Performance Modeling and Analysis of Context-Aware Mobile Software Systems *

Luca Berardinelli, Vittorio Cortellessa, Antinisca Di Marco

Dipartimento di Informatica
Università dell'Aquila
Via Vetoio, 67010 Coppito (AQ), Italy
{luca.berardinelli, vittorio.cortellessa, antinisca.dimarco}@di.univaq.it

Abstract. Context-awareness is becoming a first class attribute of software systems. In fact, applications for mobile devices need to be aware of their context in order to adapt their structure and behavior and offer the best quality of service even in case the (software and hardware) resources are limited. Although performance is a key non-functional property for such applications, existing approaches for performance modeling and analysis fail to capture the characteristics related to the context, thus resulting not suited for this domain.
In this paper we introduce a framework for modeling and analyzing the performance of context-aware mobile software systems. The framework allows to model: the software architecture, the context management, the adaptable behaviors and the performance parameters. Such models can then be transformed into performance models for analysis purposes. We tailor an integrated environment for modeling these elements in UML, and we show how to use it for performance analysis purposes. The modeling environment description and the performance analysis are driven by an example in the eHealth domain.

1 Introduction

The rapid evolution of portable devices and their increasing pervasiveness in everyday life have motivated, in the last few years, a growing interest for methodologies, techniques and tools that allow to effectively develop and analyze software systems running on such devices.

The main characteristics of portable devices are: mobility and limitation of hardware resources. Both features obviously claim for specific requirements of the deployed software systems that have to be taken into account along the whole software lifecycle.

Mobility can either physical or logical [18]. Physical mobility regards the transfers of a portable device among a certain number of physical locations. Logical mobility takes into account the re-deployment actions that certain software components can be subject to.

The limitation of hardware resources has brought to develop specific releases of software products for portable devices that require limited amounts of resources. However, some resources not only are limited, but their available amounts can change during

* This work has been supported by the Italian Project PACO (Performability- Aware Computing: Logics, Models, and Languages) funded by MIUR.

the device usage. Hence, more recently this limitation has been tackled by providing to software the ability of adapting to changes in the environment.

Mobility and context-awareness (and, as a consequence, adaptation) obviously have a large impact on the performance of software systems. Their bad effects on performance are today passively accepted as unavoidable fees to pay in the domain of advanced portable systems. As opposite, if opportunely managed, they can become powerful instruments in the hands of software developers to maintain an acceptable level of user-perceived performance even in presence of changes and degradations in the surrounding environment [16].

Goal of this paper is to introduce a framework to model and analyze performance of context-aware mobile software systems. The framework is aimed at producing UML models that embed, besides mobility and context-awareness facets, the parameters that allow an automated model-based performance analysis of the system. The elements that build up a context-aware mobile software system model are: the software architecture, the context management, the adaptable behaviors and the performance parameters.

Within this framework different types of mobility and context-awareness can be modeled and, if needed, combined. The rationale behind an uniform modeling of such aspects is that the runtime behavior of a mobile/adaptable software system can be driven both by a mobility event and a change in its computational environment that lead to changes in the software itself, namely adaptation actions. However, besides the specific characteristics of mobility and context-awareness, the interdependencies between them can be captured, and the cross-effects on the system performance can be taken into account. Thus certain types of analysis that were not feasible with specific models (for mobility or context-awareness) can be carried out in our integrated framework. The framework is based on an existing UML tool (i.e. MagicDraw) and on existing UML profiles, such as the UML profile for Modeling and Analyzing Real-Time Embedded Systems (MARTE) [17].

The paper is organized as follows: Section 2 introduces some related work and places our contribution with respect to the existing literature, in Section 3 we present our framework under the guideline of a reference example in the eHealth domain, in Section 4 we show how different types of awareness can be merged together into an unique model, Section 5 shows the results of performance analysis experiments and highlights the potential of our approach, and finally in Section 6 we conclude the paper and discuss possible future work.

2 Related work

Several approaches have been introduced in the last few years to manage mobility and adaptation at the middleware level. Among these, very relevant work has been done within the framework of the MUSIC project [13]. The MUSIC middleware monitors the context and the resources to catch their changes and adapts the application to fulfill the users' QoS requirements. The approach uses QoS predictors and utility functions to support the adaptation process. The adaptation is based on the concept of service plan [15], i.e. a platform-independent specification containing information on service configurations, its dependencies on the environment and its QoS.

All the MUSIC contributions can be used at run time given that the application has been developed to be context-aware and QoS validated. As assessed in [15], the information the MUSIC middleware needs is specified in the service plan, but such information is collected at the design time. As opposite to MUSIC project, we provide a support to model and analyze performance properties of such systems before their implementation and deployment. For example, with our framework it would be possible to automatically generate the (MUSIC) service plan and provide the QoS models that work as predictors in the MUSIC adaptation process.

Another interesting project is DiVA [2], which aims at providing an integrated framework for managing dynamic variability in adaptive systems. DiVA exploits both Model-Driven and Aspect-Oriented technologies to define an architectural model (including base, variant and adaptation models) at design time. The composition and validation at runtime of alternative models allow: (i) the choice of the system configuration that best adapts to the changed execution context, and (ii) the deployment and execution of the chosen configuration supported by a reflective middleware. However, such approach does not provide any support for non-functional analysis.

Grassi et al. in [8] have proposed a modeling framework for QoS-aware self-adaptive software applications that present several similarities with our framework. Such framework, based on the definition of an intermediate pivot language (i.e. D-KLAPER), is aimed at providing instruments to transform software models into non-functional models and analyze QoS characteristics while changes in the application and/or its environment may occur.

Our work improves the approach in [8] for several aspects: (i) context-awareness and mobility are based in our approach on a set of attributes whose evolution is modeled through Statecharts, whereas in [8] a set of triggers has to be specified in isolation; (ii) the previous difference allows us to introduce dependencies among events that cannot apparently modeled with D-KLAPER; (iii) we have implemented our approach in UML, so to prove that such language has the potential to represent triggers and (simple) adaptation mechanisms, whereas this part of D-KLAPER still does not find any correspondence in UML. However, on the other side, the work in [8] also presents some advantages, such as: (i) to explicit represent adaptation actions, (ii) to take into account the non-functional costs of such actions, (iii) to generate a Markov Reward Model that allows to study non-functional properties even in non steady states of the system.

Finally, our idea of managing all context- and mobility-related aspects with statecharts is very close to the concept of *modes*. Modes has been proposed in [11] to extend the Darwin ADL for modeling Service Oriented Computing systems. Modes are also language primitives in the Architecture&Analysis Description Language (AADL) [1] for modeling Real-Time&Embedded Systems. In both cases they can be used to model the structural evolution of software architecture at runtime. Besides components, AADL allows the modal specification of all its modeling elements like system, connectors and properties. Thus, our logical mobility and hardware managers can be modeled as AADL component's modes whereas the overall context manager as system's modes. In AADL, it is also possible to model the physical mobility by means of system's modes. However, in this case, it can't be associated to a system user as we do associating the manager to UML Actors. Therefore, differently to AADL, our UML-based modeling approach

can be (i) ”sized” for different definitions of context and (ii) used as a general ”modal-based” modeling approach for software system of multiple domain.

3 Modeling Performance-annotated Context-aware Software Systems

In this section we describe our approach for designing UML models of context-aware software systems that embed performance annotations. The description is driven by a reference example in the eHealth domain. In [4] an extended description of our approach has been reported, where more technical details are provided and a general scope of the approach is illustrated.

The envisaged eHealth service supports the doctor's everyday activities, such as the retrieval of mixed media information on his patients that combines text with or without different kinds of images referring to their personal data, their medical histories and patient-related diseases. The results can be displayed on the doctor's handheld device.

The UML model we devise is organized in three views:

The **Service View** (SV) represents the services provided by the software system as perceived and used by external actors (Use Case Diagram, UCD) along with their behavioral specifications (Sequence Diagram, SD). A *Physical Mobility Manager* (Statechart, SC) is assigned to each nomadic user that exploits the system services while moving with his mobile device [9][6].

The **Component View** (CV) represents a software architecture (Component Diagram, CD) integrated with mobility annotations that allow to distinguish logically mobile from fixed software artifacts. A *Logical Mobility Manager* is associated to each component whose implementation is (even partially) mobile [9] .

The **Deployment View** (DV) represents: (i) the current/allowed allocation of software artifacts on execution environments (e.g. handheld devices) that can physically move across different places (*Dynamic* Deployment Diagram, dynDD), and (ii) several detailed hardware device specifications (*Hardware* Deployment Diagram, hwDD). A *Hardware Configuration Manager* is associated to each resource whose state (that may represent its current amount) may vary at runtime.

To enable model-based performance analysis the UML model has to be annotated with additional information coming from several profiles. We have adopted the UML Profile for Modeling and Analysis of Real-Time Embedded Systems (MARTE) [17] and the UML Profile for Mobile Systems [9]. In addition, we have defined a Context Modeling profile to model the *managers* that handle the different types of awareness. The driving criterion in our profiling task has been to re-use existing profiles wherever possible[1].

[1] Hereafter we denote with `typewritten` words model variables whereas with the *italicized* ones the stereotypes of profiles. Note, however, that the UML diagrams have been suitably tailored to fit the page limitation and to preserve their readability, whereas a machine-readable complete UML model of our eHelath example can be downloaded at *http://www.di.univaq.it/cortelle/docs/eHealthSystemModelASE.rar*.

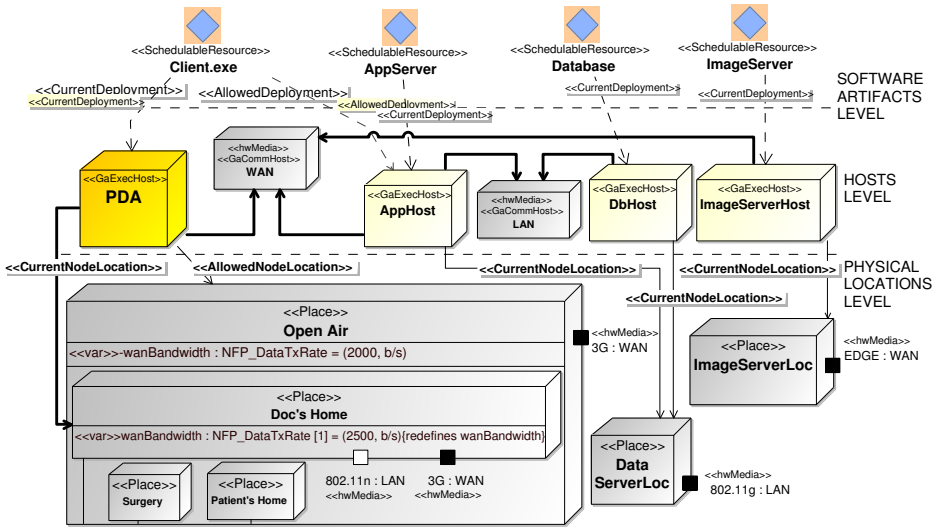

Fig. 1. The Dynamic Deployment Diagram.

3.1 Modeling the Software Architecture

The retrieval of patient related information (that hereafter we refer as RequestPatientInfoPages) is supposed to be the most frequently invoked service (basing on the user profile). In our modeling approach, each service is provided by a component-based system whose architectural description is given by a Component Diagram (CD). The CD identifies the software components, their interconnections and the executable artifacts implementing them. Moreover, it specifies which component is mobile and the performance parameters needed for the analysis.

In Figure 1 the *dynamic* DD of the application is shown. It is inspired by the diagram introduced in [9], as it basically contains two types of information: (i) the allocation of the software artifacts (*SchedulableResources*) on the execution environments (*GaExecHost*) through deployment relationships (*CurrentDeployment*, *AllowedDeployment*) that go from the Software Artifact level to the Hosts level, and (ii) the positioning of the execution hosts (e.g. PDA) on different physical locations using associations (*CurrentLocation*, *AllowedLocation*) that go from the Hosts level to the Physical Locations level. The dynamic nature of dynDD derives from the need to change the current and allowed relationships between levels whenever logical and/or physical mobility events take place.

Looking at Figure 1 we deduce that the RequestPatientInfoPages service is available if the user PDA is able to connect to a WAN network (*hwMedia*). Different *Places* can provide different types of network connections (i.e. typed Ports of *Places*), but some of them might not be exploitable by the service (such as the white-colored port 802.11n:LAN at Doctor's Home) due to particular design choices and/or hardware limitations.

3.2 Modeling the Context-Awareness

In this section we model the context-awareness of the eHealth application, whereas in Section 3.3 we describe how this awareness can influence the service behavior. Each type of awareness is handled by a manager modeled as an UML Statechart.

Physical Location-Awareness -

The modeling of the Physical Location Awareness takes inspiration from previous works [6][9]. We define a *Physical Mobility Manager* for each nomadic user (i.e. the doctor in our case).

An UML Statechart is defined for each manager, where a state represents the current physical location and the resources in the surroundings (together referred as physical configuration, *PhyConfig*) at the time when users demand for services. The transitions are triggered either by physical moves of the nomadic users or by changes in physical resources in the surroundings. Figure 2(a) shows the doctor mobility pattern (i.e. the one of his PDA) where the physical transfer from his home to the patient's one is highlighted along with the probabilities of the moves[2]). Hence each *PhyConfig* refers to some platform device (in this case the PDA), and to the deployment diagram that embeds it. A *PhyConfig* state determines the ends of the *Current-* and *AllowedNode-Locations* relationships among mobile execution hosts and places on dynDD (Figure 1.

Logical Location-Awareness -

Logical mobility is informally defined as the *capability to dynamically change the bindings between code fragments and the location where they are executed* [7]. We adopt the solution proposed in [9] that is based on an UML Statechart called *Logical Mobility Manager*. In a Logical Mobility Manager a state corresponds to the current allocation (*CurrentDeployment*) of the software components (*MobileCode*) to the proper execution platforms (*GaExecHost* in Figure 1). State transitions represent the possible re-deployments of mobile software artifacts through the communication channels (*HwMedia* in Figure 1) to other platform devices (*AllowedDeployment*).

For example, when the client artifact (i.e. client.exe in Figure 1) actually runs on the PDA, it can migrate back and forth to the application host due to some design reason (e.g. performing heavy tasks on the server side when resources on the PDA are scarce).

Hardware Platform Awareness - The third dimension of the context-awareness, as defined in this paper, takes into account the detailed hardware specification of the execution environment. We illustrate in Figure 2 the *Hardware Configuration Managers* for hardware resources whose internal configuration *HwConfig* can influence the service behavior (see [4] for a detail of these resources).

Figure 2(b) illustrates the CPU, BATTERY and DISPLAY managers as separate UML Statecharts that model the states and transitions of corresponding hardware components. Each state specifies a set of *nfpConstraints* (based on variables defined on the configured hardware component) [17] to be held in the current configuration *HwConfig*.

In addition, a remote firing transition (i.e. BATTERY2LowPowerDowngrade) is illustrated to highlight how the remaining BATTERY capacity (`currCapacity`) can

[2] For each state the probabilities of the outgoing transitions at most sum to 1, where the gap to 1 implicitly corresponds to the probability of the self-transition.

influence the configuration of the CPU by firing a remote transition that limits its clock frequency (`currFrequency`).

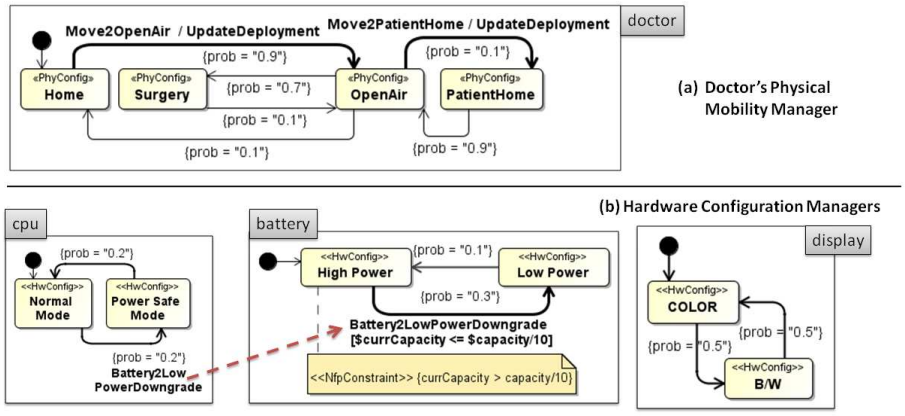

Fig. 2. Physical and Hardware Configuration Managers.

3.3 Modeling the Service Behaviors

The eHealth modeling is completed by the specification of service's behaviors.

The left side of Figure 3 shows an UML Sequence Diagram associated to the RequestPatientInfoPages service. When the doctor, once logged in, invokes the distributed service, the server-side components are in charge of retrieving data from a local (i.e. connected by LAN) database and, if suited, from a remote (i.e. connected by WAN) image server for patients' x-rays or disease-related images. Finally the result is displayed on the client.

The service behavior can be determined by the current context conditions defined by the values of the managers' model variables. Figure 3 represents indeed an Interaction Overview Diagram (IOD) that models the behavior alternatives and the conditions that determine the current behavior, as expressed at the topmost branching point of the figure. Hence, the same service can have multiple implemented behaviors whose activation is driven by the logics expressed within the managers.

Besides the StandardBehavior described above, the right side of Figure 3 reports a box for a ResourceConstrained behavior that will be executed in case of scarce resources and that excludes the interactions with the image server (i.e. the white lifeline and the bold labeled messages in Figure 3)[3].

[3] For sake of readability, in Section 5 we simplify the conditions for the activation of ResourceConstrained behavior by basing only on the display characteristics.

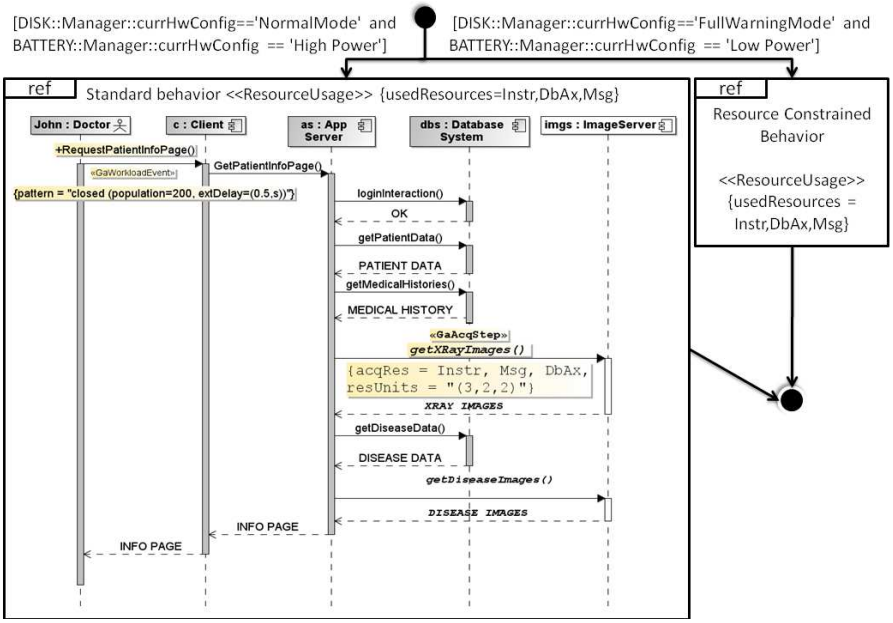

Fig. 3. The context-aware RequestPatientInfoPages service behavior.

3.4 Adding Performance Annotations

The eHealth model described so far also contains additional information related to performance. Such information is necessary to obtain performance models through automated model-to-model transformations [5]. In particular, the previously illustrated diagrams include:

- The workload (*GaWorkloadEvent*) for each service (Figure 3).
- The resource demand vector (`GaAcqStep`) that represents the amount of resources that an operational step needs to be completed (Figure 3); in particular, a resource demand vector provides values (i.e.*resUnits* tag of *GaAcqStep* [4]) for the ordered list (*acqRes*) of available *logical resources* (i.e. Instr, DbAx, and Msg *ResourceUsage* in Figure 3) necessary to execute the step.
- The multiplicity, service time and scheduling policy of each *hardware resource* such as CPUs, DISKs and NETWORKs (e.g. `wanBandwidth` in Figure 1).

Resource demand vectors represent the platform-independent annotations related to performance, in that they are abstract quantifications of resource consumption. In order to associate these annotations to platform specifications and build a solvable performance model (following the approach in [19]), the characteristics of platform devices have to be also specified (see [4]).

[4] We assume that the resource units are implicitly released at the end of each step.

In a context-aware domain the platform device characteristics can change depending on the context. In our case, for example, the available network connections can have different non functional properties that affect the quality of the service provision. In particular, the RequestPatientInfoPages performance can be affected by the network bandwidth `wanBandwidth` (Figure 1) whose value is bound to the *CurrentNodeLocation* association and varies when the doctor moves across the other allowed physical locations (i.e. *AllowedNodeLocation*).

4 An unique model for mobility and context-awareness

On the basis of the modeling approach introduced above, all considered awareness (in this paper the physical location-, the logical location- and the hardware platform-awareness) must be properly combined. Each of them can be defined in isolation or, through remote firing, can affect the other ones. For sake of performance analysis they can be considered together or in isolation, depending on the facets of interest of the software system. For example, one can investigate only the performance degradation due to an extremely high physical mobility of users without considering at all the states of resources on portable devices.

Therefore the types of statecharts that model the evolution of context dimensions, as the ones described in Section 3, can be lumped when necessary for analysis purposes in one statechart that models the runtime evolution of a mobile context-aware software system.

Each state of such statechart (that we call *superstate*) represents a possible context, and it is obtained from the combination of a certain number of states (in this paper three states), one for each statechart modeling a context dimension evolution (in this paper physical mobility, logical mobility, hardware platform evolution). Obviously not all the combinations are allowed, for example a certain configuration of hardware devices cannot allow a certain deployment of software components to devices. Therefore, in order to build a consistent unique model, only superstates that are feasible combinations of states have to be considered.

Once this set of superstates has been defined, a list of provided services and their corresponding behaviors have to be associated to each state. In fact, if multiple behaviors for some services are available, then the behavior to be adopted must be specified in each superstate where the service can be provided. We remark that this type of association does not need human processing, as it can be automated by parsing an Interaction Overview Diagram (see Figure 3). The latter, in fact, represents the behavior alternatives guarded by predicates over model variables. A superstate is uniquely characterized by the values assumed from model variables. Hence, the model variable values that determine a certain superstate drives the choice towards the appropriate behavior alternative among the ones modeled in the Interaction Overview Diagram.

Transitions have to be defined in this unifying statechart. Being each superstate obtained by lumping a certain number of states of respective statecharts, the transitions outgoing these latter states have to be opportunely combined (along with their probabilities) to build up transitions outgoing the superstate. The Harel's theory on statecharts

[10], along with the Hermanns et al.'s work on stochastic statecharts [14], provide sufficient results to automate this step in most cases.

In Figure 4 we report the unifying statechart obtained by lumping some of the awareness managers introduced in Section 3.2. In particular, we have considered the doctor's Physical Mobility Manager and the PDA Display Hardware Configuration Manager (Figure 2). This choice allows, on one end to keep the example as simple as possible, and on the other end to keep into account two different types of awareness, as we will show in Section 5.

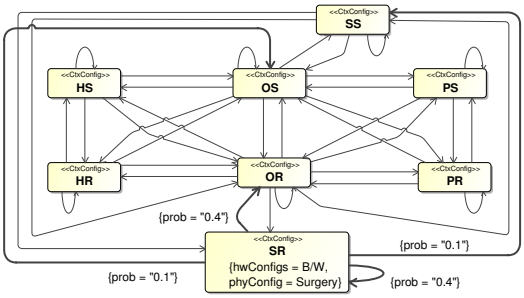

Fig. 4. The eHealth system unifying statechart.

The lumping process of these two managers brings to an unifying statechart where all eight potential superstates are feasible (i.e. the cartesian product of the manager state spaces). In Figure 4 each superstate is a different context *CtxConfig*, and is labeled with two letters: the first one recalls the state of the Physical Mobility Manager it comes from (i.e. H for Home, O for OpenAir, S for Surgery and P for PatientHome), the second one recalls the behavior adopted in the state as a consequence of the Display state (i.e. S means StandardBehavior, that is adopted when the display is in Color state, whereas R means ResourceConstrainedBehavior, adopted when the display is in B/W). For sake of illustration the SR superstate has been completely represented, along with example probabilities on its outgoing transitions.

5 Performance analysis

Several interesting experiments can be conducted on the model that we have built to study the system performance vs different model parameters. For example, the utilization of a certain platform device can be analyzed while varying the intensity of traffic due to user mobility, or the response time of a certain service can be analyzed in different superstates (or across superstates).

In fact, as outlined in Section 3.4, our model embeds all the performance parameters necessary to apply an automated transformation that generates a performance model, such as a Stochastic Petri Net or a Queueing Network. In this specific case we have used the approach illustrated in [5] [5].

In Figure 5 an Execution Graph [19] of the RequestPatientInfoPages service has been reported as obtained through a model transformation of the Sequence Diagram in Figure 3. An Execution Graph is a platform-independent model that represents the

[5] For sake of space we do not enter into technical details of such transformations; readers interested can refer to [3] for a recent survey on this topic.

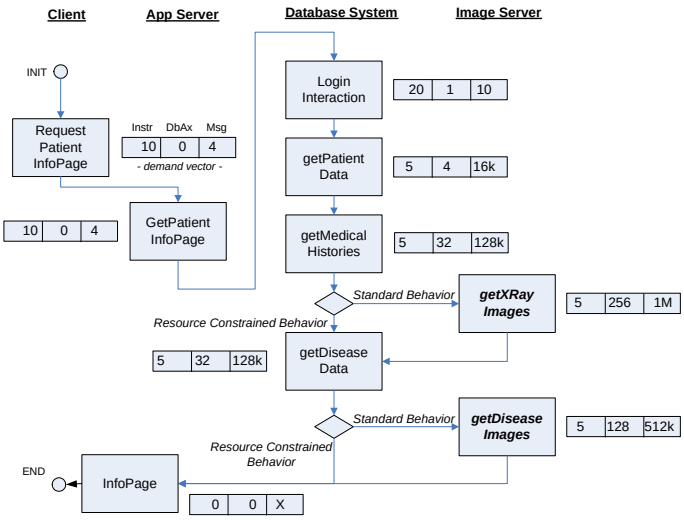

Fig. 5. Execution Graph of RequestPatientInfoPages service.

software dynamics along with its requests of resources. In Figure 5 square blocks represent the basic operations that the components perform to provide the service. Beside each block a demand vector is shown that reports (from the annotations of the Sequence Diagram) the amount of logical resources that are required to complete the block (see Section 3.4). In particular: (i) Instr represents the number of high-level instructions to be executed from a CPU, (ii) DbAx represents the number of mass memory blocks to be acceded on DISK (each block is sized 32 bytes), and (iii) Msg represents the number of bytes to be exchanged on the network (i.e. WAN/LAN).

Decision points have been generated in Figure 5 to embed the (bold labeled) blocks that are executed only in the StandardBehavior (similar labeling of the Sequence Diagram). The topmost labels indicates the names of the components that execute the underlying blocks.

The last block before the end of the graph represents the return of patient data to the doctor's PDA. The demand vector of such block cannot be uniquely identified, in that it brings over the WAN (connecting PDA and AppHost) the data retrieved. These data are different depending on the behavior executed. Therefore an X value is placed in the Msg field of this block demand vector, and X holds either 272kB or 1.772MB in case of, respectively, Resource Constrained and Standard Behavior[6].

The platform characteristics that we have considered in all experiments are reported in [4].

We have considered three scenarios for our software system, namely: Basic, High Mobility and Powerful Display. In the Basic scenario we devise a low mobility of the doctor (and hence of his PDA) that for most of time operates in the Surgery room, and

[6] These X values are obtained by summing up the amount of bytes of Msg fields of blocks executed in the two different behaviors.

an equal probability for the display to be color or b/w (i.e. equal probability for the two service behaviors). In the High Mobility scenario we introduce frequent doctor's relocations with respect to the Basic scenario. In the Powerful Display scenario, reported in [4], we instead introduce, with respect to the Basic scenario, a much higher probability for the Display to be in Color state (i.e. higher probability of adopting a StandardBehavior).

5.1 A Basic scenario

The transition probabilities for the Physical Mobility Manager and the Display Hardware Configuration Manager in the Basic scenario are annotated in Figure 2.

In consequence of the lumping operation, the transition probabilities of the unifying statechart of Figure 4 for the Basic scenario are the ones reported in Table 1. It is easy to observe that each probability has been obtained by multiplying the probabilities of the corresponding transitions in the original manager statecharts, following the canonical theory of merging probabilistic statecharts [14].

The statechart shown in Figure 4 can be interpreted as a Markov Model that describes the stochastic behavior of a software system with respect to its mobility and context-awareness. Hence, the solution of such model provides, among other, the steady-state probabilities of each superstate [20]. This result represents a measure of how often the system will be in a certain superstate, and it is therefore a crucial parameter for many types of non-functional analysis. The solution of such Markov Model in the Basic scenario leads to the steady-state probabilities reported in Table 2.

Table 1. Transition probabilities for the unifying statechart in the Basic scenario.

	HS	OS	SS	PS	HR	OR	SR	PR
HS	0.05	0.45			0.05	0.45		
OS	0.05	0.05	0.35	0.05	0.05	0.05	0.35	0.05
SS		0.05	0.45			0.05	0.45	
PS		0.45		0.05		0.45		0.05
HR	0.05	0.45			0.05	0.45		
OR	0.05	0.05	0.35	0.05	0.05	0.05	0.35	0.05
SR		0.05	0.45			0.05	0.45	
PR		0.45		0.05		0.45		0.05

Table 2. Steady-state probabilities of superstates in the Basic scenario.

HS	OS	SS	PS
0.0067	0.0608	0.4250	0.0067
HR	OR	SR	PR
0.0067	0.0608	0.4250	0.0067

We have considered the response time of the RequestPatientInfoPages service as the performance index of interest in our experiments. Minimum, maximum and average values of such index are evaluated overall the superstates for each scenario. Note, however, that since scenarios differ from each other only for transition probabilities (while keeping software and platform characteristics unchanged), the minimum and maximum response times are invariant across scenarios. As opposite, the average response time is computed as the weighted sum of response time in each superstate, where the weights are the steady-state probabilities. Therefore different values of average response time are obtained in different scenarios.

In order to obtain the response time in each superstate the Execution Graph shown in Figure 5 has to be synthesized to obtain an unique demand vector for each Execution Host [19]. For example, the demand vectors of the four blocks executed by Database System in Figure 5 have to be summed up to obtain the demand of resources addressed

to DbHost where Database System is deployed. Thereafter, each synthesized demand vector has to be combined with the corresponding platform characteristics specified in [4] to obtain the amount of time spent in each platform device to complete the service.

Summarizing, on the basis of the Steady-state probabilities of superstates (Table 2), the Demand vectors synthesized from the Execution Graph (Figure 5), and the Platform characteristics (see [4]), the response time values of RequestPatientInfoPages service in the Basic scenario are reported in Table 3, where values are expressed in seconds.

Since minimum and maximum values are invariant with respect to the scenario, the two leftmost values reported in Table 3 also hold for the other two scenarios. For these values we have reported in the bottommost row of Table 3 the superstate name where this value is achieved. A maximum response time of 82.069 seconds is obtained when the doctor's PDA is in OpenAir and its display works in color (i.e. OS state), whereas a minimum response time of 1.17 seconds is obtained when the doctor's PDA is in Surgery and its display works in black and white (i.e. SR state).

Table 3. Response time values in the Basic scenario.

	Max	Min	Average
Response Time	82.069	1.17	14.59
Superstate	OS	SR	-

The average response time obviously depends on the steady-state probabilities. In particular, as it can be observed in Table 2, in this case most of time is spent in Surgery (i.e. either SS or SR state) where the network bandwidth is quite large ([4]). Therefore the average response time is much closer to its lower bound than its upper bound.

5.2 High Mobility scenario

The transition probabilities of the Physical Mobility Manager in the High Mobility scenario are reported in Table 4, whereas the probabilities for the Display Hardware Configuration Manager are the ones adopted in the Basic scenario.

Table 4. Transition probabilities for Physical Mobility Manager in the High Mobility scenario.

	Home	OpenAir	Surgery	PatientHome
Home	0.5	0.5		
OpenAir	0.25	0.25	0.25	0.25
Surgery		0.5	0.5	
PatientHome		0.5		0.5

Similarly to the Basic scenario, after the lumping operation and the solution of the corresponding Markov Model, the steady-state probabilities of the High Mobility scenario are reported in Table 5.

Table 5. Steady-state probabilities of superstates in the High Mobility scenario.

HS	OS	SS	PS	HR	OR	SR	PR
0.1	0.2	0.1	0.1	0.1	0.2	0.1	0.1

For this scenario we have obtained an average response time of RequestPatientInfoPages service of $RT = 26.32$ seconds.

This value of the response time is quite larger than the one obtained in the Basic scenario, and this is mainly due to the following reason. In this scenario the doctor moves more often than in the Basic scenario, and therefore it experiences very different network bandwidths in a quite homogeneous distribution.

From a qualitative viewpoint this result is quite obvious, but we like to remark that our approach allows to quantify such differences among performance indices, and hence

it represents a powerful instrument in the hands of software designers to support their decisions. For example, sensitivity analysis can be conducted on response time while varying the probability of moving among pairs of locations.

6 Conclusions

We have introduced a framework for modeling and analyzing the performance of context-aware mobile software systems. Context-awareness is intended to be a composite concept, with different types of awareness concurring to its definition. No assumption underlies our framework about the types of awareness that can be modeled, as each awareness is simply represented by a statechart whose states and transitions are based on model variables.

Three main aspects represent the potential of our framework: (i) the rigorous definition in UML 2 of all necessary instruments to build a model of such an application is mostly based on reusing existing profiling, thus it does not represent "yet another profile" for context, but a promising approach to the modeling of context-related concepts, (ii) the process of lumping statecharts together in an unique stochastic model for context-awareness and mobility represents a powerful unifying approach to the more general modeling and analysis of non-functional properties, (iii) the existing mature approaches for automation in the performance model generation and solution allow to conceive, even in this specific domain, the performance analysis a viable and effective activity in the daily practice of software designers. Besides, our definition of context is extensible and/or shrinkable because any set of system attributes can enter the context as long as a manager statechart is defined for it.

This work opens the view on a plethora of problems that can be faced and solved on the basis of the promising results shown here.

First of all the validation of such approach against real case studies would lead feedback on its actual usability and effectiveness to capture performance issues.

Performance models that represent the resource contention should be addressed (possibly using existing model transformation approaches) in order to conduct a sensitivity study of such models vs. increases of system workload (i.e. a large number of users). Yet other types of performance indices could be useful in this domain, such as the utilization of certain devices across contexts.

However our models at the moment have some limitations on which we are working. First, certain scenarios involving remote firing transitions are complex to be managed in the lumping operation. We are working on parallel compositions of stochastic processes to remove this complexity. Besides, due to intrinsic constraints of UML 2, in our models the managers cannot change state during the execution of a service, but only between one invocation and another. We are trying to introduce this characteristic in our framework without needing a heavyweight extension of the UML metamodel. Moreover we are looking at more complex forms of adaptation that, for example, completely replace the internal structure and behavior of a certain component if needed [12].

We retain that such type of analysis, as well as the analysis of other non-functional attributes like reliability, can be of great support to the decision of system modelers. As shown also in our example, the validation of certain non-functional properties over a

system model allows not only to qualitatively validate possible modelers' intuitions, but also to quantitatively study the trends of non-functional metrics depending on context changes.

References

1. Architectural and Analysis Description Language, http://www.aadl.info/ .
2. DynamIc VAriability in complex, adaptive systems, Research Project, http://www.ict-diva.eu/.
3. Balsamo, S., Di Marco, A., Inverardi, P., Simeoni, M. Model-based performance prediction in software development: A survey. *IEEE Transactions on Software Engineering*, pages 295–310, 2004.
4. Berardinelli, L., Cortellessa, V., Di Marco, A. An Unified Approach to Model Non-Functional Properties of Mobile Context-Aware Software. Technical Report 003-2009, Computer Science Department, University of L'Aquila, http://www.di.univaq.it/cortelle/docs/report-003-2009.pdf.
5. Cortellessa, V., Mirandola, R. PRIMA-UML: a performance validation incremental methodology on early UML diagrams. *Science of Computer Programming*, 44(1):101–129, 2002.
6. Di Marco, A., Mascolo, C. Performance analysis and prediction of physically mobile systems. In *WOSP*, page 132. ACM, 2007.
7. Fuggetta, A., Picco, G.P., Vigna, G. Understanding code mobility. *IEEE Transactions on software engineering*, 24(5):342–361, 1998.
8. Grassi, V., Mirandola, R., Randazzo, E. Model-Driven Assessment of QoS-Aware Self-Adaptation. In *Software Engineering for Self-Adaptive Systems*, page 222. Springer, 2009.
9. Grassi, V., Mirandola, R., Sabetta, A. A UML profile to model mobile systems. *Lecture notes in computer science*, pages 128–142, 2004.
10. Harel, D. Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274, 1987.
11. Hirsch, D., Kramer, J., Magee, J., Uchitel, S. Modes for software architectures. *Lecture Notes in Computer Science*, 4344:113, 2006.
12. Inverardi, P., Mancinelli, F., Nesi, M. A declarative framework for adaptable applications in heterogeneous environments. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages 1177–1183. ACM New York, NY, USA, 2004.
13. IST-MUSIC Project. Middleware Support for Self-Adaptation in Ubiquitous and Service-Oriented Environments. http://www.ist-music.eu/.
14. Jansen, D.N., Hermanns, H. QoS modelling and analysis with UML-statecharts: the StoCharts approach. SIGMETRICS Perform. *Eval. Rev*, 32(4):28–33, 2005.
15. Lundesgaard, S.A., Lund, K., Eliassen, F. Service Plans for Context-and QoS-aware Dynamic Middleware. In *ICDCS Workshops 2006*, pages 70–70, 2006.
16. Mikic-Rakic, M., Malek, S., Medvidovic, N. Architecture-driven software mobility in support of QoS requirements. In *Proceedings of the 1st international workshop on Software architectures and mobility*, pages 3–8. ACM New York, NY, USA, 2008.
17. Object Management Group, Inc. *UML Profile for MARTE, ptc/08-06-09*, 2008.
18. Picco, G.P., Murphy, A.L., Roman, G.C. LIME: Linda meets mobility. In *ICSE*, pages 368–377. ACM New York, NY, USA, 1999.
19. Smith, C.U., Williams, L.G. *Performance Solutions: a practical guide to creating responsive, scalable software*. Addison-Wesley Boston, MA;, 2002.
20. Tjims, H.C. Stochastic models: an algorithmic approach. *IMA Journal of Management Mathematics*.