Analysing Internet Software Retrieval Systems: Modeling and Performance Comparison*

José Merseguer† Javier Campos
Eduardo Mena
Dpto. de Informática e Ingeniería de Sistemas
Centro Politécnico Superior, Universidad de Zaragoza
María de Luna,3, 50015 Zaragoza, Spain
phone: (+34)976761949 - fax: (+34)976761914
{jmerse,jcampos,emena}@posta.unizar.es

December 7, 2001

Abstract

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 that they need. If these kinds of systems are designed using mobile agent technology the previous problem might be avoided. In this paper, we present a comparison between the performance of a software retrieval system especially designed to be used in a wireless network and the performance of a software retrieval system similar to the well-known Tucows.com web site.

In order to compare performance, 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.

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

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

†Contact author

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 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 these kind of systems to exactly know how costly this process is.

A new technology, *mobile agents* [19], can aid to reduce the network connection time. We recall in this paper a software retrieval service [13] belonging to the ANTARCTICA system [11] 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 designed to be used in wireless communication environments where net speed is a problem, about 800 bytes/sec. in a GSM network. 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".

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. The impact of the intelligent agents in the ANTARCTICA SRS will be analyzed too.

The *software performance engineering* [21] 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) [6] 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

the use of formal models to obtain performance indices. However UML lacks of the necessary formal semantics to apply analytical techniques. Thus, we propose to semi-automatically obtain *Petri nets* (PNs) [18] 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 [4], they are suitable for performance evaluation. Concretely, we use stochastic well-formed coloured nets (SWNs) [7].

There exist several works that combine design methodologies and performance modeling. In [12] 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 was not possible to test their technique in our examples to compare their solution with ours. In [22] an approach to performance prediction for the real time field is presented. It uses ROOM [20] 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 queueing networks (QNs) is being developed in [9]. However, we claim that for the case of parallel and distributed systems, as those presented in this paper, PNs improve the adequacy of QNs, since they provide general synchronization mechanisms and validation techniques for logical properties. In the field of stochastic process algebras, the work in [5] presents a language to describe functional and performance aspects of software architectures inheriting the compositional nature of process algebras. The advantages of our approach are twofold: the use of UML as design language, a more suitable tool to describe software architectures and the availability of efficient analysis algorithms and software tools for performance evaluation.

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 ANTARCTICA SRS is also modeled using UML and the models translated into SWNs. In section 5, we present a comparison between performance figures for both approaches. Finally, some concluding remarks are stressed in section 6.

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" [21]. In this section, we briefly present a software performance process, introduced in [15, 16], 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. A complementary approach to obtain the performance model, based on "design patterns"[10], is being developed, it can be found in [14].

The performance of a system is traditionally obtained from its dynamic view. So, we concentrate on developing the UML diagrams corresponding to the dynamic model. Unfortunately, UML lacks of the necessary expressiveness to accurately describe the system load, which is needed to obtain performance figures. To bridge the gap, we use a UML time extension proposed in [15] to successfully deal with performance features at the design stage. Once the UML models have been developed, we have to obtain performance indices from them. As UML lacks of the necessary formal semantics to obtain them, we use PNs with this purpose. Our approach is to provide a formal semantics to UML diagrams in terms of SWNs. Therefore, we propose a translation from the UML time annotated diagrams into SWNs. This translation will be performed using the techniques given in [15]. From SWNs, performance indices may be computed by applying quantitative analysis techniques already developed in the literature. The techniques that we will use are those implemented in the *GreatSPN* tool [8].

2.1 Modeling the Behaviour of a System Using Performance Annotated UML Diagrams

The process begins by modeling the system dynamics in a conventional way. The UML dynamic diagrams [6] will guide the process. *Use case diagrams*, *sequence diagrams* and *statechart diagrams* will be used in this paper. *Activity diagrams* are not used because we are not interested in modeling the concrete actions performed by the systems that we model.

Figure 1 depicts an example of use case and 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.b). Messages sent among objects on different computers, those which travel through the net, will consume time

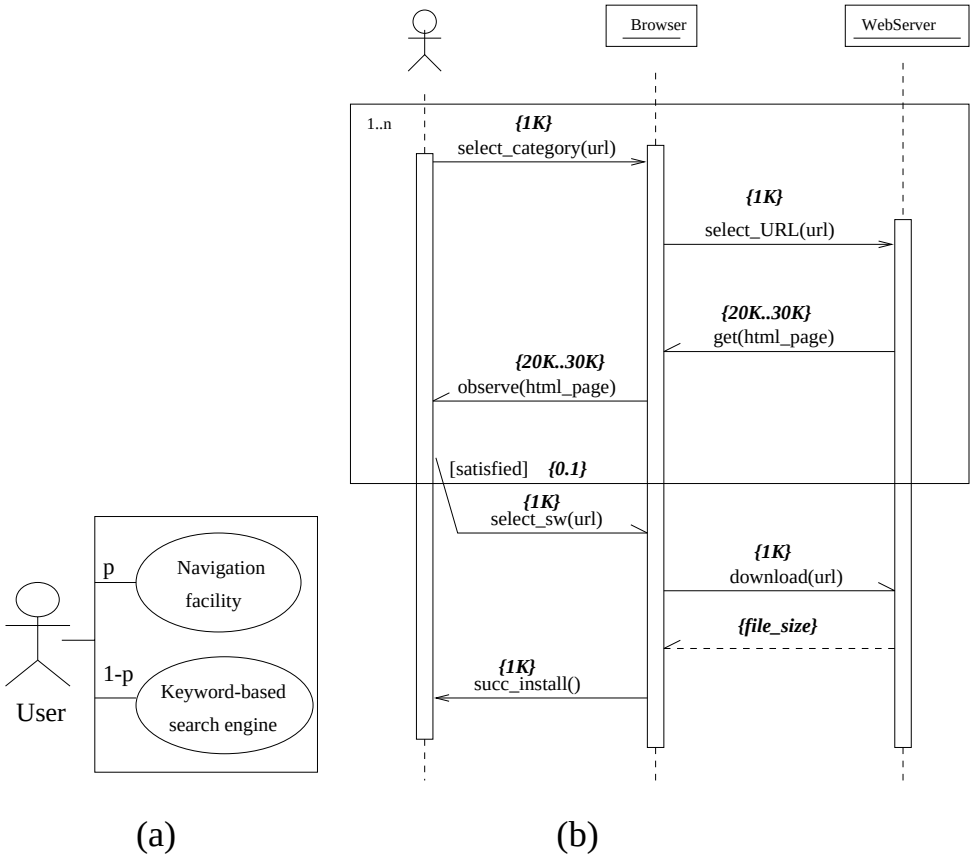


Figure 1: (a) Use cases for the Tucows-like system and (b) annotated sequence diagram for the *navigation facility* use case.

as a function of the message size and the net speed (for instance, in Figure 1.b 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 K} in Figure 1.b. 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.b). 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.b).

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.b). If the probability is unknown, the annotation is a label representing a performance parameter.

In order to get a complete view of the system dynamics, a statechart for each class with relevant dynamic behaviour must be developed, as those depicted in Figure 2. To study perfor-

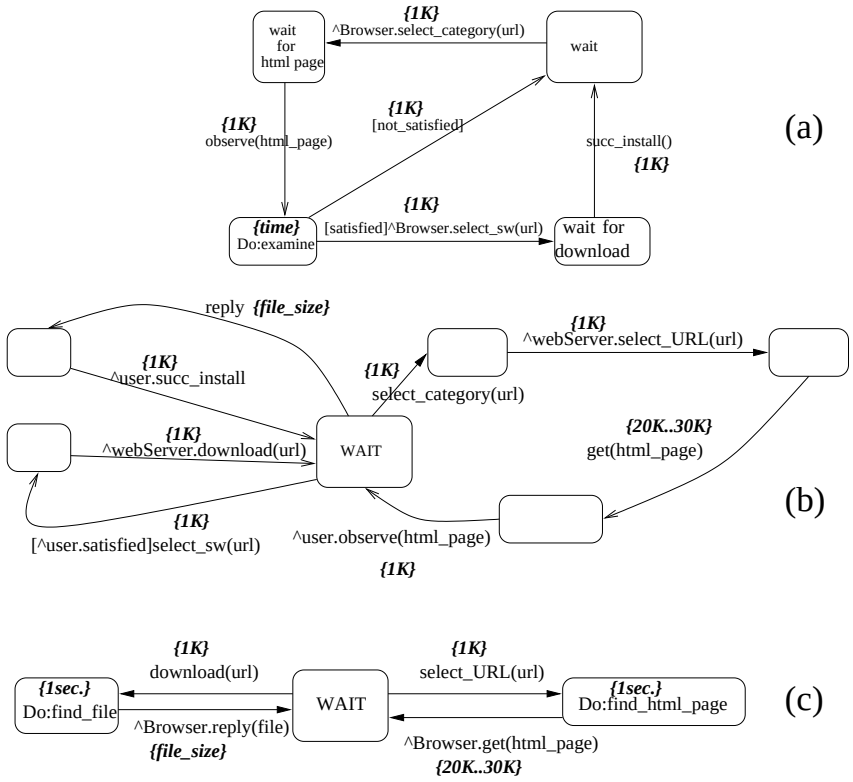


Figure 2: Annotated statecharts for Tucows-like system: (a) user, (b) browser, and (c) the web server.

mance aspects on a statechart, 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 2.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 the statecharts to gain readability. In the same way, the size of the messages may be annotated or omitted.

2.2 Translation of the Performance Annotated UML Diagrams into SWNs

From the performance annotated UML diagrams it is interesting to obtain performance indices for the system. But, as we said before, 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 statechart.* These SWNs will be called *component nets*. They represent the behaviour of each class with the underlying SWN formal semantics. To obtain a component net from a statechart several rules must be applied, details can be found in [15]; the following are the most important:

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

   As example, see in Figure 3 the component nets obtained from the statecharts in Figure 2.

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

3 Modeling Tucows-like Software Retrieval Systems

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 and by navigating through categories especially designed to make this 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 losing reality with respect to performance aspects. We will refer to these kind of systems as *Tucows-like* systems. Our intent is to evaluate performance indices for a Tucows-like system in order to compare them with those obtained for the ANTARCTICA SRS.

The keyword-based search engine helps users that know some features of the wanted software. This search facility will not be considered in this paper since it can not be used by naive users that do not known the concrete software that they need. The navigation facility consists of several web pages residing on 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 to different operating systems, say Windows 2000, Windows 95/98, Linux or Unix. The user selects the desired category and a new web page with several topics like multimedia, browsers or Internet tools is loaded; in this way the user can continue the search of the software. 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 describe and model the navigation facility of the Tucows-like systems. According to the software performance process described in the previous section, we are going to model the Tucows-like system using UML enhanced with performance annotations. Later, this model, represented by a number of diagrams, will be converted into an SWN by applying the rules given in [15] and summarized in section 2. Therefore, we will obtain a formal model

as the input for the analytical techniques in order to obtain the desired performance indices for the system.

3.1 System Description of a Tucows-like System and Modeling Assumptions

In a Tucows-like system, the *user* navigates, with the help of a *browser*, 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.

In short, the process of selecting software by navigating HTML pages is as follows: The user "clicks" on a category, then the browser requests the *web server* for the corresponding HTML page. The web server returns the HTML page to the browser, which presents it to the user. After reading this page, the user can "click" on another link in order to access a new web page with other categories or a list of software under the current category. This process is repeated until the user finds a software that fulfills her/his needs. Then the browser requests the selected software, which is downloaded into the user computer.

It must be considered that the user spends some time *reading* the information presented by the system. An exponentially distributed random variable with rate $\lambda_{examine}$ ($\lambda_{examine}$ 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 by variables with rates $\lambda_{findHTML}$ and $\lambda_{findFile}$.

The browser, on the client machine, sends messages through the net to the web server on the server machine and vice versa. A variable with rate λ_{m_i} models the time spent by the message i navigating through the net. Notice that local messages sent between the user and the browser do not consume net resources.

It could be argued that exponential assumption is not realistic for the modeling of network delays, and that heavy tailed distributions would be better. However, a performance model must many times lose in accuracy of the representation of reality in order to be able to be an-

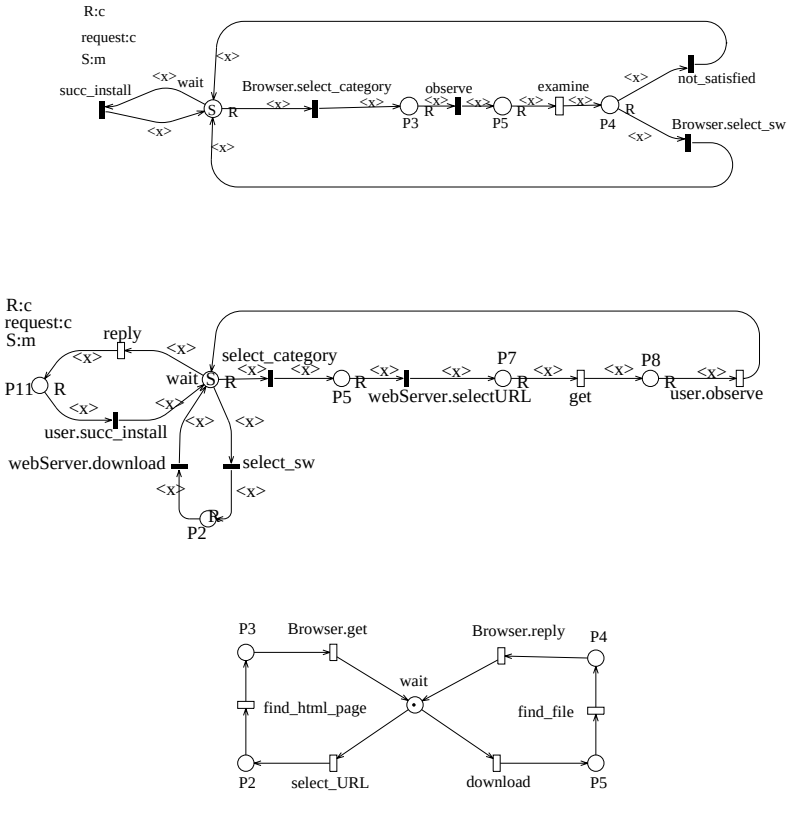


Figure 3: Component PNs for the Tucows-like system: (a) user, (b) browser, (c) web server.

alyzed. Anyhow, the possibility of representing delays with non-exponential distributions could be considered in the future if simulation techniques are used instead of the analytic approach followed here.

3.2 UML Diagrams with Performance Annotations for a Tucows-like System

In this section we model the dynamic view of the Tucows-like system using UML notation as proposed in [15]. The *use case* diagram (see Figure 1.a) shows the two possible scenarios for the system: the *navigation facility* and the *keyword-based search engine*. It has been developed following the notation given in [9] where p means the probability that the user executes the scenario. We assume that p=1 because we are not interested in the keyword-based search engine, in this way all user executions correspond to the navigation facility.

The *sequence diagram* in Figure 1.b shows a detailed description of the "navigation facility" scenario. It 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 sent through the net, taking into account the assumptions made in the previous section.

The sequence diagram 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 an 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 statechart for each class with relevant dynamic behaviour.

User statechart diagram. In Figure 2.a, 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 a 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 web address 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 statechart diagram. Figure 2.b shows the browser's statechart. 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 it, 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 a 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).

Web server statechart 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 2.c shows the web server's statechart diagram.

3.3 Modeling the Tucows-like System with SWNs

PNs are a suitable formalism for the modeling of concurrent phenomena. There are situations that cannot be expressed with UML diagrams but they can be perfectly described with PNs. For example, with a PN we can exactly model how many concurrent requests to download software the system might serve; UML diagrams cannot express that.

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 [15] 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 3 represents the *component nets*, those obtained from the statecharts, for the Tucows-like system and Figure 4 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 cannot 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 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 retrieval 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 the web server has launched 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

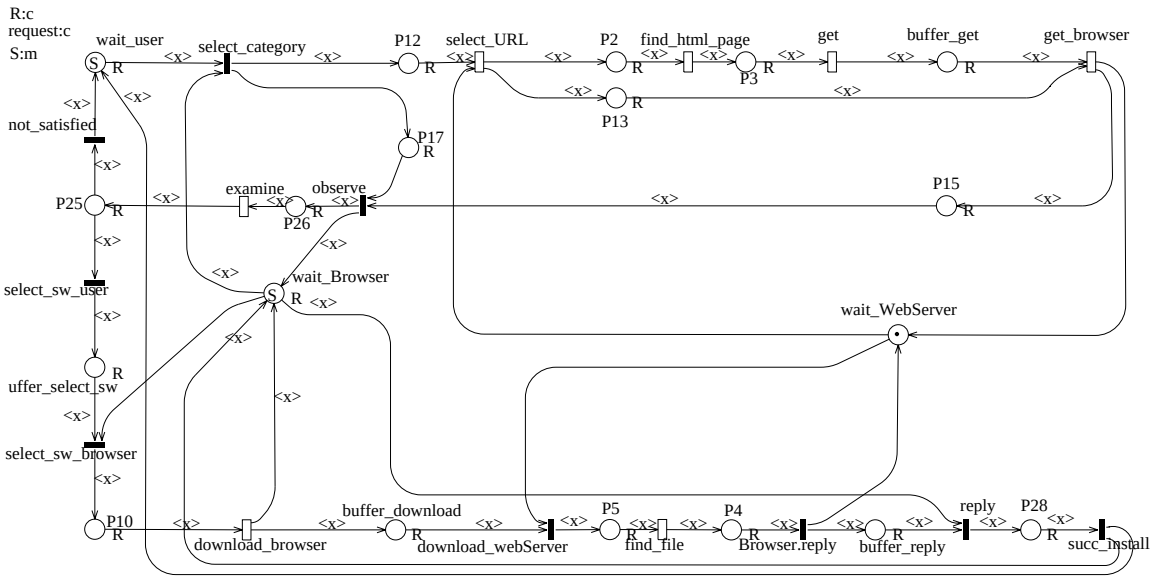


Figure 4: Complete SWN for the Tucows-like system.

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 4, is obtained by applying the rules given in [15]. These rules basically state that if two transitions in different nets represent the same message (the sender and the receiver), they must be connected using an intermediate buffer place (no wait semantics) or they must be synchronized in a unique transition (wait semantics). If the message travels through the *net*, then the transitions are timed and two situations can arise:

- If they are connected in the complete net by using an intermediate buffer place then only one of them remains as a timed transition, and the other will be converted into an immediate transition. As an example, see the timed transition named download in the web server component net (Figure 3.c) and the timed transition named webServer.download in the browser component net (Figure 3.b). In the complete net (Figure 4) the timed transition named download_browser models the time spent by the message navigating through the net and the immediate transition named download_webServer models the reception of the message.

- If the synchronization in the complete net is modeled using only one transition, it remains

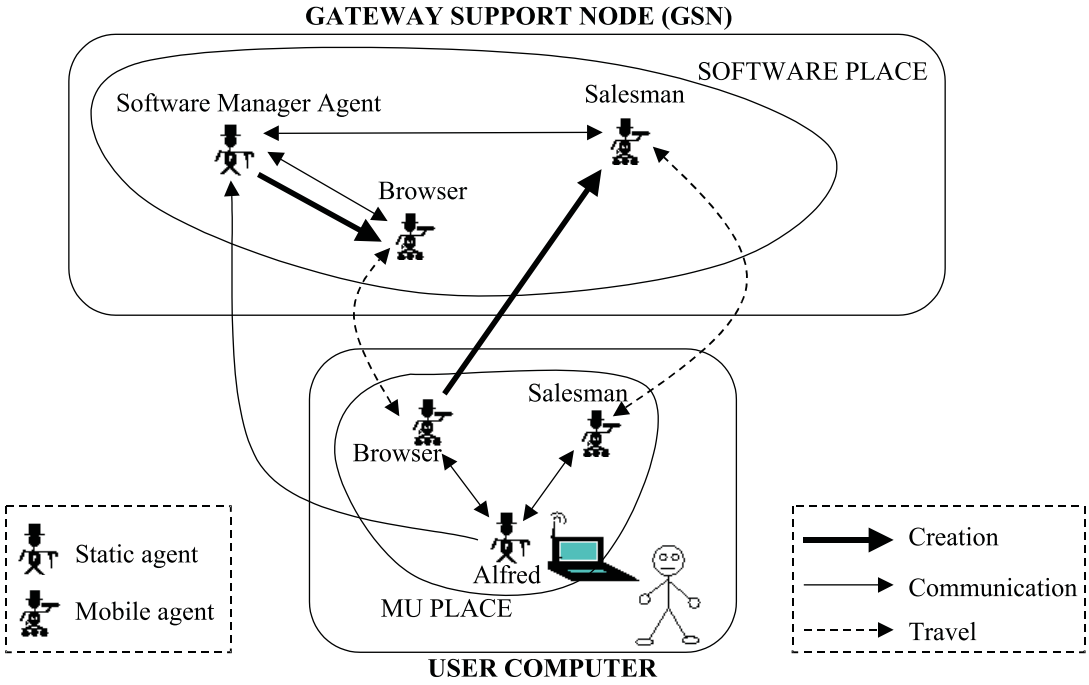


Figure 5: Architecture for the ANTARCTICA SRS.

as a timed transition. As example, see the timed transition named select_URL in the web server component net (Figure 3.c) and the timed transition named webServer.select_URL in the browser component net (Figure 3.b); in the complete net (Figure 4) the timed transition named select_URL models the time spent by the message navigating through the net and the reception of the message.

4 The ANTARCTICA Software Retrieval Service

In this section we present the ANTARCTICA SRS [13]. The goal of the system 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 ANTARCTICA SRS is situated in a concrete server called the GSN[1]. Agents are executed in contexts denominated *places* [17]. Mobile agents can travel from one place to another. The service incorporates two places: one place on the user computer called the *Mobile User place*, and other situated on the GSN, called the *Software place* (see Figure 5).

Some of the advantages of the use of mobile agents, related to accessing remote information, are the following:

- They encapsulate communication protocols.
- They do not need synchronous remote communications to work.
- They can act in an autonomous way and carry knowledge to perform local interactions at the server system instead of performing several remote procedure calls.
- They can make use of remote facilities and perform specific activities at different locations.

4.1 System Description of the ANTARCTICA SRS and Modeling Assumptions

The procedure that the ANTARCTICA SRS supports for the software retrieval process is the following: the user sends requests for software to an agent (*Alfred*). The request is sent to the GSN and an agent (the *browser*) is created. The user receives the visit of the browser, which helps the user to select the most appropriate software by browsing a catalog customized to that concrete user. The user can request more detailed information until s/he finally selects a piece of software. Then a new agent arrives to the user computer (the *salesman*) with the selected piece of software.

In the following such agents are described, grouped in two categories:

1. *The user agent.* Alfred is an efficient majordomo that serves the user and is in charge of storing as much information about the user computer, and the user her/himself, as possible.
2. *Information exploitation.* The software manager agent creates and provides the browser agent with a catalog of the available software, according to the needs expressed by Alfred (on behalf of the user), i.e., it is capable to obtain customized metadata about the underlying software. For this task, the software manager consults an ontology. The software

[1]The *Gateway Support Node* is the proxy that provides services to computer users.

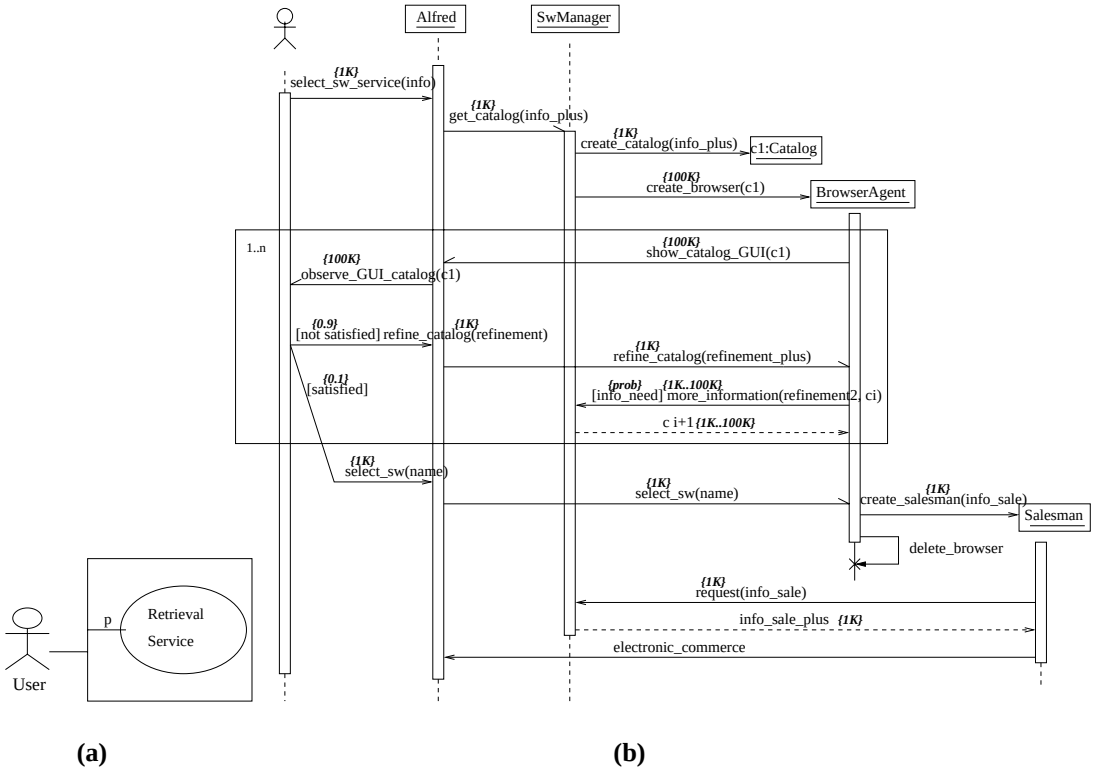


Figure 6: (a) Use case for the ANTARCTICA SRS and (b) annotated sequence diagram for the *retrieval service* use case.

itself can be either stored locally on the GSN or accessible through the web in external data sources. Thus, the GSN can have access to a great number of distinct software for different systems, with different availability, purpose, etc. The goal of the browser agent is to interact with the user in order to refine a catalog of software until the user finally chooses a concrete piece of software. When this is done, the salesman agent carries the program selected by the user to her/his computer, performs any electronic commerce interaction needed (which depends on the concrete piece of software), and installs the program, whenever possible.

The assumptions given in section 3.1 for the Tucows-like system remain valid for the ANTARCTICA SRS with the following interpretations: Now, the user spends some time *reading* the catalog presented by Alfred; the probability that the user finds the software by selecting n catalogs models different kind of users; the software manager consumes time consulting an ontology to create the catalog or to find the piece of software; some of the messages sent among the browser, the software manager, the salesman and Alfred travel through the net.

Moreover the following assumptions must be taken into account for the ANTARCTICA SRS: As the browser is an intelligent agent, sometimes it does not ask for information to the software

manager, but if it does it, then it will performed using *remote procedure call* (RPC) or traveling through the net; the size of the catalog could be parameterized.

In the following the software performance process is applied to the ANTARCTICA SRS, as we did for the Tucows-like system in section 3. Therefore, the UML diagrams, the SWNs components and the SWN net for the system will be obtained. The SWN net for the system (see Figure 9) will be used in the next section to obtain performance results. It is 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).

4.2 UML Diagrams with Performance Annotations for the ANTARCTICA SRS

As in section 3.2, we model the system using the performance extended UML notation. Figure 6.a depicts the *use case* diagram needed to describe the dynamic view of the system. It shows that the user is the unique actor that interacts with the system. The probability p=1 reflects that only one scenario is possible for the system.

The *sequence diagram* in Figure 6.b describes in detail the "retrieval service" scenario. It begins with the request select_sw_service(info) performed by the user to the majordomo and the message sent by it to the software manager in order to compose the first catalog, get_catalog(info_plus). Immediately, the catalog and the browser are created and the system advances according with the process previously defined.

For each agent a statechart is modeled, they are briefly commented in the following:

Alfred statechart diagram. Alfred is always present in the system, no creation event is relevant for our purposes. Its behaviour is typical for a server object. It waits for an event requesting a service (select_sw_service, show_catalog_GUI, refine_catalog or select_sw). For each request it performs the necessary activities and it returns to its wait state to serve another request. Figure 7.a shows Alfred's statechart. The stereotyped transition $\ll more\_services \gg$ means that Alfred may attend other services that are not of interest here.

User statechart diagram. In Figure 7.b, the behaviour of a user is represented. The user is in the wait state until s/he activates a select_sw_service event. This event sets the user in the waiting_for_catalog state. The observe_GUI_catalog event, that could be sent by Alfred,

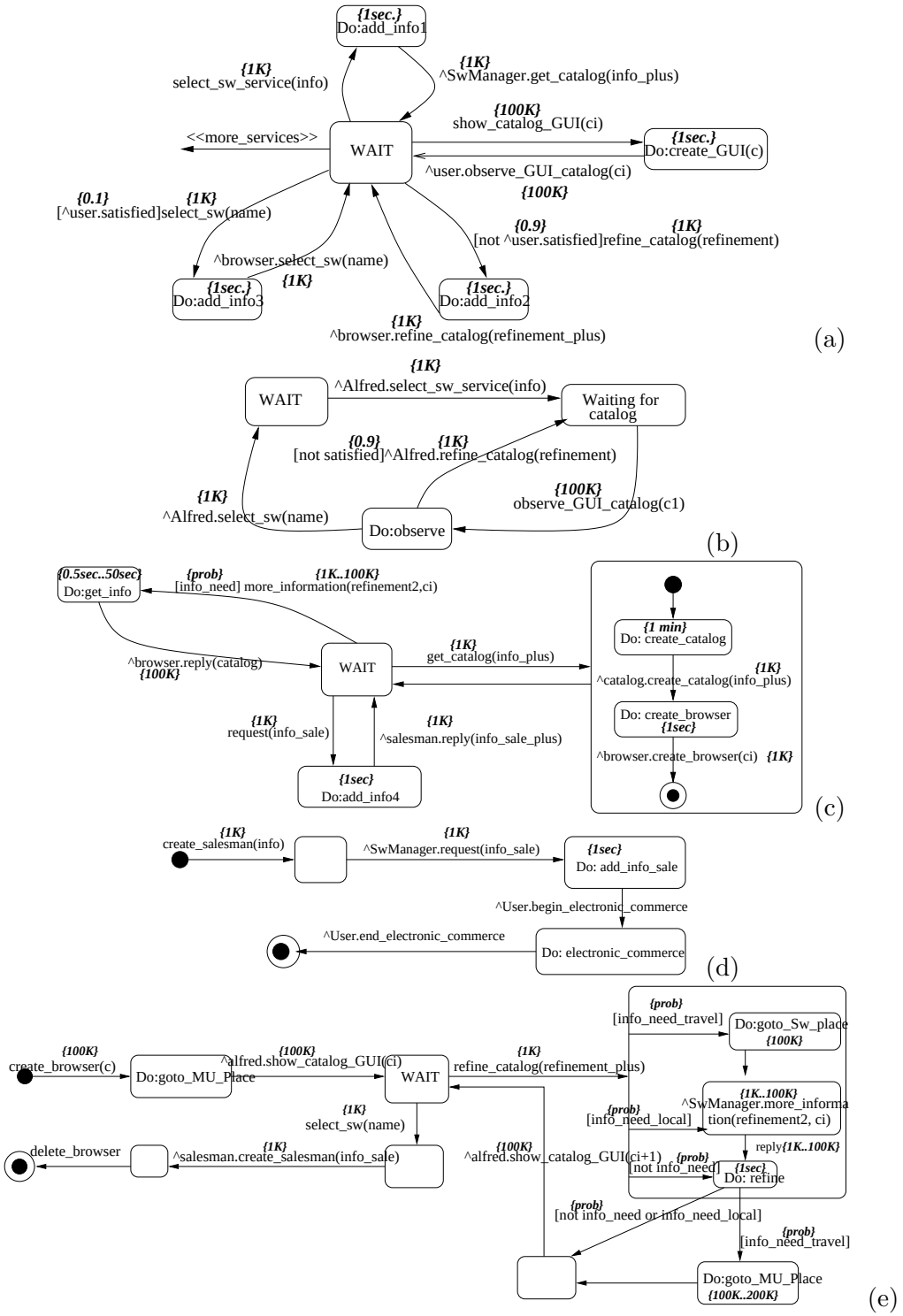


Figure 7: Annotated statecharts for the ANTARCTICA SRS: (a) Alfred, (b) user, (c) software manager, (d) salesman and (e) browser.

allows the user to examine the catalog in order to look for the desired software. If it is in the catalog, the user sends the select_sw event to Alfred, in other case s/he sends the refine_catalog event.

Software manager statechart diagram. Like Alfred, the software manager behaves as a server object. It is waiting for a request event (more_information, get_catalog, request). When one of them arrives, it performs the necessary activities to accomplish it. Figure 7.c shows its statechart diagram. It is interesting to note the actions performed to respond the get_catalog request: first, the catalog with the available software is created, after that, the browser is created.

Salesman statechart diagram. The salesman's goal is to give e-commerce services, as we can see in Figure 7.d. After its creation it asks the software manager for sale information. With this information the e-commerce can start. This is a complex task that must be described with its own use case and sequence diagram which is out of the scope of this work.

Browser statechart diagram. The statechart diagram in Figure 7.e describes the browser's life. It is as follows: once the browser is created it must go to the MU place, where it invokes Alfred's shows_catalog_GUI method to visualize the previously obtained catalog. At this state it can attend two different events, refine_catalog or select_sw. If the first event occurs there are two different possibilities: first, if the browser has the necessary knowledge to solve the task, a refinement action is directly performed; second, if it currently has not this knowledge, the browser must obtain information from the software manager, by sending a more_information request or by travelling to the software place. If the select_sw event occurs, the browser must create a salesman instance and die.

4.3 Modeling the ANTARCTICA SRS with SWNs

As performed in section 3.3 for the Tucows-like system, in this section the nets obtained by applying the process for the ANTARCTICA SRS are described.

Figure 8 represents the *component nets* for the ANTARCTICA SRS and Figure 9 represents the net for the whole system, obtained by synchronizing the component nets. In the following, we highlight the most remarkable characteristics of them and their relations with the modeling assumptions. The net for the salesman is not commented due to its simplicity.

Alfred component net. It is important to note that all the transitions add_infoX model the

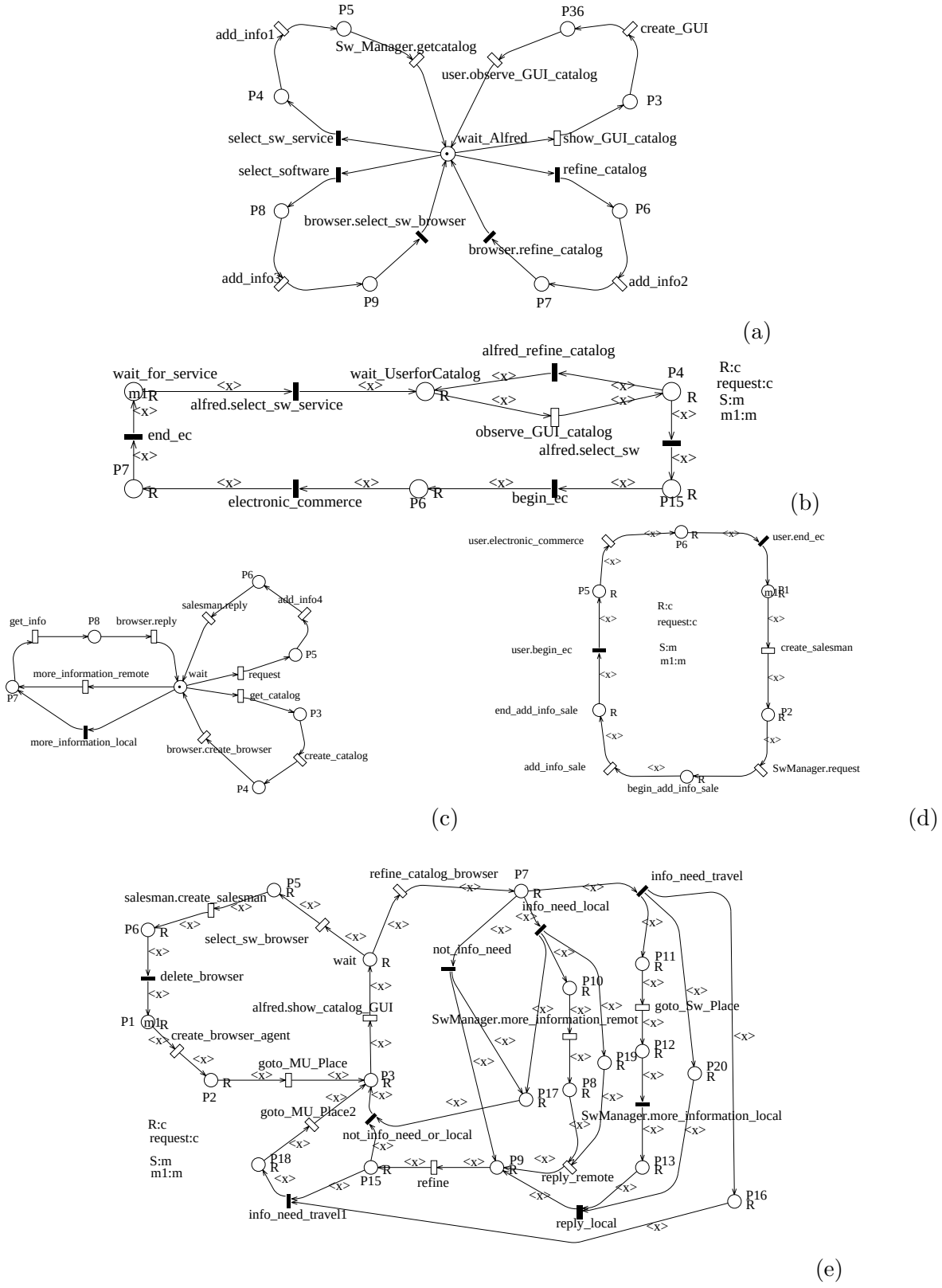


Figure 8: Component PNs for the ANTARCTICA SRS: (a) Alfred, (b) user, (c) software manager, (d) salesman and (e) browser.

activities performed by Alfred to manage the information.

User component net. The number of tokens in the place wait_for_service models how many concurrent users supports the system. This parameter cannot be modeled in the UML diagrams. The assumption, time reading the catalog, is modeled in the transition observe_GUI_catalog. The choice between the transitions alfred_refine_catalog and alfred_select_sw models the assumption about the probability that the user finds the software by selecting n catalogs.

The firing of the immediate transition named end_ec models the arrival of a message "end electronic commerce" to confirm that the retrieval of the software has been successfully completed.

Software manager component net. The assumption that expresses that the software manager consumes time consulting an ontology to create the catalog or to find the piece of software is modeled by the transition get_info. The number of tokens in the place wait models how many concurrent browsers the software manager can attend. This parameter could not be modeled in the UML diagrams.

Browser component net. Despite the difficult readability of this net, it must be pointed out that among others it reflects the important assumption that states that sometimes the browser does not ask for information to the software manager, but if it does it, then it will performed using RPC or traveling through the net. Transitions not_info_need, info_need_local, info_need_travel, not_info_need_or_local, goto_MU_Place and goto_MU_Place2, are involved in its modeling.

5 Performance Results

The results presented in this section have been obtained from the *complete* SWNs which model the Tucows-like system and the ANTARCTICA SRS (Figures 4 and 9 respectively). 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 intelligent the browser agent in the ANTARCTICA SRS must be in order to obtain better results than the Tucows-like system.

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* [8]. The inverse of the previous result

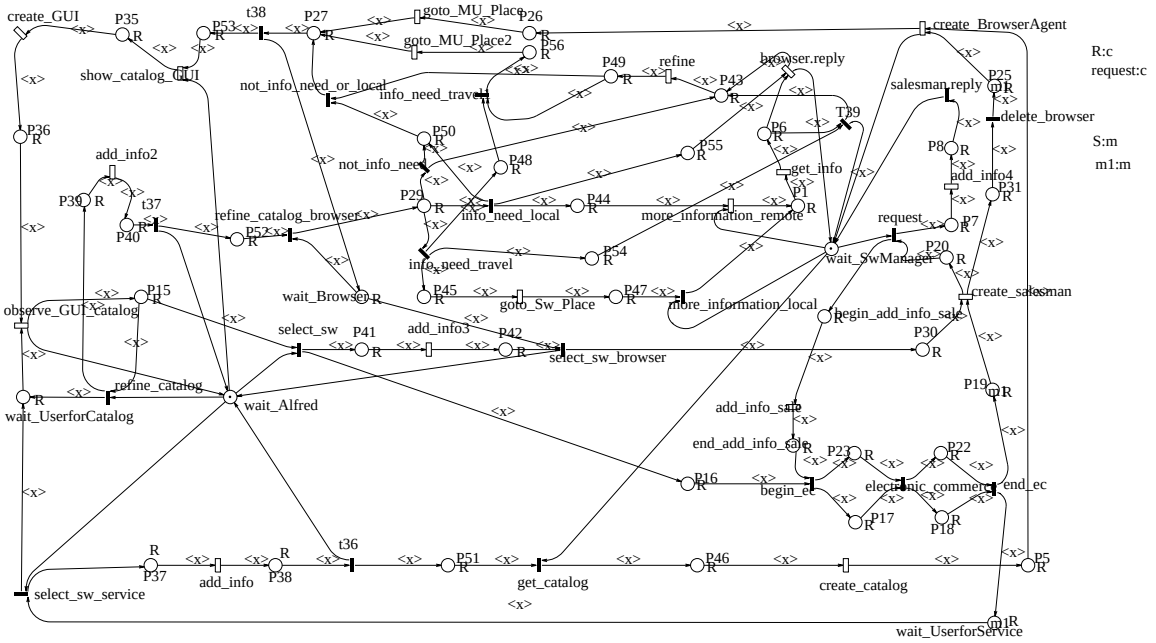


Figure 9: Complete SWN for the ANTARCTICA SRS.

gives the network time. In the ANTARCTICA SRS 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[2] (modeling different expertise of the user).

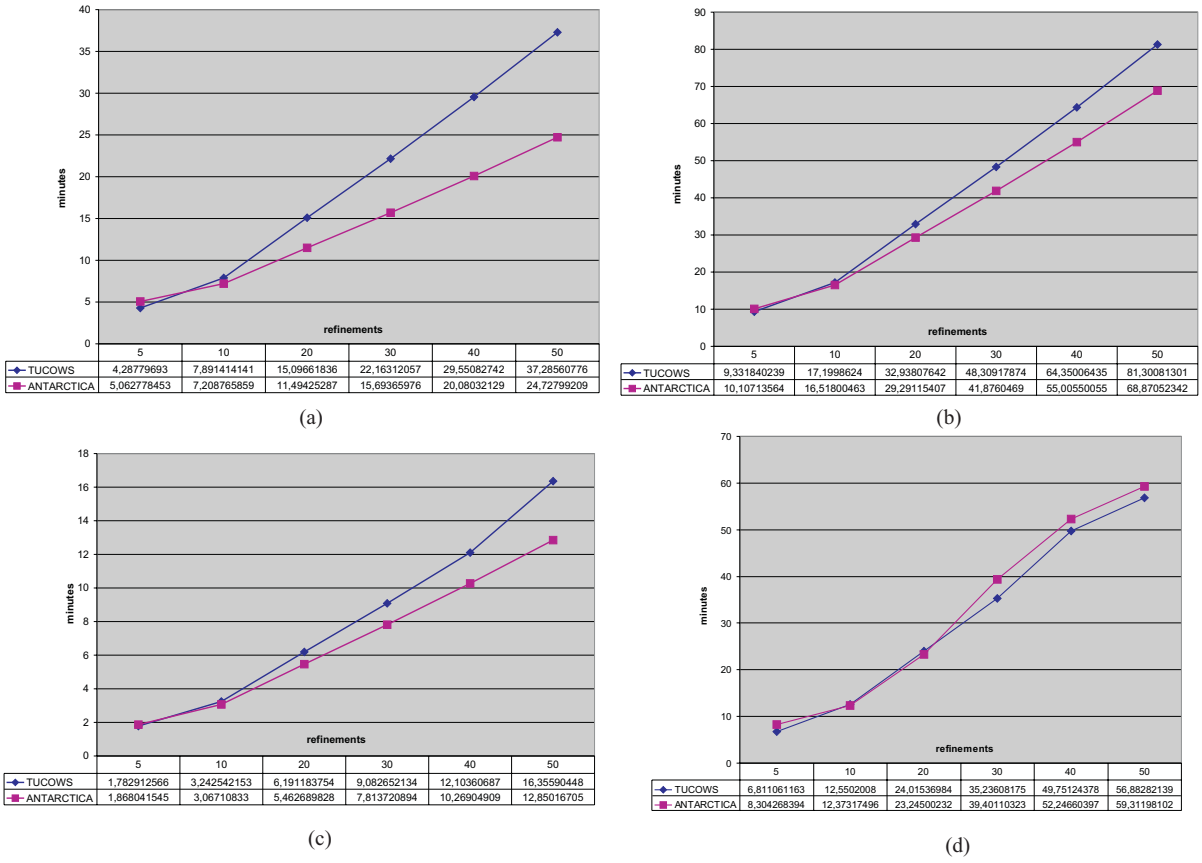
2. Two different kinds of users have been considered: a user who spends 10 sec. to study the information presented by the system (web page or a software catalog) and a user who spends 60 sec. in that task (modeling the information processing speed of the user).

3. We have considered two cases for the *net speed*: 1 K/sec. and 5 K/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 agent the

[2]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

	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

	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

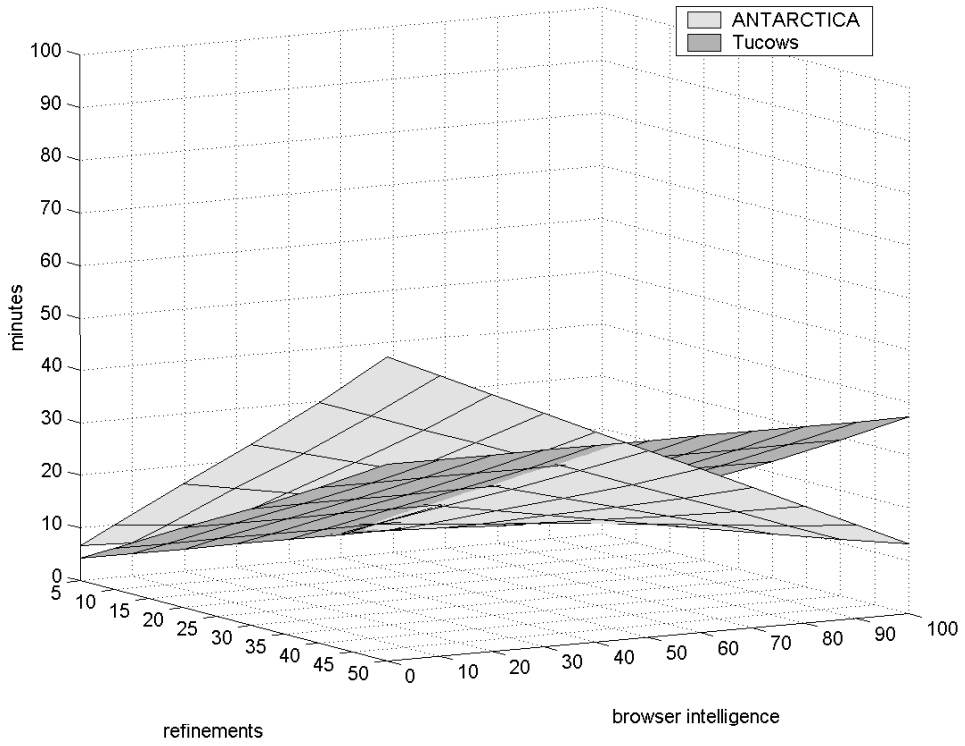
	5	10	20	30	40	50
TUCOWS	6,811061163	12,5502008	24,01536984	35,23608175	49,75124378	56,88282139
ANTARCTICA	8,304268394	12,37317496	23,24500232	39,40110323	52,24660397	59,31198102

Figure 10: Network time for different scenarios: (a) and (b) represent a net speed of 1 K/sec., (c) and (d) represent a net speed of 5 K/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%.

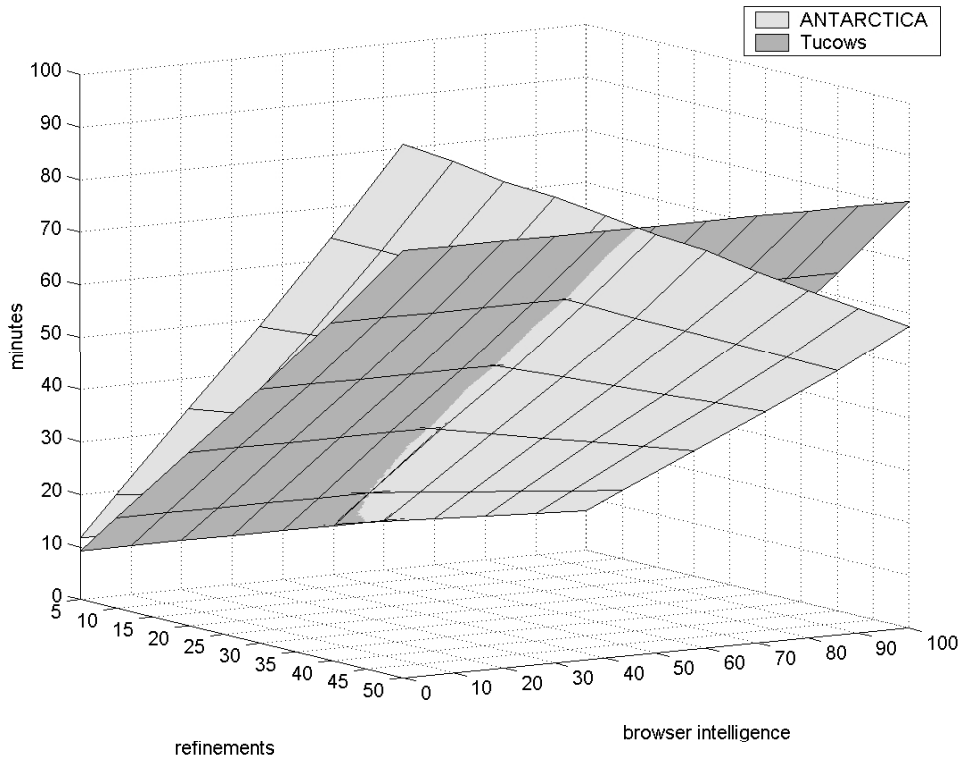
70% of the times that the user asks for a refinement. When the browser needs information, it requests the information by RPC.

2. *The size of the catalog* obtained by the browser is 50 K.

Figure 10 shows network time (in minutes) for the Tucows-like system and the ANTARCTICA SRS in different scenarios. Concretely in Figure 10.b we can observe that when the net speed is 1 K/sec., the user is naive and performs 50 refinements (the worst case), then the ANTARCTICA SRS is almost thirteen minutes faster than the Tucows-like system. The same results are obtained if the user is expert, see Figure 10.a. However, when the net speed is increased to 5 kbyte/sec. (see Figure 10.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 net-



(a)



(b)

Figure 11: (a) and (b) represent the same scenarios than Figures 10.a and 10.b, respectively, but varying the intelligence of the ANTARCTICA SRS browser.

work speed. Taking this analysis as basis we could estimate which approach is better for a given situation.

Concerning the intelligence of the ANTARCTICA SRS browser agent, Figure 11 give us interesting results. This figure shows the same scenarios than Figure 10.a and 10.b, 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 needs to ask for information the 60% of the times) the ANTARCTICA SRS behaves worse than the Tucows-like system. However, when the intelligence of the browser is greater than 40% then the ANTARCTICA SRS obtains similar or better results than a Tucows-like system.

6 Conclusions

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 or when the net speed is faster but the intelligence of the system is greater. So, the ANTARCTICA SRS is appropriate for wireless environments.

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] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, *Modelling with generalized stochastic petri nets*, John Wiley Series in Parallel Computing - Chichester, 1995.

[5] M. Bernardo, P. Ciancarini, and L. Donatiello, *AEMPA: A process algebraic description language for the performance analysis of software architectures*, Proceedings of the Second International Workshop on Software and Performance (WOSP2000) (Ottawa, Canada), ACM, September 2000, pp. 1–11.

[6] G. Booch, I. Jacobson, and J. Rumbaugh, *OMG Unified Modeling Language specification*, June 1999, version 1.3.

[7] 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.

[8] 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.

[9] 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. 58–70.

[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of reusable object-oriented software*, Addison-Wesley, 1995.

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

[12] 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.

[13] 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.

[14] J. Merseguer, J. Campos, and E. Mena, *A pattern-based approach to model software performance*, Proceedings of the Second International Workshop on Software and Performance (WOSP2000) (Ottawa, Canada), ACM, September 2000, pp. 137–142.

[15] 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.

[16] 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, (LNCS) Vol. 2047, Springer-Verlag, Heidelberg, 2001, pp. 317–332.

[17] 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*, Proceedings of Mobile Agents '98, September 1998.

[18] T. Murata, *Petri nets: Properties, analysis, and applications*, Proceedings of the IEEE **77** (1989), no. 4, 541–580.

[19] E. Pitoura and G. Samaras, *Data management for mobile computing*, Kluwer Academic Publishers, 1998.

[20] B. Selic, G. Guleckson, and P.T. Ward, *Real-time object-oriented modeling*, John Wiley & Sons, 1994.

[21] C. U. Smith, *Performance engineering of software systems*, The Sei Series in Software Engineering, Addison–Wesley, 1990.

[22] 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.