TOWARDS PERFORMANCE EVALUATION OF MOBILE SYSTEMS IN UML

Simonetta Balsamo
Moreno Marzolla

Dipartimento di Informatica
Università Ca' Foscari di Venezia
via Torino 155, 30172 Mestre (VE), Italy
e-mail: {balsamo|marzolla}@dsi.unive.it

Keywords

Process Oriented Simulation, Software Architectures, Unified Modeling Language (UML), Performance Evaluation, Mobile Systems.

Abstract

The current generation of network-centric applications exhibits an increasingly higher degree of mobility. Wireless networks allow devices to move from one location to another without loosing connectivity. Also, new software technologies allow code fragments or entire running applications to migrate from one host to another. Performance modeling of such complex systems is a difficult task, which should be carried out during the early design stages of system development. However, the integration of performance modeling analysis with software system specification for mobile systems is still an open problem, since there is no unique widely accepted notation for describing mobile systems. Moreover performance modeling is usually developed separately from high-level system description. This is not only time consuming, but the separation of performance model and system specification makes more difficult the feedback process of reporting the performance analysis results at the system design level, and modifying system model to analyze design alternatives. In this paper we address the problem of integrating system performance modeling and analysis with a specification of mobile software system based on UML. In particular we introduce a unified UML notation for high-level description and performance modeling of mobile systems. The notation allows inclusion of quantitative information, which are used to build a process-oriented simulation model of the system. The simulation model is executed, and the results are reported back in the UML notation. We describe a prototype tool for translating annotated UML models into simulation programs and we present a simple case study.

1 INTRODUCTION

The current generation of network-centric applications exhibits an increasingly higher degree of mobility. From one side, wireless networks allow devices to move from one location to another without loosing connectivity. From the other side, new software technologies allow code fragments or entire running applications to migrate from one host to another. In this direction, design approaches based on location awareness and code mobility have been proposed where the application components can move to different locations during their execution. This should improve system quality, allowing a higher degree of flexibility and increasing its performance. Indeed, from the performance viewpoint, moving application components in a distributed environment could lead to transforming remote interactions into local ones. We consider software mobile system at a high level of abstraction and we refer to Software Architecture (SA) to describe system structure and behavior (Bass et al., 1998; Shaw and Garlan, 1996). In particular, one can define mobile SA with various mobility styles, whose definition depends on whether they require copies creation of components at new locations, or local change of components preserve their identity (mobile agent). In the former case we can further distinguish systems where the copy is created at the location of the component that starts the interaction (code on demand), or systems where it is a created at the location of the component that accept the interaction (remote evaluation). Many formalisms to represent mobile software systems and reasoning about mobility have been proposed (De Nicola et al., 1998; Milner, 1999; Nottegar et al., 2001; Picco et al., 2001). However, most of them cannot be considered architectural description languages (ADL), since they do not explicitly model components and interactions as first class entity.

Several models of SA have been proposed based on formal ADL with precise semantics and syntax (Medvidovic and Taylor, 2000). On the other side, due to the difficulties in integrating formal ADL in the design practice, other approaches consider semi-formal widely used modeling languages such as the Unified Modeling Language (UML) (Medvidovic et al., 2002; Object Management Group, 2001; Rumbaugh et al., 1999).

In this paper we consider mobile software systems at the SA level, and an UML-based system specification. Performance modeling of mobile software systems is a difficult task, which should be carried out from the early design stages of system development. The integration of quantitative performance analysis with software system specification has been recognized to be a critical issue in system design. Per-

formance is one of the most influential factors that drive system design choices. Several approaches have been proposed to integrate performance evaluation tools in the early stages of the software development life cycle (Balsamo et al., 2002; Smith and Williams, 2002). However, few of them consider mobile systems (Cortellessa and Grassi, 2002) and there is still a lack of tools for performance analysis that support the system designer in the selection of SA of mobile software systems to meet given performance requirements.

We focus on the integration of performance modeling analysis with software system specification for mobile systems. To this aim we observe that there is not a unique widely accepted notation for describing mobile systems. Moreover performance modeling is often developed separately from high-level system description. This is not only time consuming, but the separation of performance model and system specification makes more difficult the feedback process of reporting the performance analysis results at the system design level, and modifying system model to analyze design alternatives.

In this paper we address the problem of integrating system performance modeling and analysis with a specification of mobile software system based on UML. In particular we propose an integrated approach to modeling and performance evaluation of mobile systems at the architectural level based on simulation. We consider both physical mobility (devices which physically change their locations) and code mobility (code fragments which migrate from one execution host to another).

The main advantage of using a simulation approach is that it allows a high degree of model flexibility, a direct representation of system components and makes it easy to report performance results back at the SA design level. Indeed, various approaches based on simulation models of software systems have been recently proposed to evaluate SA performance (Arief and Speirs, 1999, 2000; De Miguel et al., 2000). Simulation greatly simplifies the derivation of the performance model from the software system specification, with respect to other approaches based on analytical models. However, code mobility has not been considered in these proposals.

We introduce a unified UML notation for high-level description and performance modeling of mobile systems. The software system description based on the standard UML notation is integrated with performance-oriented parameters, which are described with the UML extension mechanisms (namely stereotypes and tagged values). The notation allows users to include quantitative information, which are used to build a process-oriented simulation model of the system. We use a subset of the annotations proposed in the UML Performance Profile (Object Management Group, 2002a). Then, the annotated UML diagrams are converted into a process-oriented simulation model. The simulation model is executed to evaluate the performance of the mobile system, and simulation results are reported back the original UML diagrams as tagged values. We describe a prototype tool for translating annotated UML models into simulation programs and we present a case study.

This paper is organized as follows. Section 2 introduces the proposed approach to mobility modeling with UML Use Case, Activity and Deployment diagrams. Section 3 describes the annotations used for performance modeling of UML specifications. Section 4 presents a simple case study of the proposed methodology. Conclusions and open research are discussed in Section 5.

2 UML MOBILITY MODELING

We consider mobile software systems at the SA level, and an UML-based system specification. UML is a standard graphical notation for modeling object-oriented software, and it has been widely applied in the software development process (Object Management Group, 2001; Rumbaugh et al., 1999). The graphical notation provides several types of diagrams that represent different system views of the system. UML provides extension mechanisms that include stereotypes and tagged values, to integrate the need of specific domains, We observe that currently there is no standard way for expressing mobility in UML, although different proposals exist in the literature.

2.1 Previous work

Baumeister et al. propose in (Baumeister et al., 2003) an extension of UML Class and Activity diagrams to represent mobility. They define new stereotypes for identifying mobile objects and locations. Stereotypes are also defined for moving and cloning activities. Mobile systems are then represented by Activity diagrams using either a “responsibility centered notation”, which focuses on who is performing actions, and a “location centered notation” which focuses on where actions are being executed and how activities change their location. While this approach has the advantage of requiring only minor extensions to UML, a possible shortcoming is that it represents in the same Activity diagram both the mobility model (how objects change their location) and the computation model (what kind of computations the objects perform). For large models this could render the diagrams difficult to understand.

Some UML notation mechanisms can be used to represent mobile SA, as discussed in (Rumbaugh et al., 1999). They are based on the tagged value `location` to express a component location, and the stereotypes `copy` and `become` to express the location change of a component. They can be used in Collaboration diagrams to model location changes of mobile components. Grassi and Mirandola (Grassi and Mirandola, 2001) suggest an extension to UML to represent mobility using Collaboration diagrams. Collaboration diagrams contain a `location` tagged value representing the physical location of each component. They define the `moveTo` stereotype, which can be applied to messages in the Collaboration diagram. When the `moveTo` stereotype is present, it indicates that the source component moves to the location of the destination component before interacting with it. Sequence diagrams are used to describe the interactions between components, regardless of their mobility pattern.

Kosiuczenko (Kosiuczenko, 2002) proposes a graphical

notation for modeling mobile objects based on UML Sequence diagrams. Mobile objects are modeled using an extended version of lifelines. Each lifeline is represented as a box that can contain other objects (lifelines). Stereotyped messages are used to represent various actions such as creating or destroying an object, or entering and leaving an object. This approach has the drawback of requiring a change in the standard notation of UML Sequence diagrams, that is, lifelines should be represented as boxes, with possibly other Sequence diagrams inside. Existing graphical UML editors and processors need to be modified in order to support the new notation.

2.2 The approach

We model a mobile system as a collection of devices, which can be either processors or communication links. A computation on the system is modeled as a set of activities carried out on the devices. A configuration of the system is a specific allocation of activities on processors. So, while a mobile entity travels through the system, it activates a sequence of configurations, each representing a specific system state. Mobile entities may be both physical devices traveling in the real space, or software components which migrate from one processor to another. Once a configuration is activated, the mobile entity starts an interaction with the system. This typically includes requesting service to the devices (processors) or performing communications, which we also model as requesting service to network devices. We assume that while a mobile entity is interacting with the system, it cannot move, i.e., the system configuration cannot change. Further movements are possible when the interaction is completed.

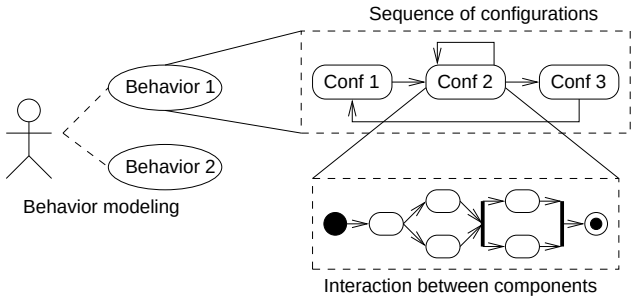


Figure 1: Overview of the mobility modeling methodology

The proposed approach to mobile system modeling involves three main steps, which are depicted in Fig. 1

1. Enumerate the various mobility strategies of the mobile entities.
2. Model the sequence of configurations which are triggered in each mobility strategy.
3. Model the interactions performed by the mobile entities in each configuration.

The first step deals with identifying the mobility pattern of the entities. For example, users of the system may exhibit a bigger or smaller probability to change their location, hence they may exhibit different degrees of mobility. Also, users may move through the system with different preferential patterns. We define a *mobility behavior* as the sequence of different configurations which are executed while the user moves. Such mobility behaviors need to be identified, and further described in the next steps. Mobility behaviors are represented by UML Use Case diagrams. Also, the set of resources (processors) which are present in the system are described using Deployment diagrams.

The next step involves the description of the sequence of configurations which are triggered while the mobile entities travel through the system. Such description can be easily expressed as a state transition diagram. We use Activity diagrams for this purpose. Each activity represents a particular configuration of the system. Transitions describe the order in which configurations are triggered as the user moves. We will refer to these diagrams as "high level" Activity diagrams.

The last step involves detailing what happens while the system is in each configuration. This means specifying what are the interactions between the components while each configuration is active. This is done again using Activity diagrams. Each Action state is associated to the Deployment node instance on which the activity takes place. Each node of the Activity diagrams defined in the previous step is expanded as an interaction. This can be readily expressed using standard UML notation as Activity diagrams have a hierarchical structure, that is, each action state may be exploded into another diagram. We will refer to these diagrams as "low level" Activity diagrams.

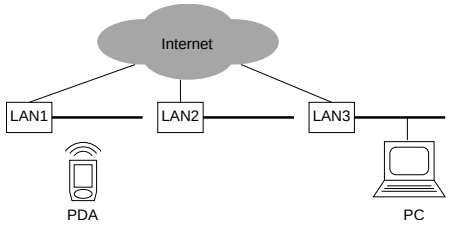
In order to illustrate the proposed approach we introduce an application example of software mobile system. Let us consider the example of software system illustrated in Fig. 2. There is a mobile user that is connected to a PC using a PDA with a wireless network card. The user is viewing a video stream, which is generated by a video server residing on the PC.

Three different Local Area Networks (LAN1, LAN2 and LAN3) are connected through the Internet. Each LAN allows wireless connections as well as wired ones. The PC is connected to LAN3 and does not move, while the user with the PDA travels through the different LAN. In the configuration C_1 of Fig. 2(a) the communication between the PDA and the PC travels through the path LAN1–Internet–LAN3. In the configuration C_2 of Fig. 2(b) the communication is routed through the path LAN2–Internet–LAN3, and in the configuration C_3 of Fig. 2(c) the communication between the PC and the PDA is routed through LAN3 only.

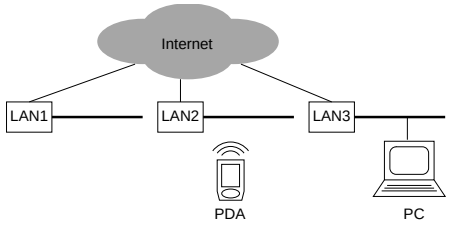
2.3 Modeling the choice of Mobility

As the very first step, it is necessary to provide the physical structure of the system. This can be done by using UML Deployment diagrams, which describe the processing resources available on the system. Such resources include both CPUs and also communication links. The Deployment diagram describing the system in the example is illustrated in Fig. 3.

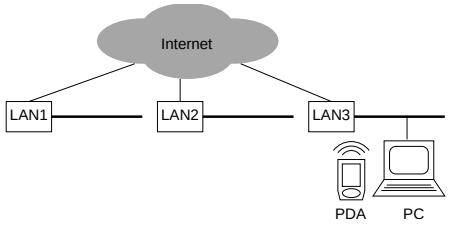
In the example above, we suppose that the mobile user can behave in two different ways. In behavior B_1 he joins the



(a) Configuration C_1



(b) Configuration C_2



(c) Configuration C_3

Figure 2: A mobile user travels through three different LAN

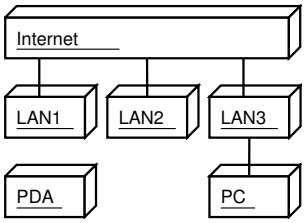


Figure 3: Deployment diagram for the example

system in LAN1, then travel to LAN2 and finally to LAN3 and leaves the system. In behavior B_2 the user joins the system in LAN2, travel to LAN1 and leaves the system. We model these behaviors with the Use Case diagram in Fig. 4.

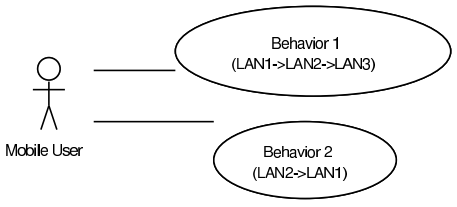
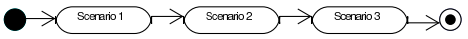


Figure 4: UML representation of different mobility possibilities

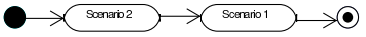
The diagram shows an Actor (mobile entity) that can perform one of the associated Use Cases, each representing one possible mobility behavior. An Actor represents each class of mobile entity. The Use Cases associated with that Actor represent the different ways in which entities of the associated mobile entity class may interact with the system. Note that this is perfectly consistent with the UML semantics of Use Case diagrams (Object Management Group, 2001), as they are used to specify the behavior of an entity without specifying its internal structure.

2.4 Modeling Mobility Behaviors

The next step is to describe the order in which configurations are activated in each behavior. To do that, we associate an Activity diagram to each Use Case; each activity of the Activity diagram represents a configuration of the system. If the mobile user triggers the configuration C_j immediately after the configuration C_i, then in the Activity diagram there will be a transition between the activity representing C_i and the one representing C_j. Considering our example, the two behaviors B_1 and B_2 are represented as the Activity diagrams of Fig. 5.



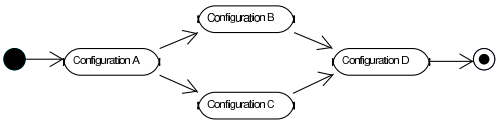
(a) Activity diagram associated to B_1



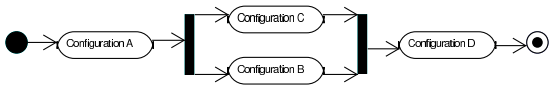
(b) Activity diagram associated to B_2

Figure 5: Activity diagrams associated to the mobility behaviors

Note that in this way it is possible to represent nondeterminism that is a behavior can have multiple successors. Fig. 6(a) illustrates an example where the mobile entity starts by activating configuration A. Then it may proceed by activating one of configuration B and C. After that, configuration D is activated. It is important to observe that the Activity diagrams associated to behaviors do not need to be acyclic. Thus, it is also possible to model situations in which the user triggers the same sequence of configurations for a number of times. Moreover, using fork and join nodes of Activity diagrams it is possible to represent the concurrent execution of different configurations. This can be used to model situations in which the mobile entity generates copies of itself, each one traveling independently through the system. Fig. 6(b) shows an example where a mobile entity starts by entering configuration A. Next, it splits in two copies, one executing configuration B and the other executing configuration C in parallel. This means that the two copies of the mobile entity can move to different locations and perform different interactions with the system. After that, the two copies synchronize and collapse into one instance, which proceeds by executing configuration D.



(a) Nondeterministic behavior



(b) Concurrent agent execution

Figure 6: Modeling nondeterministic mobility behavior and mobile concurrent execution of multiple agent instances

2.5 Modeling Component Interactions

The final step is describing the activities carried out in each configuration. To do that, we use the hierarchical structure of the Activity diagrams to associate the interactions to each action state identified in the previous step. Namely, we expand each action step representing a configuration into the Activity graph describing the sequence of actions which are taken during the interaction between the mobile entity and the system. It is necessary to specify where the actions are executed. To do so it is possible to use "swimlanes", which are a means for specifying responsibility for actions. The name of the swimlanes denotes the Deployment node instance on which the actions execute. As some graphical UML editors do not support swimlanes, it is possible to tag each action with the `PAhost` tagged value, whose value is the name of the node instance of the Deployment diagram corresponding to the host where the action is executed. Fig. 7 shows the interactions performed while the considered system is in configuration C_1 and C_3 using the swimlane-based notation.

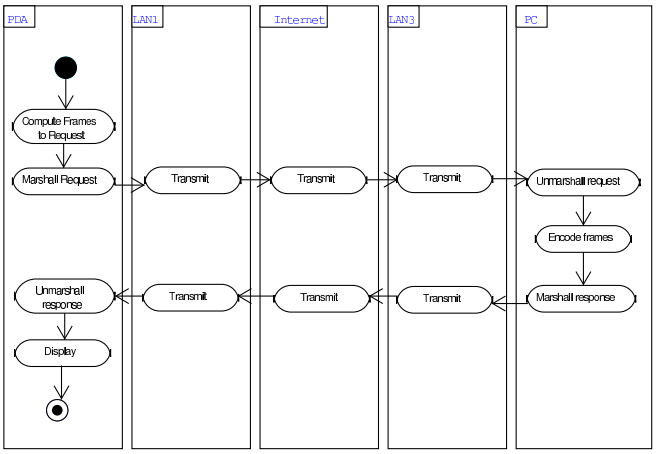
The interaction between the PDA and the PC is very simple. Basically, first the PDA computes which frames it needs. Then a suitable request is encoded and sent through the communication networks to the PC. The request is unmarshaled, and the requested frames are encoded and packed into a reply message. This message is sent back through the network to the PDA, which finally displays the frames. Note that in Fig. 7 we omitted the description of the interaction in configuration C_2, as this is basically the same as in C_1, with the only difference that LAN2 is used instead of LAN1.

2.6 Summary of the Methodology

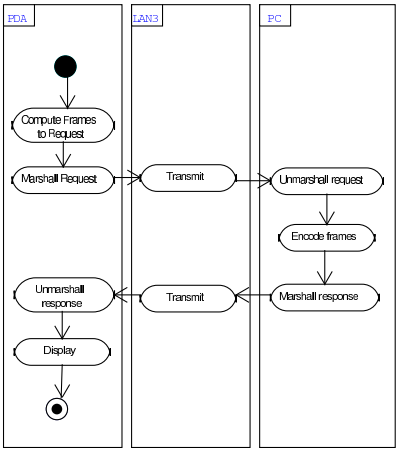
The proposed mobility modeling methodology can be summarized in the following steps:

1. Identify the processing resources (processors or networks) available in the system. Each resource is represented by a node instance in the UML Deployment diagram.

2. Identify the classes of users of the system. Users represent workloads applied to the system. Each class of users is represented as an Actor in the Use Case diagram. Actor may represent either a fixed population of users (closed workload) or an unlimited stream of users (open workload).



(a) Interaction in configuration C_1



(b) Interaction in configuration C_3

Figure 7: UML description of the interactions

3. Identify the mobility behaviors. For each class of users it is necessary to identify the different pattern of mobility they may exhibit. Each of such mobility patterns is represented by a Use Case associated to the Actor representing the class of users.

4. Provide a high level description of the mobility behaviors. An Activity diagram is association to each of the Use Cases identified in the previous step. Such Activity diagram represents the sequence of configuration changes which happens in the system while the mobile user moves.

5. Describe the interactions occurring in each system configuration. Each Action states defined in the previous step are expanded into a low-level Activity diagram describing the interactions between system entities. Each Action of the low-level diagram represents a service requested to a specific processing resource. Code mobility

is represented by associated Activities in the low-level diagram to different hosts. Physical mobility is represented by a possibly different interaction pattern associated to nodes of the high-level Activity diagram.

Quantitative informations required by the simulator can be associated to UML elements as described in the following.

3 UML PERFORMANCE MODELING

We shall now summarize the proposed approach to integrate performance modeling with UML specifications for software mobile SA. The approach derives the performance model directly from annotated UML specifications extended to include mobility as described in the previous Section. We consider the performance model and the annotations based on those proposed in the UML Performance Profile (Object Management Group, 2002a). The profile has been defined using standard UML extension mechanisms, and provides the modeler with a set of packages. Each package defines the mapping between specific domain models (schedulability, time and performance characteristics) to UML stereotypes and tagged values definitions, which represent the UML viewpoint.

We consider SA described in terms of Use Case, Activity and Deployment diagrams. We derive a process oriented simulation model of the software system that is then executed to derive performance indices. Finally, simulation results are reported back into the UML diagrams as tagged values, so they are readily available at the software designer, integrated in the UML system specification.

We now briefly sketch how the performance model is derived; a more detailed description of the derivation of the simulation model from UML diagrams for systems without mobility can be found in (Balsamo and Marzolla, 2003a,b). In the following we show how the methodology can be directly applied to modeling code and physical mobility of the application described by the SA.

Fig. 8 illustrates the structure of the performance simulation model derived from the UML diagrams. The basic object of the simulation model is a `PerformanceContext`. This object contains the other elements of the model, namely Workloads, Scenarios and Resources. Hence there are three types of simulation processes derived from the UML diagrams that are Workload, Scenarios and Resource processes.

Workloads can be open or closed, depending on whether the number of users accessing the system is unbounded or fixed. Each Workload actually drives one or more Scenarios. Each time a new user belonging to a Workload requests service to the system, one of the Scenarios associated with that workload is selected. Selection is done randomly, according to the probability associated to each scenario.

A Scenario is a set of abstract scenario steps, represented by the `AbsStep` class. Abstract scenario steps can either be composite steps (described by the `PScenario` class), or atomic steps of different kinds. Scenarios are collections of steps; exactly one of these steps is marked as the root step (starting step) of the scenario. Atomic steps can be of type `PStep_fork` for nodes representing the creation of multiple execution threads, `PStep_join` for nodes representing synchronization points between different threads, and `PStep` for normal atomic steps.

Each Scenario step executes on a single processor, to which it requires service. Processors are modeled by objects of type `AbsPRhost`. A processor is characterized, among other things, by a scheduling policy which can be one of "FIFO" (first come first served), "LIFO" (last come first served) or "PS" (processor sharing). The simulation model defines some classes derived from `AbsPRhost` which implement a specific scheduling policy.

Each UML model element (Actor, Use Case, Activity state, Node instance) can be tagged with additional quantitative information, which is necessary to derive the parameters of the simulation model. Examples of such parameters are the interarrival time of users, the service demand of action steps, the speed factor of each processor.

Concerning mobility modeling, each Use Case are tagged with the probability of its occurrence, that is, the probability that the associated Actor (motile entity) will execute that Use Case (mobility behavior) upon arriving to the system. Action states of the high-level Activity diagram associated to each Use Case are annotated with the probability of occurrence, the number of times they are repeated and the delay between repetitions. As each action state represents a configuration, the annotations allow the specify (nondeterministically) the pattern of mobility and how long the system remains in each configuration. When the simulator "executes" a configuration, it basically executes all the activities of the low-level Activity diagram embedded in the configurations.

A mobile code fragment moving from host H_1 to host H_2 is represented as follows. Let C_1 be the configuration where the code executes in H_1 and C_2 the configuration where the code executed in H_2. The low level Activity diagram describing C_1 and C_2 will contain an action state (or a whole subdiagram) corresponding to the computation. Such action state or subdiagram will be tagged with the `PAhost` tagged value, which describes the location where the activity is executed. In C_1 we set `PAhost`=H_1, and in C_2 we set `PAhost`=H_2.

Physical mobility is modeled in a similar way. If a mobile device travels through the system, as in our example, probably it will interact with different other nodes for communicating. Depending on the situation, it may even choose a different communication pattern, and hence a different interaction style with other entities. Such interaction styles will be represented by (possibly different different) structures of the low level Activity diagrams.

The derivation of the performance model from the UML specification works as follows. Each actor of the Use Case diagrams is translated into an `OpenWorkload` or `ClosedWorkload` simulation process, depending on its associated stereotype. Note that to represent mobility, each Actor represents a class of mobile entity.

Then, each Use Case associated with the Actor is examined and translated into a `PScenario` simulation process.

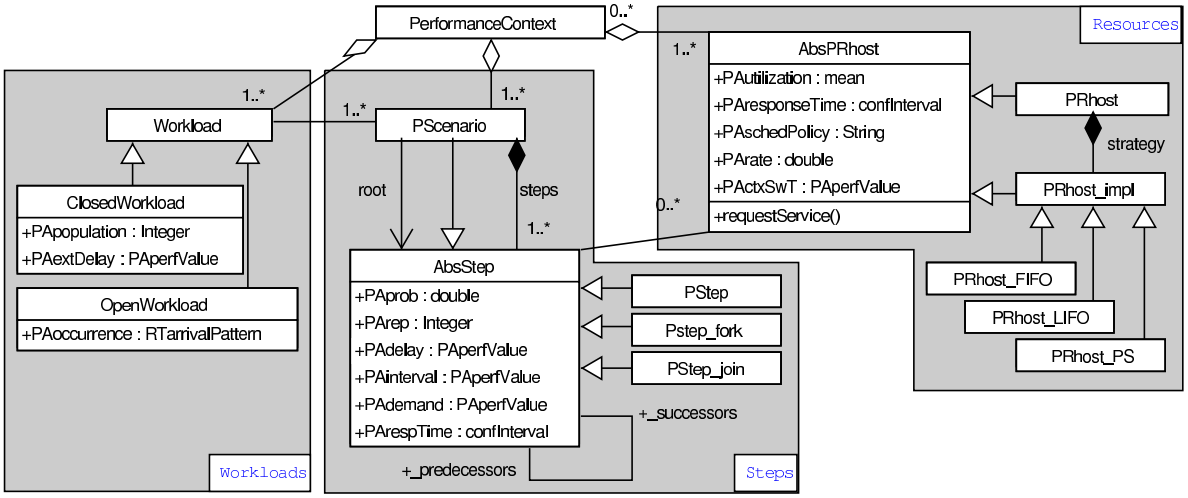


Figure 8: Structure of the simulation performance model

To define the content (sequence of steps) of the scenario, the high level Activity Diagram associated with the Use Case is examined. The structure of such diagram is translated into a network of `PScenario` simulation processes. The internal behavior of such processes is modeled by the low level Activity diagram contained in the high level action stated.

All the activities are translated into the appropriate kind of step (that is, `PStep`, `PStep_fork` or `PStep_join`) depending on the type of the UML element. The transformation is applied recursively if an activity is indeed composite of a sub-activity diagram. Each object derived from the `AbsStep` class is associated with an object representing the physical resource on which it executes. The mapping between actions and the resources where they execute can be derived by examining the `PRhost` tagged value, or the name of the swimlane containing the activity.

Finally, each node instance in the Deployment Diagrams is translated into a simulation process of type `AbsPRhost`.

We developed UML-Ψ (UML Performance SImulator), a prototype performance evaluation tool, which processes an XMI (Object Management Group, 2002b) description of UML Use Case and Activity diagrams. The UML SA has to be annotated using a simplified subset of the UML Performance Profile.

The simulation model is process oriented and its objects are derived by the analysis of the UML diagrams annotated with performance specification of the software system components, and with the extension to describe mobility, as describer in the previous Section. The simulation model is implemented as a discrete-event simulation program written in C++, whose execution provides results for a set of performance indices. We evaluate through simulation the mean response time associated with the execution of each scenario (Use Case) and each scenario step (Activity). Simulation results, i.e., the performance measures of the software components are inserted back into the original UML SA as tagged values to provide feedback to the system designer.

4 CASE STUDY

The case study illustrated in the previous sections has been simulated using the parameters reported on Table 1. A single user interacts with the system (closed workload), and the probability p that the user triggers the behavior B_1 (see Fig. 5) has been set to $p = 0.3$, while the probability that the user triggers the behavior B_2 has been set to $1 - p$.

Parameter	**Value**
(Un)Marshalling Requests	Const. $0.1s$
(Un)Marshalling Responses	Expon. mean=$5.0s$
Request Transmission Times	Exp. mean=$1.0s$
Response Transmission Times	Exp. mean=$10.0s$
Request Computation	Exp. mean=$0.1s$
Frame Encoding Time	Exp. mean=$20.0s$
Display time on the PDA	Exp. mean=$20.0s$
PDA Speedup factor	0.2
PC Speedup factor	10.0
Processors Sched. Policies	FIFO

Table 1: Simulation Parameters. “Exp.” means exponentially distributed with the given mean. “Const.” means constant.

The simulation results are shown in Table 2. They are the steady-state mean values computed at 90% confidence level; for simplicity only the central value of the confidence interval is shown. Such results are automatically inserted into the UML diagram as tagged values associated to the relevant UML elements, as described in (Balsamo and Marzolla, 2003a,b). In this way it is very easy to get immediate feedback on the model performances.

5 CONCLUSIONS

In this paper we introduced an integrated methodology for UML-based modeling and performance evaluation of mobile systems. Only minimal extensions to the semantics of UML are requested in order to apply the methodology. The performance modeling is based on a subset of the UML Perfor-

Internet utilization	0.039
LAN1 utilization	0.010
LAN2 utilization	0.028
LAN3 utilization	0.072
PDA utilization	0.832
PC utilization	0.015
Behavior B_1 response time	445s
Behavior B_2 response time	312s

Table 2: Simulation Results.

mance Profile. We depicted the structure of the simulation performance model which is automatically derived from the annotated UML diagrams. Using a simple case study, we showed the approach can be applied.

Our current research involves the development of a more compact ways to describe the UML diagrams used in the proposed approach. In fact, we note that many of the low level activity diagrams have the same structure, with only minimal differences mostly related to the location of the activities. Describing such diagrams in some "parametric" way would greatly alleviate the work of the modeler.

ACKNOWLEDGMENTS This work has been partially supported by MURST Research Project Sahara and by MIUR Research Project FIRB "Performance Evaluation of Complex Systems: Techniques, Methodologies and Tools".

REFERENCES

L. B. Arief and N. A. Speirs. Automatic generation of distributed system simulations from UML. In *Proc. of ESM '99, 13th European Simulation Multiconference*, pages 85–91, Warsaw, Poland, June 1999.

L. B. Arief and N. A. Speirs. A UML tool for an automatic generation of simulation programs. In Proceedings of WOSP 2000 Proceedings of WOSP 2000, pages 71–76.

S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni. Software performance: state of the art and perspectives. Tech Rep. TR SAH/04, MIUR Sahara Project, Dec. 2002.

S. Balsamo and M. Marzolla. A simulation-based approach to software performance modeling. Tech. Rep. TR SAH/44, MIUR Sahara Project, Mar. 2003a.

S. Balsamo and M. Marzolla. Simulation modeling of UML software architectures. In D. Al-Dabass, editor, *Proc. of the European Simulation Multiconference*, pages 562–567, Nottingham, June 2003b.

L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley, 1998.

H. Baumeister, N. Koch, P. Kosiuczenko, and M. Wirsing. Extending activity diagrams to model mobile systems. In M. Aksit, M. Mezini, and R. Unland, editors, *NetObjectDays*, volume 2591 of *Lecture Notes in Computer Science*. Springer, 2003. ISBN 3-540-00737-7.

V. Cortellessa and V. Grassi. A performance based methodology for early evaluate the effectiveness of mobile software architecture. *J. of Logic and Algebraic Programming*, 51: 77–100, Apr. 2002.

M. De Miguel, T. Lambolais, M. Hannouz, S. Betgé-Brezetz, and S. Piekarec. UML extensions for the specifications and evaluation of latency constraints in architectural models. In Proceedings of WOSP 2000 Proceedings of WOSP 2000, pages 83–88.

R. De Nicola, G. Ferrari, R. Pugliese, and B. Venneri. KLAIM: a kernel language for agents interaction and mobility. *IEEE Trans. on Soft. Eng.*, 24(5):315–330, 1998.

V. Grassi and R. Mirandola. UML modelling and performance analysis of mobile software architectures. In M. Gogolla and C. Kobryn, editors, *UML*, volume 2185 of *Lecture Notes in Computer Science*. Springer, 2001. ISBN 3-540-42667-1.

P. Kosiuczenko. Sequence diagrams for mobility. In J. Krogstie, editor, *Proc. of MobIMod workshop*, Tampere, Finland, Oct. 2002. Springer.

N. Medvidovic, D. S. Rosenblum, and D. F. Redmiles. Modeling software architectures in the Unified Modeling Language. *ACM Trans. on Soft. Eng.*, 11(1):2–57, 2002.

N. Medvidovic and R. N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Trans. on Soft. Eng.*, 26(1):70–93, 2000.

R. Milner. *Communicating and Mobile Systems: The π-Calculus*. Cambridge University Press, 1999.

C. Nottegar, C. Priami, and P. Degano. Performance evaluation of mobile processes via abstract machines. *IEEE Trans. on Soft. Eng.*, 27(10):338–395, 2001.

Object Management Group. Unified modeling language (UML), version 1.4, Sept. 2001.

Object Management Group. UML profile for schedulability, performance and time specification. Final Adopted Specification ptc/02-03-02, OMG, March 2002a.

Object Management Group. XML Metadata Interchange (XMI) specification, version 1.2, Jan. 2002b.

G. P. Picco, G.-C. Roman, and P. J. McGann. Reasoning about code mobility in mobile UNITY. *ACM Trans. On Soft. Eng. and Methodology*, 10(3):338–395, 2001.

Proceedings of WOSP 2000. Ottawa, Canada, Sept. 2000. ACM Press.

J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.

M. Shaw and D. Garlan. *Software Architecture: perspectives on an emerging disciline*. Prentice-Hall, 1996.

C. U. Smith and L. Williams. *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Addison–Wesley, 2002.