Software Language Engineering of Architectural Viewpoints

Elif Demirli and Bedir Tekinerdogan

Department of Computer Engineering, Bilkent University, Ankara 06800, Turkey
{demirli,bedir}@cs.bilkent.edu.tr

Abstract. A common practice in software architecture design is to apply architectural views to design software architecture for the various stakeholder concerns. Architectural views are usually developed based on architectural viewpoints which define the conventions for constructing, interpreting and analyzing views. So far most architectural viewpoints seem to have been primarily used either to support the communication among stakeholders, or at the best to provide a blueprint for the detailed design. In this paper we provide a software language engineering approach to define viewpoints as domain specific languages. This enhances the formal precision of architectural viewpoints and leads to executable views that can be interpreted and analyzed by tools. We illustrate our approach for defining domain specific languages for the viewpoints of the Views and Beyond approach.

Keywords: Architectural Viewpoints, Software Language Engineering, Domain Specific Modeling, Tool Support.

1 Introduction

An architectural view is a representation of a set of system elements and relations associated with them to support a particular concern. Having multiple views helps to separate the concerns and as such support the modeling, understanding, communication and analysis of the software architecture for different stakeholders. Architectural views conform to viewpoints that represent the conventions for constructing and using a view. An architectural framework organizes and structures the proposed architectural viewpoints. Different architectural frameworks have been proposed in the literature [2]. Organizing the system as a set of viewpoints has also been addressed in enterprise application system using so-called enterprise architecture frameworks [12][13]. The notion of viewpoint now plays an important role in modeling and documenting architectures. So far most architectural viewpoints seem to have been primarily used either to support the communication among stakeholders, or at the best to provide a blueprint for the detailed design. From a historical perspective it can be observed that viewpoints defined later are more precise and consistent than the earlier approaches but a close analysis shows that even existing viewpoints lack some precision. Moreover, since existing frameworks provide mechanisms to add new viewpoints the risk of introducing imprecise viewpoints is high. The development of a proper and effective architecture is highly dependent on the corresponding documentation. An

I. Crnkovic, V. Gruhn, and M. Book (Eds.): ECSA 2011, LNCS 6903, pp. 336–343, 2011.

incomplete or imprecise viewpoint will impede the understanding and application of the viewpoints to derive the corresponding architectural views, and likewise lower the quality of the architectural document.

The key premise in this paper is that a viewpoint can be considered as a domain specific language, and views are models or programs of that language. As such, to enhance the definition of the viewpoints we think that these should be also formally defined as domain specific languages. In this paper we provide a software language engineering approach to define viewpoints as domain specific languages. This will enhance the formal precision of architectural viewpoints and likewise helps to share the additional benefits of domain specific languages, i.e. defining executable views. In the paper, we illustrate our approach using an example viewpoint: decomposition viewpoint of Views and Beyond (V&B) [2] approach.

The remainder of the paper is organized as follows. In section 2 we define the background of architecture framework and software language engineering. In section 3, we show the definition of domain specific language for decomposition viewpoint of the V&B approach. Section 4 presents the related work. Section 5 provides the conclusions.

2 Model-Driven Development

Architecture design is basically about *modeling* the system from different perspectives. Historically, *models* have had a long tradition in software engineering and have been widely used in software projects. The primary reason for modeling is usually defined as a means for communication, analysis or guiding the production process. Models are different in nature and quality. Mellor et al. [9] make a distinction between three kinds of models, depending on their level of precision. A model can be considered as a *Sketch*, as a *Blueprint*, or as an *Executable*. According to [9] an executable model is a model that has everything required to produce the desired functionality of a single domain. Executable models are more precise than sketches or blueprints, and can be interpreted by model compilers.

In model-driven software development the concept of *models* can be considered as executable models as defined by the above characterization of Mellor et al. [9]. This is in contrast to model-based software development in which models are used as blueprints at the most.

The language in which models are expressed is defined by meta-models. As such, a model is said to be an instance of a meta-model, or a model *conforms to* a meta-model. A meta-model itself is a model that conforms to a meta-meta-model, the language for defining meta-models. In model-driven development, models are usually organized in a four-layered architecture. The top (M3) level in this model is the so called meta-metamodel, and defines the basic concepts from which specific meta-models are created at the meta (M2) level. Normal user models are regarded as residing at the M1 level, whereas real world concepts reside at level M0.

2.1 Architectural Description from a Model-Driven Development Perspective

In fact we can state that the current architectural modeling practices can be categorized as *model-based development*, rather than *model-driven development.* In the last two to

three decades architectural modeling and the corresponding notations have evolved from simple sketches to more precise models as defined by architectural view concept. Yet, the view models can usually not be considered as executable models. Moreover, the link between architectural models, and the link from architectural models are merely implicit and not formal.

In architecture modeling literature the notion of meta-model is not explicitly used. The concepts related to architectural description are formalized and standardized in ISO/IEC 42010:2011 [7]. The standard holds that an architecture description consists of a set of *views*, each of which conforms to a *viewpoint*. Here the concept of view appears to be at the same level of to the concept of *model* in the model-driven development approach. The concept of viewpoint, representing the language for expressing views, appears to be on the level of meta-model.

Although the ISO/IEC 42010 standard does not explicitly use the terminology of model-driven development the concepts as described in the standard seem to align with the concepts in the meta-modeling framework. In Fig. 1, we provide a partial view of the standard that has been organized around the meta-modeling framework. An *Architecture Description* is a concrete artifact that documents the *Architecture* of a *System of Interest*. The concepts *System-of-Interest* and *Architecture* reside at layer M0. *System-of-Interest* defines a system for which an *Architecture* is defined. *Architecture* is described using *Architectural Description* that resides at level M1. *Architectural Description* includes one or more *Architectural Views* that represent the system from particular stakeholder concern's perspective. Architectural views are described based on *Architectural Viewpoint*, the language for the corresponding view. *Architectural Viewpoints* are organized in *Architectural Framework.* The latter two reside at level M2. The standard does not provide a concept that we could consider at level M3, and as such we have omitted this in Fig. 1.

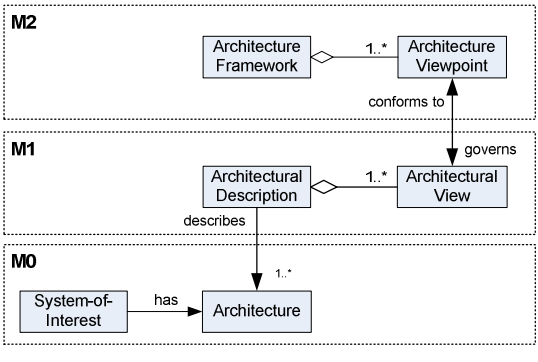

Fig. 1. Architectural Description Concepts from a meta-modeling perspective

2.2 Elements of Domain Specific Languages

Meta-models define the language for the models. The application of a systematic, disciplined, quantifiable approach to the development, use, and maintenance of these languages is usually called *software language engineering* [8]. A proper definition of meta-models is important to enable valid and sound models. In both the software

language engineering [8] and model-driven development domains [9], a meta-model should include the following elements:

- *Abstract Syntax*: the vocabulary of concepts provided by the language and how they may be combined to create models.
- *Concrete Syntax*: the notation that facilitates the presentation and construction of models or programs in the language. It can be visual or textual.
- *Well-formedness rules (Static Semantics)*: definitions of additional constraint rules on abstract syntax that are hard or impossible to express in standard syntactic formalisms of the abstract syntax.
- *Semantics*: the definition of the meaning of the concepts in the abstract syntax.

Given these elements of a language we can also evaluate viewpoints, the languages for defining views. A coarse-grained evaluation would be to check whether these elements are defined for the viewpoints. This does not really provide much information since all the viewpoints seem to somehow describe the above elements albeit in a different degree. To be able to define the degree to which each element is addressed we propose the evaluation framework as defined in Table 1. The table distinguishes among four levels L1 to L4 indicating the quality and completeness of the element. As it can be seen in the table, a lower quality indicates that the corresponding element has not been described (missing, not defined) whereas a higher value indicates that the given element is completely defined and validated.

Table 1. Assessment framework for evaluating Architectural Viewpoints

	L1	L2	L3	L4
Abstract Syntax	Missing or vague	Clear textual description	Meta-model defined. Non-validated models	Validated models
Concrete Syntax	Not defined	Informal	Semi-Formal	Formal
Static Semantics	Not defined	Incomplete constraints in natural language	Complete constraints in natural language	Formal, Executable constraints

3 Defining Viewpoints as Domain Specific Languages

In this section we will illustrate the modeling of viewpoints as domain specific languages to show how existing viewpoints can be even further formally specified to lift these to the level of executable models. We have chosen the decomposition style of the V&B framework [2], as example viewpoint. We will follow the process as defined in the previous section. For the DSL, we first present the abstract syntax that defines the language abstractions and their relationship. The abstract syntax is defined after an analysis of the viewpoint description in the corresponding textbook [2].

Based on these descriptions and the defined meta-model we provide the grammar which defines syntactic rules of the language together with textual concrete syntax.

The grammar is defined using Xtext a language development framework provided as an Eclipse plug-in [4]. The grammar of the language is defined in Xtext's EBNF grammar language and the corresponding generator creates a parser, an AST-meta model as well as a full-featured Eclipse Text Editor from that. The visual concrete syntax is defined using Graphical Modeling Framework (GMF) plug-in of Eclipse [4]. Constraints on viewpoint elements and relations are implemented as static semantics which is implemented writing validation codes in Java. We consider only the elements as defined in Table 1 and do not consider the discussion on semantics. After presenting the language for decomposition viewpoint, a short discussion of the viewpoint specification with respect to our evaluation framework is provided.

3.1 Decomposition Style

Based on these descriptions and the defined meta-model we provide the grammar which defines syntactic rules of the language together with textual concrete syntax. The *Decomposition style* is used to show how system responsibilities are partitioned across modules and how these modules are decomposed into submodules. The decomposition view of the architecture depicts the overall structure of the architecture which is reasonably decomposed into modular implementation units. It is regarded as a fundamental view of the architecture since it serves as an input for other views (e.g. work allocation view) and helps to communicate and learn the structure of the software. We have defined a DSL for decomposition style based on the textual specification given in [2]. The meta-model elements of this style are provided below.

3.1.1 Abstract Syntax

A model of the abstract syntax for the decomposition style is given in the left part of Fig. 2. The root element is DecompositionModel. A valid decomposition model consists of Elements. An element can either be a Module or Subsystem. Module denotes principal unit of implementation. Subsystem differs semantically from the module in the way that it can be developed, executed and deployed independent of other system parts. The decomposition relation between elements is established via the aggregation relation indicating that an element consists of other subelements. Element can have two types of properties: Interface and Simple property. The element's interface is documented with interface property. An element's interface can be declared as a reference to one of its children's interface. Simple property is a generic property which allows specifying new properties in view document.

3.1.2 Grammar and Concrete Syntax

The grammar for decomposition style is given in the right part of Fig. 2. An example decomposition view implemented using our DSL is shown in Fig. 3. The textual concrete syntax is defined for both elements and properties of the elements. The visual concrete syntax is defined only for elements. No explicit relation is modeled in order to express decomposition. Subelements are directly placed into the parent element.

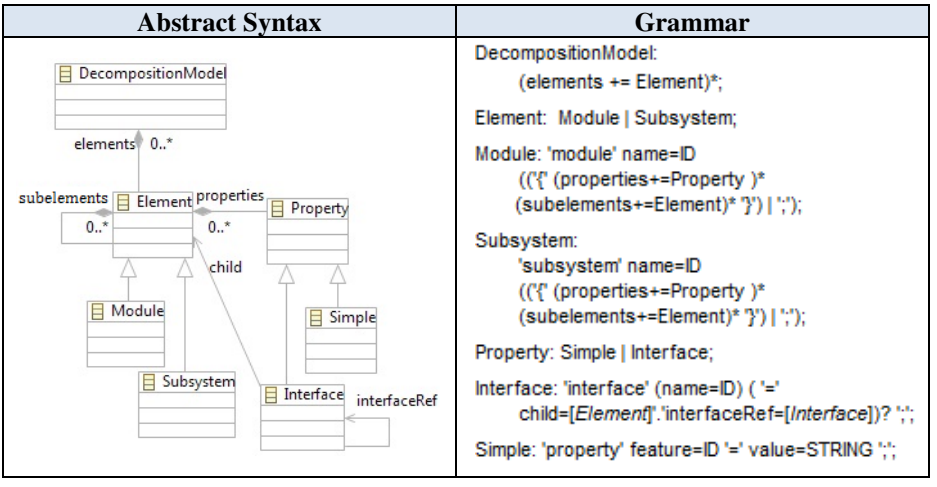

Fig. 2. Abstract Syntax and Grammar for Decomposition Style

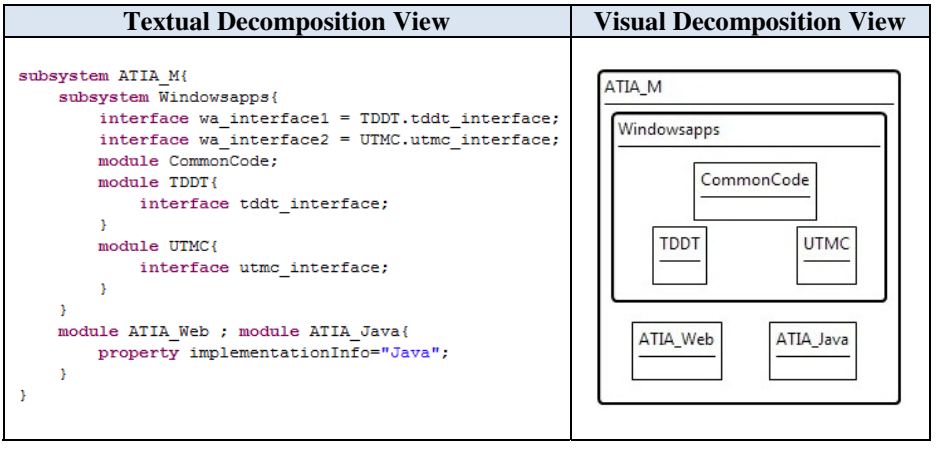

Fig. 3. Example decomposition view with textual and visual concrete syntax

3.1.3 Static Semantics

In addition to extracting the abstract syntax and the grammar we can also derive the well-formedness rules of views, the static semantics, from the viewpoint descriptions. In the decomposition style, two constraints have been defined: no loops are allowed in decomposition graph and a module can have only one parent. From the language perspective, those constraints are too high level to implement. We merged these constraints and shortly defined that no element can have the same name. Doing so we prevented both <A contains B, B contains A> case and <A contains B, C contains B> case. We have implemented this constraint in Java as a validation rule that applies on the language model.

3.1.4 Evaluation

The above results show that we could map a viewpoint to a domain specific language that can be used to define executable models or views. However, the overall effort also provides us insight in the degree of formal precision of the current viewpoint description. When we apply our evaluation framework on decomposition style specification of V&B framework, we get the following results. The abstract syntax definition falls into L2 of our evaluation framework. The concepts to be used in the language are defined textually. The textual description is clear; it can be easily translated to a formal model. However, no meta-model or grammar is provided to describe the concepts. Since both informal and semiformal notations are provided the concrete syntax definition can be considered at level L3. Finally, the well-formedness rules on the concepts of the language are properly specified in natural language. However, they are too high level to directly implement as executable well-formedness rules. Therefore, we consider these at level L3. It should be noted that with the domain specific language engineering approach we have lifted the precision degree to level L4.

4 Related Work

In the enterprise architecture (EA) design community several authors have focused on the formalization of architectural viewpoints. Different attempts have been made before to model viewpoints as domain specific languages. ArchiMate [1] is an EA modeling language that is specified by concepts that focus on business, applications and technology domains. Those concepts form the base metamodel of ArchiMate language. A set of viewpoint languages are defined by composing the concepts available in the metamodel. Contrary to their approach, our viewpoint languages do not depend on a predefined set of concepts. Each viewpoint has an independent language that defines its own concepts. This design choice makes it easy to introduce new viewpoints to the framework. However, it is difficult to define new viewpoints in ArchiMate if the required concepts are not available at the base metamodel. An additional extension mechanism is needed for this purpose [10].

Another example to attempts on formalizing EA viewpoints is about RM-ODP viewpoints. Vallecillo et al. initially focused on formally specifying the abstract languages provided by viewpoint specifications using a rewriting logic based framework Maude [3]. Later on, they also tackle the viewpoint formalization problem from model-driven development perspective and defined UML profile for viewpoints of RM-ODP [11]. Lastly, they define textual notation for ODP specifications together with tool support [5]. The main difference of their approach and our study is the level of formality of the targeted viewpoint specifications. RM-ODP is specified by a standard [6] that precisely defines the syntax and semantics of the language. So, the task of formalizing RM-ODP viewpoint specifications is transforming the present languages to executable languages and defining notations for using the language. However, in our work, we also address viewpoint specifications those are not specified precisely as languages. We offer software language engineering as a method for lifting existing viewpoint specifications to formal language level and provide a complete description of the method.

5 Conclusions

In this paper, we have illustrated the adoption of software language engineering approach for modeling architectural viewpoints. The key premise behind this assumption is that viewpoints are in fact domain specific languages, and as such should be considered and developed like that. To validate our statement we have analyzed the viewpoints in the Views and Beyond approach [2], and defined all these viewpoints as domain specific languages. In the paper, as an example, we have presented the definition of decomposition viewpoint DSL.

We believe that by adopting a software language engineering approach for architectural viewpoints we have also shown the connection with software architecture design modeling and the fields of software language engineering and model-driven software development in general. We hope that this work has paved the way for further research in this direction.

In our future work we will apply the same approach to other architecture viewpoint frameworks. The V&B approach was a case study for us but we do not foresee serious obstacles in applying the same approach for other software architecture viewpoints and enterprise architecture viewpoints. We will elaborate on the tool and consider the integration of viewpoints for nonfunctional concerns. Further, we plan to enhance the tool for supporting architectural analysis.

References

[1] Archimate 1.0 Specification, The Open Group, Tech. Rep. C091 (February 2009)
[2] Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Merson, P., Nord, R., Stafford, J.: Documenting Software Architectures: Views and Beyond, 2nd edn. Addison-Wesley, Reading (2010)
[3] Durán, F., Vallecillo, A.: Formalizing ODP Enterprise specifications in Maude. Computer Standards & Interfaces 25(2), 83–102 (2003)
[4] Eclipse Modeling Framework Web Site, `http://www.eclipse.org/emf/` (accessed on June 2011)
[5] González, D.R., Vallecillo, A., Romero, J.R.: On the Synchronization of ODP Textual and Graphical Specifications. In: Proc. of WODPEC 2010, Vitoria, Brazil, October 25, pp. 376–381 (2010)
[6] [ISO/IEC 10746-2:1996] International Organization for Standardization & International Electrotechnical Commission. Information Technology - Open Distributed Processing - Reference Model: Foundations (ISO/IEC 10746-2) (1996)
[7] [ISO/IEC 42010:2011] Systems and Software Engineering – Architecture Description (ISO/IEC 42010) (2011)
[8] Kleppe, A.: Software Language Engineering: Creating Domain-Specific Languages Using Metamodels. Addison-Wesley Longman Publishing Co., Inc., Boston (2009)
[9] Mellor, S.J., Scott, K., Uhl, A., Weise, D.: MDA Distilled: Principle of Model Driven Architecture. Addison Wesley, Reading (2004)
[10] Peña, C., Villalobos, J.: An MDE Approach to Design Enterprise Architecture Viewpoints. In: IEEE 12th Conference on Commerce and Enterprise Computing (CEC), November 10-12, pp. 80–87 (2010)
[11] Romero, J.R., Troya, J.M., Vallecillo, A.: Modeling ODP Computational Specifications Using UML. The Computer Journal 51, 435–450 (2008)
[12] TOGAF 1995 -The Open Group Architecture Framework, Version 8.1.1 (1995), `http://www.opengroup.org/architecture/togaf8-doc/arch/`
[13] Zachman, J.A.: A Framework for Information Systems Architecture. IBM Systems Journal 26(3), 276–292 (1987)