Performance Evaluation for the Design of Agent-based Systems: A Petri Net Approach*

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. Software design and implementation using mobile agents are nowadays involved in a scepticism halo. There are researchers who question its utility because it could be a new technology that does provide new skills but it could introduce new problems. Security and performance are the most critical aspects for this new kind of software. In this paper we present a formal approach to analyse performance for this class of systems. Our approach is integrated in the early stages of the software development process. In this way, it is possible to predict the behaviour without the necessity to carry out the complete implementation phase. To show the approach, we model a software retrieval service system in a pragmatic way, later, the corresponding formal model is obtained and analysed in order to study performance.

Keywords: Software performance, Petri nets, UML, mobile agent

1 Introduction

In the last years, distributed software applications have increased their possibilities making use of Internet capabilities, positioning distributed software development as a very interesting approach. The client/server model has become the key paradigm to support distributed software development. It is widely recognised that there are four main technologies which advocate for client/server developments: relational database management systems (RDBMS), TP monitors, groupware and distributed objects. It is well accepted that distributed objects in conjunction with *mobile agents* [15, 11] technology are a very interesting approach to address certain kind of software domains like e-commerce, information retrieval and network management and administration.

Although there are researchers who question mobile software, it takes sense in distributed environments [9] because it is a technology with appropriate new skills for these kind of systems. But it could introduce new problems as the inappropriate use of the net resources. In this way time consuming could become a problem for users. So, we are concerned to develop new techniques and methods which minimize these problems. In this context, *software performance* [18] appears as a discipline inside software engineering to deal with model performance on software systems design. Like many people concerned about software

* This work has been developed within the project TAP98-0679 of the Spanish CICYT.

performance, we believe that the performance evaluation must be accomplished during the early stages of the software development process.

Unified Modeling Language (UML) [2] is widely accepted as a standard notation to model software systems. Unfortunately, UML lacks of the necessary expressiveness to accurately describe performance skills. There have been several approaches to solve this lack [19, 20, 16]. One of the goals of this paper is the study of the performance indices in mobile agent systems, thus, we propose a *UML with performance annotations* (pa-UML) to deal with performance skills on these kind of systems. Our approach to solve the problem is as follows: we model the problem domain using pa-UML, describing static and dynamic views when necessary. pa-UML models will give us the necessary background to obtain the corresponding formal model expressed as *Petri nets* [13]. From pa-UML, we derive a time interpretation of Petri nets leading to Generalized Stochastic Petri Nets (GSPN) [1]. Thus, we implicitly give a semantics for pa-UML in terms of Petri nets. Performance indices may be computed for GSPN by applying quantitative analysis techniques already developed in the literature.

The rest of the paper is organised as follows. In section 2, we describe a system, based on agents, which has been taken from [12]. In section 3, we give our proposal to annotate system performance aspects in UML (pa-UML) and we develop the pa-UML models for the system presented in section 2. Section 4 is dedicated to transform pa-UML diagrams into Petri nets in order to achieve the desired formal model. Finally, some performance results and conclusions are presented.

2 An example: the Software Retrieval Service in the ANTARCTICA system

In this section we briefly present ANTARCTICA[1]. The system has been taken from [12] and it will be used as an example along this paper to study performance on mobile agent systems.

The goal of the system is to provide mobile computer users with different services that enhance the capabilities of their computer. One of these services is the Software Retrieval Service, that allows users to select and download new software in an easy and efficient way. This service has been thought to work in a wireless network media and 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.

In the following, we briefly describe the system paying attention in its components. There is a "majordomo" named *Alfred*, which is an agent specialised in

[1] Autonomous ageNT bAsed aRChitecture for cusTomized mobIle Computing Assistance.

user interaction. There is a *Software Manager* agent whose task is to create a catalog which will help the user to select the required software. Another agent, the *Browser* will help the user in selecting the software. Finally, a *Salesman* agent is in charge of performing any action previous to the installation of the selected software, like e-commerce.

The system was proposed in [12] using different technologies, namely CORBA [14], HTTP and mobile agents. Some performance tests were applied to different implementations, in order to select the best way of accessing remote software. Conclusions were the following:

- Time corresponding to CORBA and mobile implementations are almost identical for a wide range of files to be downloaded.
- Mobile agent approaches are fast enough to compete with client/server approach.

Although considering the importance and the relevance of the results of the work [12], we would like to stress the enormous cost of implementing different prototypes in order to evaluate the performance of the different alternatives. In the rest of this paper, we model the system in a pragmatic way using pa-UML, annotating consistently the system load (we have annotated the system load taking as a basis the experiments and experience of the authors of the cited paper). After that, we can interpret the pa-UML model in terms of Petri nets and derive the corresponding performance model which will be properly analysed. This analysis is used to evaluate the system.

3 Modelling the system using pa-UML

In the previous section, we have explained the general features of the target system. Now, we focus on modelling it using pa-UML notation. We have considered UML and not the notation of methodologies such as OMT [7], OOSE [10] or Fusion [6] because of its wider acceptance in the software engineering community.

The system description in UML accomplishes with static and dynamic views in order to give a complete description of the system. For the sake of simplicity and for the convenience of our problem, we only describe the dynamic view of the system.

Figure 1 shows the use cases needed to describe the dynamic behaviour of the system. We deal with three different use cases, "show services", "software retrieval service" and "e-commerce". Also, we can see the unique actor which interacts with the system, the "user". The use cases are described in the following.

Show services use case description.

- Principal event flow: the use case goal is to show to the user the available services that the system offers. The Software Retrieval Service is one of those services and it is also described as a use case.

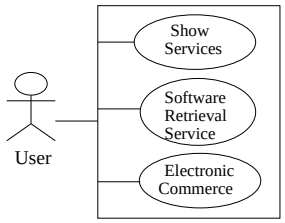


Fig. 1. Use Cases

Software retrieval service use case description.

- Principal event flow: the user requests the system for the desired software. The Browser gets a catalog and the majordomo, Alfred, shows it to the user, who selects the software s/he needs.
- Exceptional event flow: if the user is not satisfied with the catalog presented, s/he can ask for a refinement. This process could be repeated as many times as necessary until the user selects a concrete piece of software.

Electronic commerce use case description.

- Principal event flow: the goal is to provide the user an e-commerce activity and the download of the software selected.

Show services and e-commerce use cases are out of the scope of this article, thus, we concentrate on the Software Retrieval Service.

Pragmatic object-oriented methodologies such as [6, 7, 10] do not deal with performance skills. So, we can say that there is not an accepted *method* to model and study system performance in the object-oriented software development process. This lack implies that there is not a well-defined language or *notation* to annotate system load, system delays and routing rates. On the contrary, formal specification languages, such as LOTOS [17], or Petri nets [13], have considered and studied the problem in depth. Thus, there are several proposals where we can learn from.

As we remarked, it is our objective to propose a UML extension (pa-UML) to deal with performance on the software development process at the design stage. We consider that our proposal must accomplish with both, the method and the notation. First, the method will give us the process to model the system and the relevant parameters to be taken into account. We advocate for a pattern-oriented approximation. Lately, design patterns [8] have gained relevance in software development due to their simplicity and flexibility. But this will be subject of future research. Second, concerning the notation, it will be treated in this work.

In order to have a complete performance notation, the UML behavioural and structural models must be considered. Also, performance will play a prominent role in the implementation diagrams. In this paper, we are interested only in behavioural aspects, concretely in the sequence diagram and the state transition diagrams. Future works will deal with the rest of the UML diagrams to describe behaviour (use case diagrams, activity diagrams, collaboration diagrams), structural aspects, and implementation diagrams.

The UML notation to deal with time is based on the use of time restrictions. This restrictions are expressed as time functions on message names, e.g., {(messageOne.receiveTime - messageOne.sendTime) $<$ 1 sec.}. We consider more realistic to annotate the message size. In this way, we could calculate performance for different net speeds.

3.1 Sequence diagrams

In order to understand the problem, it is interesting a more detailed description of the Software Retrieval Service use case. Thus, a *sequence diagram* [2] has been developed to treat accurately the mentioned use case, see figure 2.

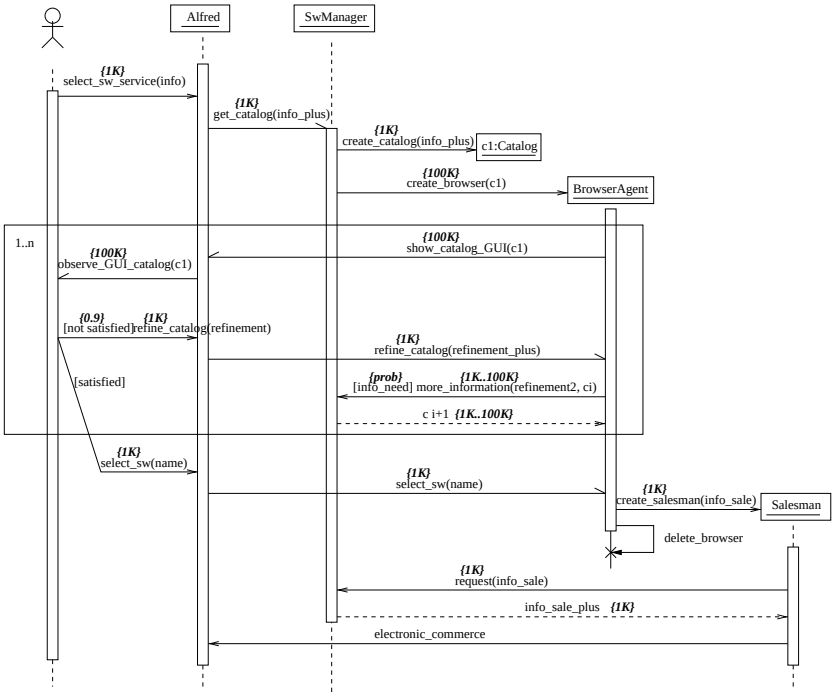


Fig. 2. Sequence diagram for the Software Retrieval Service use case

A sequence diagram represents messages sent among objects. Usually, a message is considered as no time consuming in the scope of the modelled system. But in a mobile agent system, we distinguish between messages sent by objects on the same computer and messages sent among objects on different computers, those which travel through the net. The first kind of messages will be considered as no-time consuming. The second kind will consume time as a function of the message size and the net performance (speed). Here an annotation, inside braces, will be made indicating the message size. For instance, in Figure

2, select_sw_service message is labelled with {1 Kbyte}, while show_catalog_GUI requires the movement of {100 Kbytes}. Also, it will be possible to annotate a range for the size in the UML common way, like in more_information message, where a {1K..100K} label appears.

In a sequence diagram, conditions represent the possibility that the message that they have associated with could be sent. An annotation, also inside braces, expressing the event probability success will be associated to each condition. A range is accepted too. See, for instance, the probability {0.9} associated in Figure 2 to the condition not_satisfied. Sometimes, it is possible that the probability is unknown when modelling. Also, it could be that the probability a message occurs is a parameter subject to study. In our example, the condition info_need associated to the more_information message is critical for the system, because it reveals how much intelligent the Browser is; so, we want to study it. In such situations, we will annotate an identifier, corresponding to the unknown probability.

3.2 State Transition diagrams

Sequence diagrams show how objects interact, but to take a complete view of the system dynamics, it is also interesting to understand the life of objects. In UML, the *state transition* diagram is the tool that describes this aspect of the system. For each class with relevant dynamic behaviour a state transition diagram must be specified.

In a state transition diagram two elements will be considered, the *activities* and the *guards*. Activities represent tasks performed by an object in a given state. Such activities consume computation time that must be measured and annotated. The annotation will be inside braces showing the time needed to perform it. If it is necessary, a minimum and a maximum values could be annotated. See, for example, bold labels between braces in Figures 4, 5, 6 and 7. Guards show conditions in a transition that must hold in order to fire the corresponding event. A probability must be associated to them. It will be annotated in the same way as guards were annotated in the sequence diagram, and the same considerations must be taken into account. See, for instance, label {0.9} joined to condition [not ^user.satisfied] in Figure 4.

Message size may be omitted since this information appears in the sequence diagram. In the example, we have duplicated this information to gain readability.

We now present the state transition diagrams for our system using the paUML notation.

User state transition diagram. In Figure 3, 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, sent by Alfred, allows the user to examine the catalog to look for the desired software, if it is in the catalog, the users selects the select_sw event, in other case s/he selects the refine_catalog event.

Alfred state transition diagram. The example supposes that Alfred is always present in the system, no creation event is relevant for our purposes. So the state transition diagram begins when a view_services event is sent

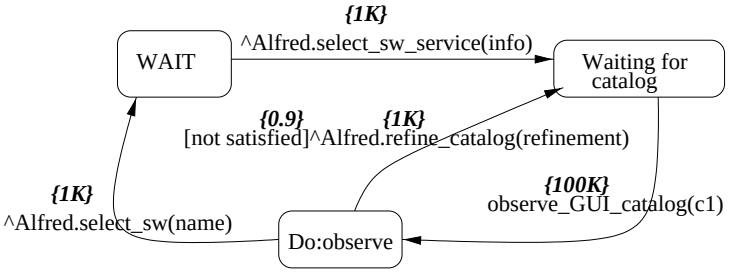


Fig. 3. State transition diagram for the user

to the user. Alfred's behaviour is typical for a server object behaviour. It waits for an event requesting a service (select_sw_service, show_catalog_GUI, refine_catalog or select_sw). For each of these requests it performs a concrete action, and when it is completed, a message is sent to the corresponding object in order to complete the task. After the message is sent, Alfred returns to its wait state to serve another request. Figure 4 shows Alfred's behaviour. The stereotyped transition $\ll more\_services \gg$ means that Alfred may attend other services that are not of interest here.

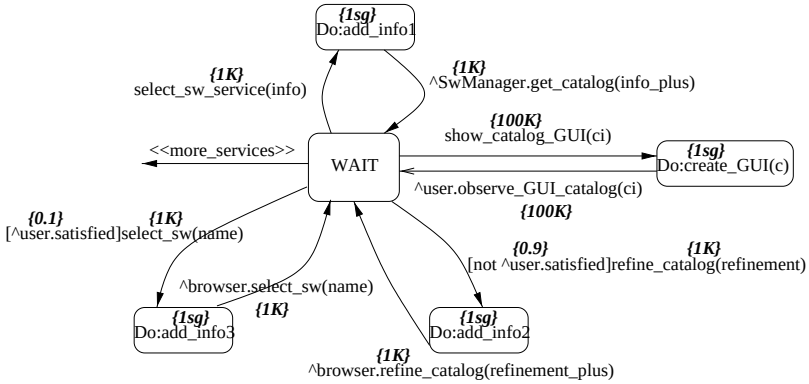


Fig. 4. State transition diagram for Alfred

Software Manager state transition diagram. Like Alfred, the Software Manager behaves as an server object. It is waiting for a request event (more_information, get_catalog, request) to enable the actions to accomplish the task. Figure 5 shows its state transition diagram; it is interesting to note the actions performed to respond the get_catalog request. First, an ontology is consulted and, after that, two different objects are created, those involved in task management.

Browser state transition diagram. The state transition diagram in Figure 6 describes the Browser's life. It is as follows: once the Browser is created it

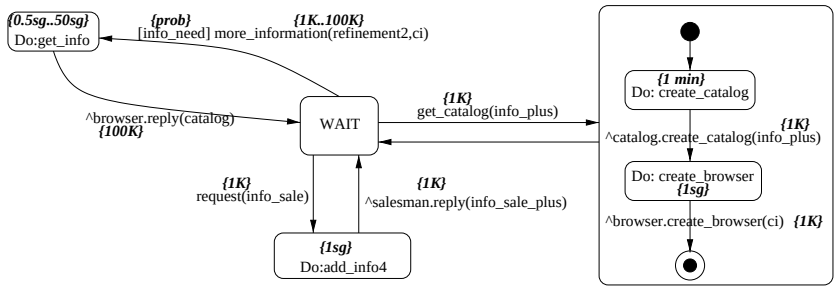


Fig. 5. State transition diagram for the Software Manager

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 background, 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.

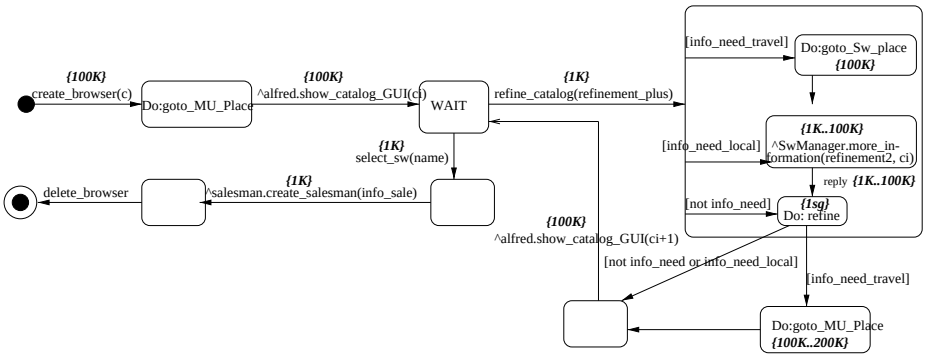


Fig. 6. State transition diagram for the Browser

Salesman State Transition Diagram. The Salesman's goal is to give e-commerce services, as we can see in Figure 7. 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 paper.

The pa-UML models that we have developed are expressive enough to accomplish with different implementations. A necessary condition to design methods

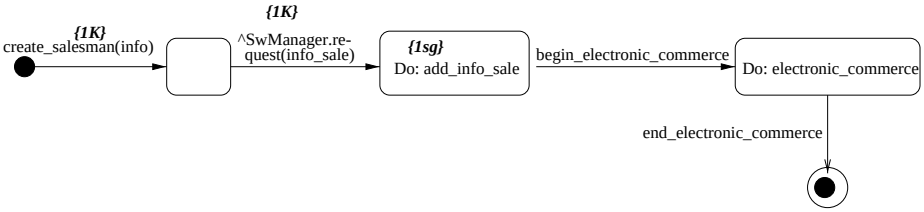


Fig. 7. State transition diagram for the Salesman

is their independence of final implementation decisions. In that way, we can use these models to develop applications based on CORBA, mobile agents, etc. But this gap between design and implementation could be undesirable in certain cases. For example, in the system that we are treating we are not sure how many majordomos should attend requests, how many concurrent users can use the system, etc. However, a formal modelling with Petri nets solves these questions satisfactorily.

The design proposed in [12] deals with one user and one majordomo. Petri nets allow to represent cases such as:

1. One user and one majordomo (the proposed system).
2. Several users served by one majordomo.
3. Many users served by many majordomos, once per request.

Thus, increasing the modelling effort, it could be possible to avoid the necessity of implementing the system for predicting performance figures.

4 Modelling with Petri nets

At this point, we have modelled the system with pa-UML notation, taking into account the load in the sequence diagram and the state transition diagrams. So, a pragmatic approach of the system has been obtained. But this representation is not precise enough to express our needs. Remember that we want to predict the system behaviour in different ways. First, we want to study how the system works with only one user served by one majordomo. On the other hand, it is also of our interest to know the system behaviour when several users are served by only one majordomo, or by several majordomos.

In order to obtain answers to our questions, we need to apply performance analytic techniques to the developed pa-UML diagrams. But there is a lack in this field because no performance model exist for UML, so the pragmatic model is not expressive enough. Also, we need to express system concurrency, but UML models concurrency in a very poor way. Thus, it is required a formal model of the system with concurrency capabilities.

To solve these lacks, we have chosen Petri nets as formal model, because it has the remarked capabilities and also there are well-known analytic techniques to study system performance in stochastic Petri net models. Thus, we propose some *transformation rules* to obtain Petri nets from pa-UML diagrams.

In the following, we model with Petri nets the first two proposed systems, the third one will be developed in a future work. For the first system, one user and one majordomo, GSPN have the expressive power to accomplish the task. To study the second system, several users served by one majordomo, stochastic well-formed coloured Petri nets [3] are of interest. Once the systems are modelled, we use analytic techniques implemented in GreatSPN [4] tool to obtain the target performance requirements.

4.1 Petri net model for a system with one majordomo and one user

First, we are going to obtain a Petri net for each system class, the *component nets*. Obviously, every annotated state transition diagram will give us the guide, and the following general transformation rules will be applied:

Rule 1. *Two different kinds of transitions can be identified in a state transition diagram. Transitions which do not spend net resources and transitions which do. The first kind will be translated into "immediate" transitions (that fire in zero time) in the Petri net. The second kind will be "timed" transitions in the Petri net. The mean of the exponentially distributed random variable for the transition firing time will be calculated as a constant function of the message size and net speed. More elaborated proposals like those given in [5] could be taken into account, but we have considered more important to gain simplicity.*

Rule 2. *Actions inside a state of the state transition diagram are considered as time consuming, so in the Petri net model they will be consider as timed transitions. The time will be calculated from the CPU and disk operations needed to perform the action.*

Rule 3. *Guards in the state transition diagram will become immediate transitions with the associated corresponding probabilities for the resolution of conflicts.*

Rule 4. *States in the state transition diagram will be places in the Petri net. But there will be not the unique places in the net, because additional places will be needed as an input to conflicting immediate transitions (obtained by applying Rule 3).*

Figures 8, 9, 10, 11 and 12 represent the nets needed to model our *system components* taking into account the previous transformation rules. According to GSPN notation [1], immediate transitions (firing in zero time) are drawn as bars (filled), while timed transitions are depicted as boxes (unfilled). Timed transitions are annotated with firing rates, while immediate transitions are annotated with probabilities for conflict resolution.

The sequence diagram will be the guide to obtain a *complete* Petri net for the system using the previous component nets. We must consider that UML distinguishes, in a concurrent system, two different kind of messages in a sequence diagram:

- those represented by a full arrowhead (*wait semantics*), and

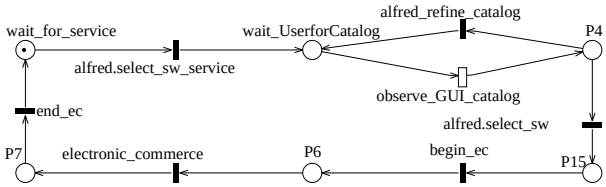


Fig. 8. User Petri net component

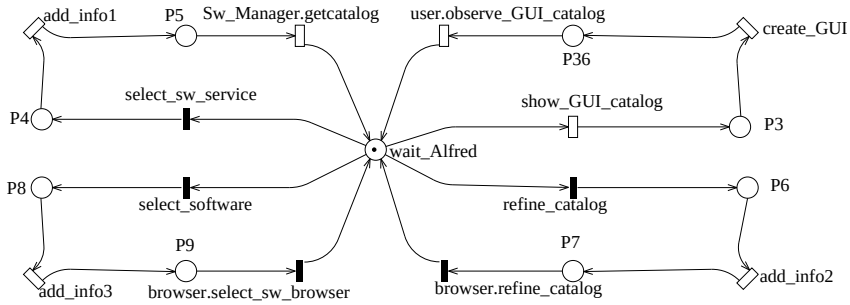


Fig. 9. Alfred Petri net component

– those represented by a half arrowhead (*no-wait semantics*).

The following transformation rules will be used to obtain the net system. But first, it must be taken into account that, for every message in the sequence diagram, there are two transitions with the same name in two different component nets, the net representing the sender and the net representing the receiver.

Rule 5. *If the message has wait semantics, only one transition will appear in the complete net system; this transition will support the incoming and outcoming arcs from both net components.*

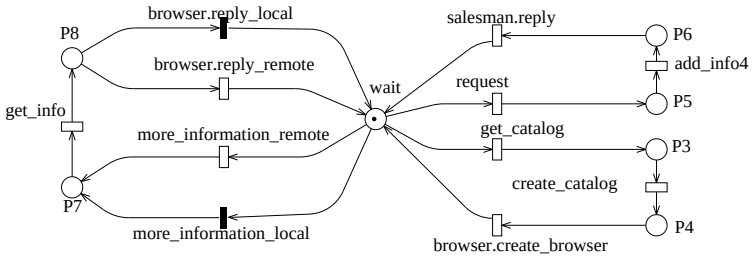


Fig. 10. Software Manager Petri net component

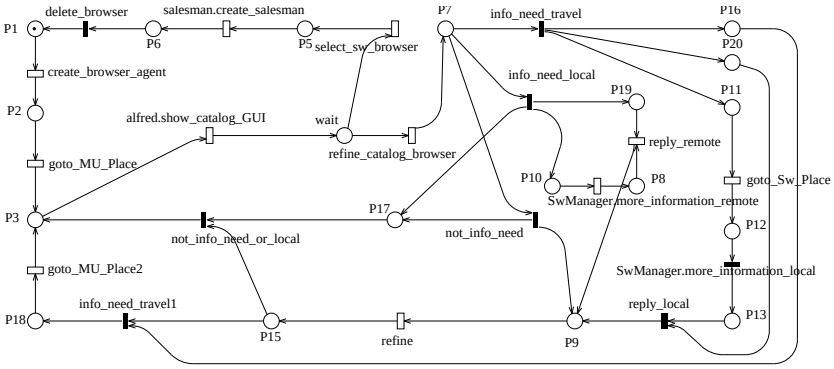


Fig. 11. Browser Petri net component

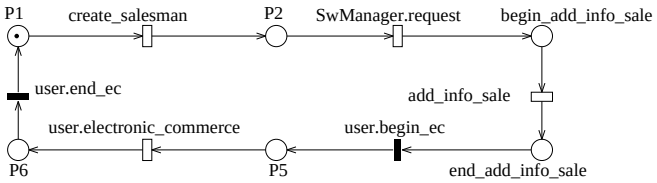


Fig. 12. Salesman Petri net component

Rule 6. *If the message has no-wait semantics, the two transitions will appear in the net system and also an extra place will be added modelling the communication buffer. This place will receive an arc from the sender transition and will add an arc to the receiver transition.*

The net system for the example is shown in Figure 13. In order to understand how to apply the previous rules, we are going to explain how to obtain the observe_GUI_catalog transition in the net system (Figure 13) from the observe_GUI_catalog message sent by Alfred to the user in the sequence diagram. We can observe in Alfred's net (Figure 9) and in the user's net (Figure 8) the presence of that transition. So, in the net system the transition appears with the union of the incoming and outcoming arcs of the components, synchronising in this way both objects.

Finally, we remark with an example that the concurrency expressed in UML has been achieved in the net system by synchronising component nets. When create_salesman transition fires one token is placed in P20 and one token is placed in P31, allowing a concurrent execution of the request and delete_browser transitions.

4.2 Petri net model for a system with one majordomo and several users

In order to model with Petri nets the situation of several users being served by one majordomo, we need to include several tokens in some places like, for

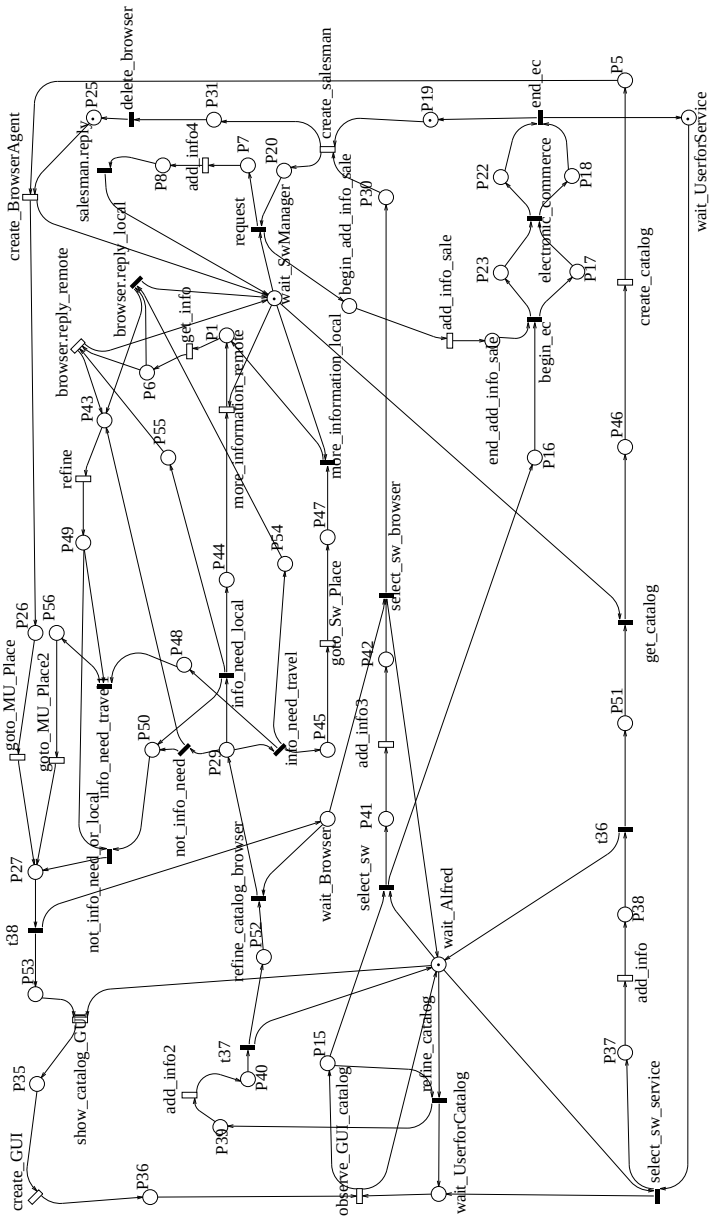


Fig. 13. The Petri net for the whole system

instance, wait_UserforService. Since the system must distinguish between different tokens (they represent different requests), we add a colour domain for the requests, thus leading to stochastic well-formed coloured Petri nets [3].

Our objective now is to reach the stochastic well-formed coloured nets for the components and for the system. Let us begin with the component nets. As in the previous system, component nets will be obtained from the annotated state transition diagrams. We begin the translation task (from pragmatic model to formal model) using the rules stated in the previous section. The Petri nets for Alfred and the Software Manager will be the same because only one instance of each is present in the system. On the contrary, the system will have as many instances of users, browsers and salesmen as required, suppose five for the example.

Now, pay attention on Figure 14, which represents the well-formed coloured Petri net for the user. The R colour means that the system deals with one to five requests and the initial marking m1 in place wait_for_service denotes that all class instances will be used. Moreover, all the places in the net have colour R and the arcs are labeled with the identity function (<x>), in this way only one request could be fired once a time.

Figure 15 shows the well-formed coloured Petri net for the Browser. It has been obtained applying the transformation rules to the Browser's STD. Initial marking m1 in place P1 shows that a maximum of five browsers could be created, one for each users request. Salesman well-formed coloured Petri net (see Figure 16) has been designed in the same way.

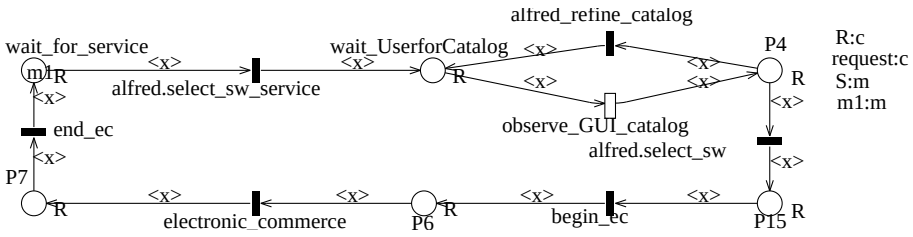


Fig. 14. User coloured Petri net component

Now, we are going to focus on the complete well-formed coloured net for the system, see Figure 17. The transformation rules given in the previous section will give us the guide to construct it. In addition, the following transformation rules will be applied concerning the colours:

Rule 7. *All colours and markings defined in the component nets will be inherited by the net system.*

Rule 8. *The places with colour and/or markings in the components nets will appear in the net system in the same way.*

Rule 9. *The arcs labelled in the component nets will appear in the net system in the same way.*

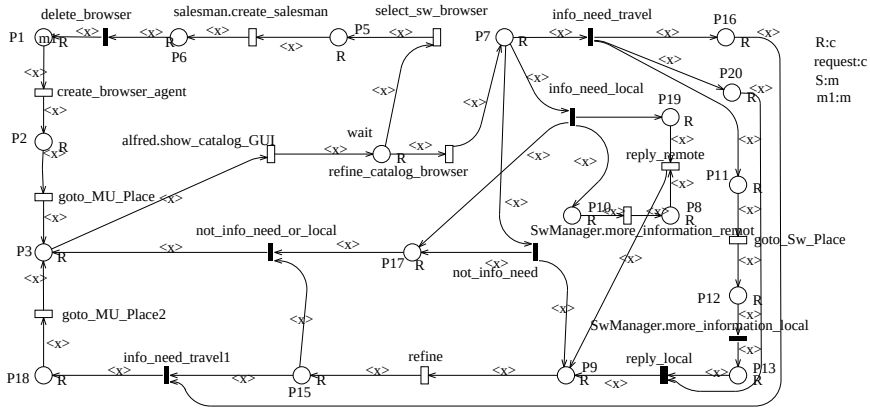


Fig. 15. Browser coloured Petri net component

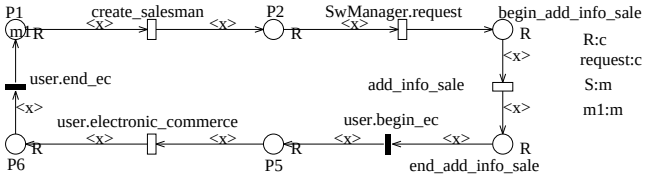


Fig. 16. Salesman coloured Petri net component

Rule 10. *Conflicting arcs are those that appear labelled in a component net but not in the component net which it is synchronised. When conflicting arcs appear, the net system must have the two arcs labelled, preserving in this way the richest semantic.*

As an example of Rule 9 see outcoming arcs for the synchronised transitions select_sw and alfred.select_sw in Figures 9 and 14 respectively.

We remark that the complete well-formed coloured net for the system describes concurrency at the same level as the complete net for the system given in the previous section. Moreover, it introduces a new level of concurrency. The use of coloured tokens models concurrent user requests of a complete service, as it can be seen in the select_sw_service transition, that can fire several tokens from place wait_UserforService representing several user requests.

5 Performance results

The results presented in this section have been obtained from the complete nets that model the examples; the complete net that models the case in which the system is used by one user, who is attended by only one majordomo and the complete net that models the case in which the system is used by several users, which are attended by only one majordomo.

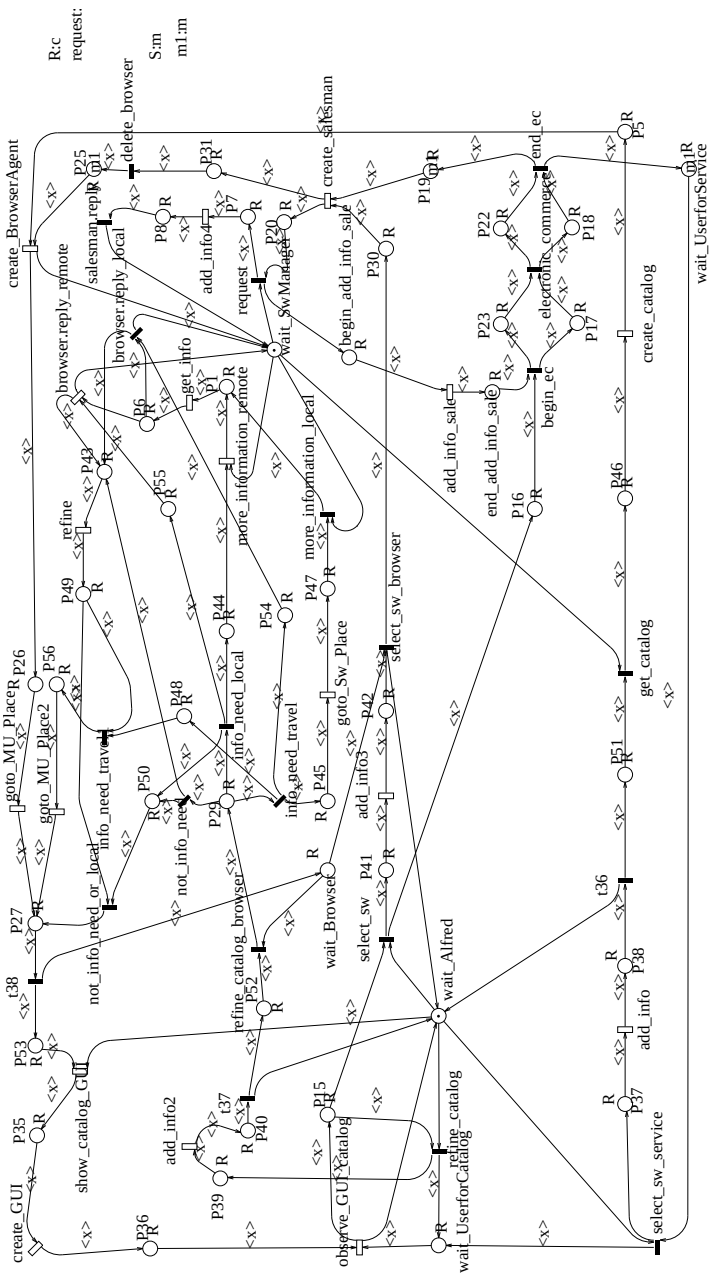


Fig. 17. The coloured Petri net for the whole system

It is of our interest to study the system *response time* in the presence of a user request. To obtain the response time, first the throughput of the select_sw_service transition, in the net system, will be calculated by computing the steady state distribution of the isomorphic *Continuous Time Markov Chain* (CTMC) with *GreatSPN* [4]; finally, the inverse of the previous result gives the system response time. *We want to know which are the bottlenecks of the system and identify their importance.* There are two possible parts which can decrease system performance. First, the trips of the Browser from the "user place" to the "software place" (and way back) in order to obtain new catalogs. Second, the user requests for catalog refinements, because s/he is not satisfied with it.

In order to study the two possible bottlenecks, we have developed a test taking into account the following possibilities:

1. When *the Browser needs a new catalog* (under request of the user) there are several possibilities:
   - The Browser has enough information to accomplish the task or it needs to ask for the information. It is measured by the not_info_need transition. We have considered an "intelligent Browser" which does not need information the 70% of the times that the user asks for a refinement.
   - When the Browser needs information to perform the task, it may request it by a *remote procedure call* (RPC) (represented in the net system by the info_need_local transition) or it may travel through the net to the Software_place (represented in the net system by the info_need_travel transition) to get the information and then travel back to the MU_Place. In this case, we have considered two scenarios. First, a probability equal to 0.3 to perform a RPC, so a probability equal to 0.7 to travel through the net. Second, the opposite situation, a probability equal to 0.7 to perform a RPC, therefore a probability equal to 0.3 to travel through the net.
2. To test the *user refinement request*, we have considered two different possibilities. An "expert user" requesting a mean of 10 refinements, and a "naive user" requesting a mean of 50 refinements.
3. *The size of the catalog* obtained by the Browser can also decrease the system performance. We have used five different sizes for the catalog: 1 Kbyte, 25 Kbytes, 50 Kbytes, 75 Kbytes and 100 Kbytes.
4. *The speed of the net* is very important to identify bottlenecks. We have considered two cases: a net with a speed of 100 Kbytes/sec. ("fast" connection speed) and a net with a speed of 10 Kbytes/sec. ("slow" connection speed).

Figure 18(a) shows system response time (in minutes), for the net in Figure 13, supposing "fast" connection speed, "expert user" and an "intelligent" Browser. One of the lines represents a probability equal to 0.7 to travel and 0.3 to perform a RPC, the other line represents the opposite situation. We can observe that there are small differences between the RPC and travel strategies. Such a difference is due to the round trip of the agent. As the agent size does not change, this difference is not relevant for the global system performance. Thus, we show that the use of mobile agents for this task does not decrease the performance.

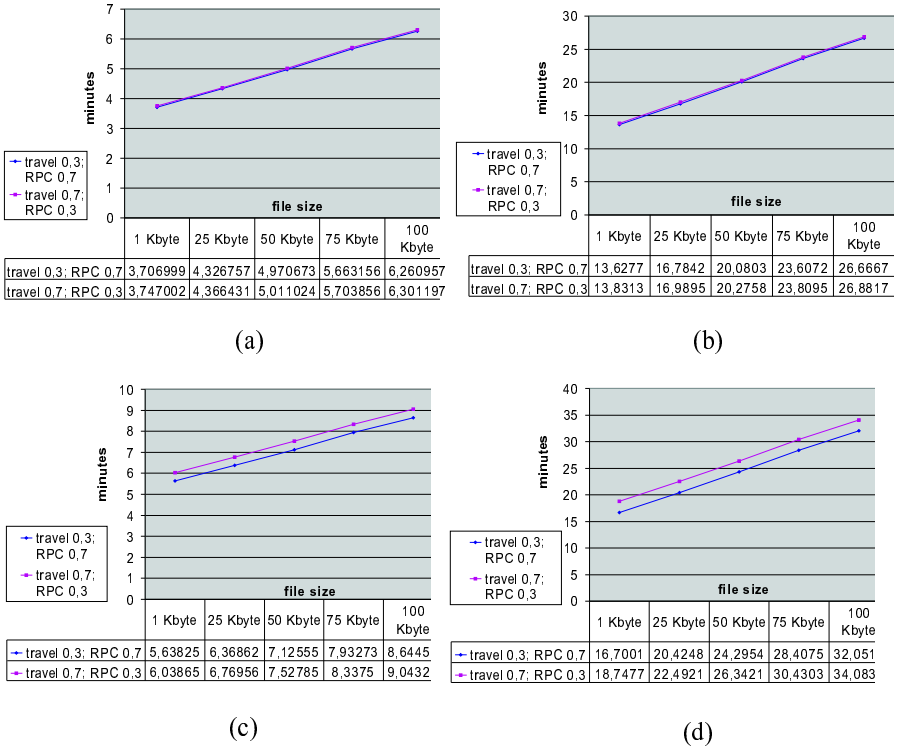
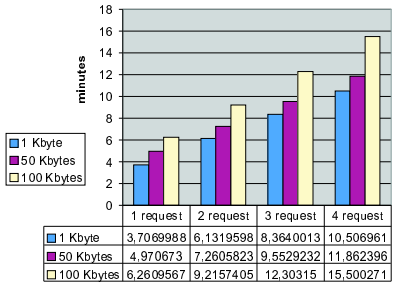


Fig. 18. Response time for a different scenarios with an "intelligent Browser". (a) and (b) represent a "fast" connection speed, (c) and (d) a "slow" connection speed; (a) and (c) an "expert user" and (b) and (d) a "naive user".

Figure 18(b) shows system response time (in minutes), supposing "fast connection", "intelligent" Browser, "naive user". The lines have identical meaning than in Figure 18(a). The two solutions still remain identical.

Someone could suspect that there exist small differences because of the net speed. So, we have decreased the net speed to 10 Kbytes/sec., (Figures 18(c) and 18(d)). It can be seen how the differences still remain non significant.

Finally, Figure 19 represents a test for an "intelligent Browser", an "expert" user, a probability for RPC equal to 0.7 and equal to 0.3 to travel. Now, we have tested the system for a different number of requests ranging from 1 to 4, thus the coloured model in Figure 17 has been used. Observe that when the number of requests is increased, the response time for each request increases, i.e., tasks cannot execute completely in parallel. Alfred and the Software Manager are not duplicated with simultaneous requests. Thus, they are the bottleneck for the designed system with respect to the number of concurrent requests of the service. Therefore, the next step in the performance analysis of the model would be to consider several majordomos (we do not include here due to space limitations).



	1 request	2 request	3 request	4 request
1 Kbyte	3,7069988	6,1319598	8,3640013	10,506961
50 Kbytes	4,970673	7,2605823	9,5529232	11,862396
100 Kbytes	6,2609567	9,2157405	12,30315	15,500271

Fig. 19. Response time for an "intelligent Browser", an "expert user", a "fast" connection and also different number of request.

6 Conclusions and further work

The main goal of this paper was to present an approximation to evaluate performance in design mobile agent software. We have used as test a system designed for providing mobile computer users with a software retrieval service. We summarise the contributions in the following items:

- A model to evaluate software performance has been integrated in the software life cycle. It has been done in the early stages of the modelling process. Thus, when performance or functional requirements change, it will be easy and less expensive to assume them. Moreover, the approach will permit to obtain the performance figures in an automatic way: Starting from the pa-UML models, the component Petri nets are systematically achieved, and from these the net system, finally the net system allows performance evaluation.
- In order to apply any technique to analyse rigorously system performance, the use of a formal model is crucial. So, we have used Petri nets to design software, avoiding the UML ambiguity.
- Concurrency is ambiguously expressed in UML, but when the translation to Petri nets is performed, a concurrent well-defined model is gained, so different kinds of concurrent systems can be analysed.
- The modelled example presents a complex system which is expensive to implement. Our approach offers an analytic way of evaluating such kind of systems without having to implement several prototypes. The results coincide with those obtained by the ANTARCTICA designers. Their results were obtained with implemented prototypes.

Concerning future work, we are interested in the following objectives:

- Software design is a complex task. So, we advocate for the reuse of the knowledge acquired in the application domain. In this way, patterns will be introduced to design software using agents. Each design pattern will deal with its own performance skills. So, we will have a pattern design library with the proper use of the performance parameters.
- As we have said, UML semantics is not defined formally, so our approach brings a formal semantics based on Petri nets to model the system. In this article, we have proposed rules to obtain the Petri nets. We will work in this line to get a formal translation from the pa-UML notation to Petri nets semantics.

References

[1] M. Ajmone Marsan, G. Balbo, and G. Conte, *A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems*, ACM Transactions on Computer Systems **2** (1984), no. 2, 93–122.

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

[3] 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.

[4] 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.

[5] J. Dilley, R. Friedrich, T. Jin, and J. Rolia, *Web server performance measurement and modeling techniques*, Performance Evaluation (1998), no. 33, 5–26.

[6] D. Coleman et Al., *Object oriented development. the Fusion method*, Object Oriented, Prentice Hall, 1994.

[7] J. Rumbaugh et Al., *Object oriented modeling and design*, Prentice-Hall, 1991.

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

[9] C. Harrison, D. Chess, and A. Kershenbaum, *Mobile agents: are they a good idea?*, Mobile Object Systems: Towards the Programmable Internet, 1997, pp. 46–48.

[10] I. Jacobson, M. Christenson, P. Jhonsson, and G. Overgaard, *Object-oriented software engineering: A use case driven approach*, Addison-Wesley, 1992.

[11] E. Kovacs, K. Röhrle, and M. Reich, *Mobile agents OnTheMove -integrating an agent system into the mobile middleware*, Acts Mobile Summit (Rhodos, Grece), June 1998.

[12] E. Mena, A. Illarramendi, and A. Goñi, *Customizable software retrieval facility for mobile computers using agents*, Proceedings of the 7th International Conference on Parallel and Distributed Systems (ICPADS'2000), Workshop International Flexible Networking and Cooperative Distributed Agents (FNCDA'2000) (Iwate (Japan)), IEEE Computer Society, July 2000.

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

[14] Object Management Group, *The common object request broker: Architecture and specification*, June 1999, Revision 2.3.

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

[16] R. Pooley and P. King, *The unified modeling language and performance engineering*, IEE Proceedings Software, IEE, March 1999.

[17] N. Rico and G.V. Bochman, *Performance description and analysis for distributed systems using a variant of LOTOS*, 10th International IFIP Symposium on Protocol Specification, Testing an Validation, July 1990.

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

[19] G. Waters, P. Linington, D. Akehurst, and A. Symes, *Communications software performance prediction*, 13th UK Workshop on Performance Engineering of Computers and Telecommunication Systems (Ilkley), Demetres Kouvatsos Ed., July 1997, pp. 38/1–38/9.

[20] 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.