A Pattern-based Approach to Model Software Performance Using UML and Petri Nets: Application to Agent-based Systems*

José MERSEGUER, Javier CAMPOS, Eduardo MENA
Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza, Zaragoza, Spain
{jmerse,jcampos,emena}@unizar.es

ABSTRACT

In this paper we present a formal approach to analyse performance for distributed systems, which is integrated in the early stages of the software development process. We propose to model the software system in a pragmatic way using as a design technique the well-known design patterns; from these models, the corresponding formal performance model, in terms of Petri nets, is obtained semi-automatically by applying a set of translation rules. Finally, the formal performance model is analysed, using analytical techniques, in order to study the performance of the system. Moreover, another benefit of the proposal is that it is possible to predict the behaviour of the system without the necessity of implementing it. To illustrate the proposal, we apply it to a software retrieval service system designed using mobile agents.

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

1. INTRODUCTION

In the last years distributed software applications have increased their possibilities making use of Internet capabilities. Although there are researchers who question mobile agents, they take sense in distributed environments because it is a technology with appropriate new skills for these kind of systems. However, 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* [11] appears as a discipline inside software engineering to deal with model performance on software systems design.

The *Unified Modeling Language* (UML) [9] is widely accepted as a standard notation to model software systems. One of the goals of this paper is the study of the performance indices in mobile agent systems, thus, we propose to extend *UML with performance annotations* (paUML) [8] 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 paUML in conjunction with design patterns [5], describing static and dynamic views when necessary (preliminary ideas were presented in [7]). The paUML models will give us the necessary background to semi-automatically obtain the corresponding formal model expressed as *Petri nets* [10]. From paUML, we derive, by applying a set of rules, a time interpretation of Petri nets leading to Generalized Stochastic Petri Nets (GSPN) [2]. Thus, we implicitly give a semantics for paUML in terms of Petri nets. Performance indices may be computed for GSPN by applying quantitative analysis techniques already developed in the literature. Moreover, as the Petri nets have been obtained as a by-product of the software life-cycle, the benefits from the pragmatic methodologies are preserved, for example the possibility of using CASE tools such as Rational Rose [1] to automatically generate code for the implementation phase or the database.

The rest of the paper is organised as follows. In section 2, we describe a system, based on agents, which has been taken from [6]. In section 3, we present the process to evaluate performance of agent systems using our proposal of "enriched" design patterns and our proposal of paUML. In section 4, we apply the first step of the process. In section 5 a set of rules is given to semi-automatically obtain the Petri nets from the paUML diagrams in order to achieve the desired formal model. In section 6 the last step of the process will be applied, consequently performance results for the proposed system will be obtained. Finally, conclusions are presented in section 7.

2. CASE STUDY: ANTARCTICA SRS

The ANTARCTICA[1] system [6], developed by the Interoperable Database Group (Country Basque University), was designed to provide mobile computer users with dif-

*This work has been developed within the projects TIC2002-04334-C03-02 of the Spanish CICYT and P084/2001 of the Gobierno de Aragón.

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

ferent services that enhance the capabilities of their computers. One of these services is the Software Retrieval Service [6] (ANTARCTICA SRS), that allows users to select and download new software in an easy and efficient way. The ANTARCTICA SRS has been chosen as a case study to apply our proposal to evaluate performance because it was designed using mobile agent technology. 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 users in the selection of the software.The system maintains up to date the information related to the available software.

In the following, we briefly describe the system paying attention to its components, see figure 1. There is a "majordomo" called *Alfred*, which is an agent specialised in 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 whenever needed.

Although considering the importance and the relevance of the results of the work [6], we would like to stress the enormous cost of implementing different prototypes in order to evaluate the performance of the different alternatives. In the next sections, we model the system in a pragmatic way using paUML combined with design patterns, annotating consistently the system load (we have annotated the system load taking as a basis the experiments and experience of the authors of the ANTARCTICA SRS). After that, we can interpret the paUML model in terms of - Petri nets and derive the corresponding performance model

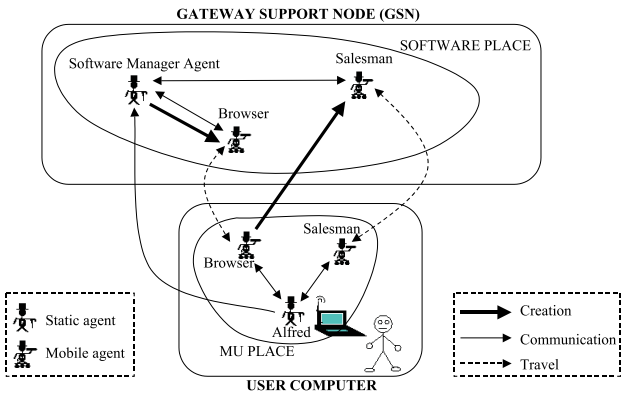


Figure 1. Architecture for the ANTARCTICA SRS.

which will be properly analysed. This analysis is used to evaluate the performance of the system.

3. PROCESS TO MODEL AGENT SYSTEMS

As we have introduced, our process combines UML and design patterns. We have considered UML because actually it has became a standard in the software engineering community. Benefits from design patterns come from their ability to achieve software reuse [5].

Pragmatic object-oriented methodologies or design techniques such as [5] do not deal with performance requirements. So, we can say that there is not an accepted *process* 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 Petri nets [10], have considered and studied the problem in depth. Thus, there are several proposals where we can learn from.

We consider that our proposal to evaluate performance in software systems must accomplish with both, the process and the notation. The process will give us the method to model the system and how to identify the relevant performance parameters that must be taken into account. We advocate for a pattern-oriented approximation to deal with performance on the software development process at the design stage. Lately, design patterns [5] have gained relevance in software development due to their simplicity and flexibility.

The idea of the design patterns was formulated initially in the architecture[2] field. This way to use and document designs was rapidly accepted in most of the engineering fields and concretely in the software design field was formulated in [5] as "*descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context*". We assume that the reader is familiar with the patterns language as proposed in [5], where twenty three design patterns, that solve a wide range of software design problems, are proposed. This language describes each pattern using the "sections" that appear in [5].

Concerning the notation, we propose a UML extension (paUML), which will be described in section 4. 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 article, we are interested only in behavioural aspects, concretely in the sequence diagram and the statechart diagrams. Future works will deal with the rest of the UML diagrams to describe behaviour (activity diagrams and collaboration diagrams), structural aspects

[2]The art and science of designing and making buildings. Not the hardware or software architecture fi eld.

(use case diagrams and class diagrams) and implementation diagrams.

Concerning the process, the following three main steps are proposed:

- **Step 1**. Model software requirements using paUML in conjunction with design patterns.
- **Step 2**. Apply the translation rules given in section 5 to the models obtained in the previous step in order to generate the corresponding formal model in terms of Petri nets.
- **Step 3**. Apply analytical techniques to solve the formal model.

The first step of our process is concerned with the description of the functional and performance aspects of the system. It does not pretend to be orthogonal to the common use of the design patterns language. On the contrary, this step proposes to use design patterns as usual in the object oriented development. So, we propose to extend the pattern language with sections that portray the performance parameters for the system, complementing in this way the functional description. This enlargement will come from two sides:

1. The "Collaborations" and "Participants" sections will be enhanced to annotate performance requirements.
2. Two new sections,"Performance goals" and "Workload definitions" will be added to the language with the same purpose.

Obviously, the performance annotations will be made using our proposal, paUML. These improvements are explained in the following.

Currently, the "Collaborations" section is described in some patterns using a sequence diagram, the rest of the patterns describe it in a textual way. We propose the mandatory use of a sequence diagram in all the patterns to describe this section. In this way, the sequence diagram will be used to annotate the performance parameters of the system that represent the load of the messages sent among objects. Which kind of annotations, their meaning in the diagram and how to obtain them are explained in section 4.1 using the running example.

The "Participants" section of the language, which describes the classes/objects participating in the pattern, must be also enhanced from our point of view. We propose that a statechart diagram for each participant must be modelled. These diagrams represent the life of the objects in the system. In order to describe the performance parameters of the system that represent the load introduced by the objects themselves, the statechart will be annotated with the events load, the probabilities of the guards and the time to perform the actions. In section 4.2, this enhancement is explained using the running example.

Now, we explain the proposal to extend the design patterns language with new sections. These sections, taken from [11], are necessary to obtain a complete description of the system performance features:

- **Performance goals**: the pattern performance objectives will be expressed. For instance, response time, throughput or utilization. Section 4.3 shows an example.
- **Workload definitions**: such as request arrival rates or the number of concurrent users. Section 4.4 shows an example.

The second and the third steps of the process are described respectively in sections 5 and 6 using the ANTARCTICA SRS as an example.

4. STEP 1: pa-UML & DESIGN PATTERNS

In the following, we apply the first step of the process proposed in section 3, i.e., to model the system described in section 2 using paUMLand design patterns.

The system is modelled according to the *mediator* pattern. The mediator pattern is advised in [5] when *an object that encapsulates how a set of objects interact, therefore it promotes loose coupling by keeping objects from referring to each other explicitly, and it lets to vary their interaction independently*. We propose a design following the mediator pattern because it is interesting that Alfred acts as a mediator object, preventing the user to interact with the rest of the objects of the system.

4.1 The Collaborations Section

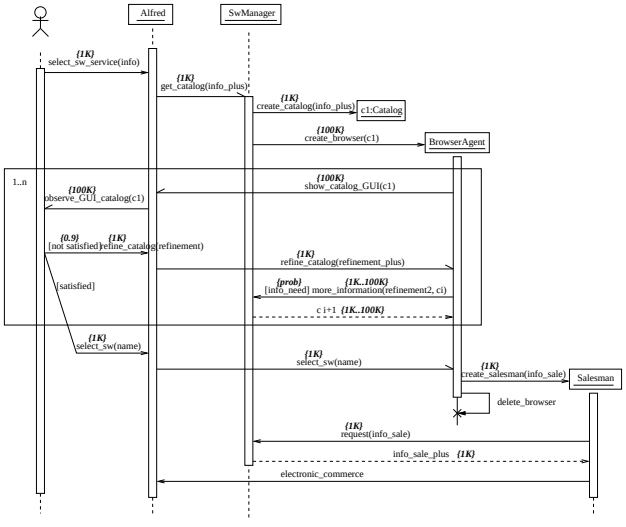


Figure 2. Sequence diagram for the Software Retrieval Service.

As we introduced in the previous section a *sequence diagram* [9] represents messages sent among objects, see figure 2. 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.

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

4.2 The Participants Section

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 *statechart* diagram is the tool that describes this aspect of the system, for these reasons we propose to incorporate them to the "Participants" section of the enhanced design pattern language. Then, for each class with relevant dynamic behaviour a statechart diagram must be specified in this section.

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

The statechart diagrams for our system using the paUML notation can be seen in [8].

4.3 The Performance Goals Section

The performance objectives for this system are: The study of the system *response time* in the presence of a user request and to identify 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.

4.4 The Workload Definitions Section

In the following, we summarize the workload considerations taken into account for the system:

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. 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) or it may travel through the net to the Software_place 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).

5. STEP 2: PNs MODELLING

The paUML models are expressive enough to accomplish with different implementations. A necessary condition to design methods is their independence of final implementation decisions. In this 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. As an example, in the system that we are treating we are not sure about how many majordomos should attend requests, how many concurrent users can use the system, etc. However, a formal modelling with Petri nets [10] solves these questions satisfactorily.

The design proposed in [6] deals with one user and one majordomo. Petri nets allow to represent cases such as: One user and one majordomo, several users served by one majordomo or 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.

At this point, we have modelled the system with paUML notation, taking into account the load in the sequence diagram and the statechart 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.

In order to obtain answers to our questions, we need to apply performance analytic techniques to the developed paUML 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 semi-automatically obtain Petri nets from paUML diagrams. Therefore, it must be underlined that a formal performance model can be obtained as a *by-product* of the software design stage.

We have modelled with Petri nets the first two proposed systems (one or several users and one majordomo), the third one (several majordomos) will be developed in a future work. For the first system, one user and one majordomo, GSPN [2] have the expressive power to accomplish the task, it is proposed in the following paragraphs. To study the second system, several users served by one majordomo, stochastic well-formed coloured Petri nets [3] are of interest, it can be consulted in [8]. Once the systems are modelled, we use analytic techniques implemented in GreatSPN [4] tool to obtain the target performance requirements.

Petri net model: One majordomo and one user

We are going to obtain a Petri net for each system class, the *component nets*. Obviously, every annotated statechart 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 statechart 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.*

Rule 2 *Activities inside a state of the statechart 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 statechart diagram will become immediate transitions with the associated corresponding probabilities for the resolution of conflicts.*

Rule 4 *States in the statechart 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).*

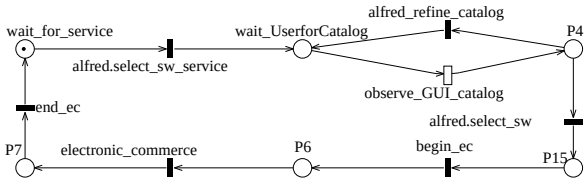


Figure 3. User Petri net component

Figures 3, 4, 5, 6 and 7 represent the nets needed to model our *system components* taking into account the previous transformation rules. According to GSPN notation [2], 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.

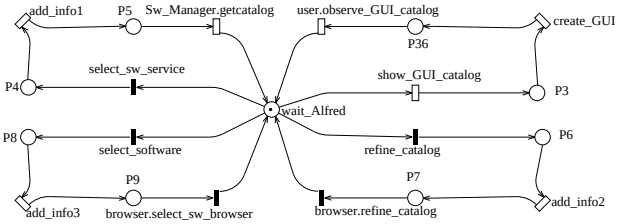


Figure 4. Alfred Petri net component

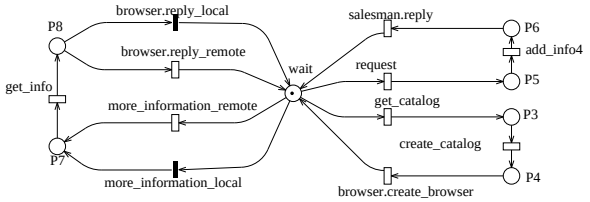


Figure 5. Software Manager Petri net component

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 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.*

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 8. In order to understand how to apply the previ-

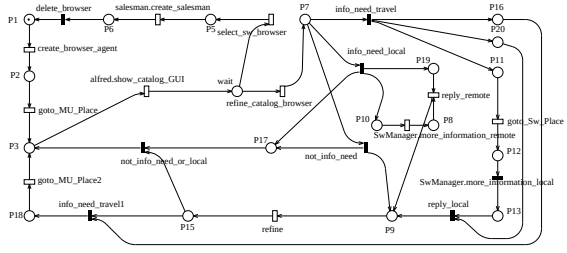


Figure 6. Browser Petri net component

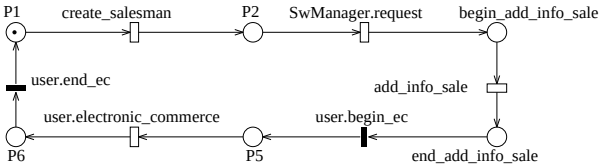


Figure 7. Salesman Petri net component

ous rules, we are going to explain how to obtain the observe_GUI_catalog transition in the net system (figure 8) from the observe_GUI_catalog message sent by Alfred to the user in the sequence diagram. We can observe in Alfred's net (figure 4) and in the user's net (figure 3) 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.

6. STEP 3: PERFORMANCE STUDY

As we expressed in the "performance goals" of the system, see section 4.3, 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. Also, we proposed as a performance goal to determine *the bottlenecks of the system and identify their importance*. In order to study the two possible bottlenecks identified in section 4.3, we have developed a test taking into account the "workload definitions" given in section 4.4 and taking into account that: Transition not_info_need measures if the Browser has enough information to create a new catalog and transitions info_need_local and info_need_travel represent respectively that the Browser performs a RPC or a travel to the Software_place to obtain information to create a new catalog.

In Figure 9, we have tested the system for a different number of requests ranging from 1 to 4, thus the coloured model in [8] 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

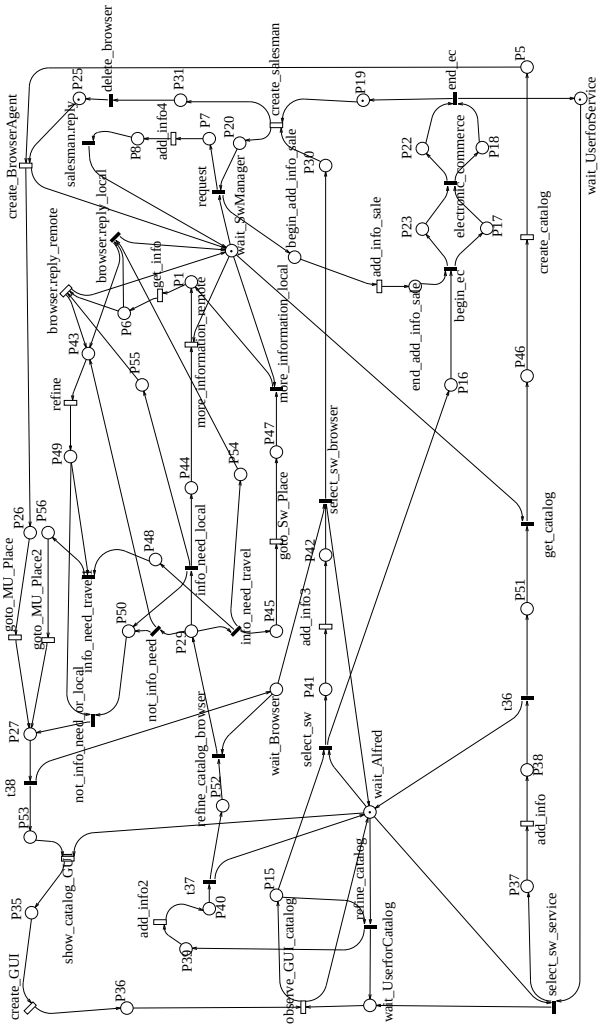


Figure 8. The Petri net for the whole system

limitations).

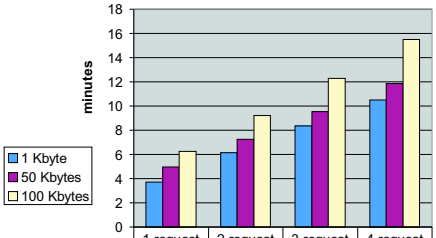
7. CONCLUSIONS

The main goal of this paper is to present an approach to evaluate performance in design mobile agents 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 process to evaluate software performance has been integrated in the early stages of the software life-cycle. Thus, when performance or functional requirements change, it will be easy and less expensive to assume them. Starting from the paUML models, the component Petri nets are systematically obtained, and from these the net system, finally the net system allows performance evaluation. Benefits from pragmatic software methodologies are preserved.

- Software design is a complex task, therefore any kind of reuse becomes interesting. In this way, design patterns have been introduced to design software using agents taking into account performance parameters.

- 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 SRS designers using implemented prototypes.



	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

Figure 9. System response time.

8. REFERENCES

[1] Rational Software Corporation., 2001. `http://www.rational.com`.

[2] 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(2):93–122, May 1984.

[3] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed colored nets for symmetric modelling applications. *IEEE Transactions on Computers*, 42(11):1343–1360, November 1993.

[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*, 24:47–68, 1995.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[6] E. Mena, A. Illarramendi, and A. Goñi. A software retrieval service based on knowledge-driven agents. In *Cooperative Information Systems CoopIS'2000*, pages 174–185, Eliat, Israel, September 2000. Opher Etzion, Peter Scheuermann editors. Lecture Notes in Computer Science, (LNCS) Vol. 1901, Springer.

[7] J. Merseguer, J. Campos, and E. Mena. A pattern-based approach to model software performance. pages 137–142, Ottawa, Canada, September 17-20 2000. ACM. ISBN 1-58113-195-x.

[8] J. Merseguer, J. Campos, and E. Mena. Performance evaluation for the design of agent-based systems: A Petri net approach. In Mauro Pezzé and Sol M. Shatz, editors, *Proceedings of the Workshop on Software Engineering and Petri Nets, within the 21st International Conference on Application and Theory of Petri Nets*, pages 1–20, Aarhus, Denmark, June 2000. University of Aarhus.

[9] Object Management Group, http:/www.omg.org. *OMG Unifi ed Modeling Language Specifi cation*, September 2001. version 1.4.

[10] M. Silva. *Las Redes de Petri en la Automática y la Informática*. Editorial AC, Madrid, 1985. In Spanish.

[11] C. U. Smith. *Performance Engineering of Software Systems*. The Sei Series in Software Engineering. Addison–Wesley, 1990.