A Development Process for Self-adapting Service Oriented Applications

M. Autili, L. Berardinelli, V. Cortellessa, A. Di Marco, D. Di Ruscio, P. Inverardi, and M. Tivoli

Dipartimento di Informatica
Università degli Studi di L'Aquila,
67100 L'Aquila, Italy
{autili,berardinelli,cortelle,dimarco,diruscio,inverard,
tivoli}@di.univaq.it

Abstract. Software services in the near ubiquitous future will need to cope with variability, as they are deployed on an increasingly large diversity of computing platforms, operate in different execution environments, and communicate through Beyond 3G (B3G) networks. Heterogeneity of the underlying communication and computing infrastructure, physical mobility of platform devices, and continuously evolving requirements claim for services to be adaptable according to the context changes without degrading their quality. Supporting the development and execution of software services in this setting raises numerous challenges that involve languages, methods and tools. However these challenges taken in isolation are not new in the service domain. Integrated solutions to these challenges are the main targets of the IST PLASTIC project.

In this paper we introduce the PLASTIC development process model for self-adapting context-aware services, in which we propose model-based solutions to address the main issues of this domain in a comprehensive way. We instantiate the process model by providing methodologies to generate Quality of Service models and adaptable code from UML service models. All these methodologies are supported by an integrated framework which is based on an UML profile that we have defined for the PLASTIC domain.

1 Introduction

Nowadays, software services need to cope with variability, as services get deployed on an increasingly large diversity of computing platforms and operates in different execution environments. Heterogeneity of the underlying communication and computing infrastructure, mobility inducing changes to the execution environments (and therefore changes to the availability of resources) and continuously evolving requirements require services to be *self-adaptive* according to the context changes. At the same time, a service should be *dependable* in the sense that it should meet the user's Quality of Service (QoS) requirements and needs. Moreover, satisfying user expectations is made more complex given the highly dynamic nature of service provision.

B. Krämer, K.-J. Lin, and P. Narasimhan (Eds.): ICSOC 2007, LNCS 4749, pp. 442–448, 2007.

Supporting the development and execution of such adaptable services raises numerous challenges that involve models, methods and tools. However these challenges, taken in isolation, are not new in the service domain. Integrated solutions to these challenges are the main targets of the IST PLASTIC project, whose main goal is the rapid and easy development/deployment of self-adapting services for B3G networks [20].

Broadly speaking, a "standard" development process focuses on activities that are traditionally divided into *development-*, *deployment-* and *run-time* activities. Each activity works on suitable system artifacts, which can be coupled with models suitable for development purposes. The evolutionary nature of services in the near ubiquitous future makes unfeasible a standard development process since dealing with self-adaptiveness would require to predict the functional and non-functional system behavior before the system is in execution. Whenever a change occurs, if service evolution has to be supported by means of adaptation, all the artifacts/models might be exploited also by the deployment and run-time activities, hence leading to a "non-standard" development process view. Thus, the main challenges in this direction are related to the support that can be offered to service developers to satisfy the user expectations in a such heterogeneous and dynamic environment.

In this paper we introduce the PLASTIC development process that relies on model-based solutions to build self-adapting context-aware services. The introduced process encompasses methodologies to generate QoS models and adaptable code from UML-based specifications. All these methodologies are supported by an integrated framework which is based on an UML profile of the PLASTIC domain.

The work described in this paper relates to multiple research areas of the existing literature, that are: (i) web-service development technologies, (ii) model-driven development, (iii) performance and reliability analysis techniques, and (iv) (self-)adapting software. For sake of space, we obviously cannot address all the recent related work in the above areas, thus in the following we shortly discuss and provide major references for each area.

Current (web-)service development technologies, e.g. [7,8,19,22,23] (just to cite some), address only the functional design of complex services, that is they do not take into account the extra-functional aspects (e.g., QoS requirements) and the context-awareness. Our process borrows concepts from these well assessed technologies and builds on them to make QoS issues clearly emerging in the service development, as well as to take into account context-awareness of services for self-adaptiveness purposes.

The PLASTIC development process adheres to the Model Driven Development (MDD) approach which claims to shift the focus of software development from coding to modeling [21]. In this respect, problems can be precisely described using specific terms and concepts more familiar to experts working in the considered domain and technological details which are unnecessary for the service description can be neglected. Model transformations are devised in our process in two directions: (i) to glue the different levels of abstractions and, by

encoding the knowledge about the technological assets, to permit the automated generation of the service code, (ii) to generate QoS models, at the same level of abstraction of the service models, that allow to validate extra-functional issues during the service development.

With regard to the latter point, up today performance and reliability models have been integrated in the PLASTIC process to support QoS validation. In this domain interesting progresses have been made in the last ten years due to the introduction of automated techniques and tools that allow to generate extra functional models from annotated software models (see, for example, [3] for performance and [6] for reliability). We have embedded some of these techniques in our service development process. Obviously some effort has been necessary to adapt the techniques to the specific domain of context-aware self-adapting services.

This work exploits also notions and concepts in the area of (self-)adaptation of software entities and self-healing system development, spanning adaptation of communication/interaction [15], performance [11], real-time behaviours [5], and synthesis of coordination/composition behaviour among semantic services [13].

The remainder of the paper is structured as follows: Sect. 2 describes the proposed development process and outlines the adopted technologies supporting it. Sect. 3 draws some conclusions and perspective works.

2 PLASTIC Development Process

In this section we introduce the PLASTIC development process for self-adapting context-aware services. By recalling Section 1, the main issues that this process addresses are: (a) service self-adaptiveness and context-awareness, and (b) service satisfaction of QoS requirements.

To address the former, at design time the possible contexts in which the service will run are specified. Models for context description are introduced to support this activity. At development time, the context specification is exploited to automatically derive, through model-to-code transformation, "generic" code that embodies a certain variability degree. Hereafter, we refer to it as adaptable code. Obviously, only the skeleton is automatically derived, i.e., its logic has to be coded by hand. At deployment time the adaptable code is processed to automatically extract, through adaptable code instantiation, the code that better fits a certain context.

The latter is addressed in two steps: (i) by allowing the designer to annotate the service model with QoS related information (i.e. QoS parameters and requirements), and (ii) by elaborating the annotated information at both design- and run-time through analysis tools whose aim is to predict and solve QoS models within the possible different contexts. The adopted QoS analysis tools use a large variety of models, from behavioral to stochastic, that can represent the system at very different levels of abstraction from requirements specification to code.

As already anticipated in Section 1, the ever growing complexity of software has exacerbated the dichotomy development/static/compile time versus execution/dynamic/interpret time thus concentrating as many analysis and validation

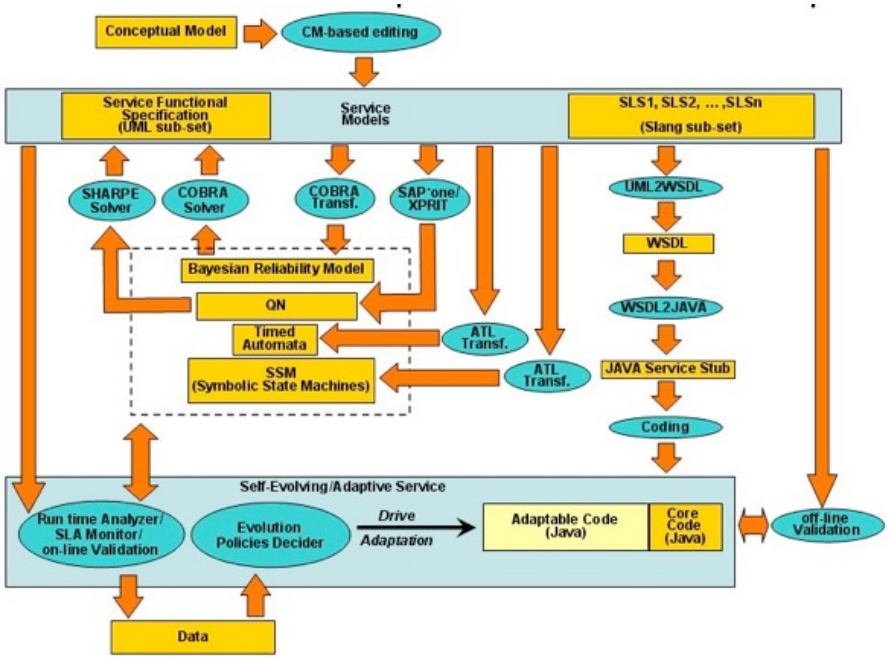

Fig. 1. The PLASTIC process for service development

activities as possible at development time. As opposite, if QoS has to be preserved through adaptation whatever the change mechanism is, at the time the change occurs, a validation mechanism must be devised at run-time. This means that models used at development time to support design decisions must be available at run-time for additional validation activities.

In Fig. 1 we illustrate the PLASTIC development process, where square boxes represent software artifacts/models and ellipses represent activities. Lifecycle time goes from the top to the bottom of the figure. All the process activities originate from a Conceptual Model where entities and relationships of the context-aware service domain are defined [2,16]. Based on these entities, a Service Model can be specified in terms of its Functional Specification and its Service Level Specification (SLS). The former describes behavioral aspects of the modelled service, whereas the latter its QoS characteristics.

The Service Model is specified by means of a UML2 [14] profile whose aim is to extend UML2 to cope with adaptable, context aware and component based software services both from structural and behavioral viewpoints along the entire software lifecycle, from requirements specification to deployment. This profile is an implementation of the Conceptual Model, and it is supported by the customization of an UML 2 tool environment (i.e. *Magic Draw*) that we have developed and described in [17]. For the sake of space, we do not describe here the profile.

Two main streams of activities originate from a Service Model, each addressing one of the issues introduced above.

In one stream of activities, Model-to-Model transformations are devised in order to derive models for performance and reliability analysis. In particular, Bayesian Reliability Models, Queueing Networks, Timed Automata, and Symbolic State Machines are considered in the current implementation of the process. Some of the Model-to-Model transformations are performed by means of the ATLAS Transformation Language (ATL) [10] that has been developed in the context of the MODELWARE European project [12].

In Fig. 1 we have reported some of the model transformation and analysis techniques that we have integrated within the PLASTIC process. As an example, SAP•one/XPRIT starts from annotated UML diagrams and generates a performance model that may be either a Queueing Network (QN) that represents a Software Architecture, if no information about the executing platform is available, or an Execution Graph (representing the software workload) and a Queueing Network (representing the executing platform) in the other case. The model solution provides performance indices that are parametric in the first case and numerical in the second one. A QN solver, like SHARPE, can provide values of performance indices. As another example, COBRA is a tool that, starting from annotated UML diagrams, generates a reliability model for component-based or service-based systems that takes into account the error propagation factor. COBRA embeds a solver that performs reliability analysis on the basis of the generated model.

Bayesian Reliability Models and Queueing Networks can also be analyzed at development time to refine/validate the Service Model characteristics that the analysis addresses. Timed Automata, Symbolic State Machines (SSM), and possibly the previous models will be made available at deployment- and run-time to allow the adaptation of the service to the execution context and for service validation. In particular, we are able to perform two kinds of validations, i.e., on-line and off-line validation (see [18] for details). Off-line validation is performed to generate test cases, before the service execution, by taking into account both the service model (in particular its SSM) and the service code. On-line validation is performed whilst the service is running and uses the generated test cases.

In the other stream of activities, Model-To-Code transformations are used to build both the core and the adaptable code of a service. The core code is the frozen unchanging portion of the service. The adaptable code portion can evolve in the sense that, basing on contextual information and possible changes of the user needs, the variability can be solved with a set of alternatives. A particular alternative might be suitable for a particular execution context and specified user needs. Each alternative can be selected by exploiting the analysis models available at run-time and the service capabilities performing the Run time Analysis/SLA Monitoring and the Evolution Policies Selection (see Fig. 1). When a service is invoked, the run-time analysis is performed (on the available models) and, basing on the analysis results, a new set of alternatives is synthesized and a new alternative is selected. The development of the adaptable service code is based on CHAMELEON [9], that is a resource-aware framework for adaptable Java applications.

Model-To-Code transformations are performed by means of a code generator based on the *Eclipse Java Emitter Template framework* (part of the EMF framework [4]). JSP-like templates explicitly define the code structure and get the data they need from the UML model of the specified service exported into EMF. With this generation engine, the generated code can be customized and then re-generated without losing already defined customizations.

We like to remark that one of the main novelties of this process is to consider SLS as part of a Service Model, as opposite to existing approaches where SLS consists, in best cases, in additional annotations reported on a (service) functional model. This peculiar characteristic of our process brings several advantages: (i) SLS embedded within a Service Model better supports the model-to-model transformations towards analysis models (in particular, the target model parametrization) and, on the way back, better supports the feedback of the analysis (i.e., reporting the analysis results on the Service Model); (ii) in the path to code generation, the SLS can drive the adaptation strategies.

3 Conclusions and Future Work

This paper proposed a development process defined in the context of the IST EU PLASTIC project [20] which aims at offering a comprehensive provisioning platform for context-aware and adaptable software services deployed over B3G networks. In particular, this work describes the instantiation of the process within an UML world. Models and techniques for developing, in UML, adaptable code of context-aware services which have to show optimal QoS within different contexts have been integrated. The approach is supported by languages and tools conceived to increase the automation in all the process steps. Service modeling is based on a PLASTIC UML profile that we have defined and whose main concepts have been inherited from other existing UML profiles and meta-models (e.g. see [1]).

Due to space limitation, in this paper, we have given an overall description of the thorough approach that supports the whole service lifecycle. The approach has been applied to a real-life example concerning the service-oriented development of an e-Health system. The treatment of this example is described in [17].

The instantiation of our process within UML can be improved by integrating a wider number of analysis techniques that may address other dimensions of QoS, such as availability and security. Besides, from a functional viewpoint, we intend to study how to tackle dynamic composition of context-aware services. We are also investigating the usage of non-UML methodologies and tools within the process, such as formal (functional and non-functional) specification of services. This would allow us to introduce in the process formal refinement and analysis techniques, such as model checking.

The application of the approach other real world case studies would obviously allow us to refine and validate the whole framework.

Acknowledgments. This work has been partially supported by the IST EU project PLASTIC (www.ist-plastic.org).

References

1. SeCSE Project, http://secse.eng.it
2. Autili, M., Cortellessa, V., Di Marco, A., Inverardi, P.: A Conceptual Model for Adaptable Context-aware Services. In: WS-MATE (2006)
3. Bernardi, S., Donatelli, S., Merseguer, J.: From uml sequence diagrams and statecharts to analysable petri net models. In: 3rd ACM Workshop on Software and Performance, ACM Press, New York (2002)
4. Budinsky, F., Steinberg, D., Merks, E., Ellersick, R., Grose, T.J.: Eclipse Modeling Framework. Addison-Wesley, Reading (2003)
5. Cortadella, J., Kondratyev, A., Lavagno, L., Passerone, C., Watanabe, Y.: Quasi-static scheduling of independent tasks for reactive systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 24(10) (2005)
6. Cortellessa, V., Singh, H., Cukic, B., Gunel, E., Bharadwaj, V.: Early reliability assessment of uml based software models. In: 3rd ACM Workshop on Software and Performance, ACM Press, New York (2002)
7. Eclipse.org. Eclipse Web Standard Tools, http://www.eclipse.org/webtools
8. IBM. BPEL4WS, Business Process Execution Language for Web Services, version 1.1 (2003)
9. Inverardi, P., Mancinelli, F., Nesi, M.: A Declarative Framework for adaptable applications in heterogeneous environments. In: ACM SAC, ACM Press, New York (2004)
10. Jouault, F., Kurtev, I.: Transforming Models with ATL. In: Bruel, J.-M. (ed.) MoDELS 2005. LNCS, vol. 3844, Springer, Heidelberg (2006)
11. Menascé, D.A., Ruan, H., Gomaa, H.: A framework for QoS-aware software components. In: WOSP '04, ACM Press, New York (2004)
12. ModelWare: IST European project 511731, http://www.modelwareist.org
13. Nezhad, H.R.M., Benatallha, B., Martens, A., Curbera, F., Casati, F.: Semi-automated adaptation of service interactions. In: WWW 2007 Web Services Track (2007)
14. OMG: UML 2 Superstructure. formal/2007-02-03 (February 2007)
15. Passerone, R., de Alfaro, L., Heinzinger, T., Sangiovanni-Vincentelli, A.L.: Convertibility verification and converter synthesis: Two faces of the same coin. In: Proc. of ICCAD 2002 (2002)
16. PLASTIC IST STREP Project: Deliverable D2.1: SLA language and analysis techniques for adaptable and resource-aware components, http://www-c.inria.fr/plastic/deliverables/plastic-d2_1-finalpdf.pdf/download
17. PLASTIC IST STREP Project: Deliverable D2.2: Graphical design language and tools for resource-aware adaptable components and services, http://www-c.inria.fr/plastic/deliverables/plastic-d2_2-finalpdf.pdf/download
18. PLASTIC IST STREP Project: Deliverable D4.1: Test Framework Specification and Architecture, http://www-c.inria.fr/plastic/deliverables/plastic_d4_1final.pdf/download
19. A-MUSE Project: Methodological Framework for Freeband Services Development (2004), https://doc.telin.nl/dscgi/ds.py/Get/File-47390/
20. PLASTIC Project: Description of Work (2005), http://www.ist-plastic.org
21. Selic, B.: The Pragmatics of Model-driven Development. IEEE Software 20(5), 19–25 (2003)
22. W3C: Web Service Definition Language, http://www.w3.org/tr/wsdl
23. Yun, H., Kim, Y., Kim, E., Park, J.: Web Services Development Process. In: PDCS (2005)