A Unified Approach to Model Non-Functional Properties of Mobile Context-Aware Software *

Luca Berardinelli, Vittorio Cortellessa, and Antinisca Di Marco

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

Abstract. Modeling context-awareness is becoming a primary activity for software engineers that design applications for mobile devices. In fact, software applications running on such devices need to be aware of their context (that may rapidly change) to adapt their services and offer the best quality (intended as a combination of non-functional properties) in any context. Thus the need of instruments to manage mobility, context-awareness and non-functional characteristics is critical to build software systems in ubiquitous and mobile domain. In this paper we introduce a framework to uniformly model different types of mobility and context-awareness so that the modeling and analysis of non-functional properties of such systems can be supported in an integrated environment. To enable non-functional analysis, we devise the integration of non-functional parameters in the modeling framework. In particular, we present an UML implementation of our framework within the MagicDraw modeling environment, with the support of existing UML profiles for modeling context-awareness and non-functional properties. We finally show an example of modeling and analysis of reliability in the eHealth domain.

1 Introduction

The evolution of portable devices and their increasing pervasiveness in everyday life have motivated 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 these aspects require specific characteristics of the deployed software systems that must be considered along the whole software lifecycle, starting from the architecture design phase.

Mobility can be either physical or logical. Physical mobility takes into account the transfers of a portable device among a certain number of physical locations. Logical mobility is achieved through online re-deployment actions on software components [20].

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

The limitation of hardware resources has brought to develop specific releases of software products. The available amount of such resources can vary at runtime, thus new approaches have been introduced to produce software able to adapt to changes in the available resources (and in the execution environment).

Mobility and limited resources obviously have a large impact on the quality of software systems (intended as a combination of non-functional properties). Their bad effects 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 quality even in presence of changes and degradations in the surrounding environment.

Goal of this paper is to introduce an unifying framework to model mobility and context-awareness aspects of software systems. The framework is aimed at producing software models that embed, beside these two aspects, the parameters needed to non-functional analysis. Beside the specific characteristics of mobility and context-awareness, in this framework the interdependencies between them can be captured and the cross-effects can be considered. Thus certain types of analysis, that were not feasible with specific models, are enabled in our framework [1].

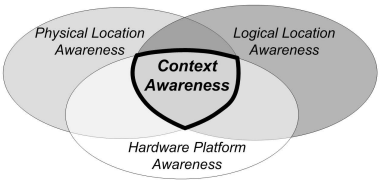

Fig. 1. Context as a combination of multiple attributes.

In Figure 1 we represent our idea of context, as the combination of attributes that can come from different domains. In particular, in this paper we focus on physical and logical location-awareness and hardware platform awareness. Our approach [5] does not limit the context definition to these domains, as it is extensible to any attribute or set of attributes that may contribute to the context definition. This modular approach allows also to provide restricted context representations in all cases where only few context attributes are necessary (e.g., logical mobility is not allowed).

Non-functional properties of mobile context-aware software systems obviously also depend on the characteristics of software services provided in each specific context. Different implementations of the same service may be available, possibly exploiting code mobility paradigms [10], so they can be dynamically

[1] Our approach has been conceived for multiple types of non-functional analysis [5], although in this paper (for sake of space) we only provide an example of reliability analysis.

chosen in order to provide the service implementation that better fits the execution context [2]. We name *context-aware, adaptable applications* the software systems able to provide services in these settings.

The paper is organized as follows: Section 2 briefly describes the basic ideas behind our framework, Section 3 presents the UML implementation of the framework driven by an eHealth application example, in Section 4 we show how to conduct a reliability analysis, Section 5 introduces the related work, and finally Section 6 concludes the paper.

2 A Unifying Modeling Approach to the Context-Awareness

In our approach, context-awareness is intended as the property of software to manage knowledge related to the context, where the context is defined here as a combination of (physical and logical) location and hardware platform characteristics.

The framework proposes a uniform representation of the location and hardware platform awareness (and the mechanisms that rule them), and allows to (separately or jointly) manage these properties within the same framework. Context-awareness of software systems is expressed through a set of attributes associated to system elements such as software components and hardware platform devices. The evolution of an attribute (or a set of attributes) is modeled by a stochastic statechart whose states model the attribute "value" (e.g., the status of a resource or the logical location for a component) whereas the transitions model the events (such as location change or resource degradation) triggering the attribute changes as well as the probability of the event occurrence. Of course, changes in the context attributes may induce adaptation actions.

On the basis of the modeling approach introduced above, physical location-, logical location- and hardware platform-awareness must be properly combined depending on the type of context-awareness suitable for the modeled application. Each of them can be defined in isolation or, through remote firing, can affect the other ones. Even for sake of reliability analysis, these facets can be considered together or in isolation, depending on the facets of interest of the software system. For example, one can investigate only the system reliability fluctuation due to an extreme physical mobility of users without considering at all the state of resources on portable devices.

Therefore the three types of statecharts have to 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 three states, one for each statechart (i.e. physical mobility, logical

[2] Note that our approach in its current version does not intend to address run-time problems, but it is a support for developing and maintaining such software systems. Runtime usage of this approach introduces additional issues (such as the complexity of analysis models) that we devise as future work.

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 service are available, then the behavior to be adopted must be specified in each superstate where the service can be provided.

In Figure 2 the synthesis of states from the three statecharts into an unique superstate is represented in an UML-like notation. A superstate represents the context as partitioned in four sectors, one for each type of awareness plus (at bottommost of figure) the list of provided services and their behaviors chosen for the specific superstate. Awareness managers (represented as statecharts) are introduced in each sector. Across physical and logical location awareness sectors the Dynamic Deployment Diagram illustrated in Figure 4 is also sketched.

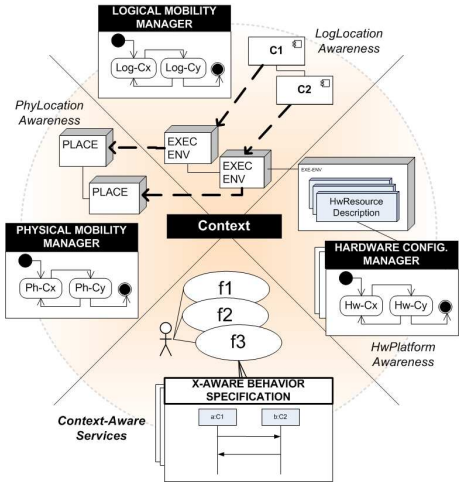

Fig. 2. The Context Superstate.

Transitions have to be defined in this unifying statechart. Being each superstate obtained by lumping three states of their respective statecharts, the transitions outgoing these latter states have to be opportunely combined (along with their probabilities) to build up transitions outgoing the superstate. Note that the problem of lumping different statecharts is very similar to the problem of combining stochastic behaviors of concurrent processes. The latter has been faced with different approaches, among which Stochastic Process Algebra seems to be a promising notation to be applied to our case as well [19].

More details on the notation-independent framework can be found in [5].

3 UML-based modeling of a context-aware software application

In this section we present the UML 2.0 implementation of our approach to the modeling of non-functional properties of context-aware software.

The UML modeling is organized in *Service*, *Component* and *Deployment Views*, as shown in Figure 3.

The **Service View** (SV) models 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* is assigned to the nomadic users that exploit the system services while moving with their mobile devices [17, 13].

The **Component View** (CV) represents the software architecture (Component Diagram, CD). It is integrated with mobility annotations that allow to distinguish logically mobile from fixed software artifacts [13]. A *Logical Mobility Manager* [13] is associated to each component whose implementation is (even partially) mobile.

The **Deployment View** (DV) models: (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 (hwDD). A *Hardware Configuration Manager* describes a resource whose state (e.g. its current amount) may vary at runtime.

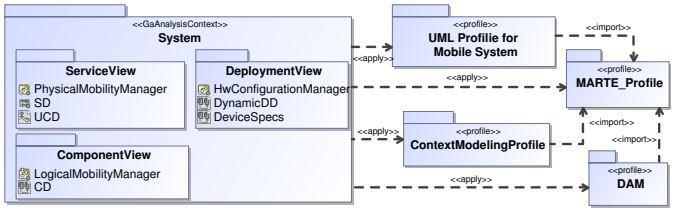

Fig. 3. The System Model.

These views are illustrated in the following subsections through a set of UML diagrams modeling a eHealth application [3].

The diagrams are suitably annotated with additional information from several profiles to enable model-based reliability analysis [7] of the specified system.

The applied profiles are MARTE [3], that includes the Dependability Analysis Model [6], and the UML Profile for Mobile Systems [13]. We have also devised

[3] The complete UML model can be downloaded at *http://www.di.univaq.it/cortelle/docs/eHealthSystemModel.rar*.

and applied a preliminary version of a Context Modeling profile to support the modeling of the *manager* statechart [4].

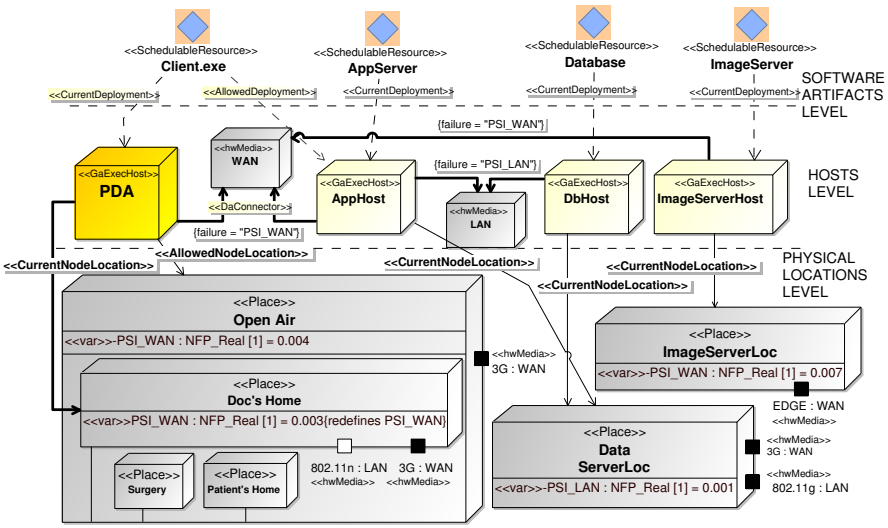

Fig. 4. The Dynamic Deployment Diagram.

3.1 Modeling the software architecture

The reference example is a eHealth system that provides a service called RequestPatientInfoPage: a doctor is able to retrieve mixed media information on his/her patients, such as text with or without different kind of images that refer to their personal data, their medical histories and patient-related diseases through his PDA (*MobileElement*)

The RequestPatientInfoPages service is supposed to be the most frequently invoked (basing on the user profile). In our modeling approach, services are 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 component failure probability needed for the analysis methodology [7] applied in Section 4.

In Figure 4 the *dynamic* DD of the application is shown. It is inspired to the diagram introduced in [13], and it basically contains two types of information: (i) the allocation of the software artifacts (*SchedulableResources*) on the execution

[4] Hereafter the `typewritten` words are model variables (following the MARTE [3] recommendations), whereas the *italicized* ones are stereotypes of profiles.

environments (*GaExecHost*) through deployment relationships (*CurrentDeployment*, *AllowedDeployment*) between the Software Artifacts and the Hosts levels, and (ii) the positioning of the execution hosts (e.g. PDA) on different physical locations using associations (*CurrentLocation*, *AllowedLocation*) between Hosts and Physical Locations levels. The dynamic nature of the dynDD derives from the need to change the current and allowed relationships between levels whenever logical and/or physical mobility events take place.

In Figure 4, the RequestPatientInfoPages service on the user-side is available if the user PDA is able to connect (*DaConnector*) to a WAN network (*HwMedia*). Different *Places* can provide different types of network connections (i.e. typed Ports of the *Places*), but not all of them are 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. Moreover the available connections can have different non-functional characteristics that affect the properties of the service provision: in our case, the service reliability is influenced by the network failure probability (i.e. `PSI_WAN`) [7] whose value is bound to the *CurrentNodeLocation* association and varies when the doctor moves across the other allowed physical locations (i.e. *AllowedNodeLocation*).

Lastly, Figure 5 shows the hardware components making up the doctor's PDA [3]. In particular the PDA can connect to a WAN network using its *HwEndPoint* 3G card. Indeed we specify that a 3G WAN network is available (i.e. the black-colored port named 3G:WAN) both when the doctor is at his/her home or outside, but with different reliability characteristics (see the redefined `PSI_WAN` variable in Figure 4). We also declare here those variables (e.g. DISK's `memorySize`, BATTERY's `capacity`) that will be referred by the hardware configuration managers (see Section 3.2).

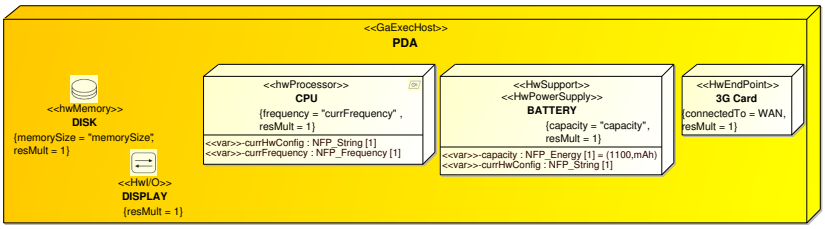

Fig. 5. The hardware platform specification of the doctor's PDA.

3.2 Modeling the Context-Awareness

In this section we model the context-awareness of the eHealth application following our approach. In Section 3.3, we will describe how the *-awareness can influence different behavior specifications of the same system service.

Physical Location-Awareness - The modeling of the Physical Location Awareness takes inspiration from some previous works [13, 17]. We define a *Physical Mobility Manager* associated to each system nomadic user (i.e. in our case the doctor).

A UML StateMachine is defined where each state represents both the current physical location (*Place*) and physical 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 6 shows the doctor mobility pattern (i.e. the one of his/her PDA) where the physical transfer from his/her home to the patient's one is highlighted along with the probabilities of the moves (*GaStep*). [5]. A *PhyConfig* state defines the actual ends of the *Current-* and *AllowedNodeLocations* relationships among execution hosts and places on the dynDD (Figure 4, [13]).

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* [10]. We adopt the UML-based solution proposed in [13] based on a 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*, Figure 4). State transitions represent the possible redeployment of the mobile software artifacts through the communication channels (*HwMedia* on Figure 4) to other platform devices (*AllowedDeployment*).

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 specify in Figure 6(b) the *Hardware Configuration Managers* for hardware resources of Figure 5 whose internal configuration (*HwConfig*) can influence the service behavior.

Figure 6 illustrates the BATTERY, Display and CPU's managers as separate statechart that model the states and transitions of corresponding hardware components (Figure 5). Each state specifies a set of *nfpConstraints* (based on variables defined on the configured hardware component) [3] to hold the current configuration (see the battery manager in Figure 6(b)).

In addition, a remote firing transition (BATTERY2LowPowerDowngrade) highlights how the remaining BATTERY capacity (`currCapacity`) influences the status of the CPU by firing a remote transition that limits its clock frequency (`currFrequency`).

3.3 Modeling the service behaviors

The eHealth modeling is completed by the specification of *-aware service's behaviors. Figure 7 shows a Interaction Overview Diagram associated to the RequestPatientInfoPages, with two possible behaviors.

[5] Note that self-transitions on all states are represented, because they are implicitly activated with the remaining amount of probability to achieve 1 on the sum of outgoing transition probabilities of a state.

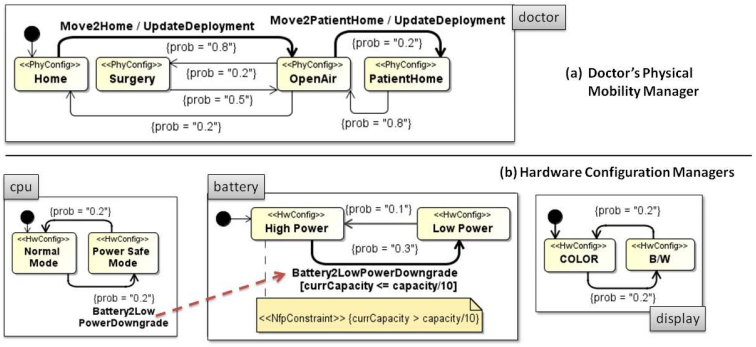

Fig. 6. Physical Mobility and Hardware Configuration Managers.

In the Standard Behavior the doctor, once logged in, invokes the distributed service, and the server-side components *DaComponent* are in charge of retrieving data from a local database and, if needed, from a image server for patients' x-rays or disease-related images. Finally the result is displayed on the client (Figure 7). The service behavior can be determined by the current context condition (see Figure 6) defined by the current values of the managers' model variables. Hence, the same service can have multiple implemented behaviors whose activation is driven by the logics expressed within the managers.

Figure 7 shows two alternative behaviors, enabled by conditions that refer to the actual state of several hardware configuration managers. In particular the StandardBehavior allows the retrieval of both text and images about the patient only if enough DISK space and BATTERY power are available (reading the current value of the `currHwConfig` vars), otherwise the ResourceConstrained behavior will be executed that excludes the interaction with the image server (the "white" lifeline and related messages in Figure 7) [6].

3.4 Lumping the awareness in a unique model

For the sake of the reliability analysis made in Section 4 we lump some of the awareness managers introduced in Section 3.2 to obtain the unifying (UML) statechart. In particular, we have considered the doctor's Physical Mobility Manager and the PDA Display Hardware Configuration Manager (both in Figure 6(b)). 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 4. The lumping process brings to an unifying statechart with eight feasible potential superstates, i.e. the cartesian product of the manager state spaces.

[6] For sake of illustration, in Section 4 we simplify the conditions for the activation of ResourceConstrained behavior by basing only on the display characteristics.

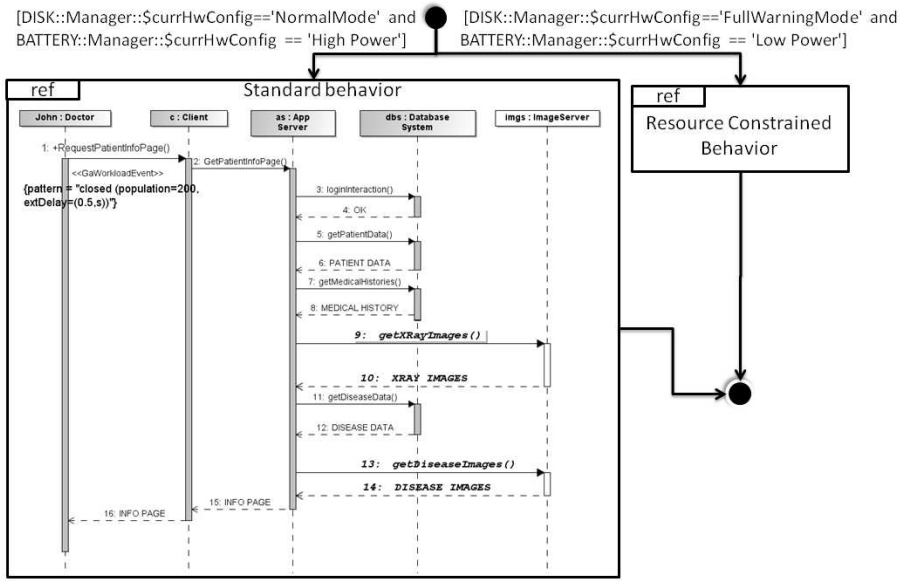

Fig. 7. The Interaction Overview Diagram for the RequestPatientInfoPage service.

4 An example of reliability analysis

In this section we show an example of reliability analysis, as a proof of concept that our approach not only allows to introduce all information necessary to model non-functional attributes, but it also enables non-functional analysis that supports critical decisions in a context-aware domain.

The target of our analysis here is a model introduced in [7] to study the reliability of a software architecture as a function of the reliability of its components, its connectors and other structural parameters. We have tailored that model to the case of context-awareness and we have used it to estimate the reliability of an example service (as modeled in Section 3) in different superstates.

As illustrated in Section 2, we assume that the set of services available to users can be different across the superstates due to changes in the resource characteristics. Moreover, different implementations of the same services can be provided in different superstates in order to adapt the service behavior to different hardware configurations. In order to denote superstate-specific services (and components implementing them), as well as other system parameters, we introduce the following notation: $x|_\sigma$. It denotes the interpretation of term x in the σ superstate.

Let S_k be one of the system services (e.g. RequestPatientInfoPages of Section 3); let $S_k|_\sigma$ be the implementation of S_k provided in the σ superstate; and finally let $C_i|_\sigma$ $(i = 1, \ldots, N)$ be the set of components implementing $S_k|_\sigma$.

Let $\theta_i|_\sigma$ be the probability of failure of $C_i|_\sigma$, and let $bp_i|_\sigma$ be the number of activations of $C_i|_\sigma$ within the dynamics of $S_k|_\sigma$.

Let $\psi_{<i,j>}|_\sigma$ be the probability of failure of the connector between $C_i|_\sigma$ and $C_j|_\sigma$, and let $int_{<i,j>}|_\sigma$ be the number of interactions between these two components within the dynamics of $S_k|_\sigma$.

Hence, we have reformulated the model in [7] to the case of context-aware service reliability as follows:

$$\mathcal{R}(S_k|_\sigma) = \prod_{i=1}^{N}(1-\theta_i|_\sigma)^{bp_i|_\sigma} \cdot \prod_{<i,j>}(1-\psi_{<i,j>}|_\sigma)^{int_{<i,j>}|_\sigma} \quad (1)$$

In a certain superstate, the reliability of a service is given by the probability that all components and connectors implementing it do not fail all the times they are used (i.e. numbers of activations over components and interactions over connectors).

The $\theta_i|_\sigma$'s can be extracted from an annotated Component Diagram modeling the software architecture of the application, $\psi_{<i,j>}$'s from a Deployment Diagram like the one in Figure 4, and structural parameters $bp_i|_\sigma$ and $int_{<i,j>}$ from a Sequence Diagram like the one in Figure 7.

Following the process illustrated in Section 2, the managers introduced for this example in Section 3 can be lumped in a unique statechart that represents the context management of the eHealth application. For sake of space we do not show this process here, but we compute the reliability of RequestPatientInfoPages service (RPIP-service) in two different contexts, that are: OpenAir/ColorDisplay (CD) and OpenAir/BWDisplay (BWD). We assume that in the former context the Standard Behavior of the Application Server is adopted, whereas the Resource Constrained Behavior is adopted in the latter case, thus we can show the effect of changing context on the service reliability.

In Table 1 the values adopted for the reliability model parameters have been summarized, where the parameters referring to the components are placed in the top rows and the ones referring to the connectors in the bottom rows. Note that all values are invariant with respect to the context σ, with the exception of the number of activations and interactions of the Application Server. This is due to the different behaviors that this component adopts in the considered contexts.

By applying the equation (1) the following values for the service reliability are obtained: $\mathcal{R}(RPIP-service|_{CD}) = 0.9386$ and $\mathcal{R}(RPIP-service|_{BWD}) = 0.9557$. The service reliability is higher in the context where a back-and-white display is used basically due to the adapted behavior of the Application Server that does not interact with the Image Server to collect the patient's images. This behavior allows to avoid additional activations and interactions that in the best case do not increase the service reliability, and in all other cases (due to non-zero failure probabilities of components and connectors) worsen it, as shown in this simple numerical example.

This example results would be even different if superstates with different physical locations of a mobile device are considered, as in that case the failure probabilities of connectors may also change. Yet many other scenarios can be

Components	**Client**	**AppServer**	**Image Server**	**DB**
$\theta_i\|_\sigma$	0.010	0.003	0.003	0.001
$bp_i\|_\sigma$	1	1	2 ($\sigma = CD$) 0 ($\sigma = BWD$)	4

Connectors	**Client, AppServer**	**AppServer, Image Server**	**AppServer, DB**
$\psi_{<i,j>}\|_\sigma$	0.010	0.003	0.001
$int_{<i,j>}\|_\sigma$	2	4 ($\sigma = CD$) 0 ($\sigma = BWD$)	8

Table 1. Reliability model parameters.

studied by simply comparing the service reliability in different contexts (e.g. the failure probability of components may change if different versions of the same service are implemented with different types of components).

Beyond the numerical analysis shown in this section, our approach opens multiple possibilities for the system analysis. The construction of a single statechart for context management allows to make analysis across different contexts. For example, by solving the lumped statechart as a Markov Chain, the steady state probabilities can be obtained that represent the probability to sojourn in each state. The average reliability of a certain service across contexts can be simply obtained by summing the service reliability in each superstate weighted by the steady state probability.

5 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 [15]. Basing on a combined resource and context model the MUSIC middleware monitors the context and the resources to catch their changes and adapts the application to fulfill the users' QoS (Quality of Service) requirements. The approach uses QoS predictors and utility functions to support the adaptation process. The adaptation is based on the concept of service plan [16], that is a platform-independent specification containing information on service configurations, its dependencies on the environment and its QoS and characteristics.

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 [16], the information the MUSIC middleware needs to properly work is specified in the service plan, but such information is collected at the design time.

As opposite to MUSIC project and other middleware contributions (e.g. ReMMoC [11]), we provide a support to model and analyze non-functional 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. Differently from other approaches where the dynamic adaptation is handled at code level [14], DiVA exploits both Model-Driven and Aspect-Oriented technologies to define an architectural model (including base, variant and adaptation models) at design time [9]. 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 [18, 4]. However, such approach does not provide any support for non-functional analysis.

Grassi et al. in [12] 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 [12] 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 [12] 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 [12] 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 [8] 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 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.

6 Conclusions

We have introduced an unifying approach to the modeling of location- and hardware platform-awareness aimed at analyzing non-functional properties of software systems. As a proof of concept, we have provided an example of reliability analysis.

One of the peculiar properties of our approach to the context and mobility modeling is its modularity. We allow to separately model with statecharts any characteristic (single or multiple attributes) that may change and affect non-functional properties. Such separate models can be used in isolation or variously combined to study the behavior of such system under different types of changes.

Besides, with previous modeling approaches cross-dependencies among attributes were hard to capture because they were represented with different models or not represented at all. With our unifying approach all attributes are integrated within the same model, dependencies are explicitly modeled, and basic adaptation actions that derive by context changes can be managed with the same instruments.

This work has been conceived to be extendable, and two main directions we see at the moment for extending it. On one side, the concept of context can be extended by embedding other attributes that can affect non-functional properties. As long as this evolution can be represented through a state machine, the mechanisms presented here can be applied to integrate these additional concepts to the context evolution. We are also working to increase the complexity of statechart remote firing mechanisms without increasing the complexity of the lumping step through the support of process algebras-based techniques. On the other side, these types of analysis can be carried out for other non-functional properties, such as performance, and further tradeoff analysis between different attributes can be devised.

As future work we plan to make explicit the adaptation actions (beside the simple replacement of a service behavior) in order to take into account their non-functional properties. Also, we are studying the possibility to adopt this approach at runtime as a support, for example, of self-adaptive systems. Finally, the idea of using a Markov Reward Model [12] as a general target model is very interesting to allow more complex types of analysis.

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. UML Profile for MARTE, OMG document ptc/08-06-09, Object Management Group, Inc. (2008), *http://www.omgmarte.org/Documents/Specifications/08-06-09.pdf*.
4. Nelly Bencomo. *Supporting the Modelling and Generation of Reflective Middleware Families and Applications using Dynamic Variability.* Phd thesis, Computing Department, Lancaster University, 2008.

5. Di Marco A. Berardinelli L., Cortellessa V. 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.
6. Simona Bernardi, José Merseguer, and Dorina C. Petriu. Adding dependability analysis capabilities to the marte profile. In *Proc. of MoDELS '08*, pages 736–750. Springer-Verlag, 2008.
7. Vittorio Cortellessa, Harshinder Singh, and Bojan Cukic. Early reliability assessment of uml based software models. In *Proc. of WOSP '02*, pages 302–309. ACM, 2002.
8. Jeff Magee Dan Hirsch, Jeff Kramer and Sebastian Uchitel. Modes for software architectures. In *European Workshop on Software Architecture (EWSA '06)*, pages 113–126. LNCS 4344, 2006.
9. Franck Fleurey, Vegard Dehlen, Nelly Bencomo, Brice Morin, and Jean-Marc Jézéquel. Modeling and validating dynamic adaptation. *Models@Runtime Workshop in conjunction with MODELS 2008*, 2008.
10. Alfonso Fuggetta, Gian Pietro Picco, and Giovanni Vigna. Understanding code mobility. *IEEE Trans. Softw. Eng.*, 24(5):342–361, 1998.
11. Paul Grace, Gordon S. Blair, and Sam Samuel. A reflective framework for discovery and interaction in heterogeneous mobile environments. *Mobile Computing and Communications Review*, 9(1):2–14, 2005.
12. Vincenzo Grassi, Raffaela Mirandola, and Enrico Randazzo. Model-driven assessment of qos-aware self-adaptation. In *Software Engineering for Self-Adaptive Systems*, pages 201–222, 2009.
13. Vincenzo Grassi, Raffaela Mirandola, and Antonino Sabetta. A uml profile to model mobile systems. In *UML*, pages 128–142, 2004.
14. P. Inverardi, F. Mancinelli, and M. Nesi. A declarative framework for adaptable applications in heterogeneous environments. In *Proc. of SAC '04*, pages 1177–1183, 2004.
15. IST-MUSIC Project. Middleware Support for Self-Adaptation in Ubiquitous and Service-Oriented Environments. http://www.ist-music.eu/.
16. Sten A. Lundesgaard, Ketil Lund, and Frank Eliassen. Service plans for context- and qos-aware dynamic middleware. *Distributed Computing Systems Workshops, International Conference on*, 0:70, 2006.
17. Antinisca Di Marco and Cecilia Mascolo. Performance analysis and prediction of physically mobile systems. In *Proc. of WOSP*, pages 129–132, 2007.
18. Brice Morin, Franck Fleurey, Nelly Bencomo, Jean-Marc Jézéquel, Arnor Solberg, Vegard Delhen, and Gordon Blair. An aspect-oriented and model-driven approach for managing dynamic variability. In *MODELS'08*, volume 5301/2008 of *LNCS*, pages 782–796, 2008.
19. Rocco De Nicola, Joost-Pieter Katoen, Diego Latella, Michele Loreti, and Mieke Massink. Model checking mobile stochastic logic. *Theor. Comput. Sci.*, 382(1):42–70, 2007.
20. Gian Pietro Picco, Amy L. Murphy, and Gruia-Catalin Roman. Lime: Linda meets mobility. In *Proc. of ICSE '99*, pages 368–377, New York, NY, USA, 1999. ACM.