Context-Aware Quality Model Driven Approach: A New Approach for Quality Control in Pervasive Computing Environments

Adel Alti[1], Abdellah Boukerram[1], and Philippe Roose[2]

[1] Computer Science Departement, Engineering Faculty,
Ferhat Abbas University of Setif, 19000 Setif, Algeria
altiadel2002@yahoo.fr
[2] LIUPPA / IUT Bayonne, 2 Allée du Parc Montaury, 64600 Anglet, France
Philippe.Roose@iutbayonne.univ-pau.fr

Abstract. This paper presents extension of MDA called Context-aware Quality Model Driven Architecture (CQ-MDA) which can be used for quality control in pervasive computing environments. The proposed CQ-MDA approach based on ContextualArchRQMM (Contextual ARCHitecture Quality Requirement MetaModel), being an extension to the MDA, allows for considering quality and resources-awareness while conducting the design process. The main idea of presented extension consists in three abstractions levels: PIM (Platform Independent Model), CPIM (Contextual Platform Independent Model) and CPSM (Contextual Platform Specific Model). At the PIM level, a model decomposed into a two interrelated models: software architecture artifacts, which reflect functional requirements and quality model. At the CPIM level a simultaneous transformation of these two models with contextual information details is elaborated and then refined to a specific platform at the CPSM level. Such a procedure ensures that the transformation decisions should be based on the quality assessment of the created models.

Keywords: MDA, Context, Quality Model, ADL.

1 Introduction

Model Driven Approach (MDA) [5] has been proposed by the OMG (Object management Group). The basic models of MDA are entities able to unify and support the development of computer systems by providing interoperability and portability. MDA approach does not address how to consider non-functional demands, i.e. how to represent and transform them.

In this paper, we present an extended Model Driven Architecture which includes support for software architecture quality control and resources requirements changes, in the framework of CQ-MDA (Context-aware Quality Model Driven Architecture). Some other works concentrate only on quality system architecture or context-aware system architecture [9, 10]. Our approach focuses on separation of two concerns: the architecture and the implementation contexts. This enables us to support them with the elaboration of quality model explicitly and to facilitate the system architecture quality control with the continuous evolution of its context.

We have previously introduced the *ArchRQMM* (ARCHitecture Requirement Quality MetaModel) [3] to address a serious gap in architectural styles quality control. *ArchRQMM* extend the common concepts of Architecture Description Languages (ADLs) with the concepts of quality requirements and quality standards [7]. The *ArchRQMM* targets the quality evaluation and selection of application styles at a high level of abstraction. However, our metamodel does not support the definition of a context-awareness and a resource-awareness metamodel.

The paper is organized as follows. Section 2 proposes the main element of CQ-MDA approach, i.e. *ContextualArchRQMM* metamodel which it is an *ArchRQMM* extension used as support for context model description and quality model definition. Section 3 describes the CQ-MDA itself. Section 4 shows an example of applying CQ-MDA for VideoConferencing system development. Section 4 summarizes related works. Section 5 concludes this article and presents some future works.

2 ContextualArchRQMM

The main idea of this proposal is to take into consideration the nonfunctional concerns (*adaptation service, communication protocol, security, QoS, etc.*) of the components by connectors at the software architecture level. Our motivation is to extend *ArchRQMM* with contextual connectors in order to support improved composability of heterogeneous components and to integrate a software architecture quality control in the framework CQ-MDA which unifies all modelling[2] approaches.

2.1 Context-Awareness MetaModel

We extend our software architecture metamodel, with a context metamodel (Fig. 1). The goal is to represent context information of system architecture at model level. Context is any information that can be collected from artifact needs, resources capacities and user preferences. *ContextualArchRQMM* uses these informations to perform a software architecture quality evaluation and selection in software development process. In our metamodel we have identified two types of context, i.e., required context (user preferences, artifacts needs) and provided context that encompasses the properties of the execution environment of an application. Context elements are realized through *Context* class, are expressed as QoS properties of the contextual architectural artifacts (*Non-Functional-Prop* class).

2.2 Resource-Awareness MetaModel

Fig. 2 depicts a resource-awareness metamodel. The hardware components are mobile devices (Class *Device*) like PDAs, PC Portables or smart phone, are constrained in their resources (memory size, CPU power, bandwith, battery, etc) and act as execution environment for architectural artifact (Class *Artifact*).

Network connections (Class *Node*) connect hardware components having a limited bandwith. A resource-awareness about current usage of processing power, memory,

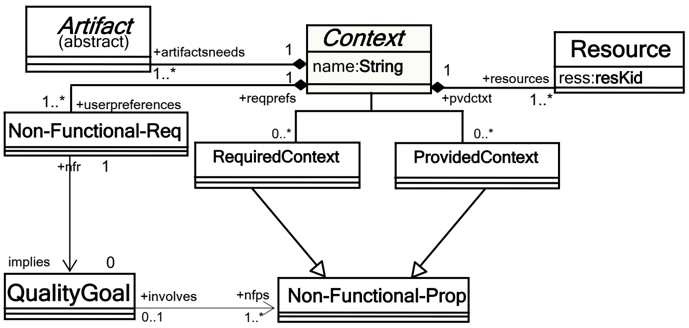

Fig. 1. The context metamodel of ContextualArchRQMM

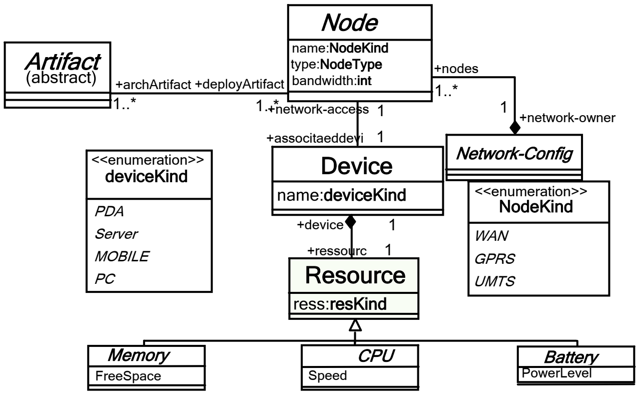

Fig. 2. The resource metamodel of ContextualArchRQMM

network bandwith, etc. is a prerequisite to guarantee a minimum quality of service. Due to heterogeneous architectural components as well as its various communications paradigms (GSM 3G, Bluetooth, ZigBee, etc.) can be specified more easily using a contextual architectural artifact to better support resource-awareness.

2.3 Contextual Architectural Artifacts

For an efficient and clear specification of connection points, we have introduced more precise port according to their global roles in a component: the *DataPort,* the *ContextPort,* the *ServiceControlPort,* the *QoSNotificationPort.* The *DataPort* is used to transfer data of any type. The *ContextPort* is responsible for the sending and receiving of the context information available at run-time when the service is active. The *ServiceControlPort* is a standard dedicated port for controlling a service. It allows the service to be (re)started, updated, relocated, stopped and uninstalled. The *QoSNotificationPort* is responsible for sending QoS information to execution platform in order to decide if a service reconfiguration is needed.

As software architecture descriptions rely on a *connector* to express interactions between components, an equivalent abstraction must be used to express a contextual and a heterogeneous interaction (i.e. various interactions paradigms). We extend an architectural connector with a contextual concern in a heterogeneous interaction (see Fig. 3). The traditional connector is not enough to design a contextual and a heterogeneous interaction because the way that a contextual component composes with a business logic component is slightly different from the composition between business logic components only. A contextual (i.e. the adaptation) connector is the central place where the auto-adaptative mechanisms are managed in a connector. Three auto-adaptative mechanisms are distinguished: **communication** (i.e. clarify the connection between various components regarding the communications paradigms), **service adaptation** (i.e. adding, suppression and substitution of adaptation services), and **QoS adaptation** (selecting parameters of service to provide adequate quality to component needs at runtime). The business logic component is adapted explicitly and automatically by a *contextual connector*. This means that context ports of *business logic* components instances, related to the context managed by a contextual connector, are all connected to that contextual connector.

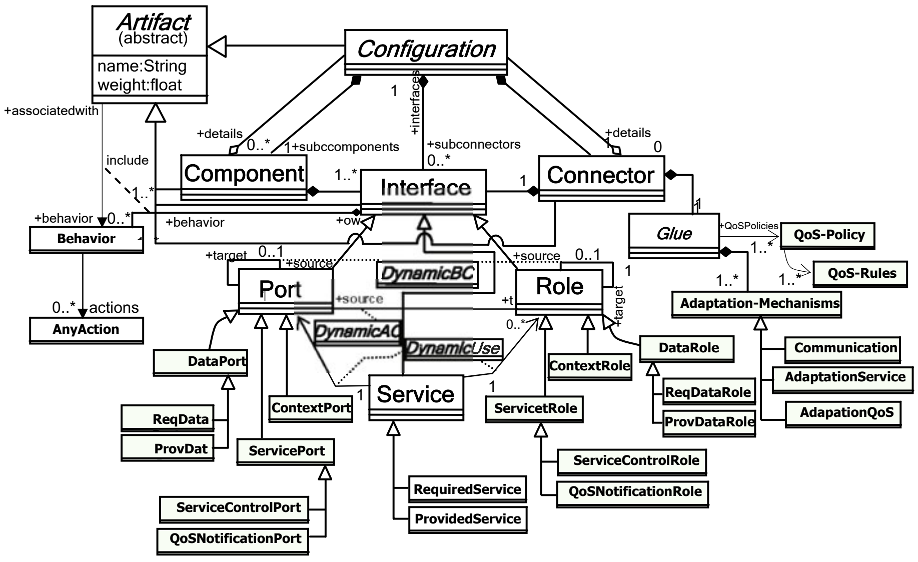

Fig. 3. Contextual architectural artifacts in ContextualArchRQMM

3 Context-Aware Quality Model Driven Architecture (CQ-MDA)

The general structure of Context-aware Quality – Model Driven Architecture (CQ-MDA) is presented in Fig. 4. The proposed structure consists in five levels representing CIM, PIM, Contextual Platform Independent Model (CPIM), Contextual Platform Specific Model (CPSM), and code. Each level decomposed into three parts: the left part represents architectural artifacts and context concepts; the right part represents quality model and measurements done for these artifacts while the center part represents requirements. Architecture quality should be controlled at each steps of the design. *External requirement*s of the system are transformed into *internal ones* for the architecture and its components. *Internal requirements* are needed for assessing designed architecture models. So, particular internal models, being instances of *ContextualArchRQMM* metamodel, are used to assess particular models of CQ-MDA, for example, the requirement reflects both functional and non-functional architects' needs are elaborated on the base of a particular set of criteria's and associated metrics. The software architecture quality model is produced by measurement done for each architectural artifact for a given factor in the context of associated requirement, for a given criteria with associated metric. The quality model is evaluated by the semantic constraints defined by the metamodel.

Two ways of using the *ContextualArchRQMM* metamodel are possible:

- The first one assumes that the software architecture quality metamodel is used for evaluating an architecture model. The architecture model is tested and validated with the semantic constraints defined by the metamodel. If the verified architecture model gets bad marks then the design process can be stopped or it can go back to the previous stage either to change requirements or to elaborate a different (better) architectural model.
- The second one, using software architecture quality metamodel considers the case when the metamodel is used for selecting the best architectural model from different choices. In this case the values of a metric are used to classify the models. A metric formula gives a note for the architecture model. The values of the metric function are used to classify the models and to choose the suitable one and we select a first model if we have the same value. After that, the selected architectural model is evaluated by the OCL constraints to remove any quality semantic violation.

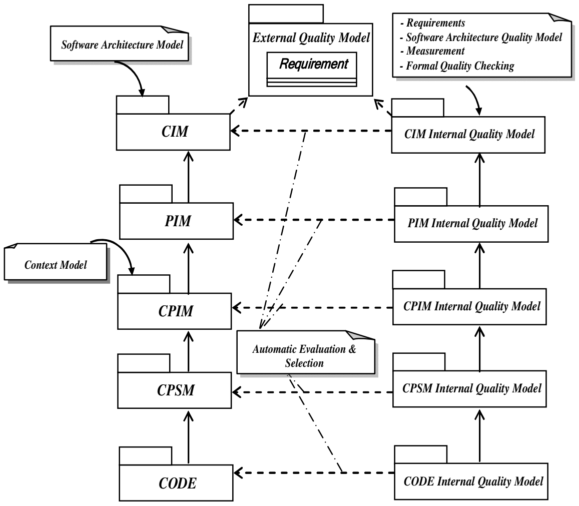
Software Architecture Model
External Quality Model
Requirement
- Requirements
- Software Architecture Quality Model
- Measurement
- Formal Quality Checking
CIM
CIM Internal Quality Model
PIM
PIM Internal Quality Model
Context Model
CPIM
CPIM Internal Quality Model
Automatic Evaluation & Selection
CPSM
CPSM Internal Quality Model
CODE
CODE Internal Quality Model

Fig. 4. Context-aware Quality Driven Model Architecture

4 Case Study: VideoConferencing System

A case study given below is intended to show applicability of CQ-MDA both for evaluation and for selection of the best architectural model from some alternatives. A case study deals with *VideoConferencing* System [13]. The following architect needs and preferences are considered:

- Recording, reviewing user' video and creating respective reports.
- Video should be delivered in quality and in period no longer than one minute from their request.

These demands are processed as external quality requirements. The first one is functional demand while second one is non-functional one. Only non-functional requirements will be considered further.

4.1 PIM Level

Several architectural models can be used to design a given system. For the *VideoConferencing* system, the model is designed with *PipesAndFilters* style. The architecture model should be evaluated, tested and validated with the semantic constraints defined by the metamodel.

According to *ContextualArchRQMM*, all these requirements should be associated with a respective architecture quality model with selected quality factors. It is proposed to use the efficiency factor with time-behavior sub-factor and the maintenability factor with modularity, analyzability sub-factors [4].

We have evaluated the PIM model with similar measurements of the whole architecture of the basic metrics (i.e. coupling, cohesion and complexity) [4]. The architectural model provides an acceptable maintainability (a high level of cohesion, a low level of coupling, a low level of complexity). This architectural model is accepted for further transformation.

4.2 CPIM Level

PIM software architecture model may be transformed, manually or automatically, into different CPIM models. The PIM model is transformed into four exemplary CPIM models. For time-behavior, three metrics proposed in [8], one of them, *TBM* (the estimated Time Behavior Metric) is selected and adapted in our case. Apart from the evaluation of time behavior sub-factor we evaluate the analyzability sub-factor to select the best CPIM model. In [17] two metrics were proposed for the dynamic adaptivity at the architectural level, but only one, *MaAC* (Minimum architectural Adaptive Cost) was used and validated for analysability assessment in our example.

We have simulated the four CPIMs models using our Java VM simulator and have varied the user (and respectively, the mobile devices) from 1 to 30. Table 1 shows the evaluation results, meaning that CPIM4 model turns out to be the best. Differences can be seen in the adaptation cost of CPIM4 and other CPIMs, which is due to the low adaptation cost compared to other CPIMs.

Table 1. CPIM models evaluation results

CPIM	$TBM^{Benefit}$(ms)	$MaAC^{Cost}$ (artifacts number)
CPIM 1	*200 ~ 400*	*0 ~ 16*
CPIM 2	*350 ~ 500*	*0 ~ 8*
CPIM 3	*470 ~ 800*	*0 ~ 8*
CPIM 4	*200 ~ 930*	*0*

5 Related Works

The first related area of research are ADLs that have been proposed for representing dynamic architectures including: ACME [12], π-ADL [6], C3 [2] and AADL [1]. However, except for ACME, most ADLs do not support the concept of evaluation function. In addition, most of them are not contextual defined. AADL [1] allows definition of non-functional requirements and their validation at model level. MARTE [15] not treat the problem of heterogeneity by a meta-model which verifies the adequacy of service regarding its context. π-ADL [6] supports dynamic software architecture and evolving software systems. However, contrary to our work, π-ADL does not support contextual connectors and not integrate quality metrics. Recently, Garlan and al. [12] extended ACME ADL in order to support evaluation function in evolution styles and their multiple decision forms. However, this work does not consider exploiting contextual connectors in heterogeneous environment where entities of different nature collaborate: software and hardware components.

The second related area of research are some works involving quality in MDA approach, like QADA (Quality-driven Architecture Design and Quality Analysis) [9] – a methodology targeted at the development of service architectures. Other works involving Context in MDA approach, e.g. Context-aware Model Driven Architecture Model Transformation [11] – a methodology targeted at the development of context-aware applications and other networked systems. These works concentrate only on quality system architecture or context-aware system architecture, while CQ-MDA insisted on the separation of the two concerns: software architecture model and context model. These models based on the quality assessment that enables us to reuse them independently and to achieve a comfortable architectural quality analysis framework.

6 Conclusion and Future Work

In this paper we presented *ContextualArchRQMM* metamodel centered on the concept of contextual connector to provide a lightweight support for the definition of some composition facilities such as contextual interfaces at the connector level. In this way, *ContextualArchRQMM* encompasses a reduced set of minor changes. Our goal is a complete ArchRQMM software architecture metamodel that supports structural and contextual description of software systems. Representing components, connectors as first class entities allows us to define the context concerns of each of concept independently and explicitly and to improve composability of heterogeneous components and lowering adaptation cost through self-adaptation policies under resources constraints. We have used our metamodel to extend the MDA's CIM-PIM-PSM with a parallel CPIM-CPSM chain of refinement, to explicitly consider quality and resources-awareness while conducting the design and implementation process. We presented an illustrative example to show the applicability of the proposed CQ-MDA approach. The results of the experiments (based on the example of *VideoConferencing system* with four CPIMs) are encouraging. The experiment shows that our approach outperforms two abstractions level in terms of some quality metrics such as adaptation ratio and time response. As future work, we will consider moving our approach to a real execution platform to validate its feasibility.

References

1. Berthomieu1, B., Bodeveix, J.P., Chaudet, C., Vernadat, F.: Formal Verification of AADL Specifications in the Topcased Environment. In: 14th Ada-Europe International Conference on Reliable Software Technologies, Brest, France, pp. 207–221 (2009)
2. Amirat, A., Oussalah, M.: First-Class Connectors to Support Systematic Construction of Hierarchical Software Architecture. JOT 8(7), 107–130 (2009)
3. Alti, A., Boukerram, A., Smeda, A.: Architectural Styles Quality Evaluation and Selection. In: 9th Conference International NOTERE 2009, Montréal, Canada (2009)
4. Alti, A., Boukerram, A.: QualiStyle: A Tool for Automatic Quality Evaluation and Selection of Architectural Styles. In: 10th Annual Conference on New Technologies of Distributed Systems, pp. 243–248. IEEE Press, Tunisia (2010)
5. Miller, J., Mujerki, J.: MDA Guide, Version 1.0. OMG Technical Report (2003), `http://www.omg.org/docs/ptc/03-05-01.pdf`
6. Oquendo, F.: π-ADL: an architecture description language based en the higher order typed π-calculus for specifying dynamic and mobile software architecture. ACM Software Engineering 29(4), 1–13 (2004)
7. Losavio, F., Chirinos, L., Lévy, N., RamdaneCherif, A.: Quality characteristics for software architecture. JOT 2(2), 133–150 (2003)
8. ISO/IEC 9126-3, Software Engineering Product quality Part 3: Internal metrics (2003)
9. Quality-driven Architecture Design and Quality Analysis, `http://virtual.vtt.fi/qada`
10. Tarvainen, P.: Adaptability Evaluation at Software Architecture Level. The Open Software Engineering Journal 2, 1–30 (2008)
11. Vale, S., Hammoudi, S.: Context-aware Model Driven Development by Parameterized Transformation. In: 3rd Workshop of MDISIS 2008, pp. 167–180 (2008)
12. Garlan, D., Barnes, J.M., Schmerl, B., Celiku, O.: Evolution Styles: Foundations and Tool Support for Software Architecture Evolution. In: WICSA 2009, pp. 16–25 (2009)
13. Laplace, S., Dalmau, M., Roose, P.: Prise en compte de la qualité de service dans la conception et l'exploitation d'applications réparties. In: Workshop GEDSIP@Inforsid (2009)
14. Raibulet, C., Masciadri, L.: Evaluation of Dynamic Adaptivity through Metrics: an Achievable Target? In: WICSA 2009, pp. 65–71 (2009)
15. Gérard, S., Petriu, D., Medina, J.: MARTE: A New Standard for Modeling and Analysis of Real-Time and Embedded Systems. In: 19th Euromicro Conference on Real-Time Systems, Pisa, Italy (2007)