Performance Analysis of Mobile Agents Tracking[*]

Elena Gómez-Martínez, Sergio Ilarri, José Merseguer
Dpto. de Inform´atica e Ingenier´ıa de Sistemas, Universidad de Zaragoza
Zaragoza, Spain
e-mail{megomez,silarri,jmerse}@unizar.es

ABSTRACT

Mobile agents have arisen as an interesting paradigm to build distributed applications, due to the unparalleled advantages they offer. However, along with the advantages they also present new challenges. One of the most relevant is that it is not easy to ensure efficient communication among agents that move continually from one computer to another.

In this paper, we apply SPE techniques to model and analyze how a mobile agent tracking approach addresses the highly dynamic movement problem in a distributed computing environment.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Measurement techniques; C.4 [**Performance of Systems**]: Modelling techniques; D.2.1 [**Software Engineering**]: Requirement/Specification; D.2.2 [**Software Engineering**]: Petri nets; D.2.8 [**Software Engineering**]: Metrics

General Terms

Performance, Design, Experimentation

Keywords

Mobile agents tracking, Software Performance Engineering (SPE), PUMA, Generalized Stochastic Petri Nets (GSPN)

1. INTRODUCTION

Mobile agents [14] have stirred up a lot of interest and research efforts. They are programs that can autonomously travel from computer to computer, and present a range of unique advantages, such as autonomy, flexibility, and effective usage of network bandwidth [11]. Due to their features, the use of mobile agent technology is very attractive in wireless [22], pervasive, and distributed computing in general.

One of the main challenges in applications based on mobile agents is how to keep track of the current locations of the agents in order to allow an efficient communication among them. If the location of an agent cannot be obtained in a short time, the agent can move to another computer before such location data is used for communication purposes; this situation may occur indefinitely, leading to *livelock* problems from the point of view of the agents that want to communicate with the agent.

The vital importance of designing efficient communication and tracking schemes for mobile agents have been highlighted in many works, such as [10, 3]. Moreover, according to the experiments in [9], this is a key issue to ensure the scalability of a mobile agent platform, especially in highly dynamic contexts.

Several models for tracking agents are conceivable; thus, the work in [3] suggests three methods to locate agents (brute force, logging, and redirection), and in [15] were proposed four (updating at the home node, registering, searching, and forwarding). A new mobile agent platform, called SPRINGS [9] (Scalable PlatfoRm for movING Software) proposes a new tracking approach; in [9] has been experimentally shown to be highly scalable and how it outperforms other popular platforms, especially in environments with a high number of mobile agents.

In this paper, we analyze the performance of the SPRINGS tracking approach. Instead of an experimental approach, we follow the SPE principles [21] and concretely the model-based PUMA approach developed in [23]. Our analysis allows us to validate the experimental results previously obtained [9], and to evaluate the platform in a variety of other hypothetical situations without the burden of real experimentation.

The structure of this paper is as follows. In Section 2, we introduce basic aspects of mobile agent technology, which is important for this work. In Section 3, we describe and model the SPRINGS architecture for tracking mobile agents. In Section 4, the latter model is annotated with performance information. In Section 5, the PUMA approach is applied in order to get the performance models corresponding to the modeled architecture. Section 6 exploits the performance models to deal with the proposed analysis goals. Section 7 revises the related literature. Finally, in Section 8 conclusions are given.

[*]This work was supported by the projects DPI2006-15390 and TIN2004-07999-C02-02 of the Spanish Ministry of Science and Technology and IBE2005-TEC-10 of the University of Zaragoza.

2. MOBILE AGENT TECHNOLOGY

In the traditional *client/server* architecture, a server at a certain computer offers a set of services to interested parties. Then, three steps take place: 1) a client located at another computer requests the execution of a service by interacting with the server, 2) the server performs the requested service, and 3) the server returns the result to the client. As opposed to this classical approach, a mobile agent [14] is a software component that can move autonomously among computers, and so it can decide itself when and where to move in order to perform its tasks.

Thanks to their mobility, mobile agents offer many interesting benefits [11]. For example, in a distributed information system a mobile agent can travel where the data are stored and process them locally, avoiding the need to communicate all the data over the network. Furthermore, in certain contexts they also exhibit a *good performance* compared with the traditional client/server approach [22, 12].

So how can an agent move to another computer and resume its execution there? Mobile agents need a specific execution environment, which we call *context*[1]. Thus, for an agent to travel to another computer, a context must be available there: an agent needs a context in the same way that a web page request needs a web server. Contexts are provided by a specific mobile agent platform [20], from which several alternatives are available (e.g., Aglets, Grasshopper or SPRINGS), and provides them with different services, such as *communication and mobility.* The two mentioned services are interrelated. Particularly, mobile agents must be able to communicate among themselves, via remote method invocation or message passing, even if they move across computers.

2.1 Contexts and Regions

The architecture of an agent platform is usually made of agents, contexts and regions, see Figure 1.

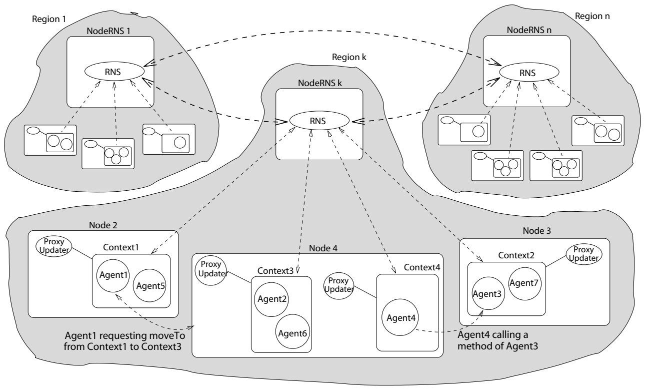


Figure 1: Architecture for mobile agent platforms.

- *Contexts* (called *places* in Grasshopper) are the environment where agents execute: a computer can host several contexts, each one assigned to a different communication port and execution process. A context provides agents with services such as a *call routing service* irrespective of the target agents' locations[2], and a *transportation service* to move to other contexts.

- A *region* is a set of related contexts. In SPRINGS, for example, the functionality of a region is provided through a remote object called Region Name Server (RNS), which can be located on any computer in the network. An RNS has several functions, such as ensuring the uniqueness of agent/context names, mapping from context names to context addresses, and assigning tracking responsibilities to contexts.

[1]In the literature of mobile agents, the term *place* is frequently used instead.
[2]Not all the existing platforms feature this property.

3. MODELING TRACKING

A key functionality of a mobile agent platform is to offer a communication service that allows an agent to communicate with another without the need of knowing its current location. Most agent platforms use the idea of *proxy* as an abstraction to communicate with an agent (if an agent $a1$ wants to communicate with another agent $a2$, it must first obtain a proxy to $a2$); *location transparency* means that the proxy routes the message to its corresponding agent efficiently, wherever it is. SPRINGS hides the proxies to the programmer and stores them in the contexts[3]: an agent can communicate with another one by just specifying the name of the target agent, without the need of using proxies explicitly.

An agent proxy stores the (remote) reference to that agent: its name and current context. If the agent moves to another context, the information contained in the proxy becomes invalid. In SPRINGS, *dynamic proxies* are considered: when an agent arrives at a new context, the *remote proxies* to that agent (i.e., held by other contexts) are updated to reflect the new agent location. These proxies are updated before resuming the agent's execution in order to maximize the probability that another interested agent succeeds in communicating with it. In each context, a *Proxy Updater* thread is in charge of updating the remote proxies to the incoming agents efficiently (see [9] for more details).

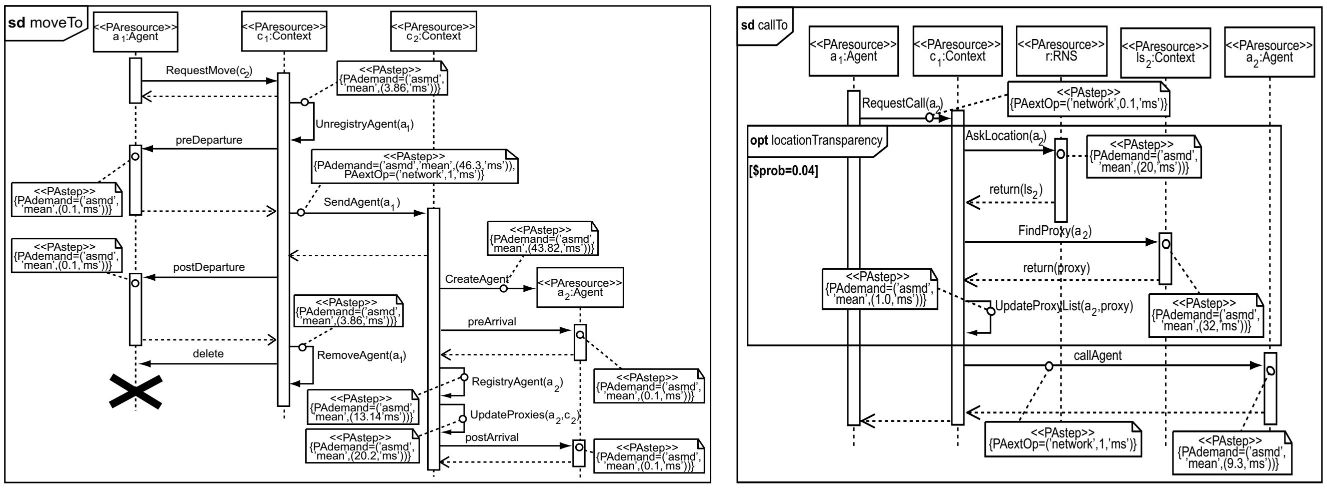
Regarding agent mobility, two important related concepts are considered:

- *Location servers*. A location server of an agent a is a context that stores a dynamic proxy to a because it has been assigned by the RNS to do so.

- *Observer contexts*. A context c is an *observer* of a certain agent a when it is interested in knowing the current location of a, which happens when: 1) a local agent has communicated recently with a, which means to include a in its *ProxyList*, or 2) it is a location server for a. An observer of an agent a always stores a dynamic proxy to the agent.

Following the concepts given so far, the sequence diagram in Figure 2(a) describes the scenario of how an agent changes its context and how the platform keeps track of it. Firstly, an agent a_1 requests to its current context c_1 to travel to a new one c_2. The origin context unregistries the agent. Just before traveling, the agent prepares its departure and when it has finished, the current context sends it to c_2. When it arrives, a new instance a_2 of the agent is created; meanwhile the old instance a_1 at origin ends its departure and it is completely removed from the origin context. Assuming

[3]All the agents in a context that want to communicate with another one use a shared proxy that points to that agent.


sd moveTo
<<PAresource>> a1:Agent
<<PAresource>> c1:Context
<<PAresource>> c2:Context
RequestMove(c2)
<<PAstep>> {PAdemand=('asmd', 'mean',(3.86,'ms'))}
UnregistryAgent(a1)
preDeparture
<<PAstep>> {PAdemand=('asmd','mean',(46.3,'ms')), PAextOp=('network',1,'ms')}
<<PAstep>> {PAdemand=('asmd', 'mean',(0.1,'ms'))}
SendAgent(a1)
<<PAstep>> {PAdemand=('asmd', 'mean',(43.82,'ms'))}
postDeparture
CreateAgent
<<PAresource>> a2:Agent
<<PAstep>> {PAdemand=('asmd', 'mean',(3.86,'ms'))}
preArrival
RemoveAgent(a1)
delete
RegistryAgent(a2)
<<PAstep>> {PAdemand=('asmd', 'mean',(13.14'ms'))}
UpdateProxies(a2,c2)
<<PAstep>> {PAdemand=('asmd', 'mean',(20.2,'ms'))}
postArrival
sd callTo
<<PAresource>> a1:Agent
<<PAresource>> c1:Context
<<PAresource>> r:RNS
<<PAresource>> ls2:Context
<<PAresource>> a2:Agent
RequestCall(a2)
<<PAstep>> {PAextOp=('network',0.1,'ms')}
opt locationTransparency
[$prob=0.04]
AskLocation(a2)
<<PAstep>> {PAdemand=('asmd', 'mean',(20,'ms'))}
return(ls2)
FindProxy(a2)
return(proxy)
<<PAstep>> {PAdemand=('asmd', 'mean',(1.0,'ms'))}
UpdateProxyList(a2,proxy)
<<PAstep>> {PAdemand=('asmd', 'mean',(32,'ms'))}
callAgent
<<PAstep>> {PAextOp=('network',1,'ms')}
<<PAstep>> {PAdemand=('asmd', 'mean',(9.3,'ms'))}

(a) For agents movement. (b) For agents communication.

Figure 2: Annotated sequence diagrams.

that the agent has successfully arrived, it prepares its arrival and then it is registered. Afterwards, its proxies have to be updated, so the *Proxy Updater* thread, embedded in the Context component, informs all observer contexts of the agent to update its dynamic proxy. After this, the arrival has finished.

The sequence diagram in Figure 2(b) models how agents communicate. Let us assume that an agent a_1 executing on context c_1 wants to communicate with another agent a_2 on context c_2. If any agent on c_1 has recently communicated with a_2, a dynamic proxy to a_2 will be locally available (c_1 is an *observer* of a_2); in this case, the *callAgent* will be directly routed through that proxy, without executing the *locationTransparency* fragment. Otherwise, c_1 must find a_2 in the following way: 1) c_1 obtains from its RNS a location server for a_2; 2) c_1 obtains a proxy to a_2 from that *location server* (ls_2), which registers c_1 as a new *observer* for a_2 (*UpdateProxyList* task); and 3) the retrieved proxy is used to route the call to a_2, and it is stored by c_1 (so the RNS will not need to be contacted the next time).

Information about agents not located in the region would be requested by the local RNS to other RNS. In that case, the sequence diagram in Figure 2(b) must be augmented with a lifeline representing the RNS which knows a location server for the called agent.

Finally, the modeling of the physical structure in Figure 1 is provided using a UML deployment diagram (DD), see Figure 3, which describes resources on the system connected through a network. The DD depicts the system architecture for one only region; new regions with their contexts can be incorporated by duplicating this structure. Since a node may host several contexts and each of them provides services to a number of agents, there may exist several instances of both agents and contexts on each node.

4. SYSTEM PERFORMANCE VIEW

In this section, we present the performance view of the proposed mobile agents tracking approach. We use the UML Profile for Schedulability, Performance and Time Specification (UML-SPT) [17] to annotate both the performance metrics, that characterize the goals of our performance analysis, and the performance parameters of the system.

4.1 Performance metrics

The first analysis goal is to validate the analytical results obtained from the UML-SPT models against those experimentally obtained in [9]. The valid performance models will be used to determine the *platform optimal configuration*. Finally, this configuration will allow to perform a sensitivity analysis, i.e., to study system response time when the agents size increases or the system is running on slow networks.

The experiments in [9] present a configuration composed of a single region with 5 contexts residing on 5 computers, one of them executing also the RNS and a variable number of agents ranging from 1 to 1500, each one assigned to a context thread. These five computers were Pentium IV 1.7 GHz with Linux RedHat 2.4.18 and 256 MBytes RAM.

The analytical experiments, Section 6, consider the same number of agents and regions as in [9], but a variable number of context threads running on separate processors, as expressed in the DD annotations, see Fig. 3.

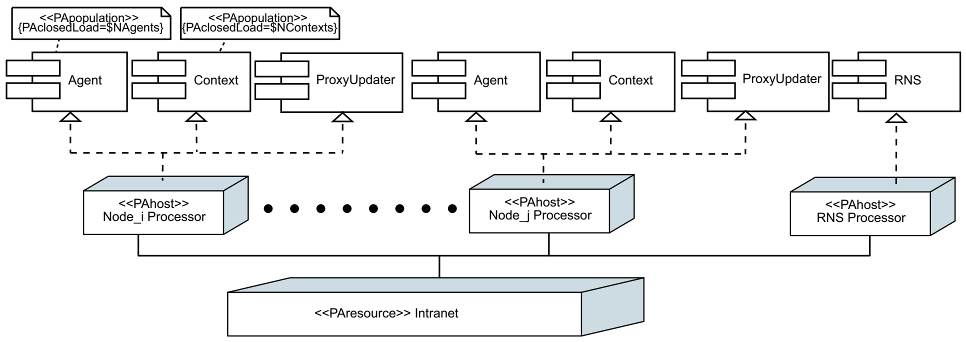
The interaction overview diagram (IOD) in Figure 4 depicts the performance scenario, where the analysis goals will be studied. Considering this IOD, an agent will change its current context (moveTo) and immediately will perform a method invocation to another agent (callTo).

The proposed analysis goals will be studied using as performance metric the one defined in the IOD, i.e. the *scenario response time*.

4.2 Performance parameters

The performance information concerning the actions duration and the messages delay has been taken from [9], they correspond to the experiment described in Section 4.1 when only one agent was executing the platform.

The actions are represented by the stereotype <<PAstep>>, where the *PAdemand* tag specifies its corresponding execution or delay time as an exponentially distributed random variable. Table 1 summarizes the mean execution times that have been annotated along the sequence diagrams. By mean execution time we mean that



<<PApopulation>> {PAclosedLoad=$NAgents}
<<PApopulation>> {PAclosedLoad=$NContexts}
Agent
Context
ProxyUpdater
Agent
Context
ProxyUpdater
RNS
<<PAhost>> Node_i Processor
<<PAhost>> Node_j Processor
<<PAhost>> RNS Processor
<<PAresource>> Intranet

Figure 3: Deployment diagram of the SPRINGS architecture.

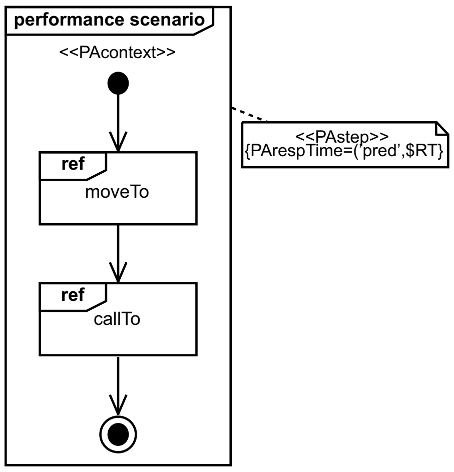


Figure 4: IOD of the performance scenario.

Operation	Mean Execution Time (ms)
preDeparture	0.1
postDeparture	0.1
preArrival	0.1
postArrival	0.1
CreateAgent	43.82
RemoveAgent	3.86
SendAgent	46.3
RegistryAgent	13.14
UnregistryAgent	3.86
UpdateProxies	20.2
RequestCall	0.1
CallAgent	9.3
AskLocation	20
FindProxy	32
UpdateProxyList	1.0

Table 1: System basic operations.

these processing times have been measured by running the system repeating 50 iterations per agent. Experiments in [9] were developed in this way to ensure accuracy in the experimental tests.

Another parameter that may impact the system performance is the probability of executing the optional fragment *LocationTransparency* in Fig. 2(b). Values close to 0 mean that the *ProxyList* of context c_1 owns knowledge enough to solve most of the RequestCall messages. Then, values close to 1 are supposed to penalize system performance.

The network is indirectly specified by the *PAextOp* tagged value, see SendAgent message in the sequence diagram of Fig. 2(a). The <<PAresource>> stereotype annotated in the lifeline of each object defines them as software components.

5. APPLYING THE PUMA APPROACH

Once, the performance scenario and its performance parameters have been defined, we apply the PUMA approach to get the performance models where to evaluate the proposed performance metric.

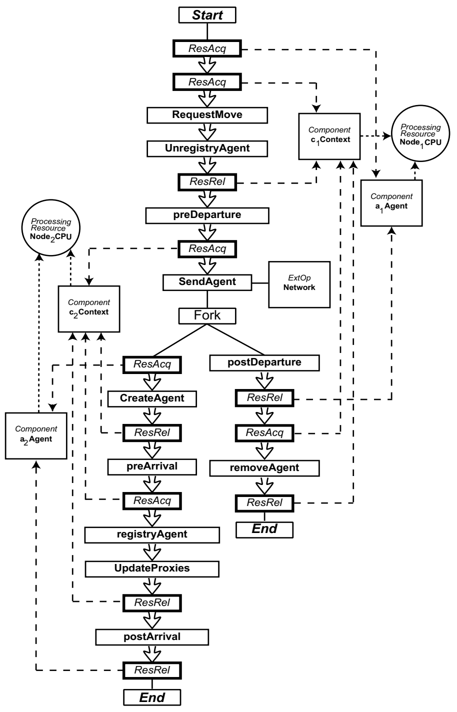
PUMA [23] is a framework that aims at extracting from a design model (UML or Use Case Maps) an intermediate model, called Core Scenario Model (CSM) [18]. PUMA describes how to translate the CSM into a target performance model, such as (layered) queueing networks or stochastic Petri nets. Concretely, we will use the translation given to obtain a Generalized Stochastic Petri Net (GSPN) [2].

5.1 Building the CSMs

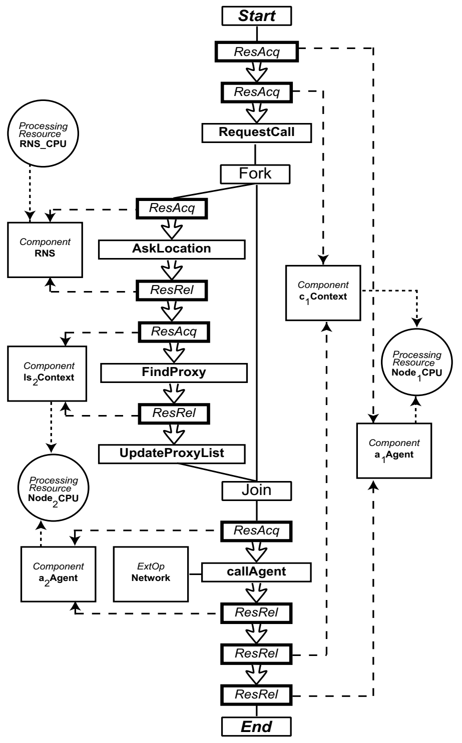
The CSM is focussed on describing performance *Scenarios*. A scenario, is a sequence of *Steps*, linked by *Connectors*. A step is a sequential piece of execution. Connectors can include branches, merges, and forks and joins. The scenario has a *Start* and an *End* points, where it begins and finishes. Start points are associated with *Workload*, which defines arrivals and customers, and may be open or closed. There exist two kind of *Resources*: *Active*, which execute steps, and *Passive*, which are acquired and released during scenarios by special *ResAcquire* and *ResRelease* steps. Steps are executed by (software) *Components* which are passive resources. A primitive step has a single host processor, which is connected through its component.

According to PUMA, each sequence diagram in the IOD of Fig. 4 has been translated into a CSM scenario. So, Figs. 5(a) and 5(b) illustrate the CSMs that represent the UML sequence diagrams in Figs. 2(a) and 2(b).

In order to make clear how PUMA proposes to generate the CSMs, a piece of execution is explained. See the RequestMove message in the sequence diagram of Fig. 2(a), it is straightforward to check that it has its corresponding step in the CSM of Fig. 5(a). Furthermore, before executing it,



(a) For moving an agent among contexts.



(b) For the communication between two agents.

Figure 5: Core Scenario Models.

it is necessary to acquire the agent a_1 and the context c_1 software components, which run on the CPU $Node_1$.

Each element in a CSM (e.g., steps, components or resources) has attributes concerning the performance information annotated in the sequence diagram. For example, the RequestMove step has a demand attribute, its value is taken from the <<PAstep>> annotation in the sequence diagram. However, we have not shown these performance attributes in our CSM scenarios due to lack of space, but they will be used when parameterizing the performance model.

5.2 Building the performance models

The next step is to translate the CSMs into GSPNs following the translation process given by PUMA. Figs. 6 and 7 depict the GSPNs that represent the CSMs in Figs. 5(a) and 5(b).

Just to outline the translation, see the UnregistryAgent step in the CSM of Fig. 5(a), it is mapped into a timed transition, see Fig. 6, being its delay defined as the *demand* attribute of the step. Previously, the agent a_1 and the context c_1 have been acquired, see transition $t\_c_1\_1$. Places representing resources, such as CPUs or software components, are marked with the amount of tokens specified by the corresponding *PAclosedLoad* tag.

Finally, the GSPNs in Figs. 6 and 7 are composed in order to obtain a performance model, i.e., a new GSPN that models the performance scenario in Fig. 4. GSPN composition is based on merging the net places that represent common elements in the CSMs, such as resources or components.

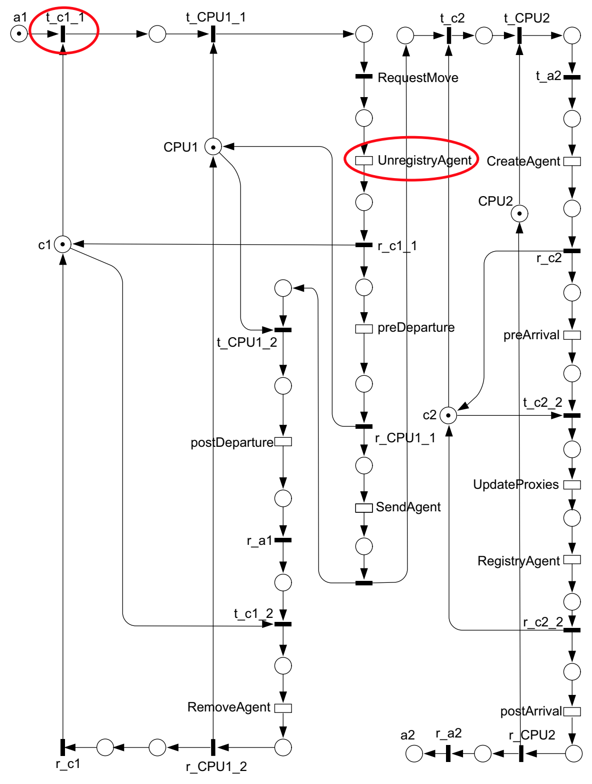


Figure 6: GSPN for agents movement.

6. PERFORMANCE RESULTS

Once the performance model has been built, we use TimeNET [1] to compute the given metric in it by means of simulation techniques. Then, in the next sections we accomplish the analysis goals proposed in Section 4.1.

6.1 Validation of experimental results

Fig. 8 depicts the response times given by the experimental test in [9] and the performance model. The configuration for each one was described in Section 4.1. As it can be observed, the results are very similar, and in both cases the response time increases linearly.

In [9], linear scalability was considered acceptable, since the experiments were carried out to test the platform in a stressful scenario for highly mobile agents [16]. Moreover, for a platform to support 1500 agents in such scenario is a real challenge. Mobile agents platforms can get lower response times when agents do not move and communicate so frequently. In the following we describe the stressful scenario.

The goal is to make agents stay on a context for a very short time and continually calling among themselves. To do so, in the experimental configuration described in Section 4.1, each agent has a communication *peer* and performs the following steps: 1) calls its peer; 2) moves randomly to another context; and 3) steps 1 and 2 are repeated without delay until reaching 50 iterations. While the number of agents increases up to 1500, the performance of agent communication decreases; thus, a target agent could move to another context before a message reaches it. So, the *scalability* of the platform is studied in terms of the time that an agent needs to perform one iteration (callTo + moveTo) as the number of agents increases.

6.2 Optimal configuration and sensitivity analysis

Optimal configuration for a platform means to minimize the number of context threads needed to execute the agents while keeping the response time.

The analytical experiment in Fig. 8 was very relaxed in this sense and it considered up to 1500 threads, i.e., a thread per agent. However, in a new experiment, depicted in Fig. 9, we drastically reduced the number of threads, in a range from 50 to only 1, while keeping the rest of the parameters. The response time grows exponentially with values under 10 threads. Finally, Fig. 10 enlarges Fig. 9 in the range from 50 to 15 threads. Now, we can observe that the response times obtained in Fig. 8 are preserved only for the range from 50 to 40 threads. Therefore, 40 threads is the optimal configuration for the agent platform, beacause values lower than 40 cause response times greater than 6sec. with 1500 agents.

The network speed may also affect the performance of the platform, influencing the time spent by agents traveling to another context, which is captured by the SendAgent task.

Using the optimal configuration with 1500 agents and considering that the SendAgent message size is 2KBytes, Fig. 11 illustrates the system response time when the delay of the SendAgent message varies. Be aware that a send delay of 10 ms corresponds with a network speed of 200 KBytes per second, while 250 ms corresponds with 8 KBytes per second. The system is sensitive to the network speed; although from 33 KBytes per second (60 ms), it does not perform better.

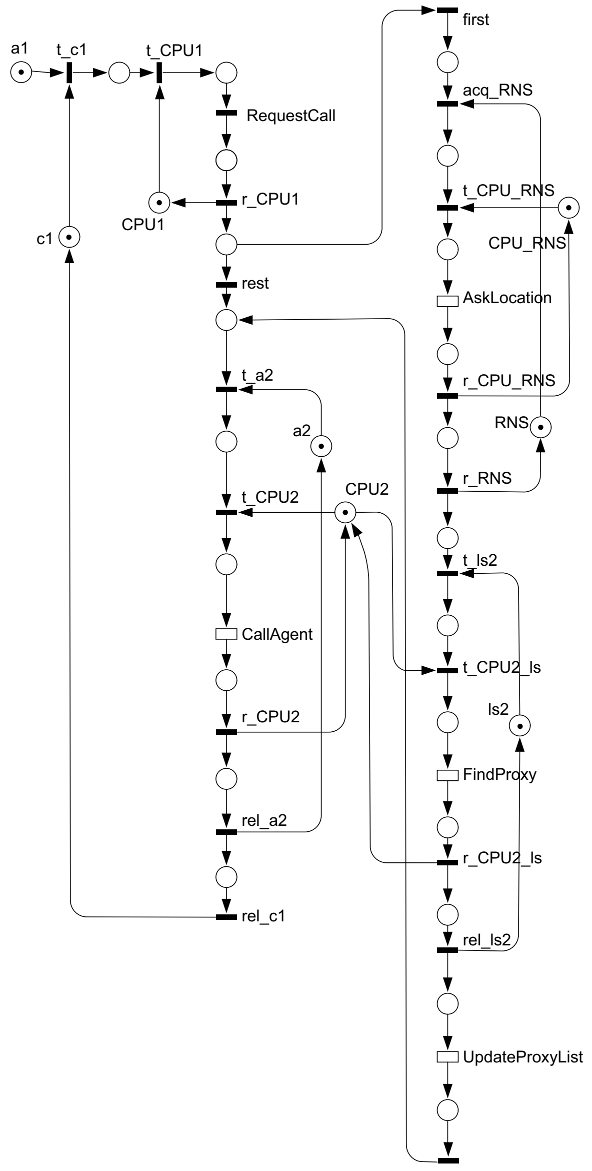


Figure 7: GSPN for agents communication.

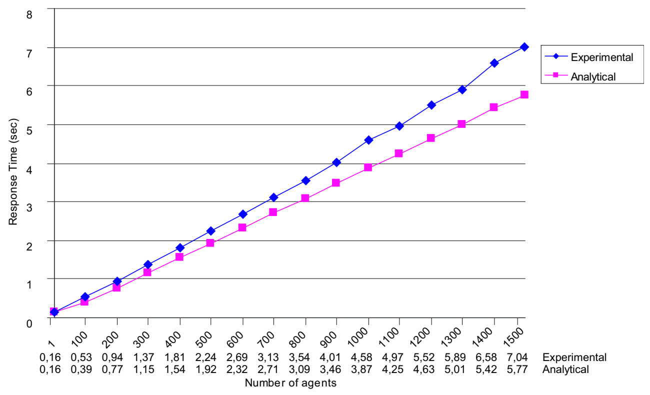


Figure 8: System response times.

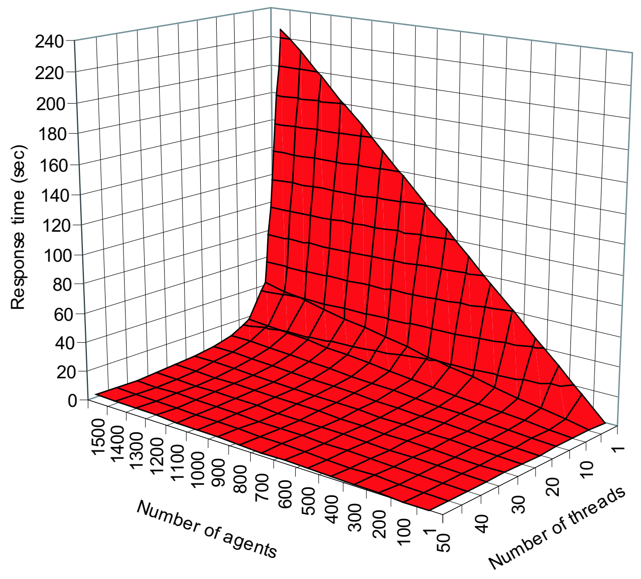


Figure 9: Response times when multithreading contexts.

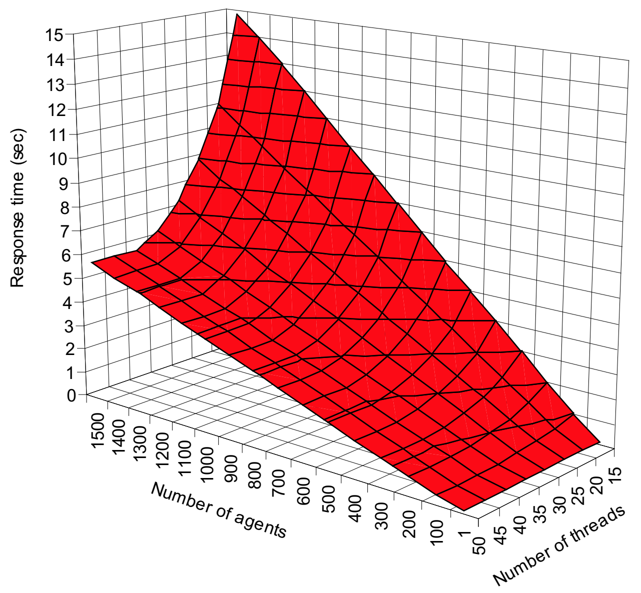


Figure 10: Detail of Fig. 9.

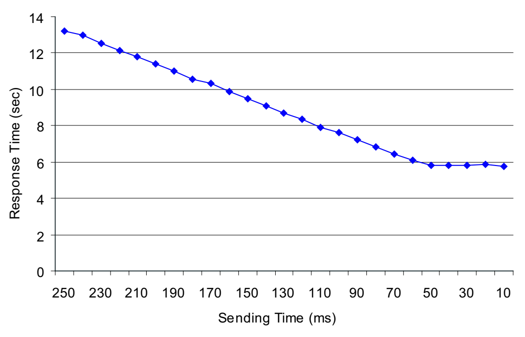


Figure 11: Sensitivity analysis.

7. RELATED WORK

To the best of our knowledge, there is no other work that analyzes the performance of mobile agent tracking strategies using a model-based approach; an experimental analysis can be found in [9]. In the following, we study the most relevant works related with the modeling of mobility with performance analysis purposes.

Currently, there is no standard way for modeling mobility, although different proposals exist, some of them based on UML [5, 4, 7, 6], while others such as [8] follow some formalism, in this case PEPA nets.

The work in [5] presents an extension of the UML class, sequence and activity diagrams to model mobile systems and performance and security characteristics, but they do not explain how to get any performance model or how to compute metrics. In [7], a non-standard UML profile for modeling mobile systems using activity, deployment and state machine diagrams is proposed, as well as an extension for collecting performance information according to the UML-SPT. The models are automatically translated into queueing networks to analyze performance. [4] describes a UML-based methodology for modeling and evaluating the performance of mobile systems using use case, activity and deployment diagrams augmented with the UML-SPT. They use simulation techniques to compute metrics. [6] proposes a framework to model performability for mobile software systems using use case, sequence, collaboration and deployment diagrams, from which Stochastic Activity Networks (SANs) are obtained.

Finally, it is worth noticing some works that use Petri nets for mobility and performance. In [19], the performance of different communication paradigms is compared using stochastic Petri nets. [13] compares the performance of two software retrieval systems applying SPE techniques using high-level Petri nets.

8. CONCLUSION

In this paper we have analyzed the performance of the SPRINGS tracking approach.

The most interesting conclusion for SPE is that it has been possible to analyze one of the key aspects concerning performance of mobile agent platforms without tailoring an SPE methodology for this purpose. Therefore, the paper shows that the PUMA approach is powerful enough to deal with complex performance problems in the mobile agents software domain.

Taking our models and results as a background, we believe that interested practitioners can use PUMA to test their mobile agent platforms in hypothetical conditions, ands they can address other performance problems in this domain following PUMA.

9. REFERENCES

[1] *The TimeNET tool.* `http://pdv.cs.tu-berlin.de/~timenet/`.

[2] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets.* John Wiley Series in Parallel Computing - Chichester, 1995.

[3] Y. Aridor and M. Oshima. Infrastructure for Mobile Agents: Requirements and Design. In *Second*

International Workshop on Mobile Agents (MA'98), Stuttgart, Germany, pages 38–49. Springer, 1999.

[4] S. Balsamo and M. Marzolla. Towards performance evaluation of mobile systems in UML. In *Proc. of ESMc'03, The European Simulation and Modelling Conference*, pages 61–68, Naples, Italy, 2003. EUROSIS-ETI.

[5] H. Baumeister, N. Koch, P. Kosiuczenko, P. Stevens, and M. Wirsing. UML for Global Computing. In *Global Computing. Programming Environments, Languages, Security, and Analysis of Systems, IST/FET International Workshop, GC 2003, Rovereto, Italy, February 9-14, 2003, Revised Papers*, volume 2874 of *Lecture Notes in Computer Science*, pages 1–24. Springer Verlag, 2003.

[6] P. Bracchi, B. Cukic, and V. Cortellessa. Performability Modeling of Mobile Software Systems. In *15th International Symposium on Software Reliability Engineering (ISSRE 2004), 2-5 November 2004, Saint-Malo, Bretagne, France*, pages 77–88. IEEE Computer Society, 2004.

[7] V. Grassi, R. Mirandola, and A. Sabetta. UML based modeling and performance analysis of mobile systems. In *Proceedings of the 7th International Symposium on Modeling Analysis and Simulation of Wireless and Mobile Systems, MSWiM 2004, Venice, Italy, October 4-6, 2004*, pages 95–104. ACM, 2004.

[8] J. Hillston and M. Ribaudo. Modelling mobility with pepa nets. In *ISCIS*, volume 3280 of *Lecture Notes in Computer Science*, pages 513–522. Springer, 2004.

[9] S. Ilarri, R. Trillo, and E. Mena. SPRINGS: A Scalable Platform for Highly Mobile Agents in Distributed Computing Environments. In *4th International WoWMoM 2006 workshop on Mobile Distributed Computing (MDC'06), Buffalo, New York (USA)*. IEEE Computer Society, ISBN 0-7695-2593-8, June 2006.

[10] G. Kastidou, E. Pitoura, and G. Samaras. A Scalable Hash-Based Mobile Agent Location Management Mechanism. In *Proceedings of the 1st International Workshop on Mobile Distributed Computing (MDC'03)*, pages 472–478, May 2003.

[11] D. Lange and M. Oshima. Seven good reasons for mobile agents. *Communications of the ACM*, 42:88–89, 1999.

[12] E. Mena, J.A. Royo, A. Illarramendi, and A. Goñi. Adaptable Software Retrieval Service for Wireless Environments Based on Mobile Agents. In *International Conference on Wireless Networks (ICWN'02), Las Vegas, Nevada, USA*, pages 116–124. CSREA Press, 2002.

[13] J. Merseguer, J. Campos, and E. Mena. Analysing Internet Software Retrieval Systems: Modeling and Performance Comparison. *Wireless Networks: The Journal of Mobile Communication, Computation and Information*, 9(3):223–238, May 2003.

[14] D. Milojicic, F. Douglis, and R. Wheeler. *Mobility: processes, computers, and agents*. ACM Press/Addison-Wesley Publishing Co., 1999.

[15] D. Milojicic, W. LaForge, and D. Chauhan. Mobile Objects and Agents (MOA). In *Fourth USENIX Conference on Object-Oriented Technologies and Systems (COOTS98), Santa Fe, New Mexico, USA*. USENIX, 1998.

[16] Amy L. Murphy and Gian Pietro Picco. Reliable communication for highly mobile agents. *Autonomous Agents and Multi-Agent Systems*, 5(1):81–100, 2002.

[17] Object Management Group. *UML Profile for Schedulabibity, Performance and Time Specification*, January 2005. Version 1.1. `http://www.uml.org`.

[18] D.B. Petriu and M. Woodside. An intermediate metamodel with scenarios and resources for generating performance models from UML designs. *Software and Systems Modeling*, 5(4), August 2006. DOI - 10.1007/s10270-006-0026-8.

[19] M. Scarpa, M. Villari, A. Zaia, and A. Puliafito. From client/server to mobile agents: an in-depth analysis of the related performance aspects. In *Proceedings of the Seventh IEEE Symposium on Computers and Communications (ISCC 2002), 1-4 July 2002, Taormina, Italy*, pages 768–773. IEEE Computer Society, 2002.

[20] A. R. Silva, A. Romão, D. Deugo, and M. M. Da Silva. Towards a Reference Model for Surveying Mobile Agent Systems. *Autonomous Agents and Multi-Agent Systems*, 4(3):187–231, 2001.

[21] C.U. Smith and L.G. Williams. *Performance Solutions*. Addison–Wesley, 2001.

[22] C. Spyrou, G. Samaras, E. Pitoura, and P. Evripidou. Mobile agents for wireless computing: the convergence of wireless computational models with mobile-agent technologies. *Mobile Networks and Applications*, 9(5):517–528, 2004.

[23] M. Woodside, D.C. Petriu, D.B. Petriu, H. Shen, T. Israr, and J. Merseguer. Performance by unified model analysis (PUMA). In *Proceedings of the Fifth International Workshop on Software and Performance, WOSP 2005, Palma, Illes Balears, Spain, July 12-14, 2005*, pages 1–12. ACM, 2005.