A Performance Engineering Case Study: Software Retrieval System*

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. This chapter presents a case study in performance engineering. The case study consists of a Software Retrieval System based on agents. The system is modelled in a pragmatic way using the Unified Modeling Language and in a formal way using stochastic Petri Nets. Once the system has been modelled, performance figures are obtained from the formal model. Finally, some concluding remarks are obtained from our experience in the software performance process.

1 Introduction

The common tasks of retrieving and installing software using Internet are provided by several sites (e.g., Tucows [3], Download.com [1], and GameCenter [2]). From a user point of view, the process of selecting and downloading software could become costly and sometimes slow, therefore the performance of these kind of services becomes *crucial*.

This chapter describes the performance study of a Software Retrieval System (SRS hereafter) proposed in [11]. This SRS, as those mention above, allows Internet users to select and download new pieces of software. The main differences between it and the others are that the SRS has been designed for wireless network systems (to satisfy mobile computer users) and it has been developed using mobile agents technology [16].

The chapter is organised as follows. In Section 2, the SRS is specified. In section 3, the SRS is modelled using the *Unified Modeling Language* (UML) and *Petri nets* (PN). In section 4 a performance analysis of the system is addresed. Finally, some concluding remarks are given.

2 System Specification

Mobile agents are intelligent and autonomous software modules that can move themselves from one *place* to another, carrying with them their whole states. A place is a context, within an agent system,[1] where an agent can execute [15].

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

[1] An agent system is a platform that can create, interpret, execute, transfer and dispose agents.

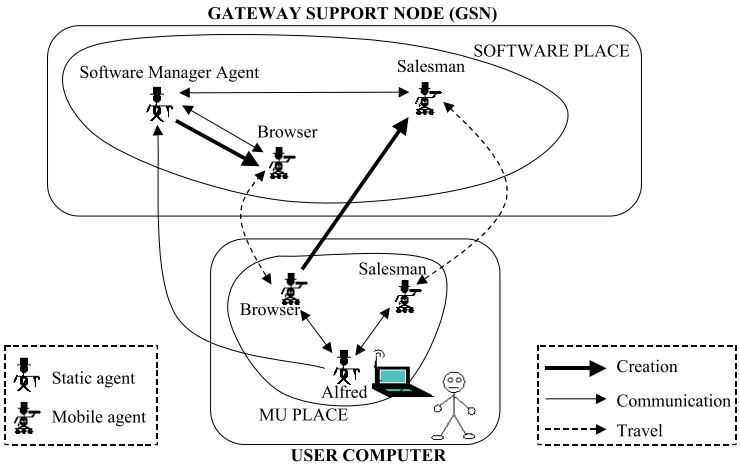


Fig. 1. Architecture for the Software Retrieval System

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

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

The main components that are involved in the SRS, with emphasis on the agents defined and how they interact, are shown in Figure 1. In the following we briefly describe how these agents interact when the user wants to retrieve software:

1. *Alfred*, an agent specialized in user interaction, allows the user to express her/his information needs. It resides in the *Mobile Unit (MU)* place.
2. Alfred communicates with the *Software Manager agent*, residing in the Software place, which obtains a catalog with the available software.
3. The Software Manager creates a *Browser agent*, a specialist in helping users to select software. This agent moves to the MU place carrying a catalog of software (see [12] for more details).
4. The Browser interacts with Alfred in order to select the wanted piece of software. The user can request a refinement of the catalog; in that case, the Browser will travel back to the *Gateway Support Node* (GSN) or, depending on the concrete refinement, it will request the information to the Software Manager. This process is repeated until the user selects the name of the software that s/he wants to install on her/his computer.

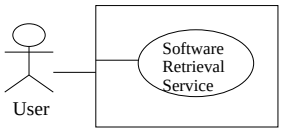


Fig. 2. Use Case diagram

5. Then, the Browser creates a *Salesman agent* on the GSN, which is on charge of performing all the tasks previous to installing the software on the user node, i.e., e-commerce, downloading of the software, etc.

Finally, the main features of the SRS are pointed out:

- Automatic generation (without user intervention) of a catalog of available software [12].
- The location and access method to retrieve remote software are transparent to users.
- The system maintains up to date the information related to the available software.
- The system is able to help the users to find software even if they are inexperienced users.

3 Modelling the Software Retrieval System

In the following, we are going to model the SRS as a previous step to obtain performance figures. First, it will be modelled using UML, but as we explain in section 5 it is not possible to obtain performance figures from UML diagrams. Therefore, in this section, we will model the system using PNs, which will be obtained from the UML diagrams. PNs will allow us to obtain performance figures for the system.

3.1 Modelling Using UML

The first step in the performance study of the SRS conveys in the development of the UML [4] diagrams that model the system. A *use case diagram*, a *sequence diagram* and several *statechart diagrams*, one for each agent present in the system, will be modelled; all them represent the dynamic view of the system.

In order to express the system load, the diagrams have been enriched following the notation presented in [14, 13]. It must be noted that for this case study all the system load can be represented in the dynamic view of the system, avoiding the necessity to develop a static view, which could be appropriately represented using a class diagram.

Figure 2 shows the only use case needed to describe the dynamic behaviour of the system. We can see in it the unique actor which interacts with the system, the "user". The use case is described in the following.

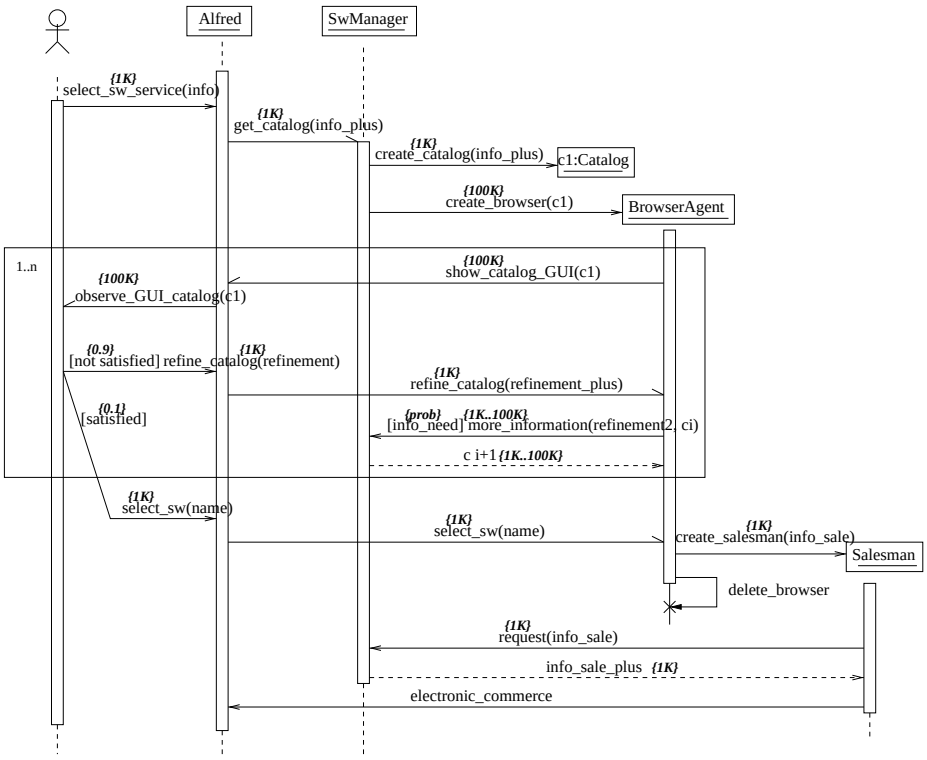


Fig. 3. Sequence diagram for the Software Retrieval System

Software Retrieval System Use Case Description.

– Principal event flow: the user requests Alfred for the desired software. Alfred sends the request to the Browser, who obtains a catalog with the available software. The Browser gives the catalog to Alfred, who shows it to the user. If the user is satisfied with the catalog then s/he selects the software, in other case 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.

The *sequence diagram* for the SRS, that describes in detail the use case, is shown in Figure 3. The diagram represents the messages sent among agents, their load and the probability associated to the guards.[2]

As it has been told, a *statechart* has been developed for each agent in the system. They represent the life of the agents as well as the load of their events, the time spent by their activities and the probabilities associated to the guards.

In the following a detailed description of the statecharts is given:

[2] It must be pointed out that these values have been obtained from our knowledge of the *problem domain*.

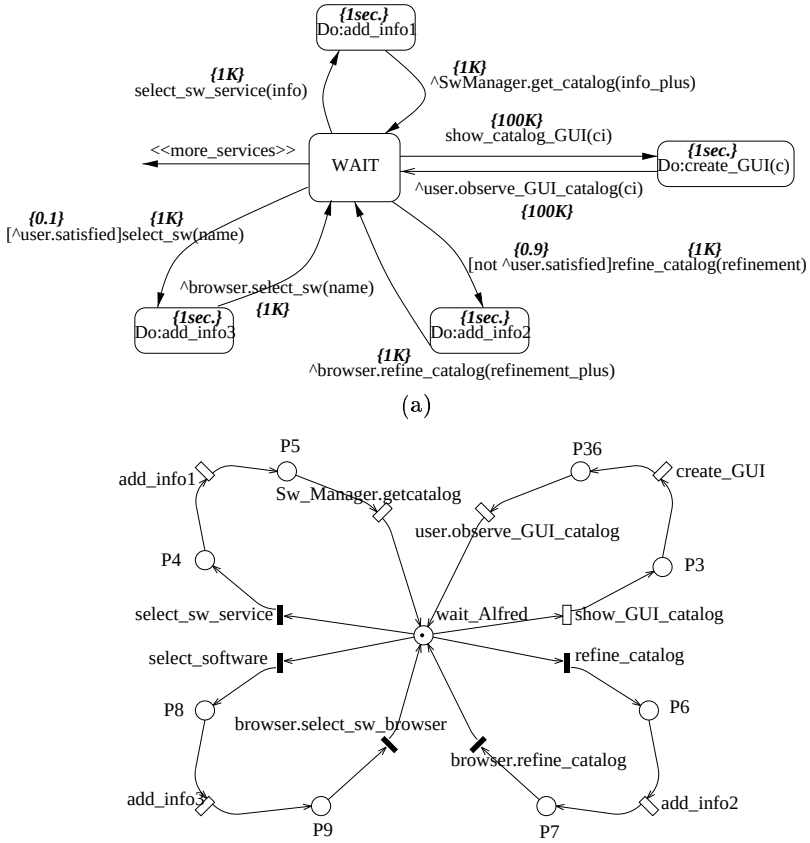


Fig. 4. (a) Statechart for Alfred, (b) SWN for Alfred

Alfred Statechart Alfred is always present in the system, no creation event is relevant for our purposes. Its behaviour is typical for a server object. It waits for an event requesting a service (select_sw_service, show_catalog_GUI, refine_catalog or select_sw). For each request it performs the necessary activities and it returns to its wait state to serve another request. Figure 4(a) shows Alfred's statechart. The stereotyped transition $\ll more\_services \gg$ means that Alfred may attend other services that are not of interest here.

User Statechart In Figure 5(a), the behaviour of a user is represented. The user is in the wait state until s/he activates a select_sw_service event. This event sets the user in the waiting_for_catalog state. The observe_GUI_catalog event, that could be sent by Alfred, allows the user to examine the catalog in order to look

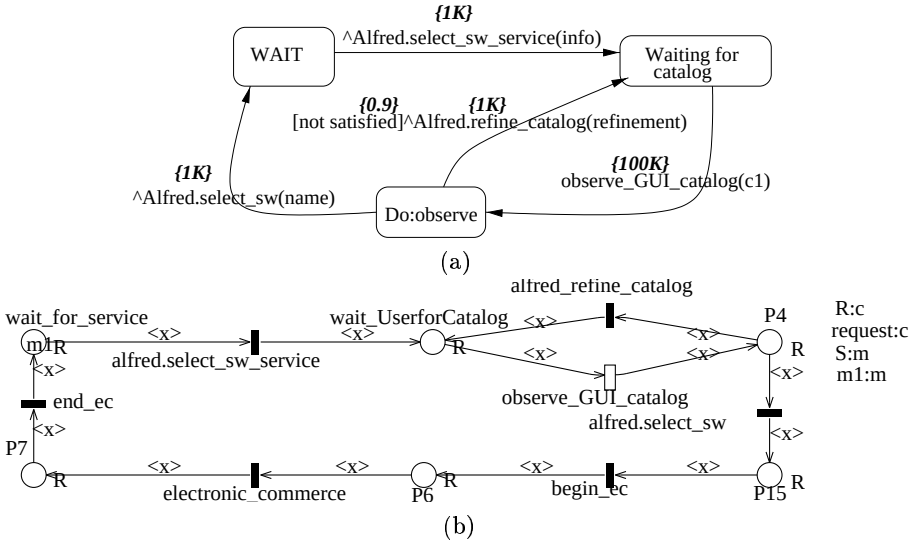


Fig. 5. (a) Statechart for the user, (b) SWN for the user

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

Software Manager Statechart Like Alfred, the Software Manager behaves as a server object. It is waiting for a request event (more_information, get_catalog, request). When one of them arrives, it performs the necessary activities to accomplish it. Figure 6(a) shows its statechart diagram. It is interesting to note the actions performed to respond the get_catalog request: first, the catalog with the available software is created, after that, the browser is created.

Salesman Statechart The Salesman's goal is to give e-commerce services, as we can see in Figure 7(a). 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 case study.

Browser Statechart The statechart diagram in Figure 8(a) describes the Browser's life. It is as follows: once the Browser is created it must go to the MU place, where it invokes Alfred's shows_catalog_GUI method to visualize the previously obtained catalog. At this state it can attend two different events, refine_catalog or select_sw. If the first event occurs there are two different possibilities: first, if the Browser has the necessary knowledge to solve the task, a refinement action is directly performed; second, if it currently has not this knowledge, the Browser must obtain information from the Software Manager,

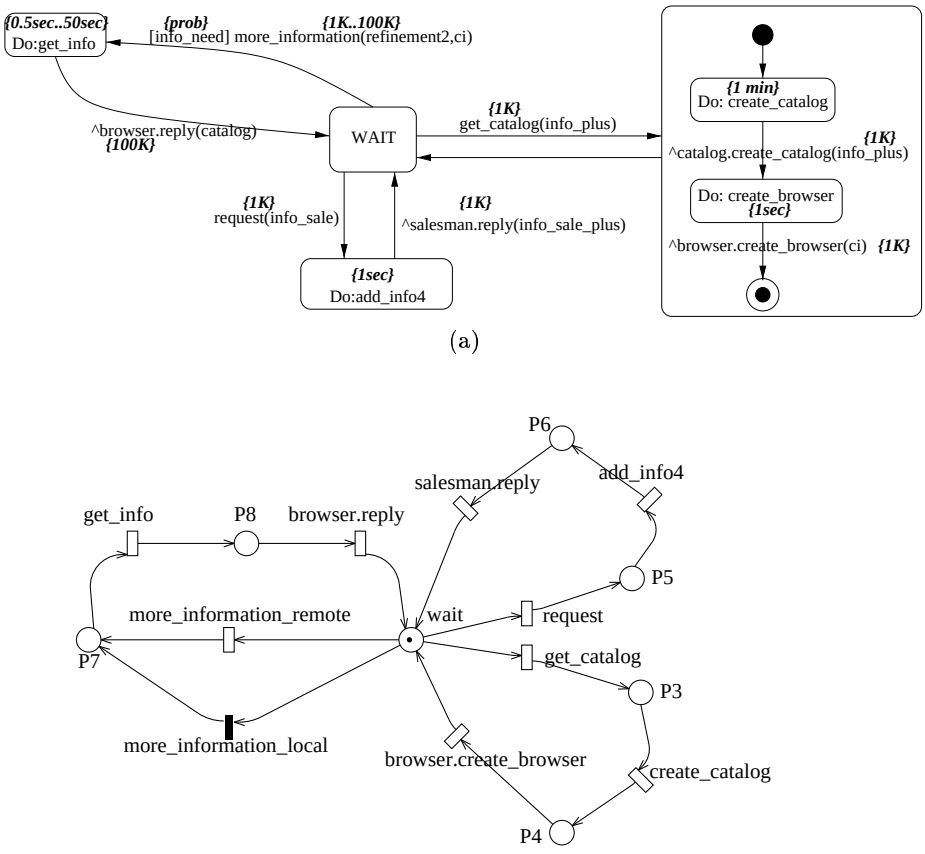


Fig. 6. (a) Statechart for the Software Manager, (b) SWN for the Software Manager

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.

3.2 Modelling Using Stochastic Petri Nets

The UML diagrams previously modelled are expressive enough to accomplish with different implementations. For example, in the system specification we did not specify how many majordomos should attend requests, how many concurrent users can use the system, etc. Several situations can arise, such as:

1. One user request served by one majordomo (no concurrency).
2. Several user requests served by one majordomo.
3. Several user requests served by many majordomos, one per request.

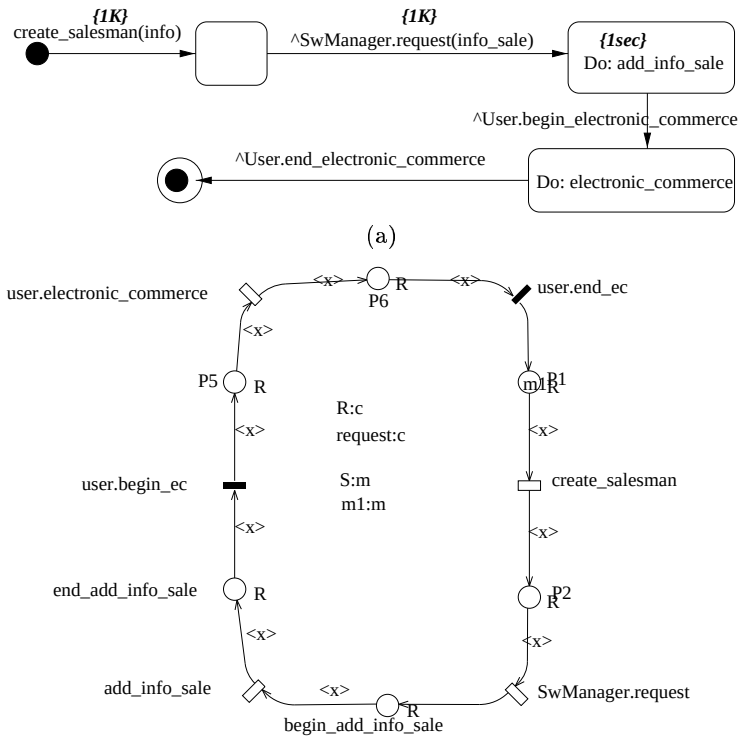


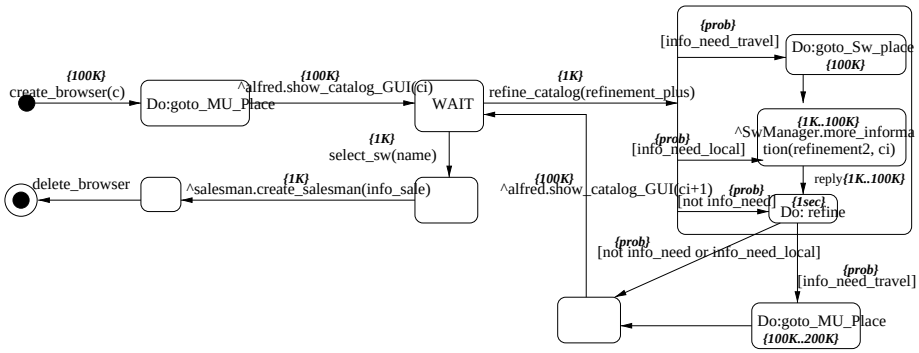
Fig. 7. (a) Statechart for the Salesman, (b) SWN for the Salesman

These kind of questions can be satisfactorily solved using a formal language, such as PNs, to model the system.

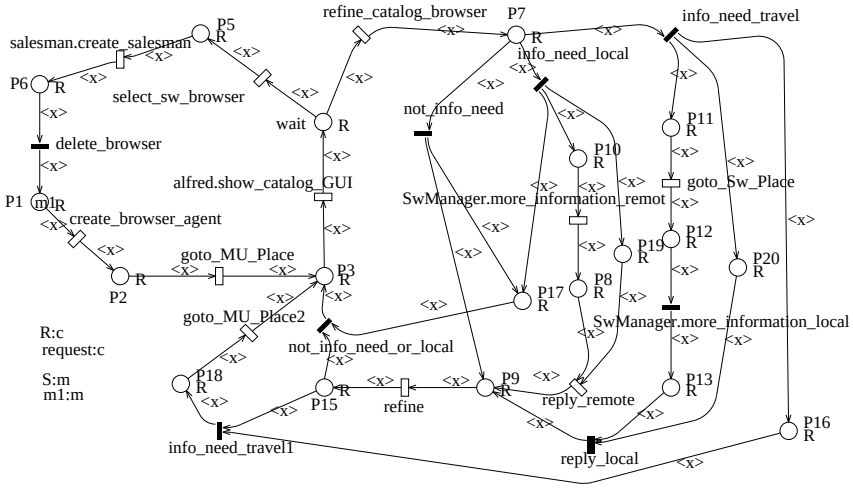
Therefore, once a statechart has been modelled, a PN (concretely a stochastic well formed coloured Petri net (SWN) [5]) is obtained from it (called *component net*). It must be noticed that the statechart and its component net provide the same information. The reasons to maintain both formalisms are given in section 5.

The component net is obtained following the *transformation rules* given in [14]. The most important facts are the following:

- Each token in the PN represents an object.
- Each state of the STD is represented by a place, with the same name, in the PN.
- For each transition in the STD arriving at a state, there will be an input transition in the PN for the place which represents the state. The name of the transition in the PN will be the same as the event that labels the transition in the STD.



(a)



(b)

Fig. 8. (a) Statechart for the Browser, (b) SWN for the Browser

- For each transition in the STD exiting from a state, there will be an output transition in the PN from the place which represents the state. The name of the transition in the PN will be the same as the event that labels the transition in the STD.
- Guards in the STD are translated into a subnet including immediate (firing in zero time) transitions.

The component nets corresponding to Alfred, the user, the Software Manager, the Salesman and the Browser statecharts are shown in Figures 4(b), 5(b), 6(b), 7(b) and 8(b).

From the component nets, and considering the sequence diagram, a *complete* PN for the system is obtained. It represents the behaviour of the whole system.

This net is shown in Figure 9. The complete PN has been obtained by synchronizing the component nets and following the *transformation rules* given in [14]. Moreover, it must be noticed 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. Basically, the transformation rules state that:

- If the message has no wait semantics (half arrowhead in the sequence diagram), then an extra place will appear in the complete net to model a communication buffer between the two component nets.
- If the message has wait semantics (full arrowhead in the sequence diagram), then only one transition appears in the complete net, that represents the fusion of the transitions in the two component nets.

4 Performance Analysis of the SRS

Once the system has been modelled, the performance analysis can be addressed. It is accomplish in this section.

4.1 Performance Assumptions

It will be of interest to study the system *response time*. There are two possible bottlenecks that can decrease the system performance. First, the trips of the Browser from the MU place to the Software place (and way back) in order to obtain new catalogs. Second, the number of user requests for catalog refinements.

In order to develop the performance tests the following realistic scenarios have been considered:

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 new 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 be requested 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, the 30% of the times the Browser performs a RPC, therefore the 70% of the times it travels through the net. Second, the opposite situation, the 70% of the times it performs a RPC, therefore the 30% of the times it travels through the net.

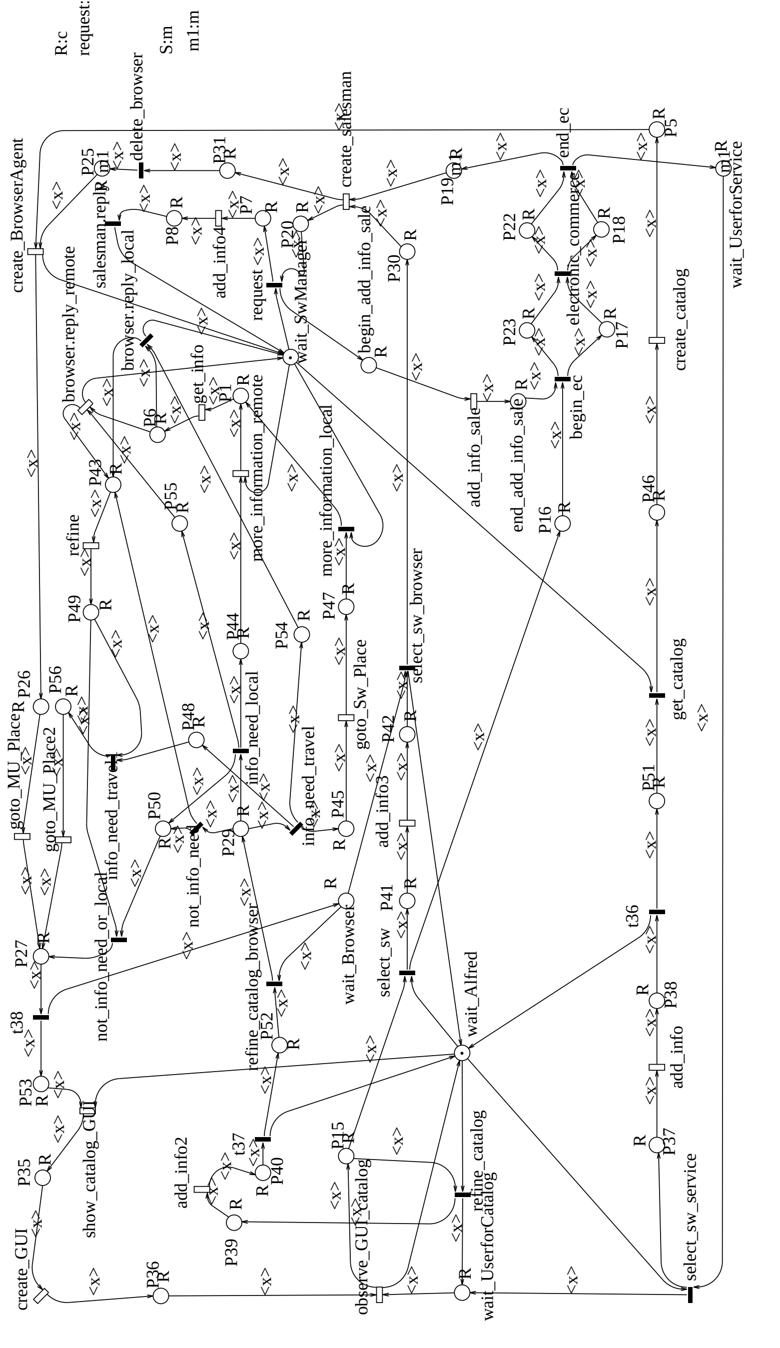


Fig. 9. The Petri net for the whole system

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, until s/he finds the wanted software.
3. *The size of the catalog* obtained by the Browser can 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. With respect to the speed of the net, two cases have been considered: a net speed of 100 Kbytes/sec. ("fast" connection speed) and a net speed of 10 Kbytes/sec. ("slow" connection speed).

4.2 Performance Results

Now, we present the results for two possible cases:

1. One user request served by one majordomo (no concurrency).
2. Several user requests served by one majordomo.

In both cases the interest is to study the system *response time*, as we said previously. The performance figures were obtained by analysing the net in Figure 9 with the tool *GreatSPN* [6]. To obtain the response time, first the throughput of the select_sw_service transition, in the net system, is calculated by computing the steady state distribution of the isomorphic *Continuous Time Markov Chain* (CTMC); finally, the inverse of the previous result gives the system response time.

One user and one majordomo Figure 10(a) shows the system response time (in minutes) assuming "fast" connection speed, an "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.

Figure 10(b) shows system response time, supposing "fast connection", "intelligent Browser" and "naive user". The two solutions still remain almost identical.

Someone could suspect that there exist small differences because of the net speed. So, the net speed is decreased to 10 Kbytes/sec. In Figures 10(c) and 10(d) it can be seen how the differences still remain non significant for a faster net speed.

Several user requests served by one majordomo Figure 11 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 tested the system for a different number of requests ranging from 1 to 4. Observe that when the number of requests is increased, the response time for each request increases, i.e., tasks cannot execute

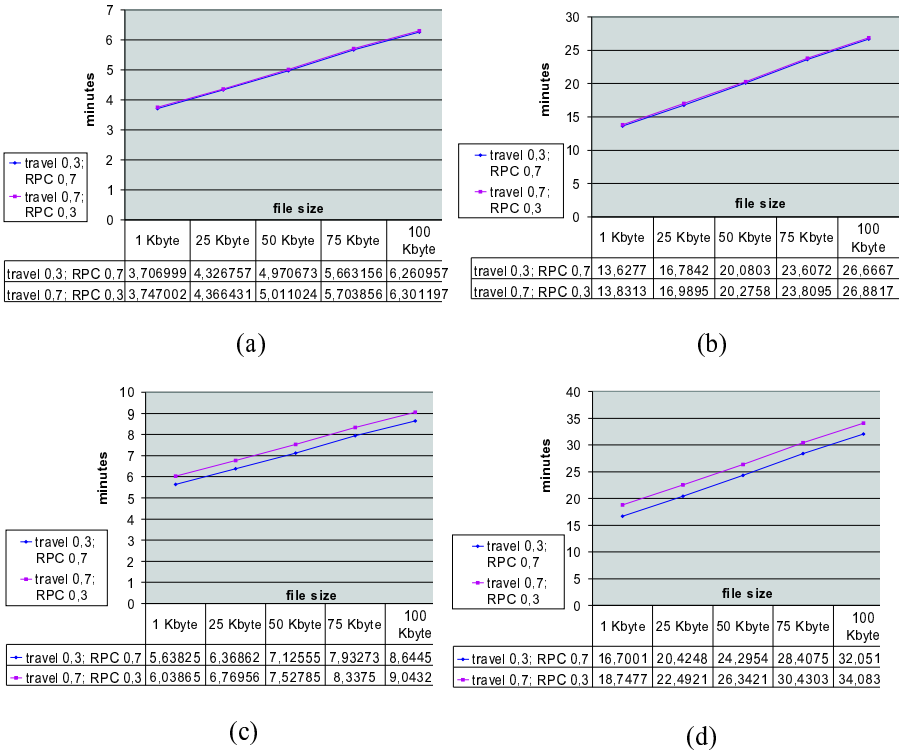


Fig. 10. Response time for different scenarios with an "intelligent Browser". (a) and (b) represent a "fast" connection speed, (c) and (d) a "slow" connection speed; (a) and (c) an "expert user", and (b) and (d) a "naive user"

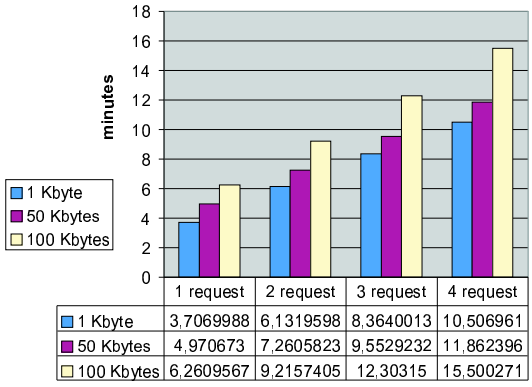


Fig. 11. Response time for an "intelligent Browser", an "expert user", a "fast" connection and different number of requests (1-4) and catalog size (1K, 50K, 100K)

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, and the impact of such bottlenecks can be evaluated using our approach.

5 Concluding Remarks

Along the Performance Engineering process several choices must be done. In the following, we stress some of them:

Why to use UML to model system requirements? Fundamentally because in the last years UML has become the standard notation to model software systems, widely accepted by the software engineering community. Unfortunately, UML lacks of the necessary expressiveness to accurately describe performance features. There have been several approaches to solve this lack [19, 20, 17, 14]. In this chapter the last one has been followed.

Why to use two modelling languages –UML and PNs– during the performance process? The reasons to maintain both formalisms are given by the advantages and disadvantages they provide.

A disadvantage of PNs is that they do not show the system load (message size, guards probability, activities duration) as clear as UML diagrams do, this information is hidden in the definition of the net transitions. Besides, much work has been done in the software engineering area in developing methodologies [18, 9] for object oriented design, from which UML take profit and PNs do not.

PNs have advantage over UML because they can be used to obtain performance figures, due to the underlying mathematical model; even more, there exist software tools that automatically obtain them [6, 21]. Moreover, PNs express concurrency unambiguously, UML do not.

Finally, it can be said that the UML diagrams are considered in this chapter as the documentation part of the system, being useful for the system analyst to express in a easy way the system requirements, including performance requirements. And PNs are considered as the mathematical tool which represent the performance model of the system.

Why to use PNs to represent the performance model instead of other mathematical formalism? As an alternative to stochastic Petri nets (SPNs), several performance-oriented formalisms can be considered as underlying mathematical tools for computing performance indices of interest. Among them, Markov chains (MCs) [7], queuing networks paradigm (QNs) [10], and stochastic process algebras (SPAs) [8]. For all of them, translation algorithms can be devised from the time-annotated UML diagrams using the general rules exposed in [14].

The main drawback of plain MCs is their poor abstraction level (each node of the underlying graph model represents a state of the system). Concerning SPAs, even if they are compositional by nature, thus specially adequated for the modelling of modular systems, the present lack of efficient analysis algorithms and software tools for performance evaluation would make them difficult to use in real applications. The use of SPNs rather than QNs models is justified because SPNs include an explicit primitive for the modelling of synchronization mechanism (a *synchronizing transition*) therefore they are specially adequated for the modelling of distributed software design. Even more, a vast amount of literature exists concerning the use of PNs for both validation of logical properties of the system (e.g., liveness or boundedness) and quantitative analysis (performance evaluation).

References

[1] *CNET Inc., 1999. http://www.download.com.*

[2] *CNET Inc., 1999. http://www.gamecenter.com.*

[3] *Tucows.com inc., 1999. http://www.tucows.com.*

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

[5] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad, *Stochastic well-formed coloured nets for symmetric modelling applications*, IEEE Transactions on Computers **42** (1993), no. 11, 1343–1360.

[6] G. Chiola, G. Franceschinis, R. Gaeta, and M. Ribaudo, *GreatSPN 1.7: GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets*, Performance Evaluation **24** (1995), 47–68.

[7] E. Cinlar, *Introduction to stochastic processes*, Prentice-Hall, Englewood Cliffs, NJ, 1975.

[8] H. Hermanns, U. Herzog, and V. Mertsiotakis, *Stochastic process algebras as a tool for performance and dependability modelling*, Proceedings of IEEE International Computer Performance and Dependability Symposium, IEEE CS-Press, April 1995, pp. 102–113.

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

[10] K. Kant, *Introduction to computer system performance evaluation*, Mc Graw-Hill, 1992.

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

[12] E. Mena, A. Illarramendi, and A. Goñi, *Automatic ontology construction for a multiagent-based software gathering service*, Proceedings of the Fourth International ICMAS'2000 Workshop on Cooperative Information Agents (CIA'2000), Springer series of Lecture Notes on Artificial Intelligence (LNAI), Boston (USA), July 2000.

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

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

[15] D. Milojicic, M. Breugst, I. Busse, J. Campbell, S. Covaci, B. Friedman, K. Kosaka, D. Lange, K. Ono, M. Oshima, C. Tham, S. Virdhagriswaran, and J. White, *MASIF, the OMG mobile agent system interoperability facility*, Proceedings of Mobile Agents '98, September 1998.

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

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

[18] J. Rumbaugh, M. Blaha, W. Premerlani, E. Frederick, and W. Lorensen, *Object oriented modeling and design*, Prentice-Hall, 1991.

[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.

[21] A. Zimmermann, J. Freiheit, R. German, and G Hommel, *Petri Net Modelling and Performability Evaluation with TimeNET 3.0*, Proceedings of the 11th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation, Lecture Notes in Computer Science, Vol. 1786, Springer, 2000, pp. 188–202.