Providing support for creating next generation software architecture languages

Ivano Malavolta
Dipartimento di Informatica, Università dell'Aquila, Via Vetoio, L'Aquila, Italy
ivano.malavolta@univaq.it

ABSTRACT

Many languages for software architectures have been proposed, each dealing with different stakeholder concerns, operating at different levels of abstraction and with different degrees of formality. It is known that a universal architectural language cannot exist since the various concerns, requirements, and domains may change. Moreover, stakeholder concerns and needs are various and ever evolving even while designing a single system. Model-driven techniques may be used to answer the need for supporting the creation of extensible, customizable and stakeholder-oriented architectural languages (i.e., next generation architectural languages). Part of this approach is developed in a framework called BYADL.

In this paper I present the big picture behind the approach, the research aspects considered in order to get BYADL closer to an ideal architectural framework and future research issues.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures—*Domain-specific architectures*; D.2.11 [**Software Engineering**]: Software Architectures; D.2.10 [**Software Engineering**]: Design

General Terms

Design, Modeling.

1. RESEARCH PROBLEM

Architectural languages [5] are fundamental while designing complex, critical systems that need methods and tools ensuring specific properties like dependability, fault tolerance, security. Since they have been proposed, their underlying philosophies, concepts and tools are continuously evolving.

Nowadays, the view of software architecture is getting broader and takes into consideration emerging concepts like multiple stakeholders concerns and their design decisions [2, 3, 5]. However, current ADLs are still not aligned with the evolved concept of software architecture. Emerging requirements like language extensibility, customization, or multiple views management are still not adequately supported by current ADLs.

In [1] a list of the requirements a next generation ADL shoud support have been traced out. *Domain specific concerns* (**R1**): allow software architects to instantiate a generic architectural language into a choosen domain by adding domain-specific modelling elements. *Multiple views* (**R2**): support for adding, deleting, merging the various views that make up the overall architectural description of the system. *Analysis features* (**R3**): if analysis is a need, analysis-specific concepts should be part of the architectural language. *Interoperability with other ADLs* (**R4**): architectural languages should be able to interoperate so to globally satisfy all the stakeholders needs [4]. *Promote architecture-centric development* (**R5**): architectural languages should be integrated into the development processes taking into account other life-cycle activities. *Tool support* (**R6**): tools are fundamental and should support features as editing, visualization, analysis and so on.

I am developing an approach that allows the creation of architectural languages satisfying the above mentioned requirements; part of the approach have been already considered in a framework called BYADL [1]. In this extended abstract I provide a high-level vision on how to support next generation ADLs; I will also present the research outcomes we had and the main goals we envision on it.

2. BACKGROUND AND RELATED WORK

Some research work has been done in the direction of developing extensible and customizable architectural languages. Acme[1] specifications may be extended by means of properties that can decorate each element; these properties are uninterpreted. xADL[2] is based on XML and so it is fully extendable; however, XML schemas do not provide advanced facilities to define the semantics of individual elements. AADL[3] provides extensibility mechanisms (annexes and property sets), but it does not provide automated support for them. XTEAM[4] provides domain-specific analysis by transforming a composed metamodel to analysis-specific notations; however this is not done via a dedicated interoperability framework.

These approaches propose extension mechanisms that only partially satisfy requirements **R1**, **R2**, and **R3**. Requirement **R4** is not properly addressed by any language (just Acme to some extent). Concrete and scalable solutions for **R5** are missing. Tools supporting the features described by **R6** can be considered not mature.

3. THE ENVISIONED APPROACH

Building on the requirements listed in Section 1 and the evolved notion of software architecture, I am investigating on how a next

[1]Acme: `http://www.cs.cmu.edu/~acme/`
[2]xADL: `http://www.isr.uci.edu/projects/xarchuci/`
[3]AADL: `http://www.aadl.info/`
[4]XTEAM: `http://www-scf.usc.edu/~gedwards/xteam.html`

generation architectural language (and the framework supporting it) should be.

R1 states that an architectural language must support *domain-specific concerns*. Given the extensibility of the language, it may be extended to consider domain-specific elements of the system as first-class entitites. The framework allows to specialize generic architectural elements to particular domains; this provides a dedicated reasoning environment for domain experts. Every stakeholder will be familiar with the concepts he is dealing with.

An architectural framework should provide also a *repository* in which architectural notations, domain informations, decisions, and architectural design rules are stored. This aspect is strongly related to both **R1** and **R2** requirements. Such a repository should be enriched with suitable metadata allowing the framework to automatize facilities like search, querying and, in general, to guide the stakeholder using it. Using a central repository promotes also the reuse of knowledge, notations, and design rules.

Regarding **R3**, assuring as early as possible the *correctness* of an SA is fundamental in order to produce quality software. Each analysis technique requires analysis-specific concepts that are part of the architectural language.

Next generation architectural languages must *interoperate* (**R4**) so to globally satisfy all the stakeholder needs; in our vision the framework may also keep notations separated, and lets them interoperate by means of suitable transformations.

According to **R5**, architectural languages should be *integrated into the development life-cycle* phases; this point has two main implications: (i) the language must offer not only design elements, but also elements related to requirements, development, testing, and so on; (ii) the framework should support the management of development teams, project timing, risks and provide facilities for collaborative development/design. This implies that the framework considers stakeholders as a first-class entity, ; it should also provide different views of the architecture depending on the current stakeholder and filter system informations depending on the stakeholder's access rights. Stakeholders may be domain experts, system developers, designers, customers, financial managers, project managers and so on. The repository should also support mechanisms for merging, splitting, composing notations with specific views, domain-specific concerns and analysis-oriented concepts. These aspects are related also to **R1** and **R2**.

Tool support (**R6**) is one of the key points for the success of an ADL. So, each facility outlined above must be implemented in a tooling environment. The tool of the language should be extensible and customizable too. Eclipse[5] is a clear example of how a framework may benefit from extensibility and customization features.

4. RESULTS AND CONTRIBUTIONS

In [1] we introduced BYADL, a model-driven framework embodying the one outlined in Section 3. BYADL provides an incremental approach to build customized and customizable ADLs starting from an already existing one. The main input of BYADL is represented by the metamodel of the ADL to be extended. BYADL contains a repository of metamodels; each metamodel is tagged depending on the concepts it contains(e.g., domain, ADL, customization). The metamodel is extended by applying specific metamodel composition operators, they are: *Match*, *Inherit*, *Reference*, *Expand*. The composition engine performs semantic checks to avoid incidental errors. The ADL obtained at the end of the process is composed of (i) *abstract syntax*, i.e. the metamodel obtained by means of the composition mechanisms, (ii) a set of *concrete syntaxes*, i.e., (semi)-automatically generated textual and graphical notations to visualize and edit models, and (iii) *semantics* describing the meaning of each language construct. The semantics of the extended language is given by means of semantic relationships between the language's elements and elements of a core set of architectural concepts called A_0 [4]. By means of such relationships, the elements of the ADL implicitly inherit the built-in semantics of A_0. Model migrators are defined in order to automatically generate model transformations able to reflect the models defined within the newly created ADL, back to the original tools. Migrators are automatically generated by higher-order transformations and are fundamental for analyzing ADL models using their original tools.

[5]Eclipse Project Home Page: `http://www.eclipse.org`

BYADL addresses the main emerging requirements of next generation ADLs at different degrees. More precisely, the composition mechanism satisfies **R1**, **R2**, and **R3** since they provide mechanisms to extend the ADL with *Domain specific concerns*, with new *Architectural views*, and with *Analysis notations*, respectively.

Focussing on the management of multiple views, if software architects represent the concepts of a viewpoint through a metamodel, then BYADL can be used to store viewpoints in the repository and organize them with suitable metadata. So, the BYADL composition engine (along with the migrators engine) may be used as a means to merge, split or customize architectural viewpoints within an integrated framework. This is one of the main research direction we are currently pursuing on BYADL.

Furthermore, the proposed operators are used also to define relationships between software architecture and development processes and methodologies, thus addressing requirement **R4**.

The requirement **R5** is satisfied by reusing **DUALL**Y, a framework for ADLs interoperability we proposed in [4]. **DUALL**Y promotes to keep notations separated and define transformations among them. BYADL reuses **DUALL**Y to achieve interoperability: once the new ADL has been generated, it interoperates with all the other notations that are already in the **DUALL**Y environment.

The **R6** requirement is satisfied because the framework of the extended ADL is the BYADL tool itself (with its textual and graphical editors, extensibility mechanisms, and migrators).

Finally, an interesting aspect of BYADL is that it may be used to create a new ADL from scratch by considering a generic metamodel for SAs as starting notation. I am investigating on this aspect, on its feasibility and on what should be the starting point for the composition process.

Acknowledgments

This work was partially supported by the Italian Government under the project PRIN 2007 D-ASAP (2007XKEHFA).

5. REFERENCES

[1] D. Di Ruscio, I. Malavolta, H. Muccini, P. Pelliccione, and A. Pierantonio. Developing next generation adls through mde techniques. In *ICSE 2010, to appear*.

[2] ISO. Fourth working draft of Systems and Software Engineering – Architectural Description (ISO/IECWD4 42010). Working doc.: ISO/IEC JTC 1/SC 7 N 000, 2009.

[3] P. Kruchten, P. Lago, and H. van Vliet. Building up and reasoning about architectural knowledge. *QoSA*, 2006.

[4] I. Malavolta, H. Muccini, P. Pelliccione, and D. Tamburri. Providing architectural languages and tools interoperability through model transformation technologies. *IEEE TSE*, 2010.

[5] R. N. Taylor, N. Medvidovic, and E. M. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. John Wiley & Sons, January 2009.