ARCHITECTURAL STYLES QUALITY EVALUATION AND SELECTION

Smeda Adel and Alti Adel
LINA, University of Nantes
2 Rue de la Houssinière, BP 92208
44322 Nantes Cedex 03, France

Keywords: Evaluation of the software architecture, Quantitative architecture factors, OCL, Metamodel.

Abstract: Today, many architectural styles have been proposed and many others are being defined. An architectural style provides a domain-specific design vocabulary and a set of constraints on how that vocabulary is used. Given the increasing complexity of architectural styles, designing a sound and appropriate architectural style becomes an important and intellectually challenging task. In order to analyze architectural styles quality factors at architecture level and meta level are needed. In this article we propose a metamodel for quality evaluation and selection of architectural style. Our metamodel includes a set of metaclasses; these metaclasses are constrained with formal OCL rules. These constraints allow us to improve the verification of the properties' quality of the architectures by modelling styles of the software system. With this metamodel, all the properties' quality of the final production are granted by the software architecture.

1 INTRODUCTION

Today, all of the software developers are regarding the quality of their productions as their own main purpose. The experiences have showed that whenever it is necessary to design a product with high demotion and complexity, a general view that is called "architecture" is needed. The meaning of architecture is to provide a formal model of the system in terms of components and connectors and how they are composed together. Architecture gives us an overall point of view of the whole system.

One of the important cornerstones of modern software architecture is the use of architectural styles. An architectural style defines a family of related systems, typically by providing a common vocabulary for architects, allowing the reuse of architectures across many products. Consequently more and more architectural styles are being defined every day (Buschman, Henney and Schmidt, 2007; Zudan and Avgeriou, 2008). They became so divers, that their evaluation becomes problematic. Unfortunately, despite significant progress in analysis of the architectures for software systems, there is relatively little work in styles evaluation and selection.

This work proposes a novel metamodel called ArchRQMM (ARCHitecture Requirement Quality MetaModel) for the evaluation of properties' quality of architectures on modelling styles for software system. The *contribution* consists in extending the core concepts of Architecture Description Languages (ADLs) to integrate the concepts of quality requirements and quality standards. Our proposal offers an automatic evaluation and selection of styles that best meet architects' needs and allows a rigorous evaluation to prove the quality of architectural styles at the architecture level.

The proposal work presented here is the result of our previous work on using profile transformations for integrating software architecture concepts into the Model Driven Architecture (MDA) platform (Alti, Khammaci, Smeda and Bennouar, 2007). We identified the need to specifically select an appropriate style that meets quality expectations. Our integration process needs a generic metamodel for the evaluation of architectural styles more precisely and more objectively.

The rest of this article is organized as follows: Section 2 presents our motivation and glances at other related works. Section 3 describes the ArchRQMM metamodel. Section 4 illustrates the applicability of the previous metamodel both for

Adel A. and Adel S. (2009).
ARCHITECTURAL STYLES QUALITY EVALUATION AND SELECTION.
In *Proceedings of the 4th International Conference on Software and Data Technologies*, pages 74-82
DOI: 10.5220/0002253100740082

evaluation and for selection of an architectural style. Finally, section 5 concludes this article and presents some future works.

2 MOTIVATION AND RELATED WORK

2.1 Management of Software Architecture Qualities

Tibermacine et *al*. (Tibermacine, Fleurquin & Sadou, 2006) assisted quality in component- based software evolution. They used a generic architecture metamodel ArchMM for architecture design and used ACL (Architecture Constraint Language) as means for formally describing architectural choices. This work concentrated on architecture evolution, while our proposed approach focuses on architectural styles available in different ADLs. Our approach is similar to Tibermacine et al. In our proposal, quality is placed on top of the system, which drives the rest of the development process, plays a central role that defines the structure of our concrete model's architecture, and decides which architectural styles are considered and which are not. Also closely related to our research is the work on evaluation of ADLs for their support to model architecture patterns (Grau and Franch, 2007a). Their works is focused on the use of patterns to design software architecture. They explored the suitability of UML and five ADLs (ACME, Wright, Aesop, Unicon and xADL) for modeling architecture patterns. They chose syntax, visualization, variability, and extensibility as criteria for selecting ADLs that can represent architecture patterns. They concluded that most of the ADLs specify strong notational, analysis and tool support to design software architectures but they still have considerable drawbacks like supporting styles. Recently, (Grau and Franch, 2007b) proposed a goal-oriented approach for the generation and evaluation of alternative architectures based on existing architectural styles. Its approach allows a well evaluation of architectural styles but lacks a catalogue of architectural styles and a formal evaluation of architectural styles. More recently, in (Zudan and Avgeriou, 2008) presented a new catalog of architectural primitives for modeling each architectural style and provides a suitable base to explicit and formally modeling styles in UML 2.0. Its approach helps in designing correct styles but suffered from a formal evaluation of complex architectures based on multiple styles for achieving an efficient framework.

2.2 Uses and Reasons of Software Architecture Importance

To investigate the importance of software architecture from the technical point of view, for three reasons (Klein, Clements, Kazman, 2002):

- *Possible reusability in architecture*: Patterns and styles are a way to reuse software development knowledge on different levels of abstractions. So it is possible that a given system described with different styles. Style selection has specific importance in developing software systems.
- *Early decision making of designation:* Software architecture includes high level decisions and trade–off that leads to produce a software system and define its characteristics. The studies show that the expense of correcting a discovered error along the requirements recognition phase or in the architecture phase is more less than correcting that error when the testing phase.
- *Architecture as a means of relation between the system stakeholders*: Software architecture system can be a common view of all stakeholders. If they have the same idea regarding the developed software, they can have a common base for discussion and evaluating the system.

2.3 Objectives and Motivations

Currently, the software architecture was designed with an arbitrary ADL and arbitrary architectural style and integrated into a given MDA platform. However, a rigorous quality evaluation of architectural styles is not considered by the model-driven software architecture integration. This results to basic drawbacks:

- The poor architecture quality results from employing wrong architectural style for quality improvement,
- Difficulty in meeting quality designs such as efficiency and maintainability,
- The quality software management depends on their architectural styles and implementation platforms.

These disadvantages can be tided, if we introduce architecture evaluation concerns in the model-driven software architecture as configuration inputs and apply a formal quality evaluation at the

architecture design step and not as an afterthought during the transformation for the final system.

3 PROPOSED METAMODEL

In order to build a solid system, it is essential to systematically take into account all the reflections regarding the system at the architecture design step and not as an afterthought during the transformation. From this point of view, we can not describe software systems with an arbitrary architectural style but we must select one style among various. Of course, style provides guidance for building a broad class of architectures in a specific kind of system, but what are the benefits that the software system gains? An architectural style, answers the architecture design needs and its quality characteristics. Therefore, we suggest placing the architecture quality evaluation as control-center for designing software architecture systems.

3.1 Description of ArchRQMM

In order to support the evaluation of quality of architectural artifacts produced in architectural level, we have defined mainly three complementary models. We use software architecture model to describe architectures based on the core concepts in each ADL, we use requirement model to represent architect's needs and the quality goals and we use architecture quality model to evaluate and analyse the quality of the whole software system as well as its architectural artifacts. The key-idea of the proposed metamodel is the combination of measurable standards at the level of architecture with OCL constraints, resulting to the overall software system quality increase and conformance.

3.1.1 Software Architecture Model

The core elements of the software architecture model are components, connectors and configurations; each of these elements has an interface to interact with its environment as shown in Figure 1. Besides, the abstract class *Artifact* gathers all the structural and behavioral information that is shared by components, connectors, and configurations and therefore does not have conceptual correspondence in traditional architectural models. *Architecture* may be composed of many artifacts. *Components* are potentially composite computational encapsulations that support multiple interfaces known as *ports*. *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). Architectural *styles* define sets of types of components, connectors, properties, and sets of constraints on how they can be combined. The software system can be described by an *architecture* with different *styles*. We have selected four styles *Layers, Pipe-Filter, Blackboard, and Client-Server* (Buschman, Henney and Schmidt, 2007) because they are the most commonly used in practice and they represent a number of different domains and concerns. Layers demands grouping of components, Pipe-Filter handles streams of data, Client-Server is frequently used in distributed systems, and Blackboard is for dynamic configurations. Although, we limit ourselves to only four styles, we emphasize that our metamodel is not meant to be exhaustive.

3.1.2 The Requirement Model

The architect's needs (*Requirement* class) should fulfil particular architecture artifacts (*Artifact* class in the model). Usually the necessities of a software system are divided in to two groups: *Functional requirements* and *Non-Functional requirements*. Functional requirements define the functional and executive purpose of the system. Non-functional requirements mostly focus on how a software system works and performs. In our case, 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 formalized using the standard ISO-9126 (ISO-IEC, 2001). Based on final quality aims, the developed architecture for a given software system can be evaluated to satisfy quality goals.

3.1.3 Software Architecture Quality Model

The model defines the 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

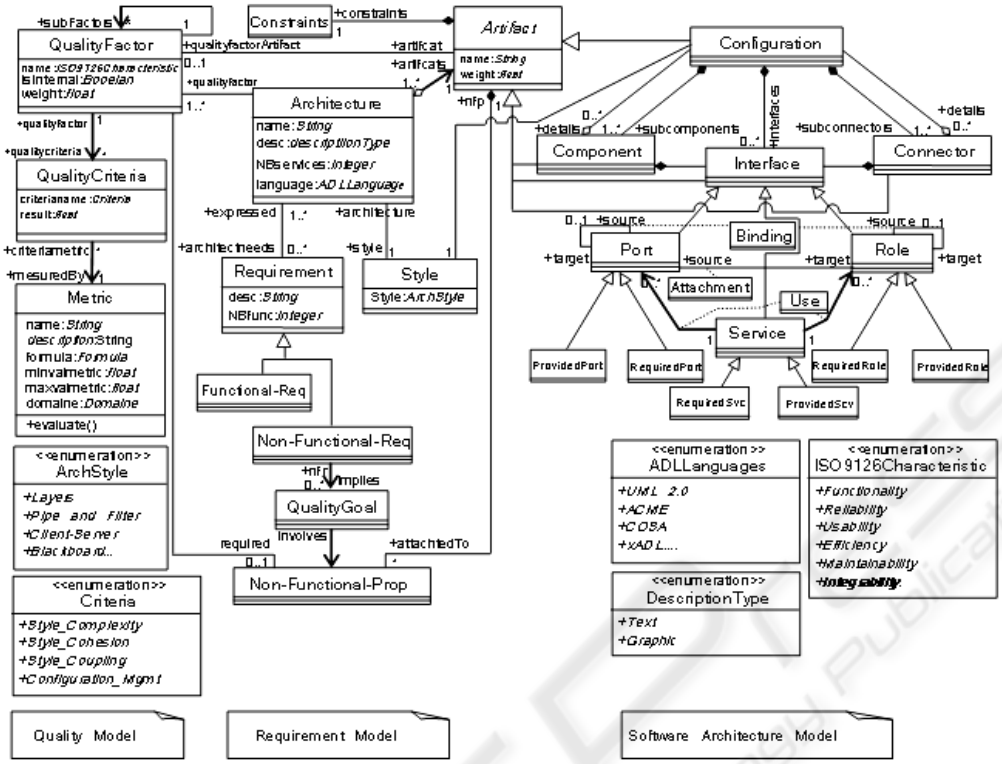

Figure 1: The ArchRQMM meta-model.

instance of *QualityFactor* class is the root of quality factors and sub-factors, and represents a given quality perspective. 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.

Such model covers the factors and sub-factors based on the quality standard ISO-9126 (ISO-IEC, 2001). Based on the norm ISO-9126, Losavio et *al.* (Losavio, Chirinos, Lévy, Ramdane-Cherif, 2003) defines six characteristics: functionality, reliability, usability, efficiency, maintainability and portability. They address the specification of software architecture requirements and its quality characteristics. However, this model has number of limitations including: many of the factors suggested by this model are not directly related to the specific issue of integration contributed to the malfunction modes of software architecture, it separates the concepts of modularity and coupling whereas the modularity of software architecture is related to the components depending. Making software architecture more modular is not sufficient if it has uncouple functions, which will provide low modularity conditions.

To overcome these limitations we propose a model that is based on the factor "criteria and metrics" as shown in Figure 2. The sub-factors of software architecture may be decomposed into two quality sub-factors: modularity and analyzability. Each sub-factor may be further expressed by a set a set of lower level quality metrics, which are directly measurable. Although, we limit ourselves to only three criteria's, we emphasize that our software architecture quality model is not meant to be exhaustive.

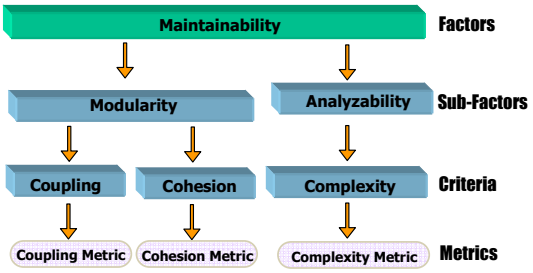

Figure 2: Software architecture integration framework.

Architecture Modularity. The sub-factor can be evaluated by controlling the modularity level of the system. The architecture 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 paper (Jihuna, Zhenbo, Zhao, Zhenhua and Ruijin, 2007.) these two metrics are proposed for measuring architecture stability. We adopted these metrics and used in our model. The coupling of the architecture is a global property relative to the exchanges between two components. Consider *T* the set of different artifacts types of the style *S*, and *n* is it's the set size. The artifact type $T_i \in T$ (i = 1,...n) is instantiated into the set $A_i = \{a_{i,1}, \ldots a_{i,m}\}$, and *m* is its the set size. The weight of the type T_i is its impact on the architecture reconfiguration by hiding dependencies at different layer of abstraction. We use the ROC (Rank Order Centroids) concept to measure a weight of the artifact type T_k:

$$weightType\ (T_k) = \left(\sum_{i=k}^{n} (1/i)\right) \Big/ n \tag{1}$$

The weight of an artifact $a_{k,j}$ is its degree in the architecture defined as follows:

$$weight\ (a_{k,j}) = weightType\ (T_k) / m \tag{2}$$

The architecture-based coupling metric for the style S is defined as follows:

$$StyleCop = \frac{\sum_{\forall T_1, T_2 \in S} Cop(T_1, T_2)}{\|S\|^2 - \|S\|} \tag{3}$$

where *Cop* (T_1, T_2) is the coupling of the artifact type T_1 to the artifact type T_2 defined as follows:

$$Cop\ (T_1, T_2) = \sum_{\forall a_{1,i} \in A_1, \forall a_{2,j} \in A_2} Cop\ (a_{1,i}, a_{2,j}) \tag{4}$$

and *Cop* ($a_{1,i}$, $a_{2,j}$) is the coupling of an artifact instance $a_{1,i}$ to an artifact instance artefact $a_{2,j}$ defined as follows:

$$Cop\ (a_{1,i}, a_{2,j}) = \frac{\sum_{k=1}^{l} assoc\_k\ (a_{1,i})}{\sum_{l=1}^{h} assoc\_l\ (a_{1,i}, a_{2,j})} \tag{5}$$

Attachments/bindings associations (*assoc*) connect port and role of the different/same types are assigned same weights (*weights = 1*). Components, connectors, configurations stand for artifacts a_1 and a_2 instantiated of respective types T_1 and T_2. At the architecture level, low architectural configuration coupling is considered to be a desirable quality for a modular software system.

The *cohesion* expresses the number of components that depend on other components in a given architecture. The architecture-based cohesion metric for the style *S* is defined as follows:

$$StyleCoh = \frac{\sum_{\forall T_i \in S} Coh\ (T_i)}{\|S\|} \tag{6}$$

where *Coh* (T_i) is the cohesion of the artifact type T_i defined as:

$$Coh\ (T_i) = \sum_{a_{i,j} \in A_i} weight\ (a_{i,j}) \times Coh\ (a_{i,j}) \tag{7}$$

and *Coh* ($a_{i,j}$) is the cohesion of the artifact instance $a_{i,j}$ defined as:

$$Coh\ (a_{i,j}) = \frac{\sum_{k=1}^{h} assoc\_target_k\ (a_{i,j})}{\sum_{l=1}^{w} assoc\_l\ (a_{i,j})} \tag{8}$$

Attachments and bindings represent different associations (*assoc*) that join different artifacts instances together (resp. *components instances*, *connectors instances*, *configurations instances*) and *assoc_target* is the number of provided ports (resp. provided roles) of an artifact instance "*a*" connected as *Target* ports (resp. *Target* roles) of attachments/bindings associations with all other artifacts in the architecture.

High cohesion and low coupling are the main facts to take into account for achieving modular architecture when applying styles. Apply these basic principles can makes a design understandable, maintainable, and of higher quality. High cohesion and low coupling are the main facts to take into account for achieving modular architecture when applying styles. Apply these basic principles can makes a design understandable, maintainable, and of higher quality.

The architecture-based modularity sub-factor for the style *S* is expressed as follows:

$$StyleMod = StyleCoh / StyleCop \tag{9}$$

Architecture Analyzability. Analyzability emphasizes on possible recognition of manners and deficiencies of an architecture. This sub-factor also tries to define the parts that must be corrected. The complexity indices for conforming style understandability and analyzability. It is believed that high complexity architecture should have high analyzability. The proposed measure of architecture complexity is based on the criteria, named *structural dependency measures* inspired by complexity metric dedicated to Component-Based Architecture (Böhme and Reussner, 2008). We adopted this metric and

used in our model. The structural dependency measure for each artifact a_i in the architecture is obtained as follows:

$$SDM(a_i) = \sum_{j=1}^{n} Dep(i,j) = \sum_{j=1}^{n} DepT(i,j,\pi_{\max}) \quad (10)$$

Where components instances, connectors instances, configurations instances stand for artifact a_i. The measure of the strength of association established by a path (i.e. direct and transitive connection) from an artifact a_i to all other artifacts in the system is defined as its structural dependency measure. Attachments and bindings represent direct dependency edges. Consequently, the sum of weighted structural dependency measures of all artifact instances of the type Ti is defined as:

$$SDM(T_i) = \sum_{\forall a_j \in I_i} weight(a_{i,j}) \times SDM(a_{i,j}) \quad (11)$$

Finally, the style structural complexity, *StyleComp*, is defined as the mean SDM of all artifacts types:

$$StyleComp = \frac{\sum_{\forall T_i \in S} SDM(T_i)}{\|S\|} \quad (12)$$

The architecture-based analyzability sub-factor for the style *S* is the architecture-based complexity metric defined as follows:

$$StyleAna = StyleComp \quad (13)$$

According to the choice made of the factors of quality and their measurement, we define the function *Quality* which measures the quality of the architecture for a given architectural style as a linear combination of each evaluated measure function. The weight associated with each function allows the software architect to modify the importance of each quality sub-factor. The final quality of the style *S* is defined as follows:

$$Quality = \alpha_1 * StyleMod + \alpha_2 * StylehAnc \quad (14)$$

With such evaluation, an architectural style that needs to be revised and improved can be determined and then given a suitable attention.

3.2 OCL Constraints

In order to assess the quality of software architecture models on modeling architectural styles in the context of all requirements as well in the context of selected requirements or factors, we use OCL 2.0 (Object Management Group, 2005) language to specify the properties and verify the model.

The focus of rigorous architecture quality analysis is to prevent the non-required affections after the design step before the early phases of system development. For example, the configuration modularity, architectural style stability and maintenability are given by the following constraints:

A configuration provides an acceptable modularity if all its subcomponents and its subconnectors present a high level of cohesion and a low level of coupling.

```
Context ArchRQMM:Configuration inv:    --Constraint C1
self.qualityfactorArtifact.subfactors.qualitycriteria
 ->exists->(sf|sf.name= #Modularity) implies
    (self.subcomponents.qualityfactorArtifact.
       qualitycriteria->select(c.criterianame=#Cohesion
          implies c.result >=0.5) -> notEmpty() and
        select(c.criterianame= #Coupling
           implies c.result <=0.66) -> notEmpty()) and
    (self.subconnectors.qualityfactorArtifact.
       qualitycriteria->select(c.criterianame=#Cohesion
          implies c.result >=0.5) -> notEmpty() and
        select(c.criterianame= #Coupling
           implies c.result <=0.66) -> notEmpty())
```

An architectural style is stable if all its architectural artifacts present a high level of cohesion (i.e. 1)

```
Context ArchRQMM:Style inv:          -- Constraint C2
self.architecture->forall(a|a.artifacts
 ->forAll(a|a.qualityfactorArtifact.qualitycriteria->
    select(c|c.criterianame=#Cohesion implies
           c.result=1)->notEmpty())) implies
   self.architecture.qualityfactor->includes(#Stability))
```

An architectural style that provides an acceptable maintainability must have structural complexity less than predefined threshold (the result exceeds such a threshold, a decision should be made about to elaborate a different (better) architectural style).

```
Context ArchRQMM:Style inv:              -- Constraint C3
self.architecture->forall(a|a.artifacts
 ->forAll(a|a.qualityfactorArtifact.qualitycriteria->
  select(c|c.criterianame=#Complexity implies
     c.result<=c.threshold)->notEmpty())) implies
self.architecture.qualityfactor->includes(#Analyzability))
```

3.3 The Evaluation and Selection of Architectural Styles

The process of the evaluation and selection started with designing the architecture model conforms to the software architecture ArchRQMM metamodel, next producing the quality model conforms to the software architecture quality 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. After that, the model is evaluated by the semantic constraints defined by the ArchRQMM metamodel. An important feature of the ArchRQMM metamodel is the possibility of architecture model(s) checking. This is verified by OCL constraints that can be checked for a given requirement, criterion, artifact, and for the whole software architecture (i.e. set of artifacts). The

results can influence the quality of architecture model. Two ways of using the ArchRQMM are possible:

- Formal Verification: The ArchRQMM 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 returned to the previous stage either to change requirements or to elaborate a different (better) architectural style.
- Quality Evaluation and Selection: The ArchRQMM metamodel is used for selecting the best architectural style from different choices. In this case the values of a metric are used classifying the models. In this case a metric formula gives a note for the architecture with each of the given styles. The values of the metric function are used to classify the models and to choose the suitable one. After that, the selected architectural style is evaluated by the OCL constraints to remove any violation.

4 CASE STUDY AND EVALUATION

In order to validate the proposed metamodel, we developped ViSAQE: (Visual Software Architecture Quality Evaluator) prototype in Eclipse 3.2. The prototype tool supports creating and managing architectures with different styles, allows graphical representation of architectures and interoperability of models using different ADLs through standard XMI, enables the automatic evaluation of architectural metrics and offers to the architects the possibility to elaborate quality models and to validate its semantics with ArchRQMM.

In current version, it translates the ACME models (Klein, Clements, and Kazman, 2002) supported by three styles: *Client-Server*, *Pipes-Filters*, *Pipes-Filters* and *Client-Server* and provides a common language for describing formal semantics with OCL. To illustrate the application of the ViSAQE for both evaluation and selection of an architectural style we use the CaPiTaLiZe system (Abi-Antoun, Aldrich, Garlan, Schmerl, Nahas, Tseng, 2005) as an example. The following architect requirements are considered:

- Recording, rewriting, updating source data and creating respective target data.
- The system should be easy maintained.

The first requirement is functional requirement while the second is non-functional. According to ArchRQMM all these requirements should be associated with a respective architecture quality model with selected quality factors. In our case study, for illustration only non-functional requirements is taken into account. The architect developer selected to use the maintainability factor with analyzability and modularity sub-factors. For the CaPiTaLiZe system, the model is designed with the prototype tool using three styles, as shown in Figure 3, Figure 4 and Figure 5. We have evaluated each kind of XMI-based models with similar measurements of the whole architecture of the basic metrics described in a previous section. The evaluation results are given in Table 1. According to the results given in Table 1, Pipe-Filter style turns out to be the best choice (scored better for coupling and complexity). This result is practically significant as well related to maintainability effort, e.g. low level of coupling, dependencies among all artifacts are losse, high number of reused artifacts (ex. number of Pipe connector instances = 4). Architecture complexity can be simplified by hiding dependencies at different layers of abstraction. An example of that is the combination of two architectural styles: *Pipe-Filter* and *Client-Server*.

Table 1: Architectural Styles Evaluation Results.

Metrics	C-S	Pipe-Filter	C-S and Pipe-Filter
Coupling	0.748	0.482	0.606
Cohesion	0.450	0.341	0.402
Complexity	0.513	0.362	0.447

From a maintenance perspective, this allows maintainers to modify and change components behind other artifacts types (e.g. *Connector Pipe*) with minimal impact on the rest of the system.

The selected architectural style (i.e. *Pipe-Filter*) for the final mapped system as shown in figure 6 is tested and validated with the semantic constraints defined by the ArchRQMM metamodel.

5 CONCLUSIONS

This paper defines ArchRQMM: a metamodel for evaluating and selecting an appropriate architectural style and formally evaluating architectural styles. We have also illustrated the usefulness and importance of non-functional requirements and quality criteria in selecting architectural style. We

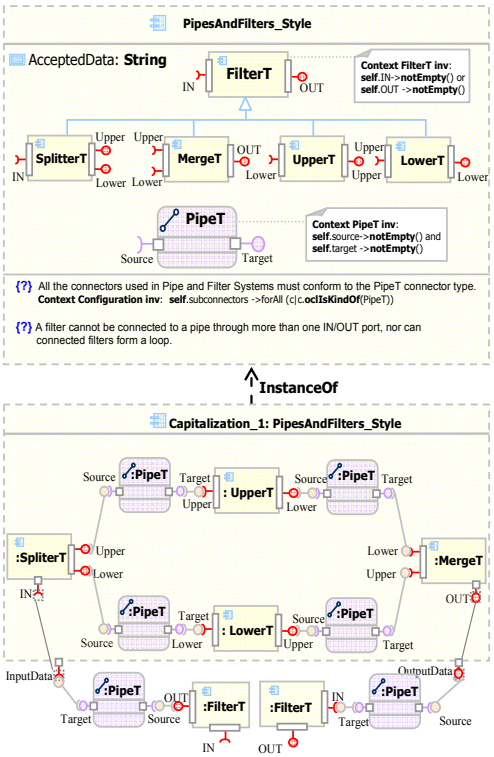

Figure 3: Capitalize System of Client-Server Architectural Style in ArchRQMM.

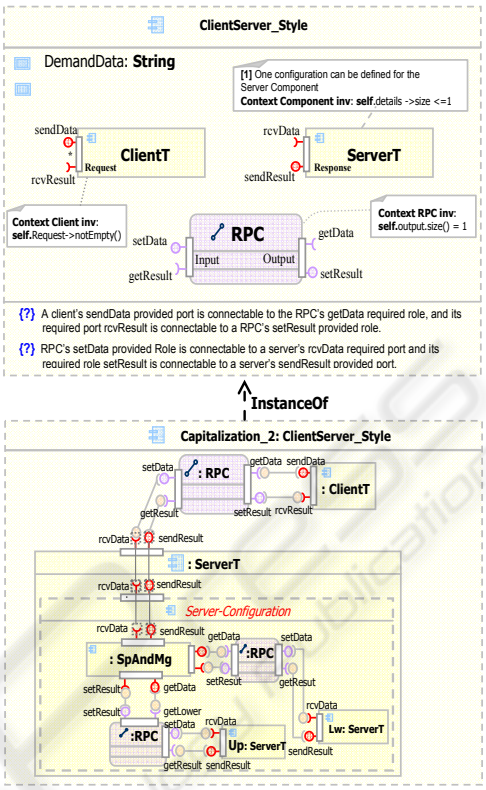

Figure 4: Capitalize System of PipesAndFilters Architectural Style in ArchRQMM.

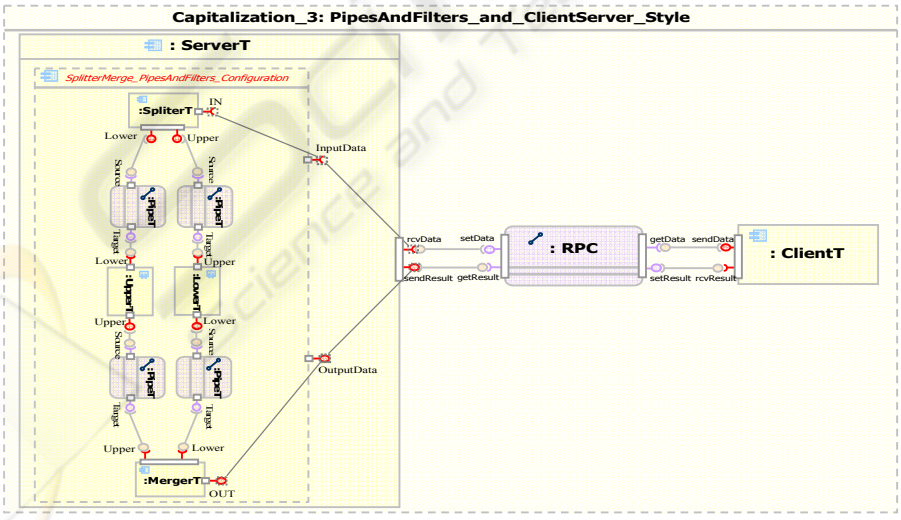

Figure 5: Capitalize System of PipesAndFilters and Client-Server Architectural Style in ArchRQMM.

presented an illustrative example to show the applicability of the proposed architecture quality metamodel. The results of the experiments (based on the Capitalize system with two known styles through ACME ADL) are encouraging. Since most ADLs provide important capabilities to express most structural aspects of software systems but lack support for architectural styles verification, having a semi-automated tool to assist in quality evaluation and selection of architectural styles is very important

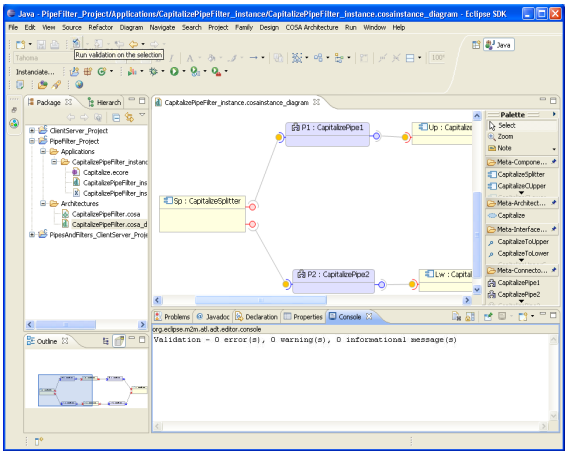

Figure 6: Validating Capitalize system of Pipe-Filter in ArchRQMM with ViSAQE.

in software architecture research area.

The limitation of our current approach is the fact that it only deals with structural properties of architectural styles and, therefore, it does not support architectural behavior, automatic generation of alternatives application models. These are open issues for our future works.

RFERENCES

Abi-Antoun, M., Aldrich, J., Garlan, D., Schmerl, B., Nahas, N., Tseng, T., 2005. Modeling and Implementing Software Architecture with Acme and ArchJava. In the *27th International Conference on Software Engineering*, pp.663-669, St Luis, USA.

Alessandro G., Thais, A.B., Awais, R.S., 2006. Driving and managing architectural decisions with aspects. *ACM SIGSOFT Software Engineering Notes*, Vol. 31 No. 5, pp. 30-37.

Alti, A., Khammaci, T., Smeda, A., Bennouar, D., 2007. Integrating Software Architecture Concepts into the MDA platform. In *ICSOFT'2007, 2nd Int. Conf. on Software and Technologies*, Barcelona, Spain.

Böhme, R. Reussner, R. 2008. Validation of Predictions with Measurements three Dependability Metrics. Springer-Verlag, LNCS 4909, pp.14-18.

Buschman, F., Henney, K., Schmidt, D., 2007. Pattern-Oriented Software Architecture, on Patterns and Patterns Languages. *Wiley Series in Software Design patterns*, Vol.5, August 2007.

Garlan, D., Monroe, R.T., David, W., 2000. ACME: architectural description of component-based systems. In *Foundations of component based systems,* pp. 47–67. Cambridge University Press.

Grau, G., Franch, X., 2007a. An Evaluation of ADLs on Modelling Patterns for Software Architecture. In *RISE'07, 3rd Int. Workshop on Rapid Integration of Software Architecture Engineering*, Luxemburg, LNCS 4063, Springer-Verlag, pp. 24-26.

Grau, G., and Franch, X., 2007b. A Goal-Oriented Approach for the Generation and Evaluation of Architectures Alternatives. *LNCS*, pp. 139-155.

ISO-IEC., 2001. ISO/IEC 9126-1 in Software Engineering– Part 1: Quality model.

Jihuna, L., Zhenbo, G. Zhao, Z., Zhenhua, Z., Ruijin, P., 2007. Towards Quantitative Evaluation of UML based Software Architecture. In *8th ACIS International Conference*, pp.663-669.

Klein, M., Clements, P., and Kazman, R., 2002. Evaluation Software Architectures: Methods and Case Studies, Addison Wesley.

Losavio, F., Chirinos, L., Lévy, N., Ramdane-Cherif, A., 2003. Quality characteristics for software architecture. *Journal of Object Technology*, Vol. 2, No.2, pp.133-150.

Object Management Group., 2005. UML OCL 2.0 Specification: Revised Final Adopted Specification. http://www.omg.org/docs/ptc /05-06-06.pdf.

Tibermacine, C., Fleurquin, R., and Sadou, S., 2006. On-Demand Quality-Oriented Assistance in Component-Based Software Evolution, *In the 9th ACM SIGSOFT CBSE'06*, Västeras, Sweden, pp. 294 - 309, LNCS 4063, Springer-Verlag.

Zudan, U., Avgeriou, P., 2008. A Catalog of Architectural Primitives for Modeling Architectural Patterns, *Information and Software Technology,* Vol. 50, pp. 1003F - 1034.