Context-aware Quality Model Driven Approach

Adel Alti, A. Bookerram, Makhlouf Derdour, Philippe Roose

Context-aware Quality Model Driven Approach

A. Alti[1], A. Bookerram[1] , M. Derdour[2], P. Roose [2]
[1] Computer Science Department, Engineering Faculty, Ferhat ABBAS University of Setif, 19000 Setif, Algeria
[2]LIUPPA / IUT Bayonne, 2 Allée du Parc Montaury, 64600 Anglet - France

***Abstract*— The explicit separation of functional and non-functional requirements is the main concern of the Contextual ARCHitecture Quality Requirement MetaModel (ContextualArchRQMM), aiming to well capturing resources-awareness and controlling quality at architecture level. This article defines 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, 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 is decomposed on 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 are 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;

I. INTRODUCTION

Model Driven Approach (MDA) 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. UML, CWM and MOF represent the core of MDA. If the number of core models available is limited, it is not yet fixed.

The notion of transformation is an essential element for MDA aiming at automated model transformations. Furthermore, system development is seen as a chain of model transformations representing different categories. There are three basic categories, according to their abstraction level; namely, Computation Independent Models (CIMs), Platform Independent Models (PIMs) and Platform Specific Models (PSMs). CIMs are focused on the model system requirements where the designed system is to be placed. Specific business modeling languages like BPMN [reference] and EPC [reference] as well as UML are used to build CIM models [12]. PIMs are in turn able to model the system's functionality without considering any particular platform. So, PIMs include such models as UML class diagrams or statechart diagrams. PSMs are refined the PIM in the way which takes the features of a given platform into account. PSM models are represented using UML or in its specific profiles. Finally, the PSM model is transformed into code. In the model transformation process the functional requirements (stated by CIM model) should be kept by PIM, and finally by PSM model. But MDA approach does not address how to consider non-functional demands, i.e. how to represent and transform them. Furthermore, MDA doesn't really take into account the quality of dynamic architectures for very limited mobile device like PDA, Smartphone, etc. – il faut expliquer ici pourquoi modéliser pour PDA/Smartphone est un peu particulier à cause des resources limitées. Although there already some works which are somehow related both to resources-awareness, quality and MDA [9, 10, 14], still there is a lot of acceptance. Il faut presenter l'apport de ce papier par rapport aux autres déjà écrits sur le domaine 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).

The main idea of the 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 is made of 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 are 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.

To provide a serious gap in software architecture quality control, we have previously introduced the ArchRQMM (ARCHitecture Requirement Quality MetaModel) [3]. It has been proposed upon four main principals: 1) - extend the common concepts of Architecture Description Languages (ADLs) with the concepts of quality requirements and quality standards [11] 2) - define the exact measurable standards at the level of architecture for goodness of software architecture 3) - define clear separation between application styles and quality factors to evaluate those styles and at last, 4) - improve a formal verification of the properties' quality of architectures on modelling styles using OCL [13]. However, our metamodel does not support the definition of a context-awareness and a resource-awareness metamodel.

We begin this paper by introducing ArchRQMM metamodel. Section 3 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 4 describes the CQ-MDA itself. Section 4 shows an example of applying CQ-MDA for VideoConference system development. Section 5 summarizes related works. Section 6 concludes this article and presents some future works.

II. AN OVERVIEW OF ARCHRQMM METAMODEL

ArchRQMM metamodel enables architectural styles quality evaluation and selection at the architecture design step and ensures formal verification of the properties' quality of architectures on modelling styles. The metamodel was described in details in [3, 4]. It was developed according to ISO/IEC 9126 standard [8]. ArchRQMM is based on a set of metaclasses for the common concepts of architectures descriptions languages (ADLs) and a set of quality characteristics based on a standard ISO quality model [11] which can be investigated and evaluated in the architecture level. Figure 1 (elle n'est pas bien lisible, il faudrait refaire la capture/l'exportation avec une meilleure qualité) presents a MOF metamodel of the ArchRQMM. An instance of this metamodel defines:

- Software architecture model (i.e. architectural artifacts) and architectural styles.
- Quality requirements model.
- Software architecture quality model, measurements and quality criteria's.

The software architecture model involves classes representing software architecture and its artifacts. *Architecture* may be composed of many artifacts. *Components* are potentially composite computational encapsulations that support multiple interfaces known as *ports*. *Attachments* define set of port/role associations. *Ports* are bound to ports on other components using first-class entities called *connectors*, which have the so-called *roles* that are attached directly to ports. *Configurations* are the abstractions that represent graphs of components and connectors. *Attachments* define set of port/role associations. *Bindings* connect two interfaces of the same type (two ports or two roles). In our metamodel, styles are reused and composed using *Pattern Templates* [15].

Requirements package represents architect's needs and quality goals. The architect's needs (*Requirement* class) should fulfill particular architecture artifacts (*Artifact* class in the model). Usually the necessities of a software system are divided into two groups: *Functional requirements* and *Non-Functional requirements*. Functional requirements derived from the architect's needs and non-functional requirements are more related to the problem's environment or context such as the system's operational environment and the problem's real word. Non-functional-requirements are associated with a quality goal (*QualityGoal* class) that must be satisfied to ensure the accomplishment of the functionality in the final software product. The non-functional-properties (*Non-Functional-Prop* class), which are related to quality requirements are specified in the requirement model. Non-functional properties are formalized using the standard ISO-9126 [8]. Based on final quality aims, the developed architecture for a given software system can be evaluated to satisfy quality goals.

The software architecture quality model defines quality of the whole software system as well as its architectural artifacts in terms of quality factors and associated metrics that are formal measured attributes of the software. An instance of *QualityFactor* class is the root of quality factors and sub-factors, and represents a given quality perspective. Only the quality factors and sub-factors that are leaves of the root are requested to have metrics assigned. Organization of the model is consistent to the interpretation of quality model as a set of factors and the relationships among them provides the basis for specifying quality requirements and evaluating quality architecture model when using a given architectural style. ISO/IEC 9126 standard [8] defines three quality perspectives (quality in use, external quality and internal quality). In the context of the paper external and internal qualities are considered (*IsInternal* as Boolean filed in *QualityFactor* class). Each quality factor is associated with quality criteria (*QualityCriteria* class); it shows the technical concepts that must be investigated at the level of architecture to ensure quality. Each quality criteria is associated with quality metrics (*Metric* class), which represents values of metric for a given architectural artifact. In this case its value is used for selection of the best artifact instead of artifact classification.

The quality of architectural styles is perceived as a constraint which has to be checked for validity during each design step. The focus of rigorous architecture quality analysis is to prevent the non-required affections before the early phases of system development. For example, the configuration modularity is given by the following OCL constraints [13]:

```
Context ArchRQMM:Configuration inv:
self.qualityfactorArtifact.subfactors.qualitycriteria->
   exists->(sf|sf.name= #Modularity) implies
(self.subcomponents.qualityfactorArtifact.qualitycriteria->
  select(c|c.criterianame=CriteriaName::Cohesion
     implies c.result >=0.5) -> notEmpty() and
  select(c|c.criterianame=CriteriaName::Coupling
     implies c.result <=0.66) -> notEmpty())and
(self.subconnectors.qualityfactorArtifact.qualitycriteria->
  select(c|c.criterianame=CriteriaName:: Cohesion
     implies c.result >=0.5) -> notEmpty() and
  select(c|c.criterianame=CriteriaName::Coupling
            implies c.result <=0.66) -> notEmpty())
```

légende

The designer may be modified as needed a *threshold* of quality associated with each criterion (je ne comprends pas cette phrase). Constraints expressed with OCL are also used to evaluate other concepts of ArchRQMM such as configurations, artifacts, metrics, criteria, etc. For more details see [4].

III. CONTEXTUALARCHRQMM

An application for heterogeneous mobile embedded and limited (low bandwidth, power consumption, etc.) device has to firstly prevent interaction and mobility limitation. Il faut l'expliquer + dans l'introduction en disant quelle est le support physique cible. We decided to extend ArchRQMM with contextual connectors in order to support improved composability of heterogeneous components and therefore to

integrate a software architecture quality control in the framework CQ-MDA (Context-aware Quality Model Driven Architecture) which unifies all modeling approaches. Our extension does not create a new concept abstraction and is strictly based on enriching the connection semantics supported by architectural connectors instead of introducing elements that elevate various interactions paradigms concepts to the architecture level. This section presents the description of ContextualArchRQMM. We present the ArchRQMM extension to support the definition a context-awareness and a resource-awareness metamodel.

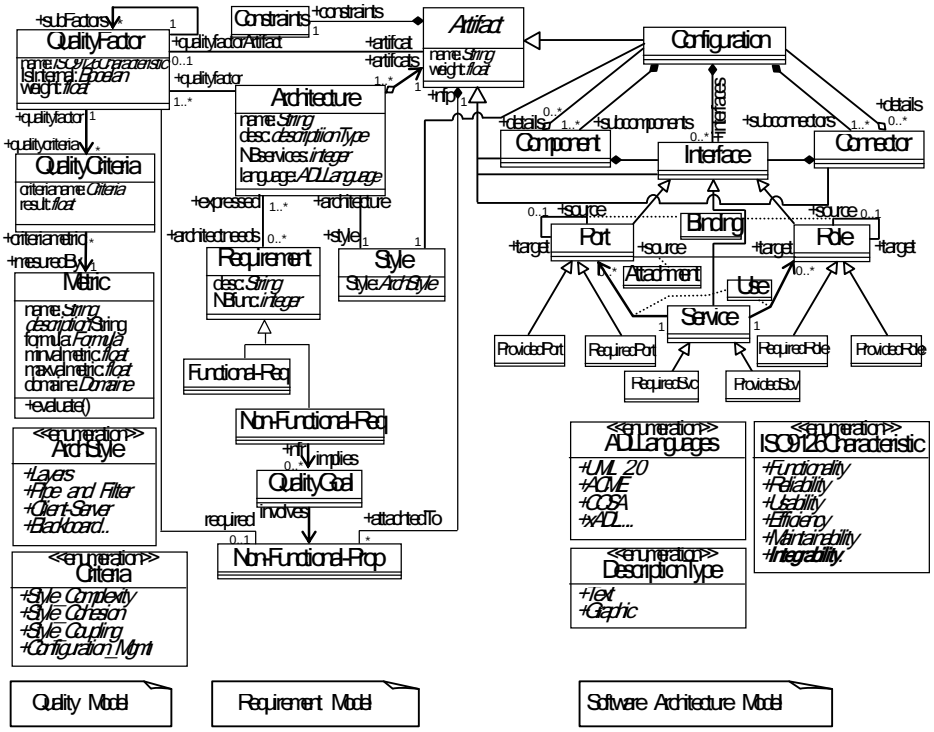


Figure 1. A MOF metamodel of ArchRQMM

A. *Context-awareness MetaModel*

We extend our software architecture metamodel, with a context metamodel (see Figure 3). The goal is to context information to measure the quality 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).

La figure 2 n'est pas lisible, il faut refaire la capture.

En fait c'est général aux figures 1, 2, 3 et 4 de l'article. On ne peut pas les lire, c'est très gênant.

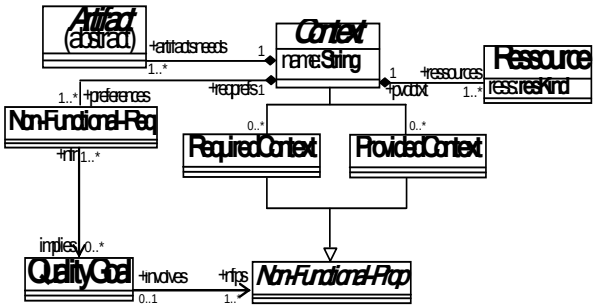


Figure 2. The context metamodel of ContextualArchRQMM

B. *Resource-awareness MetaModel*

Figure 4 depicts a resource-awareness metamodel. The hardware components are mobile devices (Class *Device*) like PDAs or smart phone, are constrained in their resources (memory size, CPU power, bandwidth, battery, etc) and acts as execution environment for architectural artifact (*Artifact* class in the model). Network connections (Class *Node*) connect hardware components having a limited bandwith. A resource-awareness about current usage of processing power, memory, network bandwith, battery lifetime, etc. is a perquisite 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.

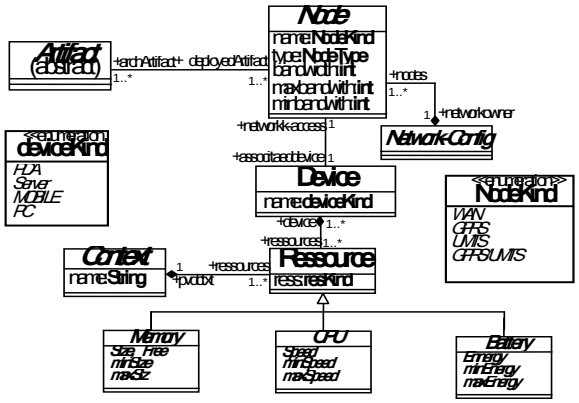


Figure 3. The context metamodel of ContextualArchRQMM

C. *Contextual Architectural Artifacts*

As software architecture descriptions rely on a *connector* to express interactions between components, an equivalent abstraction must be used to express a contextual and heterogeneous interaction (i.e. various interactions paradigms). We extend an architectural connector with a contextual concern in a heterogeneous interaction (see Figure 4). The traditional connector is not enough to model (design?) a contextual interaction because the way that a contextual component composes with a regular component is slightly different from the composition between regular components only. A contextual concern is represented by provided contextual services of a component and it can be affect both provided and required contextual service of other components which can be, in turn, regarded as contextual joint point at the architectural level. Since ADL valid configurations are those that connect provided and required services, it is impossible to represent a connection between provided contextual services of a contextual component and a provided service without extensions to the traditional notion of architectural concerns. Although ArchRQMM itself does not support a syntactic distinction between data and context ports, this distinction can be expressed using extra dedicated ports: The *context port* is responsible for the sending and receiving of the context information available at run-time when the service is active. It allows the service to be notified of new resources, and to inform other services about resource currently in use by this service. The *context control port* is a standard dedicated port for controlling a service. It allows the service to be (re)started, updated, relocated, stopped and uninstalled. The *QoS Notification port* is responsible for sending QoS information to execution platform in order to decide if a service reconfiguration is needed.

In order to express the dynamic interaction in different contexts, we extend a connector with a new kind of roles. The purpose of such a new role is twofold: to make a distinction between the element playing different roles in a dynamic interaction (i.e. affected base (business ?) components and contextual components; and to capture the way both categories of components are interconnected). The connector interface contains: 1) – data roles 2) – contextual roles, 3) – QoS notification roles, 4) control roles and 5) – a glue clause.

The data role may be connected to the data port of a component (provided or required) and the *contextual role* may be connected to a *contextual port* of a component. The distinction between a data and context roles addresses the constraint typically imposed by many ADLs about the clear separation between functional and non-functional aspects. The composition between heterogeneous components is expressed by the *glue clause*. Our contribution in this level consists in enhancing the structure of connector by defining a *glue clause.* The *glue clause* specifies the different ways of interacting heterogeneous components. There three types of contextual glue: *switch*, *conditional*, and *default*.

The *glue clause* can be simply a declaration of the glue type, or a block with *multiple declarations*, where each relates a contextual role, a data role, a service control role; a QoS notification role and a glue type (see Figure 5).

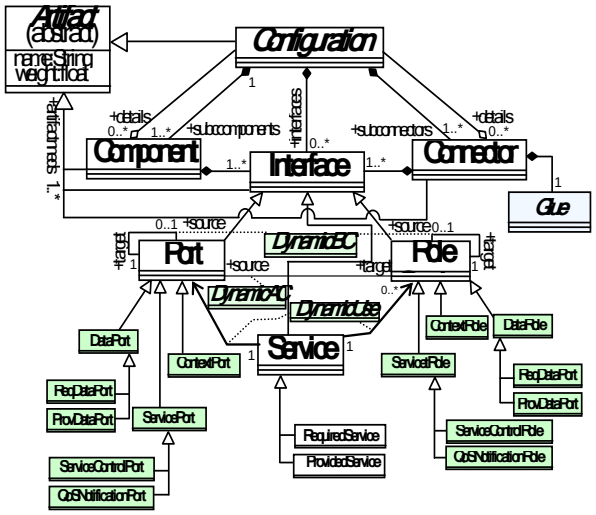


Figure 4. The context metamodel of ContextualArchRQMM

```
Connector aConnector = {
    DataRole reqDataRole, provDataRole;
    ContextRole abandwithRole;
    QoSNotifcationRole aQoSNotificationRole;
    ServiceControlRole aServiceControlRole;
    Service Conversion_dataformat1_dataformat2, Conversion_dataformat1_dataformat3;
    Glue {
        bw = abandwithRole.getBandWidth();
        conditional (bw less-than 4000) {
            DynamicUse {provDataRole, Conversion_dataformat1_dataformat3} ;
            Type = RMI
            Communication = distant ;
            Mode = Asynchronous mode ;
            setParameter new rate to aServiceControlRole ;
            getQuality quality from aQoSNotificationRole ;
            }
        default {
            DynamicUse {provDataRole, Conversion_dataformat1_dataformat2} ;
            Type = Socket
            Communication = local ;
            Mode = Synchronous mode ;
            setParameter default rate to aServiceControlRole ;
            getQuality quality from aQoSNotificationRole ;
            }
        }
}
```

Figure 5. A contextual connector in ContextualArchRQMM.

Figure 6 contains a graphical notation that we propose to represent a dynamic connector and its contextual roles. Con1 is a dynamic connector that defines a heterogeneous interaction between three components C1, C2, C3.

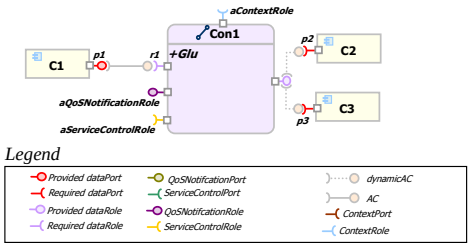


Figure 6. Graphical Notation of the Contextual Connector

This extension improves variability in architectural configurations as well as quality of composing heterogeneous components. Note that the different ways of interacting can now be modularized in a single architectural connector. Consequently heterogeneous components assembled in an easy and coherent way.

From this new structural joint points that be affect by contextual connector, many attachments should be defined, where each one binds the same data role instance to a different component port. We propose an extension of the attachments part of ArchRQMM configuration to allow to attachment sets of ports to the same role using Dynamic Attachment Connector (*DynamicAC* Class). This description allows creating a dynamic reconfiguration for several heterogeneous components. To perform a better QoS for the contextual components (composing and/or decomposing of services) when detecting the frequent changes in the environment, a *DynamicUse* connector is introduced. This connector is guided by significant evolutions of the environment, consisting of a collection of context informations (CPU usage, battery lifetime, bandwith measure…).

IV. CONTEXT-AWARE QUALITY – MODEL DRIVEN ARCHITECTURE (CQ-MDA)

The general structure of Context-aware Quality – Model Driven Architecture (CQ-MDA) is presented in Fig. 7. We consider the full software development cycle within MDA, i.e. from formulation of needs up to the code generation.

The proposed structure consists in five levels representing CIM, PIM, Contextual Platform Independent Model (CPIM), Contextual Platform Specific Model (CPSM), and code. Each level is 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.

In Fig. 7 External Quality Model represents an instance of ContextualArchRQMM metamodel. This model is used for expressing quality factors with metrics for evaluation of software system external quality at the architecture level. External Quality Model contains architects needs, resources requirements and user preferences. We assume, that system requirements reflect both functional and non-functional architect's demands.

Requirement represents internal requirements for software architecture elements. The internal requirements are elaborated on the base on external requirements (at CIM level), or internal requirements expressed on the higher level (at the lower level of abstraction, e.g. PIM, CPIM, CPSM, Code).

The contents of each MDA model could be perceived as a collection of sub-models of different types. For example, PIM model typically gathers at least two kinds of sub-models, i.e. *structural sub-model*, and *behavioral sub-model* developed without considering contextual details. Each sub-model can be transformed and adding context information to several CPIMs (i.e. several adaptations). The best selected CPIM model can generate a CPSM. The CPSM inherits quality requirements and context from CPIM. CPSM specifies operation system requirements, middleware architectures and networking. Architecture quality should be controlled at each steps of the design. External requirements of the system are transformed into internal ones for the architecture and its components. Internal requirements are needed for assessing designed architecture models. Ce paragraphe est compliqué à comprendre – il faut être un peu plus concret je pense

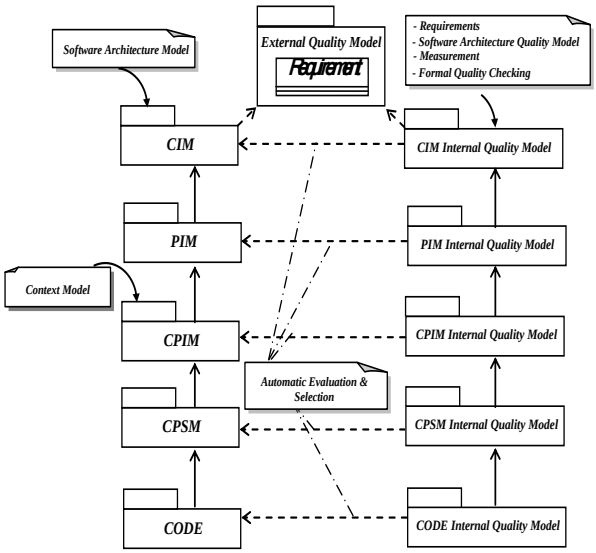


Figure 7. Context-aware Quality Driven Model Architecture

So, particular internal models, being instances of ContextualArchRQMM metamodel, are used to assess the architecture models, 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 and used to assesses a particular model of MDA cette phrase est trop longue/compliquée. For example, *CIM Internal Quality Models* are elaborated on the base of the *External Quality Models*. Similarly, PIM/CPIM/CPSM/Code Quality Models are elaborated on the base of the former quality models which is represented by dependency relationships between quality models. The elaborated Quality Model is used for expressing quality factors with metrics for evaluation of architecture models.

The process of the evaluation starts with designing the architecture model conforms to the software architecture ContextualArchRQMM metamodel, next producing the quality model conforms to the software quality ContextualArchRQMM metamodel by measurement done for each architectural artifact for a given factor in the context of associated requirement, for a given criteria with associated metric cette phrase est trop longue/compliquée. After that, the model is evaluated by the semantic constraints defined by the ContextualArchRQMM metamodel. Each Quality Model aggregates factors that are associated with metrics. The set of factors and associated metrics from External Quality Model should be transformed by a non-empty set of factors and associated metrics from Internal Quality Models. Next, similar transformation process is repeated at further levels. The set of new factors for the target model may be extended with additional factors to take into account assessment perspective of software architect.

An important feature of ContextualArchRQMM is the possibility of architecture model(s) checking. This is realized by OCL constraints that can be checked for a given requirement, criterion, artifact, context, and for the whole software architecture (i.e. set of artifacts). Results of assessment functions can influence the development process (Context-aware Quality – Model Driven Architecture).

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 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. In this case 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. After that, the selected architectural model is evaluated by the OCL constraints to remove any violation de quoi ?.

V. EXAMPLE

An example given below is intended to show applicability of CQ-MDA both for evaluation and for selection of the best architectural model from some alternatives. The example deals with *VideoConference* System [17]. *VideoConference* has the following adaptable and optional artifacts:

- Audio Encoder/Decoder: (de-)compressing the audio stream.
- Audio Filter: components for changing the frame size.
- Video Filter: reducing the video frame rate.
- Video Encoder/Decoder: (de-)compressing the video stream.

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. Only non-functional requirements will be considered further. Expliquer pourquoi on ne parle que des aspects non fonctionnels

According to ArchRQMM, all these requirements should be associated with a respective architecture quality model with selected quality factors. In our example, for illustration only non-functional requirements is taken into account. It is proposed to use the efficiency factor with time-behavior sub-factor [4]. On the CIM level some internal requirements may be specified additionally to external ones. We propose "an easy maintenance of software architecture model internal requirement" as we consider it to be important characteristic from architect point of view. This additional requirement can be expressed more precisely as "low complexity, high cohesion and low coupling are the main facts to take into account for achieving an easy maintainability architecture (subfactors of the maintainability factor [4])." The time behaviour sub-factor for software architecture model artifact cannot be evaluated at CIM level (as the software architecture is not defined yet) and should be forwarded to the next level i.e. PIM level. Therefore the CQ-MDA approach will be shown in details using the transformation of the PIM model with respective internal quality model into CPIM model with its internal quality model and the CPIM model with respective internal quality model into CPSM model with its internal quality model. In this example, we describe only the two first levels.

A. PIM level

1. PIM level – MDA artifacts

PIM model is the starting point for the considered transformation. The PIM model can consist in many sub-models describing different perspectives (e.g. behavior, structural) of the designed system. We are only interested in structural perspective which relates to the conceptual software architecture model. Several architectural models can be used to design a given system. For the *VideoConference* system, the model is designed with *PipesAndFilters* style as shown in Figure 8. At PIM level we have also formally defined set of architectural artifacts that are traced from CIM model.

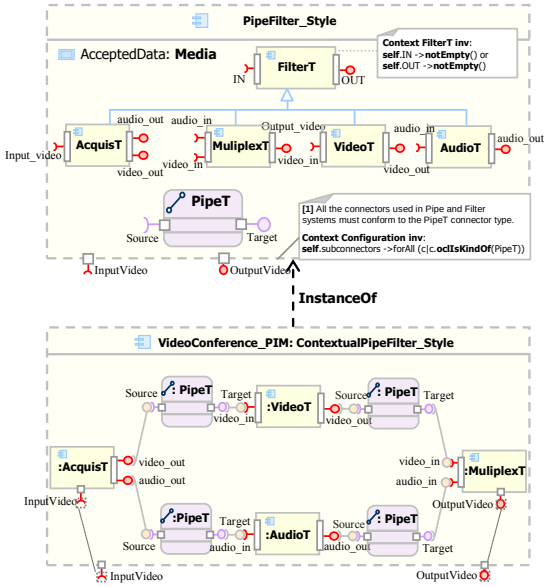


Figure 8. Examplary PIM software architecture model.

2. PIM level – Internal quality model

Internal quality model on this level is traced from the upper quality level model. So, we have to consider the factors from CIM level, i.e. efficiency factor with time-behaviour sub-factor and maintainability factor with modularity, analyzability sub-factors. The first factor is efficiency with sub-factors. Time-behavior cannot be evaluated at this level as we have not found accepted metrics for evaluation of the PIM model. This factor must be still forwarded for evaluation to the next modeling level.

The second factor is maintainability with modularity and analyzability sub-factors [4]. The first sub-factor, modularity, depends on the configuration, component and connector modularity. Indeed an architecture whose configuration has a good modularity if its components and its connectors have good modularity. If the system has been divided correctly to suitable modular, the software system can be analyzed more easily. At the architecture level, this factor can be measured with criteria, named *coupling* and *cohesion*. In [4] these two metrics are proposed for measuring architecture modularity. We used these metrics in our model. The second sub-factor, analyzability, architecture complexity metric is defined in [4] and used. The metric characterizes complexity of a structure of the architecture model. The complexity indices for conforming model understandability and analyzability. High complexity architecture should have high analyzability.

The evaluation of PIM model with measurement of the whole architecture of the basic metrics (i.e. coupling, cohesion and complexity metrics [4]). The quality model is elaborated of the whole software system as well as its architectural artifacts in terms of quality factors (i.e. analyzability and modularity) and associated metrics (i.e. coupling, cohesion and complexity) is shown in Figure 9. The evaluation results are given in Tab. 1 using a prototype implemented in Java [4].

TABLE I. PIM EVALUATION RESULTS

PIM	coupling	cohesion	complexity
PipesAndFilters	0.482	0.341	0.362

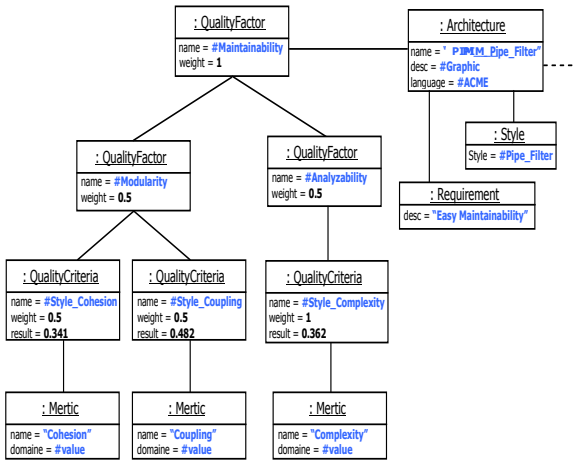


Figure 9. The elaborated quality model of PIM model

The architecture model should be 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 returned to the previous stage (i.e. CIM) either to change requirements or to elaborate a different (better) architectural model.

Il faut expliquer comment sont obtenues ces valeurs, l'unité de mesure, l'écart des valeurs (entre quoi et quoi), qu'est ce qu'une bonne valeur et une mauvaise et pourquoi. As for the architecture model from Table 1 the values of coupling is equal **0.482** and a threshold of coupling is equal **0.66**, the value of cohesion is equal **0.341** and a threshold of coupling is equal **0.5** and the value of complexity is equal **0.362** and a threshold of complexity is equal **1**, 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.

This result is practically significant as well related to maintainability effort, e.g. low level of coupling, dependencies among all architectural artifacts are loss, high number of reused artifacts (i.e. number of Pipe connector instances, m = 4).

B. *CPIM level*

1. CPIM level – MDA artifacts

PIM software architecture model may be transformed, manually or automatically, into different CPIM models. The software architecture model from Fig. 8 is transformed into four exemplary CPIM models – see Fig. 11 - and the total resource requirements are given in Table 2.

2. CPIM level – Internal quality model

At this level analyzability, time-behavior sub-factors taken from upper level are evaluated (it is worth to mention –

different metrics can be used for this purpose). The evaluation results should be helpful in choosing the best CPIM model for further transformation.

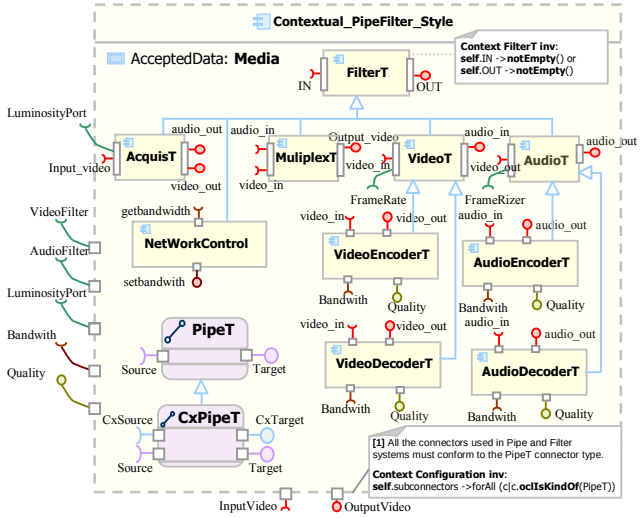
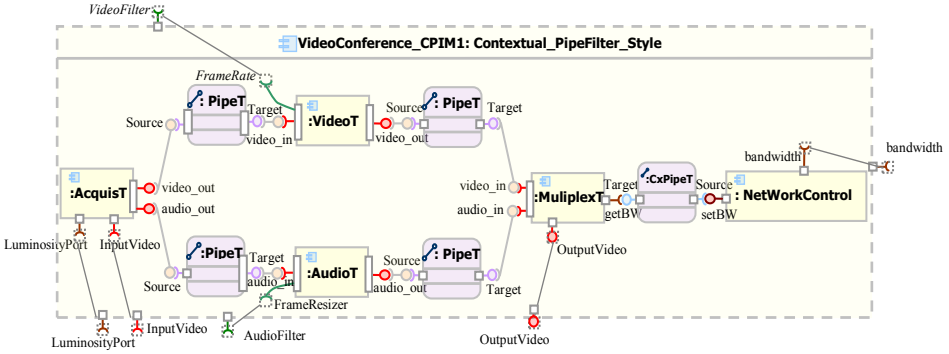


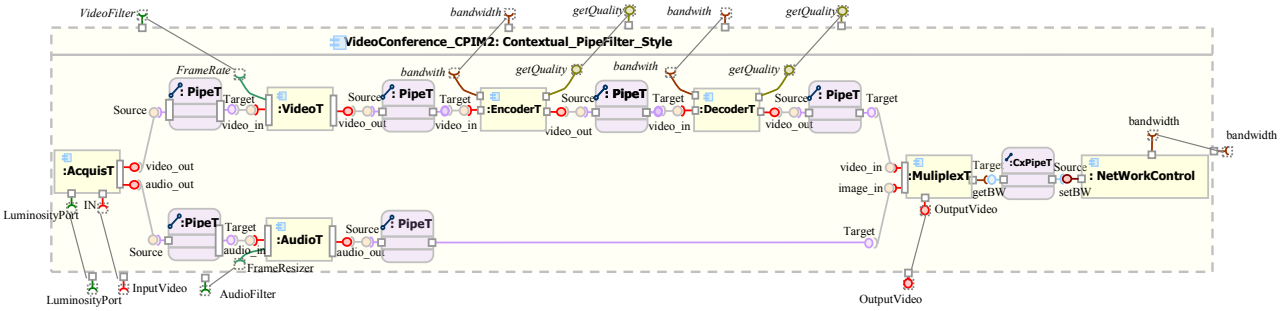
Figure 10. VideoConference with Contextual Pipe Filter Style.

TABLE II. NON-FUNCTIONAL REQUIREMENTS DETAILS.

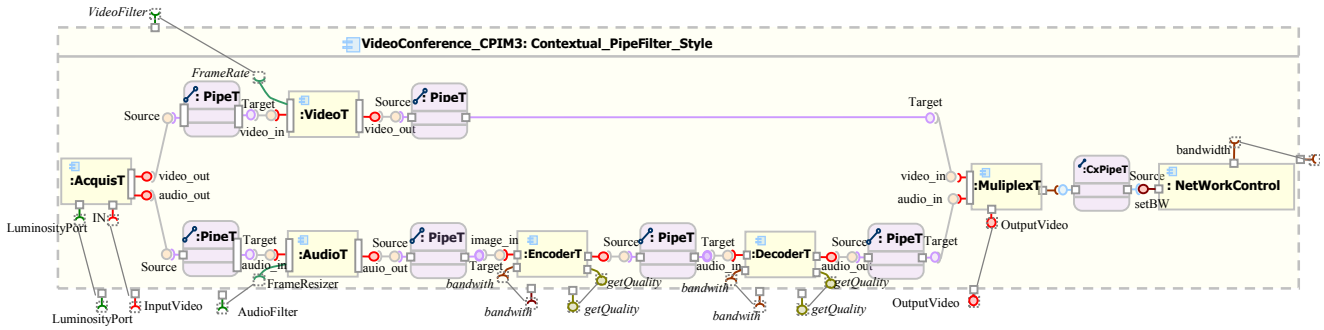
Component	*User preferences*	*CPU speed*	*Bandwith*
AudioT	-	≈ 100 MIPS	4:1 Reduction
VideoT	-	≈ 50 MIPS	2:1 Reduction
AudioEncoderT AudioDecoderT	High Quality Medium Quality Low Quality	≈ 200 MIPS	64 kbps 32 kbps 8 kbps
VideoEncoderT VideoDecoderT	High Quality Medium Quality Low Quality	≈ 800 MIPS	10:1 Reduction 20:1 Reduction 30:1 Reduction



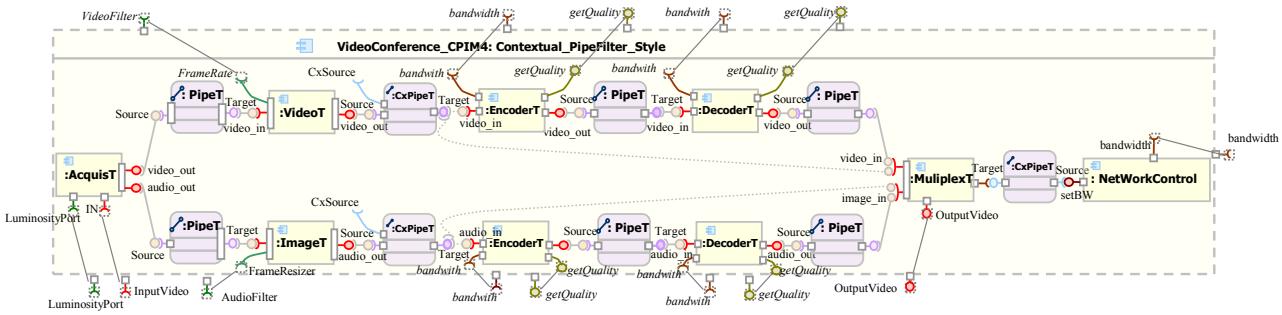
(a) Model I



(b) Model II



(c) Model III



(d) Model IV

Figure 11. Alternative versions of CPIM model

For time-behavior, two metrics proposed in [8], one of them is selected and adapted in our case. The estimated time behavior metric for a set ***A*** of artifacts of a given configuration performed with a given time in a certain context calculated as the weighted sum of TB_a metric counted for every artifact instance ***"a"***:

$$TBM^{Benefit}_{Memory_{size}, CPU_{speed}, Network_{bw}}(config) = \sum_{a \in A} w_a * TB_a \qquad (1)$$

where w_a - normalized coefficient of the number of artifact instances within a configuration

$TB_a^{context}$ – the estimated response time of artifact instance within a given data size in a certain context (CPU speed, Memory size…etc). et que représente TMB ?

Apart from the evaluation of time behavior sub-factor we evaluate the analyzability sub-factor to select the best CPIM model.

In [18] the analyzability sub-factor was investigated and the dynamic adaptivity as its indicator was validated. This adaptivity was considered at the architectural level. Two metrics were proposed in [18], but only one, *MaAC* (Minimum architectural Adaptive Cost) was validated for analysability assessment in our example. This metric related to the architectural growth indicates the minimum number of artifacts which should be added to make a system adaptive independently of the number of functionalities that it provides. Essentially, this metric expresses the fixed cost of adaptivity at the architectural level.

It may be necessary to perform a careful balancing between user preferences, limited device capacity and architectural model features. According to the choice made of the sub-factors of quality and their measurement, we define the function *Utility* which measures the utility of a given configuration as follows:

$$Utility\ (User, Env, Arch) = \frac{TBM^{Benefit}_{Memory_{size}, CPU_{speed}, Network_{bw}}(config)}{MaAC^{Cost}_{Memory_{size}, CPU_{speed}, Network_{bw}}(config)} \qquad (2)$$

Our selection strategy for a given mobile device is to assign qualities and allocate resources to result in the best configuration such that the system utility is maximized subject to device resource constraints, user preferences constraints, and architectural model features respectively. Given complete knowledge and centralized control of the system, the objective of mobile is to maximize the system utility function.

The primary use of such function is to provide feedback to an architect about the costs and quality of a given configuration in a certain context, allowing the architect to use a best configuration earlier before deployment phase by economizing major changes to reach the target architecture with acceptable time response. 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. The users use

the system as modeled as Poisson-process ????, and each mobile device's CPU speed is initialized with a random value in the range of [100, 800], and reduced automatically by a random value in the range of [0, 5] in each iteration (pourquoi ?). The simulation took not more than approximately 10 seconds on a 3.5 GHz Pentium PC using Mpeg 4 video format.

The table 3 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. This result is practically significant as well related to adaptation effort e.g. number of artifacts which should be added to make a system adaptive are very loss as consequence of self-reconfiguration for environment evolution (i.e. *CPU usage*) guided by the adaptation policies. While considering the transformation from CPIM to CPSM model, CPIM4 model should be chosen for it, trusting that all evaluated metrics are adequate indicators of efficiency and maintainability characteristic respectively.

TABLE III. CPIM Models Evaluation Results

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

VI. RELATED WORKS

The first related area of research are ADLs that have been proposed for representing dynamic architectures including: ACME [16], π-ADL [6], C3 [2, 7] 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. However, AADL does not support evaluation metrics to achieve a quality analysis. C3 [2] is concerned with the static structure of software components as well as their interactions. The authors do not mention in their proposal the need for supporting evaluation function and context information measure. In [7], Amirat and al. used UML profiles to describe the software architecture of systems. They use UML component diagrams to show the static configuration. But this work supports neither quality style evaluation nor transformation. π-ADL [6] is a formal architecture description language based on the π-calculus. It 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. [16] 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 [14] – 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, to achieve a comfortable architectural quality analysis framework.

VII. CONCLUSION AND PERSPECTIVES

The process of software systems design, according to the MDA approach, is based on the transformation of models at various levels of abstraction. As on any level the outcome of the transformation may comprise several models, the problem of choosing the best of them appears. It seems that the decisions concerning that choice should be based on the quality criteria. MDA approach is focused on transformations of models that maintain mainly functional requirements – the non-functional requirements are rather not taken into account.

This paper proposed *ContextualArchRQMM* metamodel centered on the concept of contextual connector, which take advantage of traditional architectural connectors and provides a lightweight support for the definition of some composition facilities such as heterogeneous 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.

The paper proposed also CQ-MDA approach based on ContextualArchRQMM, 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 of three abstractions levels: PIM, CPIM and CPSM. At the PIM level, a model is decomposed on 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 are 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.

The short period of experimentation of the system has shown the interest of the application of such strategy when considering the increasing use of UML. 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 *VideoConference* 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. In the future, we will consider moving our approach to a real execution platform to validate its feasibility.

ACKNOWLEDGMENT

Our thanks give to reviewers for their helpful suggestion and reviews.

REFERENCES

[1] B. Berthomieu1, J.P. Bodeveix, C. Chaudet, F. Vernadat, "Formal Verification of AADL Specifications in the Topcased Environment," *14th Ada-Europe International Conference*, 2009, pp. 207 – 221.

[2] A. Amirat and M. Oussalah, "First-Class Connectors to Support Systematic Construction of Hierarchical Software Architecture," *Journal of Object Technology*, Vol. 8. N°.7, 2009,pp. 107-130.

[3] A. Alti, A. Boukerram and A. Smeda, "Architectural Styles Quality Evaluation and Selection," *Proceedings de 9éme Conférence Internationale sur Les NOuvelles TEchnologies de la Repartition Software and Technologies (NOTERE'09)*, Montréal (Canada), 2009.

[4] A. Alti, A. Smeda, "Architectural Styles Quality Evaluation and Selection," *Proceeding of 4th International Conference on Software and Technologies* (ICSOFT'2009), Barcelona (Spain), 2009, pp. 74 - 82.

[5] J. Miller, J. Mujerki, editors. "MDA Guide, Version 1.0. OMG Technical Report,", http://www.omg.org/docs/ptc/03-05-01.pdf, 2003.

[6] F. Oquendo, "π-ADL: an architecture description language based en the higher order typed π-calculus for specifying dynamic and mobile software architecture," ACM Software Engineering Notes, vol. 29, n°. 4, 2004, pp. 1 - 13.

[7] A. Amirat and M. Oussalah, "Towards an UML Profile for the Description of Software Architecture," *Proceedings of International Conference on Applied Informatics (ICAI'09)*, 2009, pp. 226 - 232.

[8] ISO/IEC 9126-1. *In Software Engineering – Product quality – Part 1: Quality model*, ISO-IEC, 2001.

[9] QADA, http://virtual.vtt.fi/qada , 2007.

[10] P. Tarvainen, "Adaptability Evaluation at Software Architecture Level, " *The Open Software Engineering Journal*, vol. 2, Bentham Science Publishers Ltd., 2008, pp. 1-30.

[11] F. Losavio, L. Chirinos, N. Lévy, and A. RamdaneCherif, "Quality characteristics for software architecture," *Journal of Object Technology*, vol. 2. n°.2, 2003, pp. 133-150.

[12] OMG, Business Process Modeling Notation (BPMN) http://www.bpmn.org/Documents/ , 2006.

[13] OMG. UML OCL 2.0 Specification: Revised Final Adopted Specification. http://www.omg.org/docs/ptc/05-06-06.pdf, June 2005.

[14] S. Vale, S. Hammoudi, Context-aware Model Driven Development by Parameterized Transformation. *Proceedings of 3rd Workshop of Model Driven Interoperability for Sustainable Information Systems (MDISIS'2008)*, 2008, pp. 167–180.

[15] R. Bashroush, I. Spence, P. Kilpatrick, J. Brown, "Towards More Flexible Architecture Description Languages for Industrial Applications," *Proceedings of 3rd European Workshop on Software Architecture (EWSA'2006),* pp. 212- 219, Springer-Verlag, LNCS 4344, Nantes, France, September 4-5, 2006.

[16] D. Garlan, J.M. Barnes, B. Schmerl, O. Celiku., "Evolution Styles: Foundations and Tool Support for Software Architecture Evolution," *WICSA'09*, 2009.

[17] S. Laplace, M. Dalmau, P. Roose, Prise en compte de la qualité de service dans la conception et l'exploitation d'applications réparties, *In the Workshop GEDSIP@Inforsid 2009*, Toulouse, 26 mai 2009.

[18] C. Raibulet, L. Masciadri, "Evaluation of Dynamic Adaptivity through Metrics: an Achievable Target?," *WICSA'09*, 2009.