Evaluating Performance on Mobile Agents Software Design

José Merseguer, Javier Campos, and Eduardo Mena

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

Abstract. Software design and implementation using mobile agents are nowadays involved in a scepticism halo. There exist researchers who question its utility because it could be a new technology which no new skills but could introduce new problems. Security and performance are the most critical aspects for this new kind of software. In this contribution we present a formal approach to analyze 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 expected behaviour without the necessity of carry out the complete implementation phase. To show the approach, we model a software retrieval service system in a pragmatic way, after the corresponding formal model is obtained and analyzed 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. Client/server has become the key paradigm to support distributed software development. It is widely recognized that there exist 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* [14, 11] technology are a very interesting approach to address certain kind of software domains like electronic commerce, information retrieval and network management and administration.

Although there exist researchers who question mobile software, because it could be a new technology with no new skills but it could introduce new problems; mobile software takes sense in distributed environments [9]. This software could make an 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 take care about these problems. In this context, *software performance* [16] appears as a discipline inside software engineering to deal with model performance on software systems design. Like many people concerned about software performance, we believe that performance evaluation must be accomplished during the early stages of software development.

Assuming *Unified Modeling Language* (UML) [2] as a standard notation to model software systems, we propose an environment in which software analyst models using UML. Unfortunately, UML lacks of expressivity desired to describe performance skills. There have been several approaches to solve this lack [17, 18]. One of the principals of our work is the study of the performance indices in mobile agent systems, so we propose a *UML time extension* 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 UML, describing static and dynamic views when necessary. UML models will give us the necessary background to obtain the corresponding formal model expressed as *Petri nets* [1, 4]. From the time extension of UML, we derive a time interpretation of Petri nets leading to Generalized Stochastic Petri Nets (GSPN) [1]. Performance indices may be computed for GSPN by applying quantitative analysis techniques already developed in the literature.

The rest of the paper is organized as follows. In section 2, we describe a system, based on agents, which has been taken from [12], it will introduce us in a mobile agent environment. Sections 3 and 4 develop the UML models for the system and give a proposal to annotate system load. Section 5 is dedicated to transform 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 ANTARTICA system

In this section we briefly present ANTARTICA[1]. The system taken from [12] that will be used as example along this paper. It will be the reference to study performance on mobile agent system.

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

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

In the following, we briefly describe the system paying attention in its components. There exists a “majordomo” named Alfred, which is an agent specialized 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 instalation of the selected software, like electronic commerce.

The system was proposed using different technologies, namely CORBA [13], 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, we would like to stress the enormous cost of implementing different prototypes in order to evaluate the performance of the different alternatives. We could model the system with a pragmatic approach using UML and annotate consistently the system load. After that, we can interpret the UML models in terms of Petri nets and derive the corresponding performance model which will be properly analyzed. This analysis is used to evaluate the system.

Thus, we have worked with the authors of the cited paper in order to obtain the UML models corresponding to the system, and also we have all together annotated the system load, taking their experiments and experience as basis. Later, we have translated the UML diagrams into the corresponding formal ones. Results look promising.

3 Modeling the system using UML notation

In the previous section we have explained the general features of the target system. Now we focus on modeling it using 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 dynamical view of the system, since performance indices are defined in terms of the dynamic behaviour of the system.

Figure 1 shows the target use cases needed to describe the dynamic behaviour of the system. We deal with three different use cases, “show services”, “download software” and “electronic commerce”. Also we can see the unique actor which interacts with the system, the “user”. The first two use cases are described in the following.

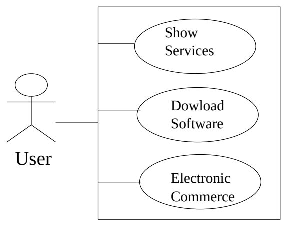

Fig. 1. Use Cases

Show services use case description.
- Principal event flow: the use case goal is to show to the user the available services offered by the system. In the list of services there would appear the download software service.

Download software use case description.
- Principal event flow: the user requests the system for the desired software. The majordomo, Alfred, is concerned to get a catalog and must show 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 of the catalog. This process could be repeated as many times as necessary until selecting a concrete piece of software.

Electronic commerce use case description.
- Principal event flow: the goal is to provide the user an electronic commerce activity.

Show services use case and electronic commerce use case are out of the scope of this article, thus we concentrate on download software operation.

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

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 to describe this aspect of the system. For each class with relevant dynamic behavior a state transition diagram must be specified. Thus, we are going to present the corresponding state transition diagrams for our system.

Alfred state transition diagram. The example supposes that Alfred is always present in the system, no event creation is relevant for our purposes. So the state transition diagram begins when a view_services event is sent to the user. Alfred's behaviour is typical for an object server 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 his wait state to serve another request. Figure 3 shows Alfred's behaviour.

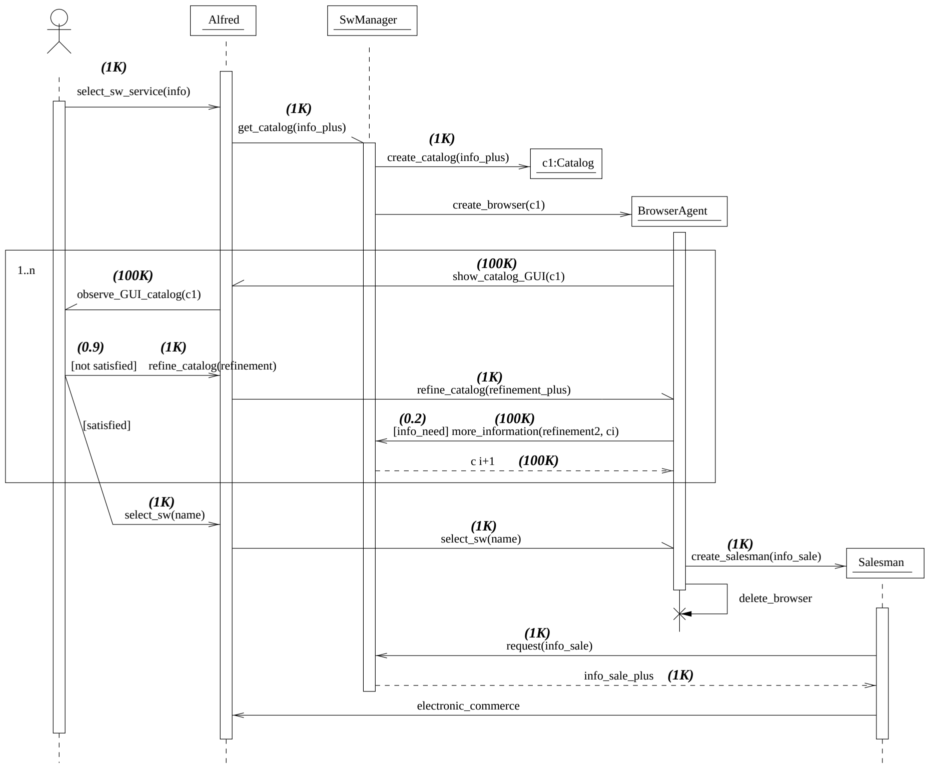

Fig. 2. Sequence diagram for download software

The stereotyped transition ≪ *more_services* ≫ points out that Alfred may attend for other services that are not of interest here.

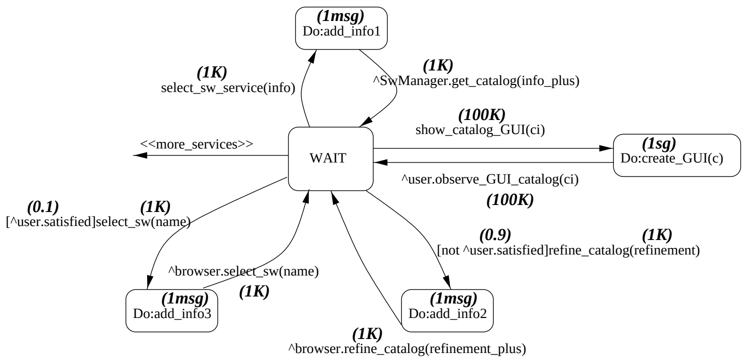

Fig. 3. State transition diagram for Alfred

Software manager state transition diagram. Like Alfred, the software manager behaves as an object server. It is waiting for a request event (more_information, get_catalog, request) to enable the necessary mechanism to accomplish the task. Figure 4 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.

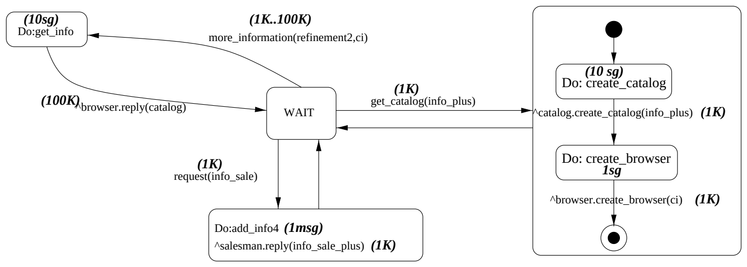
(10sg)
Do:get_info
(1K..100K)
more_information(refinement2,ci)
(100K) ^browser.reply(catalog)
WAIT
(1K)
get_catalog(info_plus)
(1K)
request(info_sale)
Do:add_info4 (1msg)
^salesman.reply(info_sale_plus) (1K)
(10 sg)
Do: create_catalog
^catalog.create_catalog(info_plus) (1K)
Do: create_browser
1sg
^browser.create_browser(ci) (1K)

Fig. 4. State transition diagram for the software manager

Browser state transition diagram. State transition diagram in Figure 5 describes the Browser's life. It is as follows: once the Browser is created it must go to the MU_Place, where invokes Alfred's shows_catalog_GUI method to offer it 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: if the browser has the necessary knowledge to solve the task, a refine action is directly performed, but if actually it has not this background, the browser must obtain information from the software manager, by sending a more_information request. If the second event occurs, select_sw event, the browser must create a salesman instance and die.

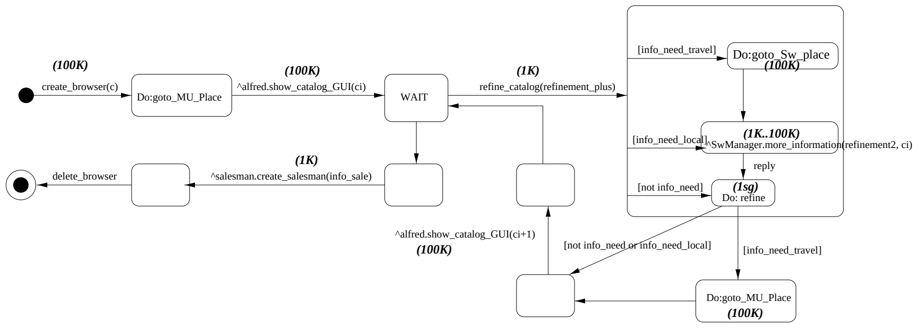

Fig. 5. State transition diagram for the browser

Salesman State Transition Diagram. The Salesman's goal is to give electronic commerce services, as we can see in Figure 6. After its creation it asks the software manager for sale information. With this information the electronic commerce can start. This is a complex task that must be described with its own use case and sequence diagram but it is out of the scope of this paper.

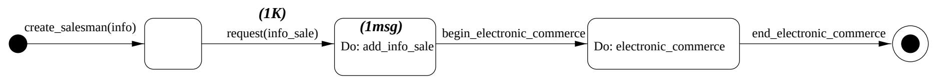

Fig. 6. State transition diagram for the Salesman

The developed models are expressive enough to accomplish with different implementations. It remarks the necessary independence between methodologies and final implementations. So, we can use these models to develop CORBA, mobile agents, etc. based applications. But this lack could be not desirable in certain cases. For example, in the system we are treating we do not know how many majordomos should attend requests, how many concurrent users can use the system, etc. However, a formal modeling with Petri nets allows that most of these questions are well solved.

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. Many users and many majordomos.

We can see as less effort offers more results, due to avoid the necessity of implementing the system for predicting performance figures.

4 UML extensions to deal with performance

Pragmatic object-oriented methodologies such as [6, 7, 10] do not deal with performance skills. So, we can say that there does not exist an accepted method to model and study system performance in the object-oriented software development process. This lack implies that there does not exist a well-defined language or notation to annotate system load, system delays and routing rates. On the contrary, formal specification languages, such as LOTOS [15], or Petri nets [1], have considered and studied the problem in depth. Thus, there exist several proposals where we can learn from.

In this context, as we remarked before, it is our objective to propose a UML extension to deal with performance on the software analysis and design stages.

We considered that our proposal must accomplish with both, the method and the notation. First, the method will give us the relevant parameters to be into account and the process to model the system. 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, the notation is treated in this work.

In order to have a complete performance notation, the UML behavioral and structural models must be considered. Also, performance will play a prominent role in the implementation diagrams. In this paper, we are going to fix our attention only in behavioral 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.

4.1 Sequence diagrams

In a sequence diagram, messages sent between objects are represented. Normally, a message is considered as no time consuming in the scope of the modeled system. But in a mobile agent system, we distinguish between messages sent by objects on the same computer and messages sent between objects on different computers, those which travel by 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, between brackets, will be made indicating the message size. Also it will be possible to annotate a range for the size in the UML common way. For instance, in Figure 2 select_sw_service message is labeled with (1 Kbyte), while show_catalog_GUI requires the movement of (100 Kbytes)

In a sequence diagram, conditions represent the possibility of a message to be sent or not. An annotation, also between brackets, expressing the event probability success will be associated to each condition. A range is accepted too. See, for instance, probability (0.9) associated in Figure 2 to the condition not_satisfied.

4.2 State Transition diagrams

In a state transition diagram two elements will be considered, the *activities* and the *guards.*

Activities represent tasks performed by the object in a given state. Such activities consume computation time that must be measured and annotated. The annotation will be between brackets indicating the time needed, with the possibility of indicating a range if the activity requires it. See, for example, bold labels between brackets in Figure 3, 4, 5, 6.

Guards show conditions in a transition that must be held in order to fire the corresponding event. A probability must be associated to them. It will be annotated in the same way that guards in the sequence diagram were annotated. See for instance label (0.9) in Figure 3.

5 Modeling with Petri nets

At this point, we have modeled the system with UML notation, and also the load has been taken into account through 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 system behaviour in two 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 UML diagrams developed. 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 concurrent capabilities and also there exist well-known analytic techniques to study system performance in stochastic Petri net models.

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

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

- Two different kind of transitions were identified. Those which do not spend net resources and those which do. The first kind will be translated into *immediate* transitions (that fire in zero time). The second kind will be *timed* transitions. The mean of the exponentially distributed time for the transition is calculated as a constant function of the message size and net speed. More elaborated proposals like [5] could be taken into account, but we have considered more important to gain simplicity.
- Actions inside a state of the state transition diagram are considered as time consuming, so in the Petri net model we consider them also as timed transitions. The time is calculated from the CPU and disk operations needed to perform the action.
- Guards in the state transition diagram will become immediate transitions with the associated corresponding probabilities for the resolution of conflicts.
- 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.

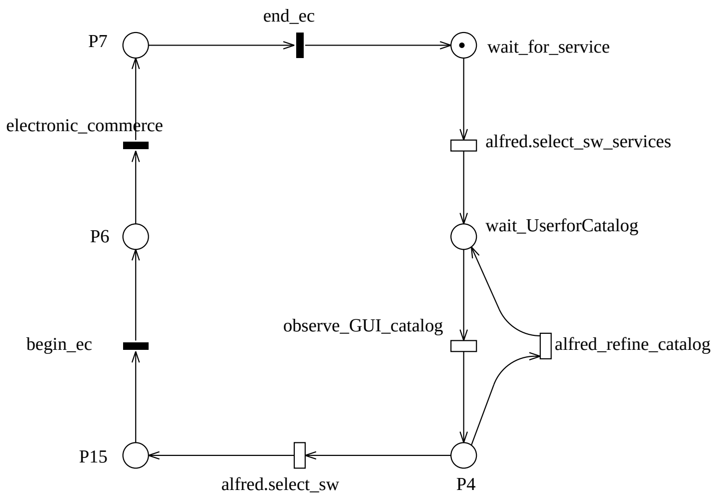

Fig. 7. User Petri net component

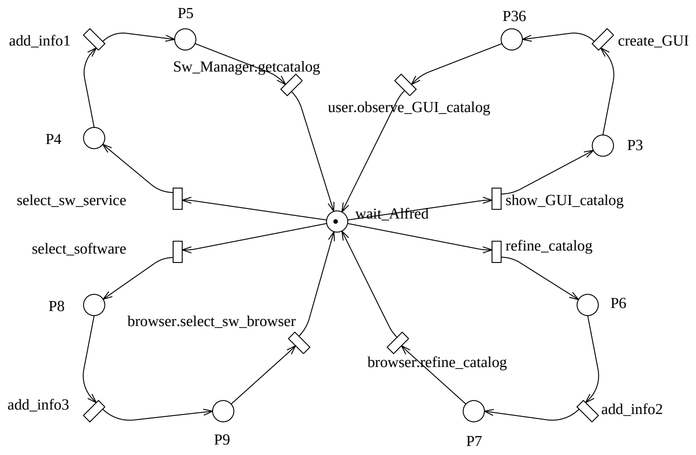

Fig. 8. Alfred Petri net component

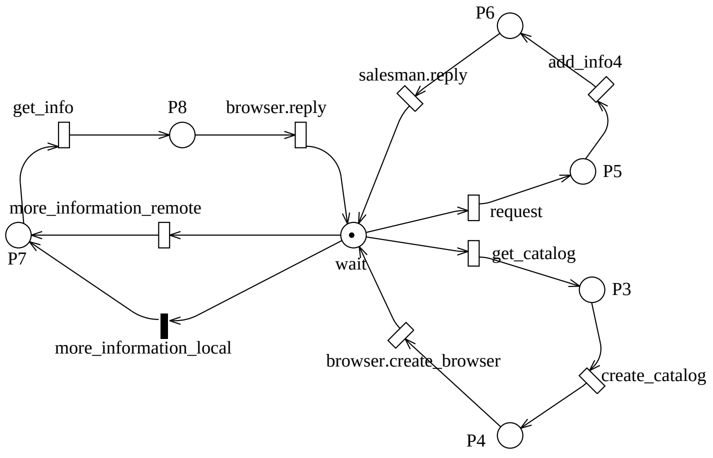

Fig. 9. SwManager Petri net component

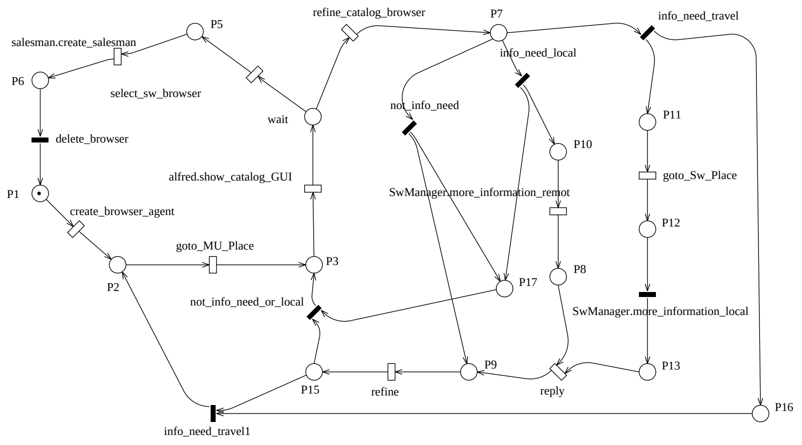

Fig. 10. BrowserAgent Petri net component

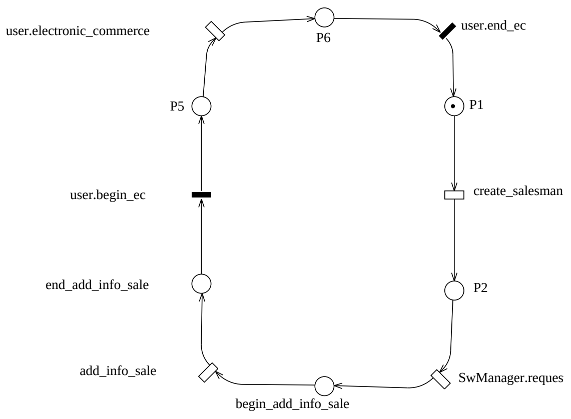

Fig. 11. Salesman Petri net component

Figures 7, 8, 9, 10 and 11 represent the nets needed to model our *system components* taking into account the previous rules.

The sequence diagram will be the guide to obtain a *complete* Petri net for the system from 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*)
– those represented by a half arrowhead (*no-wait semantics*).

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

– If the message has wait semantics, only one transition will appear in the net system, this transition will support the incoming and outcoming arcs from both net components.
– If the message has no-wait semantics, the two transitions will appear in the net system and also an extra place will be added modeling the communication buffer. This place receives an arc from the sender transition and adds an arc to the receiver transition.

The net system for the example is shown in Figure 12. It has been obtained supposing that all the messages have wait semantics. That is the case here because the example supposes only one user requesting every time.

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 12 from the observe_GUI_catalog message sent by Alfred to the user in the sequence diagram. We can observe in Alfred's net Figure 8 and in the user's net

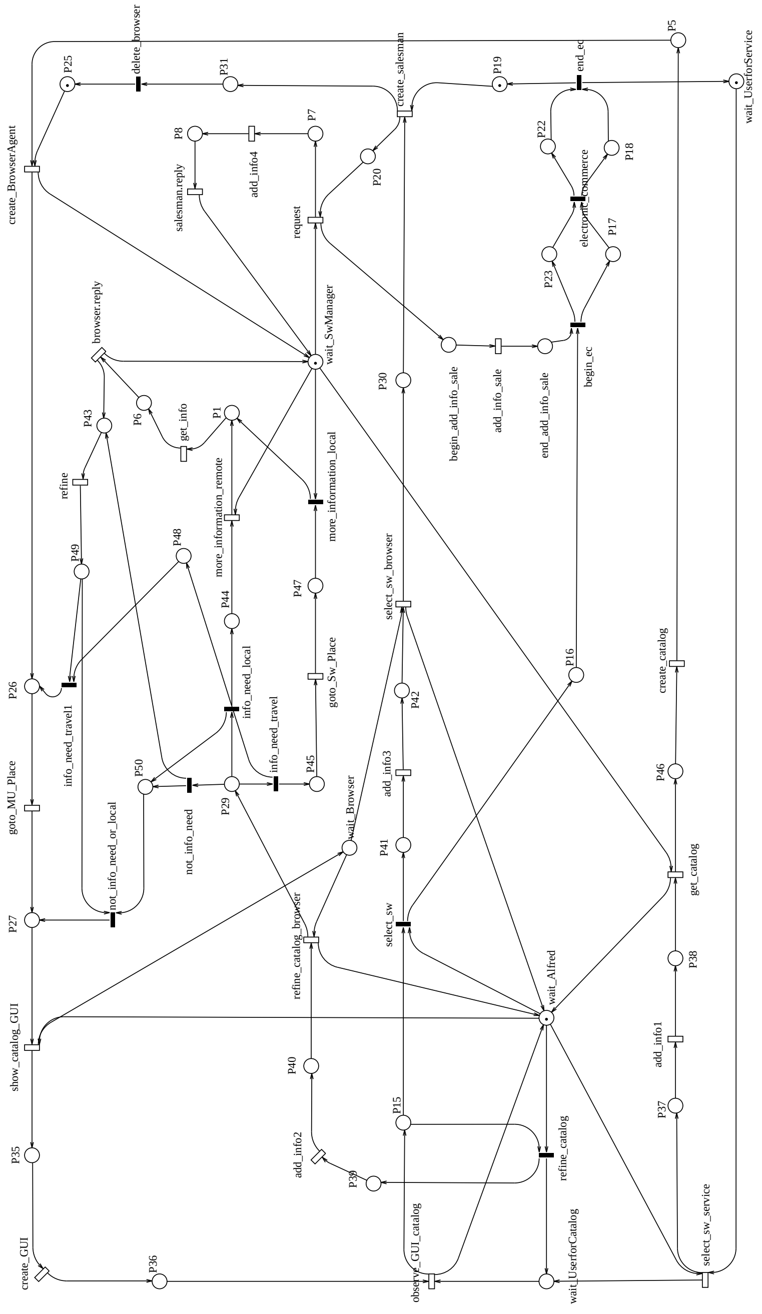

Fig. 12. The Petri net for the whole system

Figure 7 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, synchronizing in this way the lifes of the objects.

The results shown in the next section are obtained from the net system in Figure 12.

6 Performance results

As we have said in the previous section, the developed nets model the case in which the system is used by only one user who is attended by only one majordomo.

It is of our interest to study the system *response time* in the presence of a user request. To obtain 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 [3]; finally the inverse of the previous result gives the system response time. *We want to know which are the system critical parts to accomplish the request and identify its importance.* There exist two possible parts which can decrease system performance. First, the browser traveling by the net to the "software place" to obtain new catalogs. Second, the user requesting for a refinement of the catalog showed because s/he is not satisfied with it.

Table 1. System response time for an intelligent Browser

	RPC	*browser travels*
5	5,4716	5,7313
10	9,1374	9,1334
60	44,6827	49,4559
110	877,1929	980,3921

Table 2. System response time for a dummy Browser

	RPC	*browser travels*
5	6,1546	7,2369
10	10,3971	12,3915
60	50,6585	62,4219
110	1041,6666	1282,0512

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

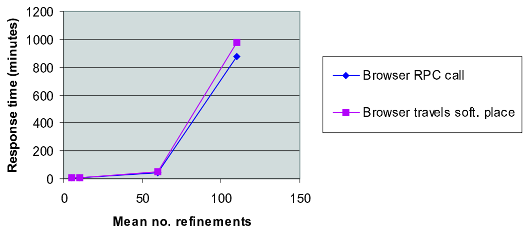

Fig. 13. RPC vs. travelling for an intelligent Browser

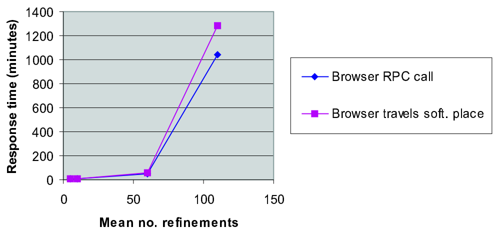

Fig. 14. RPC vs. travelling for a dummy Browser

- When the browser is requested for a new catalog there exist several possibilities:
  - The browser has enough information to accomplish the task or he needs to ask for the information. The first case supposes probability equal to 0 in the more_information transition, the second supposes probability 1 in the same transition. We have considered eleven different possibilities in a range that varies from a probability equal to 0, to a probability equal to 1, increasing the probability 0.1 each time.
  - 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 by the net to the Software_place (represented in the net system by the info_need_travel transition). In this case, we have also considered the same eleven different possibilities. As an example, if the probability to perform a RPC is 0.4 then the probability to travel by the net will be 0.6.
- To test the user refinement request, we have considered four different possibilities. A user requesting a mean of five, ten, sixty, and one hundred and ten refinements.

In Tables 1 and 2 we show some interesting results taken from our test. The complete test is not presented because of its extent.

Table 1 shows system response time (in minutes) supposing the browser has enough information to perform the task with a probability of 0.7 (an "intelligent" browser). Column one indicates a probability equal to 0.3 to perform a RPC, so there is not a probability to travel to the Software_place. Column two indicates a probability equal to 0.3 to travel to the Software_place, so there is not a probability to perform a RPC.

Table 2 shows system response time (in minutes) supposing the browser has enough information to perform the task with a probability of 0.3 (a "dummy" browser). Column one indicates a probability equal to 0.7 to perform a RPC, so there is not a probability to travel to the Software_place. Column two indicates a probability equal to 0.7 to travel to the Software_place, so there is not a probability to perform a RPC.

The results obtained in Tables 1 and 2 are shown in a graphical way in Figures 13 and 14 . We can observe as a result, that the browser's knowledge is more important for system response time than the agent travelling to the Software_place. We still work in the performance parameters to obtain more precise results.

7 Conclusions

The main goal of this paper is to present an approximation to evaluate performance in design mobile agent software. We have used a system designed for providing mobile computer users with a software retrieval service as test. We summarize 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 modeling process. Thus, when performance or functional requirements change, it will be easy and less expensive to assume them. Moreover, the approach permits to obtain the performance figures in an automatic way, beginning from the UML models the component Petri nets are systematically achieved, and from these the net system, finally the net system allows performance evaluation.
- UML semantics is not as well defined as desired, so our approach brings a formal semantics based on Petri nets to model the system. This is crucial to apply any technique to analyze rigorously system performance.
- 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 could be analyzed.
- The modeled 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.

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, *A graphical Petri net tool for performance analysis*, Proceedings of the 3^{rd} International Workshop on Modeling Techniques and Performance Evaluation (Paris, France), AFCET, March 1987.
4. 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.
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. Object Management Group, *The common object request broker: Architecture and specification*, June 1999, Revision 2.3.
14. E. Pitoura and G. Samaras, *Data management for mobile computing*, Kluwer Academic Publishers, 1998.
15. 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.
16. C. U. Smith, *Performance engineering of software systems*, The Sei Series in Software Engineering, Addisson–Wesley, 1990.
17. 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.
18. 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.