Using Model Transformation Techniques for the Superimposition of Architectural Styles

Elena Navarro[1], Carlos E. Cuesta[2], Dewayne E. Perry[3], Cristina Roda[1]

[1] Computing Systems Department, University of Castilla-La Mancha
enavarro@dsi.uclm.es, cristinarodasanchez@gmail.com
[2] Dept. LSI2 (Computing Languages and Systems II), Rey Juan Carlos University
carlos.cuesta@urjc.es
[3] Electrical and Computer Engineering Department, The University of Texas at Austin
DewaynePerry@engr.utexas.edu

Abstract. Software Architecture is a key artifact in the software development process, as it provides a bridge between the requirements of the system-to-be and its final design. Architectural description is therefore a critical step, which can be assisted by the use of Architectural Styles. Styles make it possible to reuse architectural knowledge by providing guidelines for its description, and by constraining the configuration and behavior of the target system. The architect must superimpose these constraints, but this could be an error-prone task unless some kind of automatic support is provided. Therefore, this paper presents a proposal that generates proto-architectures by superimposing architectural styles on the initial requirements' operationalization, using model-to-model (M2M) transformation techniques. Our proposal includes a tool called MORPHEUS, which applies QVT as the transformation language; a real-world example is provided to explain how the superimposition process works, and how the QVT language is used to express these style-based transformations.

Keywords: architectural style, model-driven development, architectural description, model transformations

1 Introduction

The specification of the *Software Architecture* (SA) is always a challenging activity. It is a decision making process that establishes strict compromises at the architectural level. These compromises must be reached in order to elaborate a specification that is able to meet both functional and non-functional requirements. In this context, the proper use of *Architectural Styles* can be a great asset for the process.

According to Perry & Wolf [15], an architectural style is something that "abstracts elements and formal aspects from various specific architectures". This definition is deliberately open; it can be used to describe full systems, or just a specific aspect of the architecture at hand [16], which can be composed to others. In addition, a style may also specify constraints on those elements and/or formal aspects, [18] as well as on the system behavior. Thus, they can affect the configuration of the architecture.

However, to the best of our knowledge, there are very few proposals that clearly establish how the architectural styles can be automatically or semi-automatically used to describe the SA. Many papers have focused their efforts on the classification of Architectural Styles, or their evaluation; but hardly any of them have focused on their automatic application. Our proposal is, then, a system to automatically superimpose the mandatory constraints of an architectural style on top of the system-to-be.

This work is structured as follows. After this introduction, section 2 presents a brief introduction to the methodological context in which this proposal has been elaborated, along with a case study developed for validation purposes. Section 3 describes the way in which our proposal introduces architectural styles into the generation of the proto-architecture [1], using model transformation techniques. Finally, section 4 discusses related work and section 5 presents the conclusions.

2 Context of the Work

During the description of the SA, the architect should weigh the impact of its decisions at the architectural level and the relationships they have with other decisions and requirements before realizing them in the system-to-be. The ATRIUM methodology [10] provides support in this context. It has been designed using the *Model-Driven Development* (MDD) approach and encompasses these activities:

- *Modelling Requirements*. This activity allows the analyst to identify and specify the requirements of the system-to-be by defining the *ATRIUM goal model* guiding the architect from *goals* that the systems should achieve, till *requirements* that the system should meet, and *operationalizations* that describe solutions to the established *requirements*.
- *Modelling Scenarios*. This activity focuses on the identification of the set of scenarios that defines the system's behaviour under certain architectural decisions, described in the ATRIUM goal model as *operationalizations*.
- *Synthesize and transform*. This activity has been defined to generate the *proto-architecture* [1] of the specific system. It synthesizes the architectural elements that make up the system, as well as the structure of the future system, from the ATRIUM scenario model.

They must be iterated over in order to define and refine the different models and allow the architect to reason about both the requirements and the architecture. One of the inputs for the *Synthesize and transform* activity is the selected Architectural Style. It must be taken into account that this Architectural Style imposes constraints in terms of the structure and the behaviour of the final description. For this reason, this activity must apply these constraints automatically, conforming to them during the generation of the architecture, hence avoiding a task that could be cumbersome and error-prone for the architect, if it was performed by hand.

ATRIUM has been validated in a case study that is associated with the European project *Environmental Friendly and cost-effective Technology for Coating Removal* (EFTCoR) [7]. The goal of this project is to design a family of robotic systems capable of performing maintenance operations for ship hulls. The Robotic Devices Control Unit (RDCU) integrates all the required functionality to manage the EFTCoR. We have focused our efforts in its SA because of the strict constraints that have to be satisfied in terms of safety, performance, and reliability.

3 Using M2M Transformations for Architectural Styles

As presented in section 2, the architectural elements, their behavior, and the structure of the proto-architecture (the output of the activity *Synthesize and Transform*) are

synthesized from the scenario model considering the constraints imposed by the selected architectural style.

Several languages have been proposed to define M2M transformations. Several surveys, such as that presented by Czarnecki and Helsen [4] identify the features that a M2M transformation language should satisfy in order to fulfill the MDD approach. Considering these, QVT Relations [12] was selected as our M2M transformation language because it provides facilities to manage source and target model; it defines an *incrementality* feature, i.e. it is able to update the generated model according to the changes in the source model; and it offers *directionality* and *tracing*.

The *Rules Organization* feature of QVT has been used to classify the different transformations. Specifically, they have been catalogued as follows:

- *Architectural Generation patterns*. They describe those transformations that are applicable to most of the existing architectural metamodels because they are focused on the generation of components, connectors and systems.
- *Idioms*. They describe low-level transformations that are specific to an architectural metamodel. We have made this distinction in order to facilitate the generation of the proto-architecture according to the architectural metamodel selected by the architect.
- *Architectural Styles*. These transformations are oriented to the application of the constraints imposed by the Architectural Style selected during the activity *Modelling Requirements*.

Therefore, using the same scenario model, different proto-architectures can be generated, depending on the selected architectural metamodel, by applying its specific idioms, and the selected Architectural Style.

Table 1. Head of the transformation *ScenariosToArchmodelPRISMA*, which generates PRISMA proto-architectures from ATRIUM scenario models

```
import archpatt;
import archAcrosetStyle;
transformation ScenariosToArchmodelPRISMA (scenarios: ATRIUMScenarios,
archmodel: Archmodel)
   key archmodel::System {name};
   key archmodel::Component {name};
   key archmodel::Connector {name};
   key archmodel::Port {name, ArchitecturalElement};
   key archmodel::Attachment {name, System};
   key archmodel::Attachment {name, Architecturalmodel};
```

Table 1 shows the head of the transformation in charge of generating proto-architectures from ATRIUM scenario models. It can be seen that the *Architectural Generation patterns* and the A*rchitectural Style patterns* are imported. Table 1 also illustrates the declaration of the transformation, by means of a name (i.e. *ScenarioToArchmodelPRISMA*) and the identification of the candidate models. There are two candidate models: *scenarios,* which represents a candidate model that conforms to the ATRIUM Scenario metamodel; and *archmodel,* which represents a candidate model that conforms to the selected architectural metamodel to be instantiated. Specifically, we have used the *ATRIUM Scenarios* and *PRISMA* [14] metamodels to execute the transformations. In addition, the transformation can define *keys* to uniquely identify the elements and avoid duplicate instances.

In the following, we describe in greater detail the process to perform the automatic superimposition of Architectural Styles, by dealing with the constraints imposed by them in a generative way.

During the *Modelling Requirements* activity, the Architectural Style is selected; and this means that several constraints must be satisfied. The main idea of this proposal is to transform these constraints into QVT generation rules so that the generated proto-architecture is compliant with the selected Architectural Style.

The EFTCoR system uses a Layered Style; specifically, we used ACROSET [13], which is a *Domain Specific Software Architecture* (DSSA) that specializes the Layered Style identifying three kinds of subsystems (SUC, MUC, and RUC). These styles specify some constraints in terms of compositionality, by establishing that one layer only requires the services provided by the lower layer.

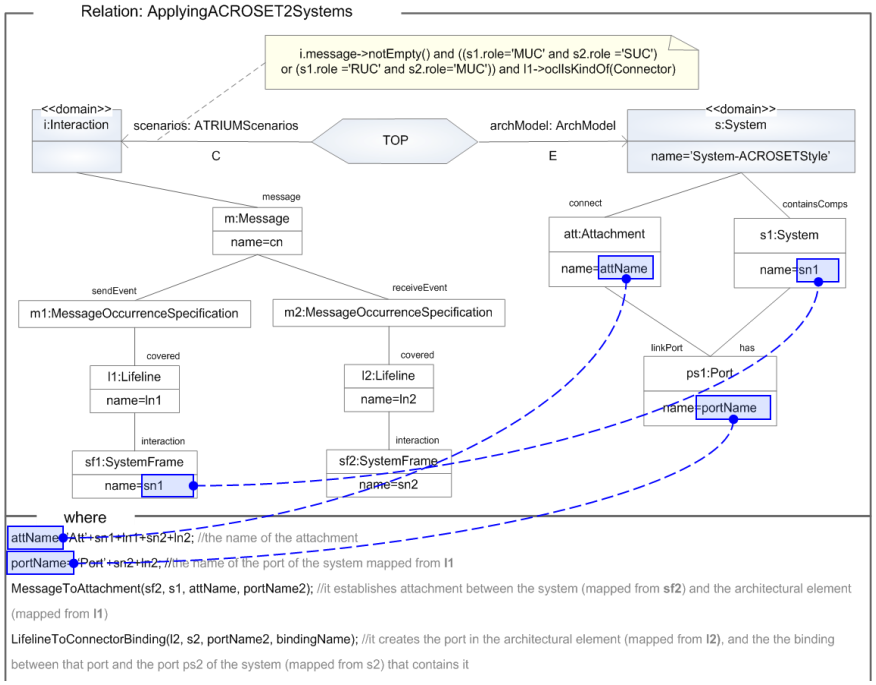

Fig. 1. QVT relation to superimpose the ACROSET Style

In order to generate the proto-architecture of the EFTCoR taking into account the constraints imposed by the ACROSET Style, a transformation was defined (see in Table 1 as *archAcrosetStyle*). A transformation in QVT is defined by means of a set of *relations* that must hold in order to apply successfully the transformation. *ApplyingACROSET2Systems* is one of these relations (see Fig. 1).

The graphical syntax of QVT has been used to enhance the legibility. The hexagon in the middle helps us to represent the transformation by identifying the candidate models (in this case, *scenarios* and *archmodel*) along with their respective metamodels (*ATRIUMScenarios* and *Archmodel*). Each arrow represents a *domain* that is labeled as either *C* or *E* to determine if the transformation is executed in

checkonly mode (it will only be checked if a valid matching exists that satisfy the relation) or in *enforce* mode (provided the matching fails, the target model will be modified to satisfy the relationship), respectively, in that direction. This allows the architect to either generate the proto-architecture or check whether inconsistencies between the proto-architecture and the scenario model arise.

In addition, whenever a relation is defined, clauses *where* and *when* can be defined. The *where* clause (see Fig. 1) specifies a condition that must be held by all the elements involved in the relation so that it can be successfully applied.

It is necessary to establish the roles played by the elements of the scenario model with regard to the Architectural Style. For this reason, the *role* attribute is defined to be used by the OCL expression in the *scenarios* domain (see again Fig. 1) to ensure that the relation is only applied when an interaction happens between elements that belong to the appropriate layers. This means that any interaction between other layers will not cause any sort of generation on the proto-architecture.

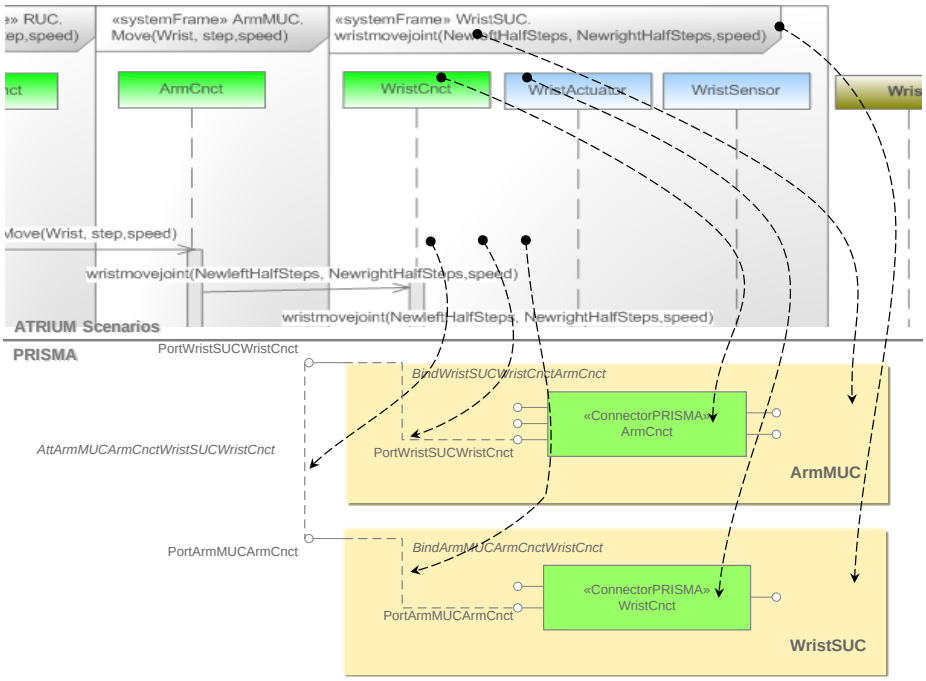

Fig. 2. ATRIUM Scenario (atop) and the generated proto-architecture

Fig. 2 shows an example of the result of this relation when it is applied on an ATRIUM Scenario. At the top of the figure, there is a scenario which establishes how should be the collaboration between the architectural and environmental elements, to meet the requirement "REQ.6" according to the operationalization "OPE.13".

As shown in Fig. 1, a matching is established between the *SystemFrame* and the *System* because both of them bind their attribute "*name*" to the same variable. This means that when the Relation is applied to the Scenario in Fig. 2, a PRISMA *System* named *ArmMUC* is created in the PRISMA model, because the *systemFrame*

ArmMUC contains the *Lifeline ArmCnct*. Both a *Port* and an *Attachment* are resolved in the *where* clause, and also created in the architectural model. These two elements are related, in order to define the connection between the System *s1*, and the System *s2* that will be created using the relation *MessageToAttachment* (see Fig. 3).

The Relation *MessageToAttachment* (Fig. 3) acts in a similar way to *ApplyingACROSET2Systems* (Fig. 1), generating the System *s2* from systemFrame *sf2* by matching their names, and attaching this to the other generated System, *s1*.

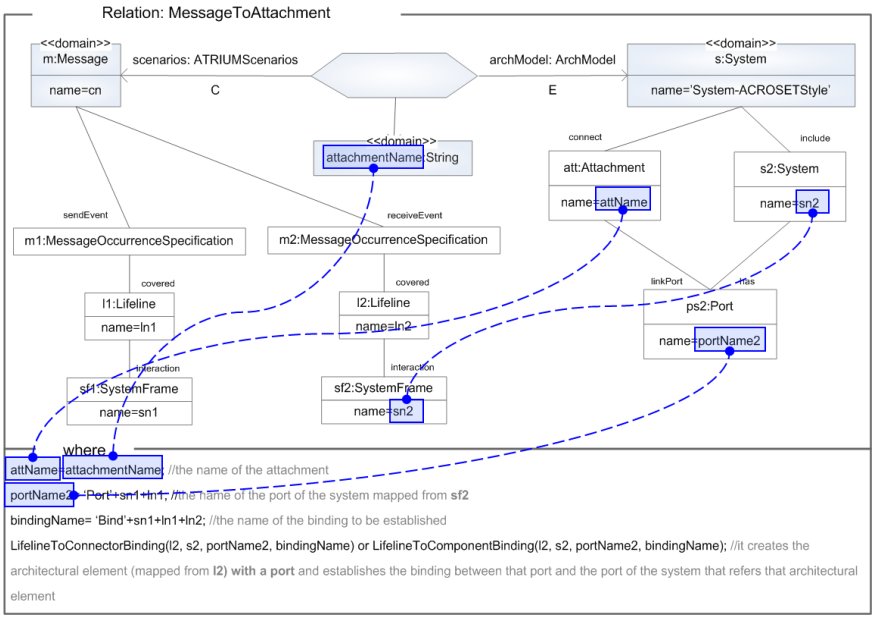

Fig. 3. QVT Relation: Establishing connections between Systems

In the *where* clause of both the relations *ApplyingACROSET2Systems* and *MessageToAttachment*, another relation, *LifelineToConnectorBinding*, is specified. This relation helps to apply the constraints of the ACROSET Style, that is, the compositionality of the *Architectural Elements* belonging to the different layers. Due to space constraints, no more details are provided about this relation.

In summary, we have been able to generate the proto-architecture with only one ATRIUM scenario. The *incrementality* feature of QVT Relations makes possible to automatically update the proto-architecture as new scenarios are defined or exiting ones are modified. Finally, it should be noted that the ATRIUM development process is fully supported by a toolset, called MORPHEUS [11].

4 Related Work

Architectural Styles have received significant attention from both academia and industry. Preliminary work in the field designed software structures specialized for specific domains, such as avionics or missile control and command [5]. This initial work on the definitions of Architectural Styles is one of the cornerstones of SA.

However, most proposals on Architectural Style have focused their attention on recurrent issues. Most of the work has been oriented towards the description of new architectural styles. Some work has focused on the classification of architectural styles, and other proposals have tried to define primitives to describe and/or compose architectural styles [9] [19]. But, as far as we know, no proposals have paid attention to their automatic superimposition. This is the reason why, in the following, we focus our analysis on proposals for *model transformations* in the area of SA.

There are some proposals which actually intend to generate software architectures. The proposal by Bruin and van Vliet [2] describes a process for the generation of SA taking as inputs both a rich Feature-Solution graph and *Use Case Maps* (UCM). This proposal introduces architectural styles as a decision in the solution space, along with decision fragments. However, they do not deal with the automatic superimposition of the architectural style, nor they provide details about how this generation proceeds.

Castro et al. [3] have defined a methodology called TROPOS to guide the process of system specification from early requirements. Requirements are elicited with the *i** framework. The methodology proposes a refinement process from requirements to the SA. However, the (agent-oriented) architectural style is applied by hand.

To the best of our knowledge, the work presented by Sanchez et al. in [17] is the only using a similar generative perspective to ours. They present a process that combines MDD and AOSD to derive Aspect-Oriented Architectures from Aspect-Oriented Requirements models. Once the set of scenarios have been defined, they are transformed into an Aspect-Oriented Architecture by means of a set of transformation rules specified using QVT [12]. However, this proposal pays no attention to the superimposition of architectural styles.

ATRIUM faces many of the issues exhibited by these proposals. Architectural Styles are selected according to the specific needs of the system-to-be, and are automatically applied by means of M2M transformations. Also, it has been designed to make possible the use of that different architectural metamodels.

5 Conclusions and Future Work

ATRIUM aims at generating the proto-architecture of the system-to-be by means of a M2M transformation problem. QVT emerged as the best solution for this purpose, as it satisfies most of the requirements.

By using QVT Relations, a set of transformations has been defined to generate the proto-architecture from ATRIUM scenario models. It provides several advantages. The first is related to its applicability to the whole set of scenarios. QVT supports the definition of keys, which guarantee that the synthesis process prevents the creation of duplicated objects. In addition, it is not necessary to provide the entire set of scenarios to generate the proto-architecture. In fact, the generation can proceed with only one scenario. Thanks to QVT's *incrementality* feature, the generated proto-architecture can be automatically updated as new scenarios are defined to be used.

One of the main concerns in the definition of ATRIUM is traceability. Top-down traceability is provided because the proto-architecture is generated automatically by establishing the appropriate transformations. Bottom-up traceability can be achieved because QVT Relations derives a *Trace Class* from each relation in order to generate traceability mappings. This ability is highly meaningful because a mapping is

established between every element in the proto-architecture and its related element/s in the ATRIUM Scenarios model.

Another challenge to be faced by our proposal is how to establish mechanisms to evaluate the proto-architecture being obtained. We are currently focusing on how to detect faults while the proto-architecture is being generated. Early detection of such faults will make a meaningful improvement in the development in terms of both quality and cost. The definition of an *evaluation model* that describes potential faults at the specification level would be a first step in this direction.

Acknowledgements. This work has been partially supported by grants (PEII09-0054-9581) from the JCCM, and also by grants (TIN2008-06596-C02-01, TIN2009-13838, and CONSOLIDER CSD2007-022) from the Spanish Ministry of Science. Professor Perry is supported in part by NSF CISE Grants IIS-0438967 and CCF-0820251.

References

[1] M. Brandozzi, D. E.Perry, Transforming Goal-Oriented Requirement Specifications into Architecture Prescriptions, Workshop from Soft. Req. to Arch., 2001, pp. 54-61.
[2] H. de Bruin and H. van Vliet, Quality-Driven Software Architecture Composition, Journal of Systems and Software, 66(3), (2003), 269-284.
[3] J. Castro, M. Kolp, J. Mylopoulos, Towards Requirements-Driven Software Development Methodology: The Tropos Project, Information Systems, 27(6), (2002) 365-389.
[4] K. Czarnecki, S. Helsen, Classification of model Transformation Approaches, IBM Systems Journal, 45(3), (2006), 621-645.
[5] N. Delisle, D. Garlan, Formally specifying electronic instruments, 5th Int. Workshop on Software specification and design, ACM New York, USA, 1989, pp.242-248.
[6] D. Garlan, M. Shaw, An introduction to software architecture, Advances in Software Engineering and Knowledge Engineering, 2, (1993), 1-39.
[7] GROWTH G3RD-CT-00794, EFTCOR: Environmental Friendly and cost-effective Technology for Coating Removal. European Project, 5th Framework Program, Spain, 2003.
[8] Medini QVT Relations, 2008. http://www.ikv.de/index.php?option=com_content&task =view&id=75&Itemid=77& lang=en.
[9] N. R. Mehta, N. Medvidovic, Composing architectural styles from architectural primitives, ACM SIGSOFT Software Engineering Note, 28(5), (2003), 347-350.
[10] F. Montero, E. Navarro, ATRIUM: Software Architecture Driven by Requirements, 14th IEEE Int. Conf. on Eng. of Complex Computer Systems (ICECCS'09), Potsdam, Germany, June 2-4, 2009.
[11] E. Navarro, A. Gómez, P. Letelier, I. Ramos, MORPHEUS: a supporting tool for MDD, 18th Int. Conf. on Information Systems Development (ISD2009), Nanchang, China, September 16-19, 2009.
[12] OMG doc. ptc/05-11-01, QVT, MOF Query/Views/Transformations final adopted specification, 2005.
[13] F. J. Ortiz, D. Alonso, B. Álvarez, J. A. Pastor, A Reference Control Architecture for Service Robots Implemented on a Climbing Vehicle, 10th Ada-Europe Int. Conf. on Reliable Software Technologies, York, UK, Springer-Verlag, Berlin, 2005, pp. 13-24.
[14] J. Pérez, N. Ali, J. Á. Carsí, I. Ramos, Designing Software Architectures with an Aspect-Oriented Architecture Description Language, 9th Int. Symp. on Component-Based SE, 2006, pp. 123-138.
[15] D. E. Perry, A. L. Wolf, Foundations for the study of software architecture, ACM SIGSOFT Software Engineering Notes, 17(4), (1992), 40-52.
[16] D.E. Perry, Generic Architecture Descriptions for Product Lines, Development and Evolution of Software Architectures for Product Families, LNCS 1429 (1998), pp. 51-65.
[17] P. Sánchez, A. Moreira, L. Fuentes, J. Araújo, J. Magno: Model-driven development for early aspects. Information & Software Technology, 52(3), (2010), 249-273.
[18] M. Shaw, D. Garlan, Characteristics of higher-level languages for software architecture. Technical Report CMU-CS-94-210, School of Comp. Science, CMU, December, 1994.
[19] U. Zdun, P. Avgeriou, A catalogue of architectural primitives for modeling architectural patterns, Information & Software Technology, 50(9-10), (2008), 1003-1034.