Evaluation Framework for Software Architecture Viewpoint Languages

Bedir Tekinerdogan
Bilkent University
Department of Computer Engineering
06800 Bilkent Ankara, Turkey

bedir@cs.bilkent.edu.tr

Elif Demirli
Bilkent University
Department of Computer Engineering
06800 Bilkent Ankara, Turkey

demirli@cs.bilkent.edu.tr

ABSTRACT

In general, software architecture is documented using software architecture views to address the different stakeholder concerns. The current trend recognizes that the set of viewpoints should not be fixed but multiple viewpoints might be introduced instead to design and document the software architecture. To ensure the quality of the software architecture various software architecture evaluation approaches have been introduced. In addition several documentation guidelines have been provided to ensure the quality of the software architecture document. Unfortunately, the evaluation of the adopted viewpoints that are used to design and document the software architecture has not been considered explicitly. If the architectural viewpoints are not well-defined then implicitly this will have an impact on the quality of the design and the documentation of the software architecture. We present an evaluation framework for assessing existing or newly defined software architecture viewpoint languages. The approach is based on software language engineering techniques, and considers each viewpoint as a metamodel. The approach does not assume a particular architecture framework and can be applied to existing or newly defined viewpoint languages. We illustrate our approach for modeling and reviewing the first and second editions of the viewpoint languages of the Views and Beyond approach.

Categories and Subject Descriptors

D.2.11 [**Software Architectures**]: Domain Specific Architectures, Languages.

General Terms

Documentation, Design.

Keywords

Software Architecture Evaluation, Architectural Viewpoints, Software Language Engineering, Metamodeling, Tool Support

1. INTRODUCTION

Architectural drivers define the concerns of the stakeholders which shape the architecture [3]. A stakeholder is defined as an individual, team, or organization with interests in, or concerns relative to, a system [15][3][23]. Each of the stakeholders' concerns impacts the early design decisions that the architect makes. A common practice is to model different architectural views for describing the architecture according to the stakeholders' concerns. An architectural view is a representation of a set of system elements and relations associated with them to support a particular concern [5][19]. 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 [18].

In the literature, initially a fixed set of viewpoints have been proposed to document the architecture. For example, the Rational's Unified Process [22] which is based on Kruchten's 4+1 view approach [21] utilizes the logical view, development view, process view and physical view. Another example is the Siemens Four Views model [16] that uses conceptual view, module view, execution view and code view to document the architecture. Because of the different concerns that need to be addressed for different systems, the current trend recognizes that the set of views should not be fixed but multiple viewpoints might be introduced instead.

To ensure the quality of the software architecture various software architecture evaluation approaches have been introduced [2][9][13][15][6][33][34]. In addition, several documentation guidelines have been provided to ensure the quality of the software architecture document. Unfortunately, the evaluation of the adopted viewpoint languages that are used to design and document the software architecture has not been considered explicitly. If the architectural viewpoint languages are not well-defined then implicitly this will have an impact on the quality of the design and the documentation of the software architecture.

We provide an evaluation framework for evaluating existing or newly defined architectural viewpoints. Our basic premise is that viewpoints can be considered as domain specific languages [7] and likewise the evaluation of the viewpoint also considers the language aspects of the viewpoint. The approach does not assume a particular architecture framework and can be applied to existing viewpoints or newly defined viewpoints. We illustrate our approach for reviewing the first and second edition of the viewpoints (i.e. styles), of the Views and Beyond (V&B) approach [4][5].

The remainder of the paper is organized as follows. In Section 2 we provide the background for architectural evaluation and define the context of this paper. In Section 3 we discuss software

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
QoSA'13, June 17–21, 2013, Vancouver, BC, Canada.
Copyright © ACM 978-1-4503-2126-6/13/06...$15.00.

language engineering and the application of this perspective on architectural viewpoints. In Section 4, we present the approach for evaluating the architectural viewpoints. In Section 5 we discuss the modeling of viewpoints as DSLs, for the V&B approach. Section 6 provides the overall summary of the evaluation of the two editions of the V&B approach. Section 7 describes the tool support that is used for the modeling and analysis of the viewpoints. Section 8 provides the related work and finally section 9 concludes the paper.

2. BACKGROUND AND CONTEXT

Since software architecture is critical for the success of a project, different architectural evaluation approaches have been introduced to evaluate the stakeholders' concerns. From a cost perspective architectural evaluation is also a strategic decision because the earlier the problems in a software project are detected, the better. Problems that are detected later on in the software life cycle will be more difficult to fix and as such require higher costs. Evaluating or reviewing the software architecture can have different meanings in different contexts. We distinguish the following three evaluation processes:

2.1 Architecture Evaluation

Architecture Evaluation process aims to analyze the software architecture design with respect to the stakeholder concerns. This is typically carried out by software architects and the corresponding stakeholders. A comprehensive overview of these architecture analysis methods is given in [9]. To compare the architectural evaluation approaches a number of frameworks have been proposed. The Software Architecture Review and Assessment (SARA) report, for example, provides a conceptual framework for conducting architectural reviews [13]. The evaluation frameworks usually compare the methods based on the criteria of context and goals of the method, required content for applying the method, the process adopted in the method, and the validation of the method. Based on the results of the frameworks we can state that the architecture evaluation approaches are useful in making design decisions explicit and supporting the refactoring of the architecture to enhance its quality.

2.2 Architecture Documentation Evaluation

In addition to evaluating the architecture, recently also approaches have been defined for evaluating the *documentation* of an architecture [4][15][28]. This is because the architectural documentation provides the tangible means for communication about the architecture. A poorly documented architecture will impede the communication and analysis of the architecture and likewise the architecture will fail to meet its goals. The documentation for an architecture consists primarily of the documentation of the different architectural views and documentation that describes the relation among the views. Likewise evaluation of the architecture document implies the review of the different architectural views and their fitness to the purposes of the stakeholder concerns. For example, in their book on the Views and Beyond approach [4][5] Clements et al. define seven rules for sound documentation including (1) Write documentation from the Reader's point of view (2) Avoid unnecessary repetition (3) Avoid ambiguity (4) Use a standard organization (5) Record rationale (6) Keep documentation current but not too current, and (7) Review documentation for fitness of purpose. A more detailed and structured evaluation approach is given by Nord. et al. [27] who provide a framework to build a set of review questions to analyze the document. Hämäläinen and Markkula [15] have proposed a question framework for assessing the quality of architectural descriptions. The framework was developed together with the industry and validated by the industry.

2.3 Architecture Viewpoint Evaluation

Both evaluation processes, i.e. evaluation of the architecture and the evaluation of the documentation, are important to ensure the effectiveness of the architecture. Ensuring that the architecture is properly designed is important to meet the quality concerns. Ensuring that the architectural documentation indeed describes the architecture as it should be described is important to support the communication among the stakeholders. Yet, the architectural views are defined based on existing viewpoints for a given architectural viewpoint framework. If the selected set of viewpoints is not properly designed, then this will have both an impact on the design and impede the key motivations for architecture description, that is, communication, guidance and analysis. Hence, complementary to the existing architecture design analysis and architecture documentation analysis approaches we believe that it is very important to analyze the quality of viewpoints. In this paper we focus on the language aspects of the viewpoints. Likewise in this paper our key concern is the evaluation of the architectural viewpoint languages.

3. SOFTWARE LANGUAGE ENGINEERING

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 and different classifications of models have been provided in the literature. Mellor et al. [26] 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 [26] 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. A similar classification of models is defined by Fowler [14] who suggests a distinction based on three levels of models, namely *Conceptual Models*, *Specification Models* and *Implementation Models*.

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. [26]. In model-*driven* software development models are not mere documentation but become "code" that are executable and that can be used to generate even more refined models or code. This is in contrast to model-*based* software development in which models are used as blueprints at the most [31].

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. Given the different levels in which the models reside 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.

3.1 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. However, the view models can usually not be considered as executable models yet. Moreover, the link between architectural models, and the link from architectural models are merely implicit and not formal.

The concepts related to architectural description are formalized and standardized in ISO/IEC 42010:2007, a fast-track adoption by ISO of IEEE-Std 1471-2000, *Recommended Practice for Architecture Description of Software-Intensive Systems* [19][24]. On one hand, it appears that in the architecture modeling literature, the notion of meta-model is not explicitly used. Yet, a closer look at the standard shows that we can identify the concepts related to the notions of metamodel and model. The standard holds that an architecture description consists of a set of *views*, each of which conforms to a *viewpoint*, but it has deliberately chosen not to define a particular viewpoint. Here the concept of view appears to be at the same level of 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.

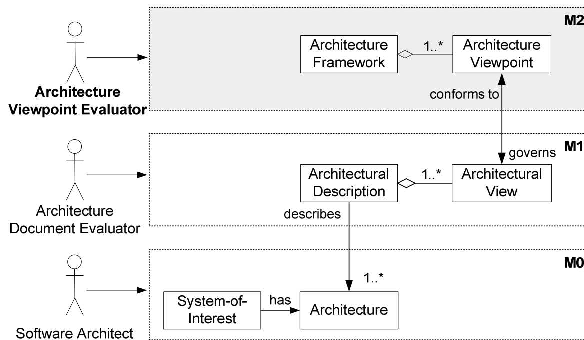

Figure 1. Architectural Description Concepts from a meta-modeling perspective

As such, although the ISO/IEC 42010 standard does not really 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 Figure 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 Figure 1. The left part of Figure 1 shows the corresponding stakeholders that focus on reviewing the architecture. Based on the discussion in the previous section we can identify three types of evaluators, that is, *Architecture Viewpoint Evaluator, Architecture Document Evaluator,* and *Software Architect.* The *Architecture Viewpoint Evaluator* is responsible for evaluating the viewpoints of selected the architecture framework, or the viewpoints that have been newly added. The *Architecture Document Evaluator* evaluates whether the architecture documentation fits the proper documentation standards. Finally, the *Software Architect* is the actor who designs the architecture by using the selected viewpoints. As stated before, in this paper, we focus on the architecture viewpoint evaluation process.

3.2 Elements of Domain Specific Languages

In the previous sub-section we have made the link between viewpoints and meta-models. Likewise, for understanding how to evaluate viewpoints we have to know how meta-models are evaluated in practice. In fact, 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* [20]. A proper definition of meta-models is important to enable valid and sound models. As described in both the software language engineering [20] and model-driven development domains [31] a meta-model should include the following elements:

- *Abstract Syntax*: describes the vocabulary of concepts provided by the language and how they may be combined to create models. It consists of a definition of the concepts and the relationships that exist between concepts.
- *Concrete Syntax*: defines the syntax, the notation that facilitates the presentation and construction of models or programs in the language. Typically two basic types of concrete syntax are used by languages: textual syntax and visual syntax. A textual syntax enables models to be described in a structured textual form. A visual syntax enables a model to be described in a diagrammatical form.
- *Well-formedness rules (Static Semantics)*: provides 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 description of the meaning of the concepts and relation in the abstract syntax. Semantics can be defined in natural language or using other more formal specification languages.

4. APPROACH FOR EVALUATING VIEWPOINTS

Evaluating architecture viewpoints can be carried out from various perspectives including the appropriateness for stakeholders, the consistency among viewpoints, and the fitness of the language. Likewise, the overall process for evaluating an architectural framework consisting of different viewpoints is shown in Figure 2. The activity *Select Viewpoint* selects a viewpoint that is provided either by a given architecture framework, or that has been newly introduced by viewpoint designers. After selecting the viewpoint it is evaluated with respect its language precision. Here, a coarse-grained evaluation would be to check whether the language elements of abstract syntax, static semantics and concrete semantics, 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, and as such the architectural viewpoint evaluation would not be of less practical value. To be able to refine the degree to which each element is addressed we propose to model each viewpoint explicitly as a domain specific language (DSL). After selecting an architectural viewpoint, the viewpoint is modeled and in parallel the evaluation of the corresponding viewpoint takes place.

After the evaluation of the viewpoint with respect to the language formalism perspective, the viewpoint is assessed for fitness with the stakeholder concerns. This activity is carried out in close interaction with the stakeholder. The feedback of the stakeholder is taken into account to enhance the viewpoint accordingly.

The subsequent step is the evaluation of the consistency between the viewpoints. This implies the coverage of the viewpoints for the stakeholders as well as the mapping of the elements between the viewpoints. After all the viewpoints have been modeled and evaluated, the overall evaluation for the architectural framework is provided. Based on the overall evaluation of the viewpoint(s) it is decided on what actions to take.

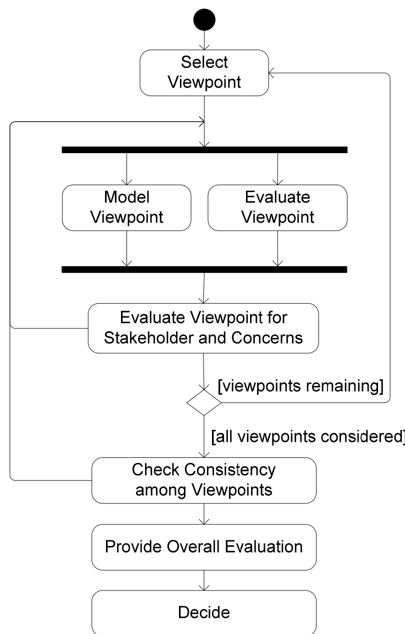

Figure 2. Overall Process for Evaluation of Architectural Framework

The activity *Model Viewpoints* defines the DSL for the selected viewpoint and the detailed steps for this are shown in Figure 3. For modeling the viewpoint, the description of the viewpoint in the literature (e.g. textbook) is analyzed. The first step in the activity *Model Viewpoints* is the identification and definition of the architectural element and relation types. This is necessary to define the abstract syntax of the viewpoint. As stated before, the abstract syntax defines both the concepts (architectural element and relation types) of the language and the relations among these concepts. To represent the abstract syntax either a model-based approach or a grammar-based approach is adopted [20][31]. In the model-based approach, typically a UML model is provided defining the language concepts and their relations. In the grammar-based approach a grammar (e.g. EBNF grammar) is defined. In our approach we provide both a UML model and an EBNF-based grammar of the viewpoint. The composition rules are identified in the activity *Identify and Model Composition Rules*. After the abstract syntax and the corresponding grammar/model have been defined the topology constraints (i.e. static semantics) are identified and modeled. The next activity is to *Identify and Define the Notation (Concrete Syntax)*. Finally, the activity *Validate using Example* aims to define example models using the modeled viewpoint. The outcome of this activity might require iterating to the previous activities.

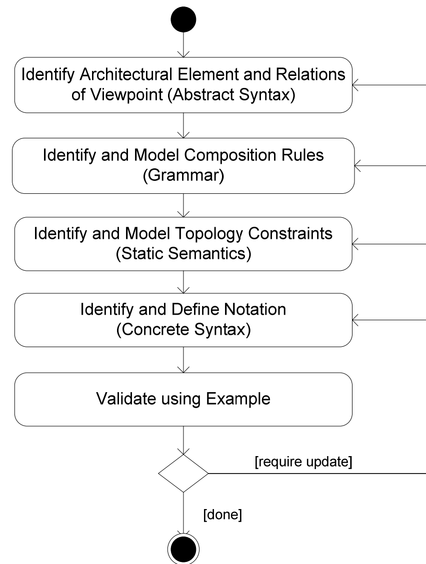

Figure 3. Activity Diagram for Activity *Model Viewpoint*

In parallel with the execution of the activity *Model Viewpoints*, also an evaluation of the viewpoint is carried out (activity *Assess Viewpoint* as shown in Figure 2). For evaluating the viewpoint we focus in particular on the elements of abstract syntax, concrete syntax and static semantics. We adopt the evaluation framework as defined in Table 1.

Table 1. Evaluation framework for evaluating Architectural Viewpoints

Evaluation Level	Description
L0	Not defined
L1	Incomplete, Informally defined
L2	Complete, Informally defined
L3	Incomplete, Formally defined
L4	Complete, Formally defined

The table distinguishes among four levels L0 to L4 indicating the quality and completeness of the corresponding element. As it can be seen in the table, a lower quality indicates that the corresponding element is incomplete or informally defined; whereas a higher value indicates that the given element is more complete and formally defined. The activity *Provide Overall Evaluation* in Figure 2 defines the summary of the overall evaluations of the viewpoints for the given architecture framework or set of viewpoints. The final activity *Decide* in Figure 2 describes the recommendations and decisions on the usage of the selected viewpoints. In case the selected viewpoint is well-defined typically no action will be undertaken and the viewpoint can be used as is. If the viewpoint is not well-defined one may decide to enhance the viewpoint of the original

viewpoint description after the evaluation process. In that case, the evaluation level (L0 to L4) will increase as well.

5. EVALUATING V&B APPROACH

In this section we provide, as an example, the evaluation of the V&B approach using the viewpoint evaluation framework as defined in the previous sections. The V&B approach consists of many predefined viewpoint descriptions, which we have all evaluated with our approach. In the V&B approach rather than viewpoints, the notion of style is adopted. The V&B approach distinguishes among three different categories of styles, module styles, component & connector styles, and allocation styles [5]. For each category of styles, several styles have been predefined. In the following we will illustrate the evaluation for two different styles in the V&B approach.

5.1 Decomposition Style

Based on the 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* [6] is used to show how system responsibilities are partitioned across modules and how these modules are decomposed into sub-modules. 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 [6]. The meta-model elements of this style are provided below.

5.1.1 Abstract Syntax

A model of the abstract syntax for the decomposition style is given in the left part of Figure 4. 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 sub-elements. *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.

5.1.2 Grammar and Concrete Syntax

The grammar for decomposition style is given in the right part of Figure 4. An example decomposition view implemented using our DSL is shown in Figure 5. 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. Sub-elements are directly placed into the parent element.

5.1.3 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 original decomposition style description, 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 have merged these constraints and shortly defined that no element can have the same name. Doing so we prevented both the constraints for avoiding loops (<A contains B, B contains A> case) and ensuring that a module can have only one parent (<A contains B, C contains B> case). We have implemented this constraint in Java as a validation rule that applies on the language model.

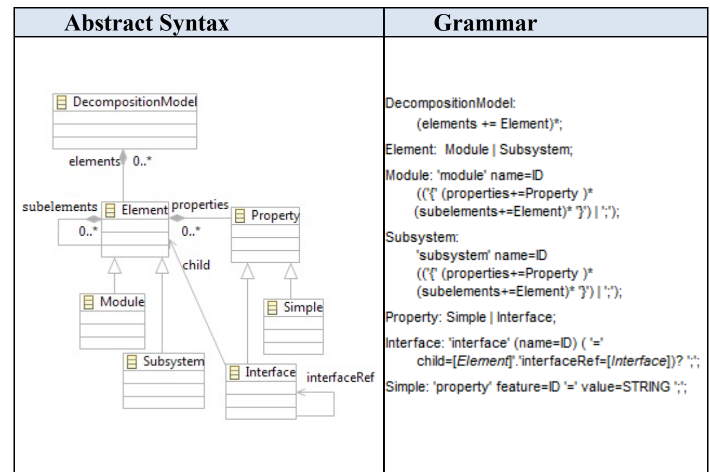

Figure 4. Abstract Syntax and Grammar for Decomposition Style

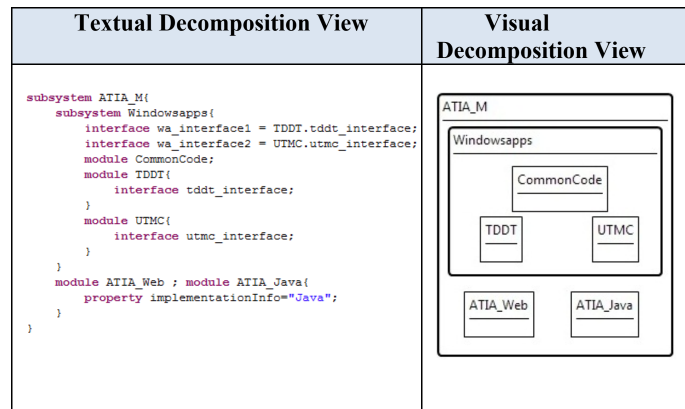

Figure 5. Example decomposition view with textual and visual concrete syntax

5.1.4 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 the 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. The well-formedness rules on the concepts of the language are properly specified in natural language. However, they are too informal and cannot be directly implemented as executable well-formedness rules. Therefore, we consider these at level L3. Finally, regarding

the semantics of the language elements we consider the viewpoint description at level L2. The concepts for module is sufficiently explained in natural language but not formally defined. It should be noted that with the domain specific language engineering approach we have lifted the precision degree to level L4 for the elements of abstract syntax, concrete syntax and static semantics.

5.2 Deployment Style

The deployment style is a style that is used to show how the software elements are allocated to hardware of a computing platform. This style is useful for analyzing and tuning certain quality attributes of the system such as performance, reliability and security.

5.2.1 Abstract Syntax

The abstract syntax defined for the deployment style is shown in Figure 6. The abstract syntax describes the elements of the language, which are software elements and hardware elements. The software elements are statically allocated to hardware elements by allocated to relation. In abstract syntax definition, we do not explicitly show this relation. It is implicit in the aggregation relation between hardware element and software element. The allocation of software to hardware does not have to be static. Migration relations are defined to support dynamic allocation schemes. There are three types of migration relations: migrates to, copy migrates to, execution migrates to. In addition to these style specific elements and relations, in order to reflect the topology of the platform connection links between hardware elements are required.

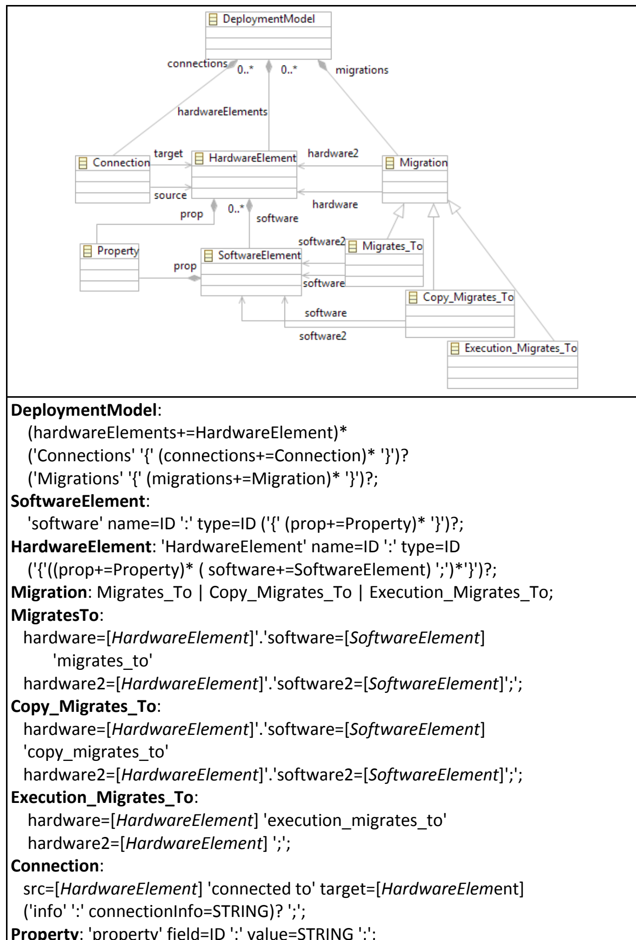

Figure 6. Abstract syntax and grammar for deployment style

5.2.2 Grammar and Concrete Syntax

The grammar for the deployment style follows the abstract syntax but due to space limitations we could not include it in this paper. An example deployment view specified using both textual and visual concrete syntax is provided in Figure 7. The visual concrete syntax defined for deployment view models software and hardware elements as elements, migrations and connections as relations. The properties of software and hardware elements are also modeled in visual concrete syntax.

5.2.3 Static Semantics

We have identified four well-formedness rules for deployment style and implemented these as validation code. These rules are: (1) Every hardware element must be connected to at least one other hardware element. (2) An element cannot connect to itself (3) All types of migration relations have to be between two distinct hardware elements. (4) The source and target software element names referenced in migrates to and copy migrates to relations must be the same (i.e. the same software migrates from one hardware element to another).

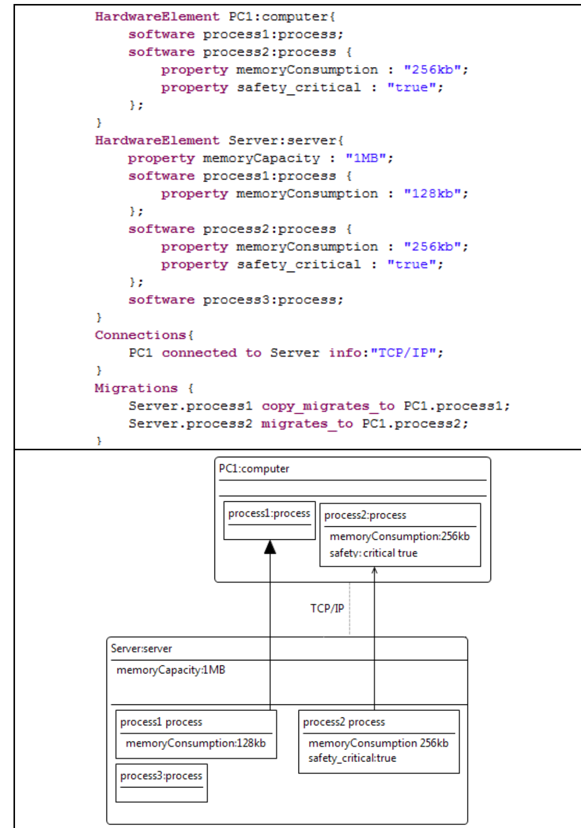

Figure 7. Example deployment view with textual concrete syntax

5.2.4 Evaluation

Although this style is a bit more complex than the previous two styles it was possible to define a DSL. 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 models are provided. Informal and semiformal notations are provided. AADL and SysML are mentioned as

formal notations. However, no guidelines for mapping deployment style constructs to those languages' constructs are specified. No example is provided. The concrete syntax definition is in L3.

Regarding the static semantics, in the constraints section of deployment style, it is stated that allocated topology is unrestricted. No further constraints are specified. However, some well-formedness rule definitions are still required. Those constraints are not explicitly described in the V&B deployment style definition. Probably this is omitted since the rules are obvious (for human architect) and there is no need to define them explicitly. However, when we look from the meta-modeling perspective in which models need to be processed by tools, we have to specify the rules explicitly and more precisely. Based on this observation we can state that the static semantics definition of deployment style is in L1.

6. OVERALL SUMMARY FOR V&B APPROACH

Throughout section 5, we have provided an evaluation for two different styles of the Views and Beyond approach. In this section, we present an overall summary of our experience in mapping V&B architectural styles to domain specific languages. For this we will use again our meta-model evaluation framework as we have defined in section 5. We have applied the framework on each style defined by V&B. In fact, we have implemented all architectural styles of V&B framework as domain specific languages and we can state that the mapping of each viewpoint and its discussion is interesting by itself. Unfortunately, we cannot present all of these due to space limitations. The adopted approach was similar as defined in the previous section. We have applied our approach to the first [4] and second version [5] of the Views and Beyond approach. We will discuss the language elements including abstract syntax, concrete syntax, static semantics and semantics separately.

6.1 Evaluation of Abstract Syntax

Figure 8 shows a dot chart that compares the precision of the abstract syntax of viewpoints in both editions of the V&B approach. With respect to the abstract syntax we can conclude that there is not much deviation between two editions of the book. Aspects, Data Model and SOA style values are under L1 for the first edition of the book, because those styles are later introduced in the second edition. The same situation also applies to the communicating processes style for the second edition of the book, since it is excluded in the second edition. For most of the remaining styles, abstract syntax definition levels overlap for both editions of the book. For generalization and publish-subscribe styles a more clear textual description is provided in the second edition.

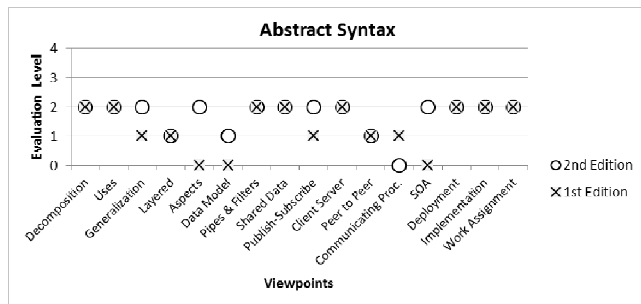

Figure 8. Abstract syntax definition levels for V&B (both editions of the book)

6.2 Evaluation of Concrete Syntax

When we consider the concrete syntax definitions the deviation between two editions of the book is higher. For the module styles (i.e. the first 6 styles of the chart in Figure 9), the concrete syntax definitions are mostly in level L3, indicating that there is semi-formal concrete syntax definition for those styles in both editions of the book. Mostly, UML is recommended as modeling notation explicitly showing how to use UML while realizing views for module styles. For component-and-connector styles (i.e. from 7th style to 13th style), the second edition of the book is still at L3. However, in the first edition of the book most of the C&C styles are in L2-informal concrete syntax level. In the first edition, UML is mentioned roughly for the overall C&C styles, however, it is not depicted how to use them for the specific styles. In the second book, UML discussion for C&C styles is again done for all styles together, however, this time the discussion is detailed enough to specify how to use UML notations required for each style. For none of the styles of the two editions, L4-formal concrete syntax level is reached. Although some formal modeling techniques such as ADLs are mentioned, it is not described how to use those ADLs for modeling with specific styles.

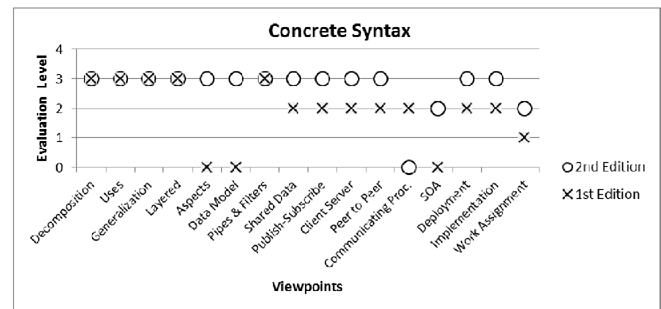

Figure 9. Concrete syntax definition levels for V&B (both editions of the book)

6.3 Evaluation of Static Semantics

The static semantics definition for no style exceeds level 3-complete constraints in natural language. The constraints are always defined in natural language. There is some refinement of the constraint definitions in the second edition compared to those described in the first edition. In the first edition, 11 styles are in L1 and L2 meaning that no constraints are specified or they are incomplete. In the second edition, four of those moves to L3 (uses, generalization, pipes&filters and publish-subscribe) meaning that they are still in natural language form however the constraints on language constructs are completely specified.

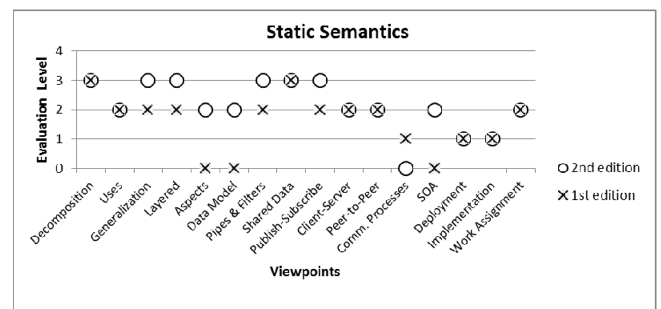

Figure 10. Static semantics definition levels for V&B (both editions of the book)

6.4 Evaluation of Semantics

The semantics of the styles in both editions of the V&B approach does not exceed level L2. None of the styles are formally defined. Some styles provide sufficient explanation in natural language and likewise can be considered at level L2, however many styles are also incomplete regarding the explanation of the component and connector types.

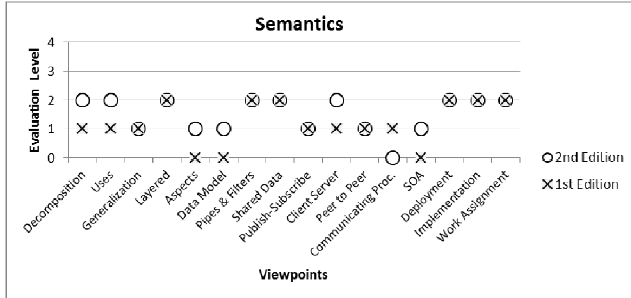

Figure 11. Semantics definition levels for V&B (both editions of the book)

6.5 Overall Evaluation

We can conclude from this analysis that abstract syntax definition for V&B styles are mostly in L2 and that these can be easily mapped to validated models as we do while defining DSLs. The concrete syntax definitions are mostly in L3. Informal and semi-formal notations are introduced and their usage is properly explained. However, no formal notations are provided. The constraints on style elements and relations are always provided in natural language form. Regarding the semantics of the viewpoints we have seen that none of the styles are above level L2. This is because semantics of the styles is provided through natural language and no formal specifications have been provided. By defining DSLs for V&B approach, we have made the style definitions in L4 for each category: abstract syntax, concrete syntax and static semantics. The semantics of each style could be formally enhanced by adopting a common formal model, based on which the elements of the styles can be explained. It should be noted that the evaluation framework is general and can be applied to other architecture frameworks, than the V&B approach. In addition the evaluation framework can also be applied to evaluate newly defined viewpoints.

7. TOOL SUPPORT

In this section we discuss the tool SAVE-BENCH [8] that we have developed in the Eclipse environment to model architecture viewpoints as DSLs. As stated before, the evaluation of the viewpoints takes place during the effort for modeling the viewpoints as DSLs. Various tools such as Xtext [36], GMF [11], EuGENia [12] and EMFatic [10] are used in the language definition process. Firstly, the viewpoint definer creates the grammar definition of the viewpoint using the Xtext editor and following the rules of Xtext's EBNF like grammar definition language. Xtext is a part of Eclipse TMF (Textual Modeling Framework) project and it enables creation of domain specific languages from grammar definitions. After writing the grammar, the Xtext language generator is run which builds the full implementation of the domain specific language for the written grammar. Subsequently, the Xtext language generator extracts the metamodel from the grammar and outputs it as an Ecore metamodel. We use this Ecore metamodel as the abstract syntax definition while defining the visual concrete syntax of the corresponding DSL. Traditionally, GMF (Graphical Modeling Framework) tools are used in order to define the visual concrete syntax based on an Ecore metamodel. GMF tools provide also a set of generative components for generating diagram editors in Eclipse. To support the easy development we have used the tool EuGENia [12] for generating the required models for GMF diagram generation from a single annotated Ecore metamodel. For annotating the Ecore metamodel with visual concrete syntax information, we have utilized EMFatic [10]. That is, using specific annotations the viewpoint definer states for each metamodel (viewpoint) element the corresponding graphical notations. The resulting Ecore metamodel is given as an input to EuGENia generator, which generates the required models for GMF diagram editor generation. Lastly, both textual and visual editors defined for viewpoint are exported as plug-ins to Eclipse. A view modeler can use those editors to model architecture views based on the viewpoint.

Figure 12 shows a sample screenshot from the SAVE-BENCH tool. SAVE-BENCH provides a user interface with 5 different panes to define the different elements of the DSL. For the evaluation of the viewpoint we have used Excell sheets that resulted in the dot graphs as shown in section 6.

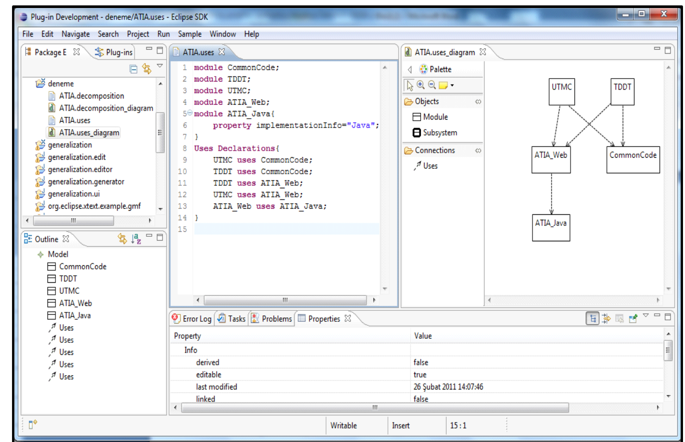

Figure 12. Snapshot of the SAVE-BENCH tool for modeling architectural views

8. RELATED WORK

Organizing the system as a set of viewpoints has also been addressed in enterprise application system using so-called enterprise architecture frameworks. Examples include the early Zachman's Framework for Enterprise Architecture [37], The Open Group Architecture Framework (TOGAF) [35], and the ISO (ISO/IEC 10746) Reference Model of Open Distributed Processing (RM-ODP) [18].

Architecture description languages (ADLs) have been proposed to model architectures. For a long time there have been little consensus on the key characteristics of an ADL. Different types of ADLs have also been introduced. Some ADLs have been defined to model a particular application domain, others are more general-purpose. Also the formal precision of the ADLs differ; some have a clear formal foundation while others have been less formal. Several researchers have attempted to provide clear guidelines for characterizing and distinguishing ADLs, by providing comparison and evaluation frameworks. Medvidovic and Taylor [25] have proposed a definition and a classification framework for ADL which states that an ADL must explicitly model *components*,

connectors, and their *configurations*. Furthermore, they state that *tool support* for architecture-based development and evolution is needed. These four elements of an ADL include other sub-elements to characterize and compare ADLs. The focus in the framework is thus on architectural modeling features and tool support. In adopting a software language engineering approach we have focused on the three language elements of abstract syntax, concrete syntax and static semantics. In fact we could analyze also existing ADLs based on the approach in this paper. That could be complementary to earlier evaluations of ADLs.

xADL has been introduced to support modularity and extensibility of architectural modeling [8]. Despite earlier ADLs xADL is not a single fixed ADL but encapsulates various ADL features in modules that can be composed to form new ADLs. This is achieved by using the extension mechanisms provided by XML and XML schemas. xADL forms the basis for the ArchStudio 4 [17], an open-source software and systems architecture development environment including tools for modeling, visualizing, analyzing and implementing software and systems architectures. It is based on the Eclipse open development platform. Similar to our tool it is an architecture meta-modeling environment that can be used to define new views. In ArchStudio, new viewpoints could be defined by extending the core language. In our approach we focus on the software language engineering elements of abstract syntax, concrete syntax and static semantics. In addition viewpoints can be defined from scratch using Xtext [36] or extended.

In the enterprise architecture 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 [29].

Romero et al. tackle the viewpoint formalization problem from model-driven development perspective and defined UML profile for viewpoints of RM-ODP [30]. 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 [18] 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

9. CONCLUSION

The discipline of software architecture description has substantially evolved in the last decades. We can characterize the evolution from the following two perspectives. First of all, there seems now a common awareness that architecture should be modeled using multiple views. Having multiple views of the architecture helps to separate the concerns and as such support the modeling, understanding, communication and analysis of the software architecture for different stakeholders. In the literature, initially views were not explicit, later a fixed set of viewpoints has been proposed to model and document the architecture. Because of the different concerns that need to be addressed for various systems, the current trend recognizes that the set of views should not be fixed but *open-ended*. The second dimension of evolution considers the formal precision of the architectural descriptions. Initially software architecture was represented using arbitrary box-and-lines notations leading to ambiguous interpretations. Later on, it was acknowledged to provide more formal support for architectural modeling, both visually and textually.

In this context, the definition of properly defined architectural viewpoints has become important. Unfortunately, it appears that the current literature does not provide yet a review process for architectural viewpoint languages. In this paper we have provided an evaluation framework for evaluating existing or newly defined architectural viewpoint languages based on software language engineering. The approach does not assume a particular architecture framework and can be applied to existing viewpoints or newly defined viewpoints. One of the recent architectural frameworks that includes a broad set of viewpoints is the Views and Beyond approach. We have been able to review the first and second edition of the viewpoints of the Views and Beyond approach [4][5]. To validate our statement we have analyzed the viewpoints in the Views and Beyond approach, and defined all these viewpoints as domain specific languages. We have compared both the first edition and second edition of the Views and Beyond approach and illustrated the differences in formal precision. 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. Finally, we will extend our evaluation framework and in addition to the language formality aspect we will also consider other aspects of viewpoints such as coverage of stakeholder concerns, orthogonality and consistency among viewpoints.

10. REFERENCES

[1] Archimate 1.0 Specification, The Open Group, Tech. Rep. C091, Feb. 2009.

[2] M.A. Babar & I. Gorton. Comparison of Scenario-Based Software Architecture Evaluation Methods, *Proceedings of the 11th Asia-Pacific Software Engineering Conference.* NSW Australia, Nov-Dec. 2004. IEEE, 2004.

[3] L. Bass, P. Clements, & R. Kazman. *Software Architecture in Practice*, 2nd ed., (Chapter 9). Addison-Wesley, 2003 (ISBN: 978-0-321-15495-8).

[4] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, J. Stafford. Documenting Software

Architectures: Views and Beyond. First Edition. Addison-Wesley, October 2002.

[5] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, J. Stafford. Documenting Software Architectures: Views and Beyond. Second Edition. Addison-Wesley, 2010.

[6] P. Clements, R. Kazman, & M. Klein. *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley, 2002.http://www.sei.cmu.edu/library/abstracts/books/020170482X.cfm

[7] E. Demirli & B. Tekinerdogan. Software Language Engineering of Architectural Viewpoints, in Proc. of the 5th European Conference on Software Architecture (ECSA 2011), LNCS 6903, pp. 336–343, 2011.

[8] E. Demirli & B. Tekinerdogan. SAVE: Software Architecture Environment for Modeling Views, in proc. of WICSA 2011: 9th Working IEEE/IFIP Conference on Software Architecture, pp. 355-358, 20-24 June 2011.

[9] L. Dobrica and E. Niemela. A Survey on Software Architecture Analysis Methods. IEEE Transactions on Software Engineering, 28(7):638–654, 2002.

[10] Eclipse Modeling Framework Technology – EMFatic Project,http://www.eclipse.org/modeling/emft/?project=emfatic, accessed February 2011.

[11] Eclipse Graphical Modeling Framework, http://www.eclipse.org/gmf/, accessed February 2011.

[12] EuGENia, http://www.eclipse.org/gmt/epsilon/doc/eugenia/, accessed February 2011.

[13] Final Report of the Software Architecture Review and Assessment (SARA) Group, Version 1.0, 2002. http://philippe.kruchten.com/architecture/SARAv1.pdf

[14] M. Fowler, S. Scott, G. Booch. UML distilled, Object Oriented series, 179 p. Addison-Wesley, Reading, 1999.

[15] N. Hämäläinen & J. Markkula. Quality Evaluation Question Framework for Assessing the Quality of Architecture Documentation. *Proceedings of International BCS Conference on Software Quality Management*. University of Tampere, SQM, 2007

[16] C. Hofmeister, R. Nord, and D. Soni. Applied Software Architecture. Addison-Wesley, NJ, USA.

[17] ISR, Institute for Software Reseach. Archstudio 4.0 tool set for the xadl language, http://www.isr.uci.edu/projects/archstudio/

[18] [ISO/IEC 10746-2:1996] International Organization for Standardization & International Electrotechnical Commission. In-formation Technology - Open Distributed Processing - Reference Model: Foundations (ISO/IEC 10746-2). 1996.

[19] [ISO/IEC 42010:2007] International Organization for Standardization & International Electrotechnical Commission. Systems and software engineering—Recommended practice for architectural description of software-intensive systems (ISO/IEC 42010), July 2007.

[20] A. Kleppe. Software Language Engineering: Creating Domain-Specific Languages Using Metamodels. Addison-Wesley Longman Publishing Co., Inc., Boston, 2009.

[21] P. Kruchten. The 4+1 View Model of Architecture. IEEE Software, 12(6):42–50, 1995.

[22] P. Kruchten. The Rational Unified Process: An Introduction, Second Edition. Addison-Wesley, Boston, MA, USA, 2000.

[23] A.J. Lattanze. Architecting Software Intensive Systems: A Practitioner's Guide, Auerbach Publications, 2009.

[24] M. W. Maier, D. Emery, and R. Hilliard. Software Architecture: Introducing IEEE Standard 1471. IEEE Computer, 34(4):107–109, 2001.

[25] N.Medvidovic & R.N. Taylor. A classification and comparison framework for Software Architecture Description Languages, IEEE Trans. on Software Engineering, Vol. 26, No.1 pp. 70-93, 2000..

[26] S.J. Mellor, K. Scott, A. Uhl, D. Weise. MDA Distilled: Principle of Model Driven Architecture, Addison Wesley, Reading , 2004

[27] R.L. Nord, P.C. Clements, D. Emery, and R. Hilliard, A Structured Approach for Reviewing Architecture Documentation, TECHNICAL NOTE, CMU/SEI-2009-TN-030, December 2009.

[28] D.L. Parnas & D.M. Weiss. Active Design Reviews: Principles and Practices, 215-222. *Pro-ceedings of 8th International Conference on Software Engineering*, 1985. Reprinted in Hoffman, D. and Weiss, D., *Software Fundamentals,* 2001.

[29] C. Peña, J. Villalobos. An MDE Approach to Design Enterprise Architecture Viewpoints, IEEE 12th Conference on Commerce and Enterprise Computing (CEC), vol., no., pp.80-87, 10-12 Nov. 2010.

[30] J. R. Romero, J. M. Troya, A. Vallecillo. Modeling ODP Computational Specifications Using UML, The Computer Journal 2008 51: 435-450.

[31] T. Stahl, M. Voelter. Model-Driven Software Development, Addison-Wesley, 2006.

[32] B. Tekinerdogan, A. Moreira, J. Araújo, and P. Clements. Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design. In: Workshop Proceedings. University of Twente, TR-CTIT-04-44, October, 2004.

[33] B. Tekinerdogan. ASAAM: Aspectual Software Architecture Analysis Method, in Proc. of 4th Working IEEE/IFIP Conference on Software Architecture (WICSA), pp. 5-14, June, 2004.

[34] B. Tekinerdogan, A. Moreira, J. Araújo, P. Clements, "Early aspects: aspect-oriented requirements engineering and architecture design", Report Early Aspects Workshop at AOSD, Lancaster, UK, March, 2004.

[35] TOGAF 1995 -The Open Group Architecture Framework, Version 8.1.1., 1995. http://www.opengroup.org/architecture/togaf8-doc/arch/

[36] Xtext – Language Development Framework, http://www.eclipse.org/Xtext/, accessed on February 2011.

[37] J.A. Zachman. A Framework for Information Systems Architecture. IBM Systems Journal, Vol. 26. No 3, pp. 276-292, 1987.