ByADL: an MDE framework for building extensible Architecture Description Languages[*]

Davide Di Ruscio, Ivano Malavolta, Henry Muccini,
Patrizio Pelliccione, Alfonso Pierantonio

University of L'Aquila, Dipartimento di Informatica
{davide.diruscio,ivano.malavolta,henry.muccini,
patrizio.pelliccione,alfonso.pierantonio}@univaq.it

Abstract. In order to deal with evolving needs and stakeholder concerns, next generation ADLs should support incremental extension and customization. In this direction we proposed BYADL (Build Your ADL), a framework which allows software architects to (i) extend existent ADLs with domain specificities, new architectural views, or analysis aspects, (ii) integrate an ADL with development processes and methodologies, and (iii) customize an ADL. This paper presents the BYADL tool and its features.

1 Introduction

A broader view of Software Architecture (SA), which is being accepted today, is far beyond the traditional perception of an SA as a set of constituting elements (such as components, connectors and interfaces) and looks at the multiple stakeholder's concerns and their design decisions [1–3]. Many Architecture Description Languages (ADLs) have been proposed in the last years. Software engineers found existent ADLs inadequate for modelling concerns judged unavoidable by system's stakeholders. Furthermore, stakeholders's concerns vary from system to system and from domain to domain; this demonstrates that it is not possible to define a general, optimal ADL once and forever. Therefore, ADLs should be extensible in order to be able to adapt to different stakeholder's concerns and to different domain specificities. However, first attempts of extensible ADLs do not deal with semantic aspects of extensions in a satisfactory way.

BYADL (Build Your ADL) [4] is a framework that supports a software architect in defining its own ADL, which is optimal according to specific stakeholder's concerns, starting from an existent ADL. BYADL provides extensibility mechanisms for: (i) adding domain specificities, new architectural views, or analysis aspects, (ii) integrate ADLs with development processes and methodologies, (iii) customize ADLs by fine tuning them. These mechanisms are implemented in the BYADL tool that is the main objective of this paper. The tool supports different features such as model editing, visualization with different views, extensibility, interoperability, and analysis. The tool has been applied on a real system called Integrated Environment for Communication on Ship (IECS); the case study comes from a project developed within Selex Communications, a company mainly operating in the naval communication domain.

[*] This work is partly supported by the Italian PRIN d-ASAP project.

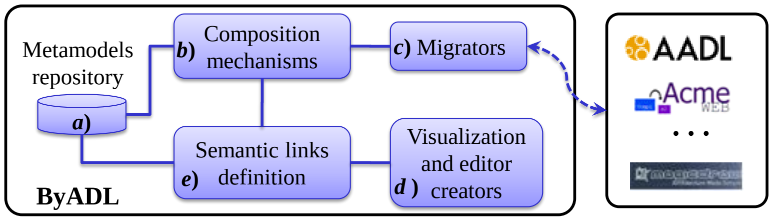

Fig. 1. The high-level design of the BYADL framework

The paper is organized as follows: Section 2 presents the main aspects of the BYADL framework, Section 3 presents the tool, and finally Section 4 concludes the paper.

2 The ByADL framework

Figure 1 shows the high-level design of the BYADL framework. The main input of BYADL is the metamodel of the ADL to be extended. For this reason BYADL contains a repository of metamodels as shown in Figure 1.a. The metamodel is extended by means of composition mechanisms (see Figure 1.b) that rely on specific composition operators for metamodels.

It is important to note that, even though this could be technically possible, we do not allow to compose two different ADLs since this could lead to the creation of a “chaotic” and “vague” language. In such cases we believe that it is better to keep the two ADLs separated, and to use interoperability techniques to translate from one ADL to a different one [5]. The composition operators are: *Match*, *Inherit*, *Reference*, and *Expand*, whose semantics is detailed in [4]. The composition engine performs also semantic checks to avoid incidental errors. The *Migrators* (see Figure 1.*c*) component is used to automatically generate model transformations able to reflect the architectural models defined within the newly created ADL, back to the original tools. In the BYADL framework, migrators are automatically generated by means of higher-order model transformations. The ADL obtained at the end of the process is a modeling language consisting of (i) an *abstract syntax*, i.e. the metamodel obtained by means of the composition mechanisms, (ii) a set of *concrete syntaxes*, i.e., textual and graphical notations to visualize and edit models conforming to the composed metamodel (see Figure 1.*d*), and (iii) *semantics* describing the meaning of the language constructs [6] (see Figure 1.*e*). The semantics of the extended language is given by means of semantic relationships between the language’s elements and elements of a target semantic domain called A_0 [5].

3 The ByADL tool

The BYADL tool[1] is implemented as a plugin of the Eclipse[2] platform. More specifically, it extends the ATLAS Model Management Architecture (AMMA)[3]. Metamodel compositions are specified as weaving models defined by using the ATLAS Model

[1] BYADL Web site: `http://byadl.di.univaq.it/`
[2] Eclipse Project Home Page: `http://www.eclipse.org`
[3] AMMA: `http://wiki.eclipse.org/AMMA`

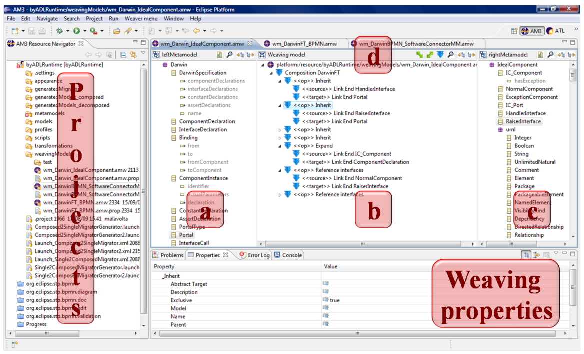

Fig. 2. Metamodels composition user interface

Weaver (AMW). All the involved model transformations are based on the ATLAS Transformation Language (ATL), a hybrid model transformation language with declarative and imperative constructs. Models and metamodels are managed by means of EMF[4], a modeling framework for Eclipse. In the following we present how the features of the BYADL framework described in Section 2 are realized in its supporting tool.

Metamodels import. Importing a metamodel into the *Metamodels repository* is a three-steps process:
1. *Obtaining a metamodel*. The BYADL tool works on EMF metamodels. If the metamodel of an architectural notation is not available, many techniques exist to obtain such a metamodel, like using TCS[5] in case of textual notations, or using the **DUALLY** importing engine in case of UML-based notations.
2. *Tagging a metamodel*. Tagging metamodels guides ADLs extensions and keeps the involved metamodels organized. The tags available in BYADL reflect the different kinds of metamodel involved in the composition scenarios, namely: `ADL`, `Domain`, `Analysis`, `Viewpoint`, `Process`, `Methodology`, `Customization`. The importing wizard allows the user to associate one or more tags to the metamodel being imported.
3. *Providing semantics to a metamodel*. As stated in Section 2, in BYADL a kind of translational semantics is provided to a metamodel by linking it to the A_0 semantic domain. We extended the AMW interface so that the semantic links to the A_0 metamodel guide the application of the composition operators. More precisely, once applying an operator, the BYADL tool highlights as target the metaclasses that are semantically compatible with the source metaclass.

Metamodels composition. Software engineers create a new ADL by composing two metamodels already imported into the *Metamodel Repository*. Metamodels are composed by specifying weaving models which represent the application of the composition operators. Figure 2 shows the AMW graphical interface we extended for metamodel composition. The woven metamodels are rendered into two lateral panels (points *a* and *c* in the figure) using the standard EMF tree-based interface. The central panel (point *b* in the figure) represents the composition weaving model. Our extension consists of a specific weaving metamodel defining the four kinds of composition operators and a

[4] Eclipse Modeling Framework (EMF) project Web site: `http://www.eclipse.org/emf`
[5] TCS: `http://wiki.eclipse.org/TCS`

dedicated weaving toolbar (see Figure 2.*d*). The composed metamodel is generated by clicking on a button of the BYADL weaving toolbar; the result is a metamodel which is automatically loaded into the *Metamodels repository* and tagged as *ADL*.

Model migrators generation. A model migrator is a specific ATL transformation; its inner logic is represented by the operators applied in the weaving model during the composition phase. The BYADL weaving toolbar contains functionalities (see Figure 2.*d*) to automatically generate the migrators starting from the current composition weaving model. Model migrators may be used also outside the BYADL tool.

Editors generation. In BYADL there are three possibilities to produce an editor for the ADL being developed: *tree-based*, *textual*, and *graphical*. Each editor has different levels of usability and requires different efforts for the customization (if needed). The tree-based editor, with its collapsible and hierarchical structure, is automatically provided by EMF. The textual editor is automatically generated and conforms to the Human-Usable Textual Notation (HUTN) specification[6]. The produced textual editor supports syntax highlighting and automatic conformance check with respect to the metamodel of the new ADL. The graphical editor is based on the EuGENia[7] tool: exploiting specific annotations of the metamodels involved in the composition (included A_0), a graphical editor is automatically generated. Obviously the generation of the editor is limited to elements for which EuGENia annotations are provided. Special policies regulate the choice of the graphical element to be used when more than one metamodel provide EuGENia annotations for a specific concept.

4 Conclusions

In this paper we presented the BYADL tool. Starting from an existing ADL, BYADL allows software architects to incrementally extend and customize an ADL according to stakeholder's concerns. BYADL ensures the compatibility with existing tools by means of automatically generated migrators. Our tool also supports the generation of textual and graphical editors for the newly created ADL. The generation of graphical editors is in its prototypal version, this aspect is one of the main future work directions.

References

1. ISO: Fourth working draft of Systems and Software Engineering – Architectural Description (ISO/IECWD4 42010). Working doc.: ISO/IEC JTC 1/SC 7 N 000, IEEE (2009)
2. Kruchten, P., Lago, P., van Vliet, H.: Building up and reasoning about architectural knowledge. Quality of Software Architectures (2006)
3. Taylor, R.N., Medvidovic, N., Dashofy, E.M.: Software Architecture: Foundations, Theory, and Practice. John Wiley & Sons (2009)
4. Di Ruscio, D., Malavolta, I., Muccini, H., Pelliccione, P., Pierantonio, A.: Developing next generation adls through mde techniques. In: ICSE 2010, IEEE Computer Society (2010)
5. Malavolta, I., Muccini, H., Pelliccione, P., Tamburri, D.: Providing architectural languages and tools interoperability through model transformation technologies. IEEE TSE **36**(1) (2010)
6. Cuadrado, J.S., Molina, J.G.: A model-based approach to families of embedded domain specific languages. IEEE TSE **99**(RapidPosts) (2009) 825–840

[6] HUTN specification: `http://www.omg.org/spec/HUTN/`.
[7] EuGENia: `http://www.eclipse.org/gmt/epsilon/doc/eugenia/`.