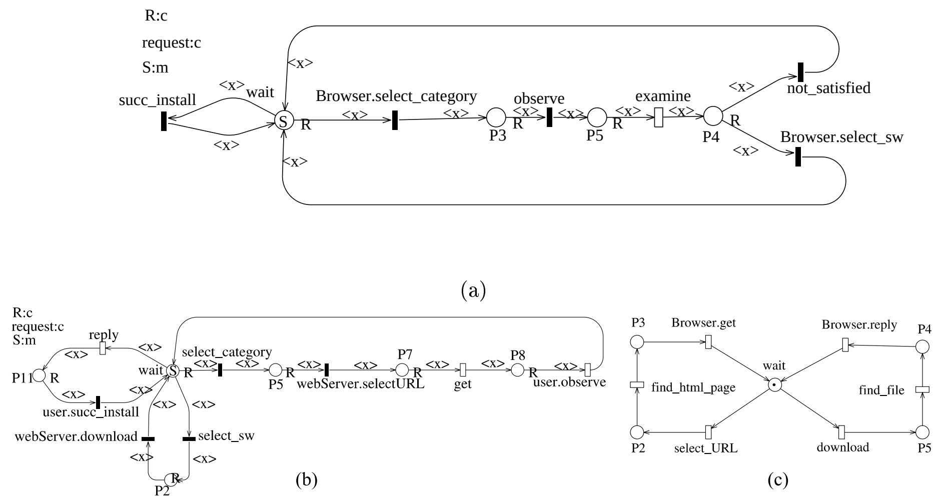


Fig. 2. Component PNs for the Tucows-like system: (a) user, (b) browser, (c) web server.

Web server state transition diagram. As the browser, the web server behaves as a server object. It is waiting for a request (select_URL and download) from the browser. For each request, the web server performs the corresponding actions to serve it (find_html_page and find_file). When the actions are completed, it sends the corresponding message to the browser. Figure 1.c shows the web server's state transition diagram.

3.3 Modeling the system with SWNs

In this section, we detail the SWN model for the Tucows-like system. The nets have been obtained by applying the translation rules, given in [11] and schematically shown in section 2. In order to model situations that the Petri nets can express but the UML notation cannot, the appropriate decisions will be taken and commented.

Figure 2 represents the *component nets*, those obtained from the STDs, for the Tucows-like system and Figure 3 represents the net for the whole system, obtained by synchronizing the component nets. We start describing the component nets.

User component net. The number of tokens in the place wait models how many concurrent users supports the system. This parameter can not be modeled in the UML diagrams.

The firing of the transition named Browser.select_category models the dispatch to the browser of a message to specialize the current HTML page, it will arrive when the transition named observe fires. The firing of the transition named

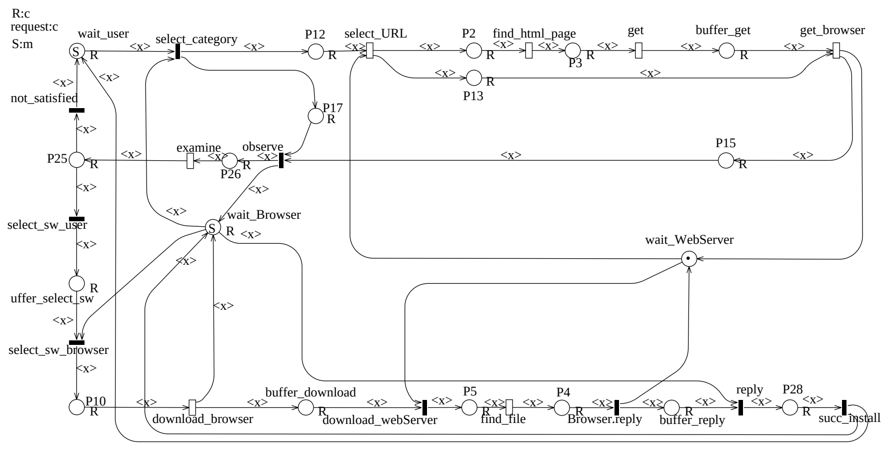


Fig. 3. Complete SWN for the Tucows-like system.

examine models the time spent by the user reading the information presented in the new HTML page. After the end of the reading, a choice will determine whether the user is satisfied with any of the products shown (firing of the immediate transition Browser.select_sw), or not (firing of the immediate transition not_satisfied).

The firing of the immediate transition named succ_install models the arrival of a message to confirm that the retrieve of the software has been successfully completed.

Browser component net. The number of tokens in the place wait models how many concurrent browser access to the system. The colour is the same as in the user component net to identify each browser with a user. This parameter cannot be modeled in the UML diagrams.

The firing of the transition named select_category models the arrival of messages from the user requesting for a specialization of the category that s/he has examined, the transition is immediate because both, the sender and the receiver, are in the client machine. The request is sent to the web server through the *net*, therefore consuming time by firing the timed transition named webServer.select_URL. The firing of transitions get and user.observe models, respectively, the obtaining of the HTML page with new categories and its dispatch to the user.

The firing of the transition named select_sw models the arrival of messages from the user requesting a concrete piece of software. The request is sent to the web server through the *net* by firing the timed transition named webServer.download. The firing of the timed transition named reply models the obtaining of the file requested by previous transition. Finally, the firing of the transition named user.succ_install models the advertisement to the user that the retrieve of the software has been successfully completed.

The select_category and select_sw transitions will be synchronized in the complete net with the transitions in the user component net with the same name.

Web server component net. The number of tokens in the place wait models how many concurrent processes has opened the web server to attend browser's requests. This parameter could not be modeled in the UML diagrams.

The firing of the timed transition named select_URL models the arrival of a remote message to request for a new HTML page. The firing of the timed transition named download models the arrival of a remote message to request for a concrete piece of software. The firing of the timed transition named find_html_page models the completion of the search for a new HTML page. The firing of the timed transition named find_file models the completion of the search for a requested piece of software. The firing of the timed transition named Browser.get models the dispatch of the HTML page to the browser. Finally, the firing of the timed transition named Browser.reply models the dispatch of the file to the browser.

The select_URL, download, Browser.get and Browser.reply transitions will be synchronized in the complete net with the transitions in the browser component net with the same name.

As we said before, the net for the whole system, depicted in Figure 3, is obtained by applying the rules given in [11].

4 The ANTARCTICA Software Retrieval Service

The goal of the ANTARCTICA SRS [9] is to provide mobile computer users with a service to select and download software in an easy and *efficient* way. *Efficient* because the system optimizes battery consumption and wireless communication costs. It provides several interesting features:

– The system manages the knowledge needed to retrieve software without user intervention, using an ontology.
– The location and access method to remote software is transparent to users.
– There is a "catalog" browsing feature to help user in software selection.
– The system maintains up to date the information related to the available software.

The software performance process given in [11] was applied to the ANTARCTICA SRS in [12, 10]. The UML diagrams, the SWNs components and the SWN net for the system were obtained. The SWN net for the system will be used in the next section to obtain performance results. It will be interesting to compare the performance of the ANTARCTICA SRS with the performance of the Tucows-like system, in order to obtain conclusions about the impact of mobile agent technology in a wireless network (low speed, costly, disconnections).

5 Performance results

The results presented in this section have been obtained from the *complete* SWNs which model the Tucows-like system (see Figure 3) and the ANTARCTICA SRS (see [10]). It is of our interest to study how much time the systems need to be connected to the net, *network time*, in the presence of a user request. Also, it is interesting to know how much intelligent the browser agent in the ANTARCTICA SRS must be, to obtain the same or better results than the Tucows-like system.

In order to obtain the network time in the Tucows-like system, the throughput of the succ_install transition will be calculated by computing the steady-state distribution of the isomorphic *Continuous Time Markov Chain* (CTMC) with *GreatSPN* [6]. The inverse of the previous result gives the network time. In the ANTARCTICA software system the target transition is select_sw_service.

To study the network time, we have developed a test taking into account the following scenarios:

1. To test the *user refinement request*, we have considered six different possibilities. A user requesting a mean of 5, 10, 20, 30, 40 and 50 refinements[1] (modeling different expertise of the user).
2. Two different kinds of users have been considered: a user who spends 10 sec. to study the information presented by the system (web page or a software catalog) and a user who spends 60 sec. in that task (modeling the information processing speed of the user).
3. We have considered two cases for the *net speed*: 1 Kbytes/sec. and 5 Kbytes/sec. By considering these low speed values we want to compare the performance of both approaches in a wireless computing environment (*real* GSM network speed is around 800 bytes/sec).

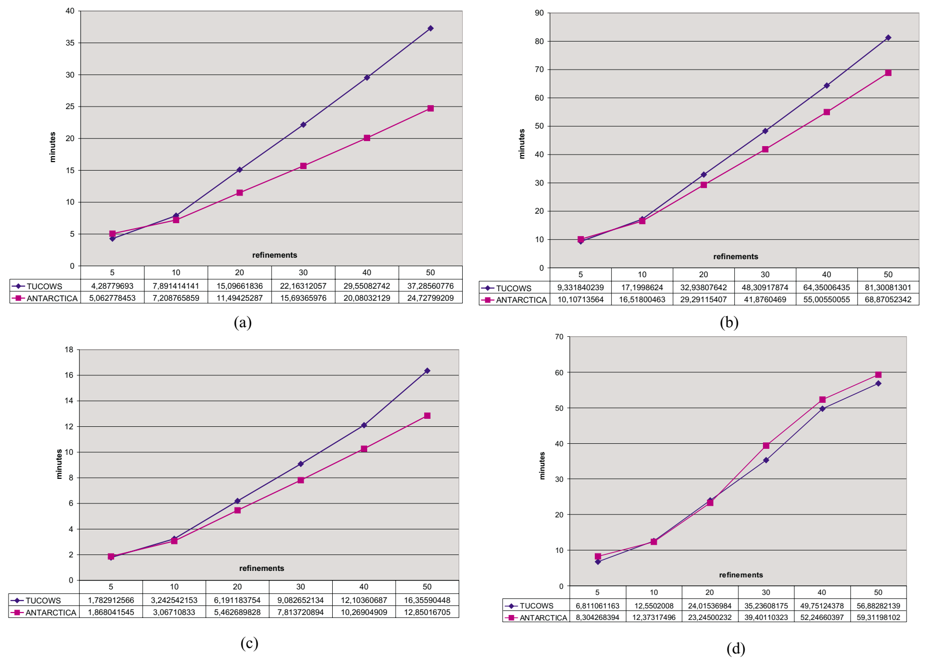
For the ANTARCTICA SRS, we have also considered:

1. A browser which does not need to ask for information to the software manager the 70% of the times that the user asks for a refinement. When the browser needs information, it requests the information by a *remote procedure call* (RPC).
2. *The size of the catalog* obtained by the browser is 50 Kbytes.

Figure 4 shows network time (in minutes) for the Tucows-like system and the ANTARCTICA SRS in different scenarios. Concretely in Figure 4.b we can observe that when the net speed is 1 kbyte/sec., the user is naive and performs 50 refinements (the worst case), then the ANTARCTICA SRS is almost thirteen seconds faster than the Tucows-like system. The same results are obtained if the user is expert, see Figure 4.a. However, when the net speed is increased to 5 kbyte/sec. (see Figure 4.d), the differences decrease, and if the user performs more than thirty refinements the ANTARCTICA SRS behaves worse. In conclusion, we can say that the ANTARCTICA approach behaves much better than a Tucows-like system for low network speed. Differences between the two approaches become less significant for a higher network speed. Taking this analysis as basis we could estimate which approach is better for a given situation.

About the intelligence of the ANTARCTICA SRS browser agent, Figure 5 give us interesting results. This figure shows the same scenarios than Figure 4.a and 4.b, but varying the intelligence of the ANTARCTICA SRS browser,

[1] We mean by refinement a “click” in a Tucows-like system and a catalog refinement in the ANTARCTICA SRS.



	5	10	20	30	40	50
TUCOWS	4,28779693	7,891414141	15,09661836	22,16312057	29,55082742	37,28560776
ANTARCTICA	5,062778453	7,208765859	11,49425287	15,69365976	20,08032129	24,72799209

(a)

	5	10	20	30	40	50
TUCOWS	9,331840239	17,1998624	32,93807642	48,30917874	64,35006435	81,30081301
ANTARCTICA	10,10713564	16,51800463	29,29115407	41,8760469	55,00550055	68,87052342

(b)

	5	10	20	30	40	50
TUCOWS	1,782912566	3,242542153	6,191183754	9,082652134	12,10360687	16,35590448
ANTARCTICA	1,868041545	3,06710833	5,462689828	7,813720894	10,26904909	12,85016705

(c)

	5	10	20	30	40	50
TUCOWS	6,811061163	12,5502008	24,01536984	35,23608175	49,75124378	56,88282139
ANTARCTICA	8,304268394	12,37317496	23,24500232	39,40110323	52,24660397	59,31198102

(d)

Fig. 4. Network time for different scenarios: (a) and (b) represent a net speed of 1 Kbyte/sec, (c) and (d) represent a net speed of 5 Kbyte/sec., (a) and (c) represent a "user delay" of 10 sec., (b) and (d) represent a "user delay" of 60 sec. The intelligence of the ANTARCTICA's browser has been set to 70%.

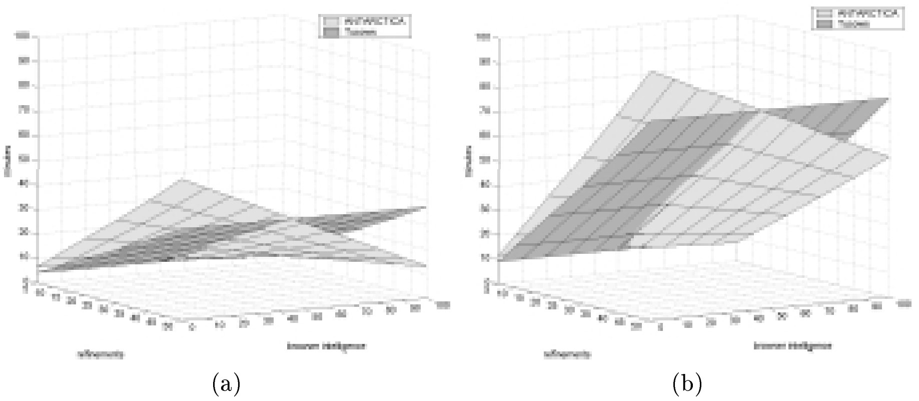


Fig. 5. (a) and (b) represent the same scenarios than Figures 4.a and 4.b, respectively, but varying the intelligence of the ANTARCTICA SRS browser.

from a browser that needs to ask for information the 100% of the times to a browser that needs to ask for information the 0% of the times. When the intelligence of the browser is less than 40 (it does not need to ask for information the 40% of the times) the ANTARCTICA SRS behaves worse than the Tucows-like system. However, when the intelligence of the browser does not need to ask for information the 40% of the times or more, then ANTARCTICA SRS obtains similar or better results than a Tucows-like system.

6 Conclusions and Related work

In this paper we have compared the performance between a classical software retrieval system (the so-called Tucows-like system) and another one proposed using mobile agents (the ANTARCTICA SRS). The comparison has been performed by applying to each of them a software performance evaluation process, which has as a major advantage that it is integrated in the early stages of the software life cycle.
We would like to stress the following points:

- The combination of a UML performance extension and SWNs is expressive enough to model complex distributed software systems even taking into account different technologies. It must be remarked that a performance formal model (SWN) can be obtained semi-automatically from the UML performance annotated diagrams in the context of the software life cycle.
- Different scenarios can be tested easily by using this process without investing time in implementing prototypes.
- As a result of our tests, we can affirm that the ANTARCTICA SRS behaves better than Tucows-like system when the net speed is slow. So, the ANTARCTICA SRS is appropriate for wireless environments.
- As a future work, we plan to give semantics to the UML diagrams in terms of Petri nets.

Concerning related work, in [8] stochastic Petri nets are obtained from UML diagrams with performance evaluation purpose. In our opinion, it is not clearly stated the translation process, and it is not clear how the performance model is obtained. Therefore, it has not been possible to test their technique in our examples to compare their solution with ours. In [17] an approach to performance prediction for the real time field is presented. It uses ROOM [15] as a development method and a layered queuing network as a performance model, a prototype tool has been developed to assist the process. A parallel work to our software performance process using UML and SWNs is being developed in [7]. However, we claim that for the case of parallel and distributed systems, as those presented in this paper, Petri nets improves the adequacy of queue network, since they provide general synchronization mechanisms and validation techniques for logical properties.

Acknowledgements: We would like to thank Laura Recalde for all the helpful collaboration.

References

1. *CNET Inc., 1999. http://www.download.com.*
2. *CNET Inc., 1999. http://www.gamecenter.com.*
3. *Tucows.com inc., 1999. http://www.tucows.com.*
4. G. Booch, I. Jacobson, and J. Rumbaugh, *OMG Unified Modeling Language specification*, June 1999, version 1.3.
5. G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad, *Stochastic well-formed coloured nets for symmetric modelling applications*, IEEE Transactions on Computers **42** (1993), no. 11, 1343–1360.
6. G. Chiola, G. Franceschinis, R. Gaeta, and M. Ribaudo, *GreatSPN 1.7: GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets*, Performance Evaluation **24** (1995), 47–68.
7. V. Cortellesa and R. Mirandola, *Deriving a queueing network based performance model from UML diagrams*, Proceedings of the Second International Workshop on Software and Performance (WOSP2000) (Ottawa, Canada), ACM, September 2000, pp. 58–69.
8. P. King and R. Pooley, *Using UML to derive stochastic Petri nets models*, Proceedings of the Fifteenth Annual UK Performance Engineering Workshop (J. Bradley and N. Davies, eds.), Department of Computer Science, University of Bristol, July 1999, pp. 45–56.
9. E. Mena, A. Illarramendi, and A. Goñi, *A software retrieval service based on knowledge-driven agents*, Cooperative Information Systems CoopIS'2000 (Eliat, Israel), Opher Etzion, Peter Scheuermann editors. Lecture Notes in Computer Science, (LNCS) Vol. 1901, Springer, September 2000, pp. 174–185.
10. J. Merseguer, J. Campos, and E. Mena, *Evaluating performance on mobile agents software design*, Actas de las VIII Jornadas de Concurrencia (Cuenca, Spain) (Diego Cazorla, ed.), Universidad de Castilla-la Mancha, June 2000, pp. 291–307.
11. J Merseguer, J Campos, and E. Mena, *Performance evaluation for the design of agent-based systems: A Petri net approach*, Proceedings of the Workshop on Software Engineering and Petri Nets, within the 21st International Conference on Application and Theory of Petri Nets (Aarhus, Denmark) (Mauro Pezzé and Sol M. Shatz, eds.), University of Aarhus, June 2000, pp. 1–20.
12. J. Merseguer, J. Campos, and E. Mena, *A performance engineering case study: Software retrieval system*, Performance Engineering. State of the Art and Current Trends (R. Dumke, C. Rautenstrauch, A. Schmietendorf, and A. Scholz, eds.), Lecture Notes in Computer Science, Springer-Verlag, Heidelberg, 2001, To appear.
13. T. Murata, *Petri nets: Properties, analysis, and applications*, Proceedings of the IEEE **77** (1989), no. 4, 541–580.
14. E. Pitoura and G. Samaras, *Data management for mobile computing*, Kluwer Academic Publishers, 1998.
15. B. Selic, G. Gulleckson, and P. Ward, *Real-time object-oriented modeling*, Wiley, 1994.
16. C. U. Smith, *Performance engineering of software systems*, The Sei Series in Software Engineering, Addison–Wesley, 1990.
17. M. Woodside, C. Hrischuck, B. Selic, and S. Bayarov, *A wide band approach to integrating performance prediction into a software design environment*, Proceedings of the 1st International Workshop on Software Performance (WOSP'98), 1998.