DUALLY: Putting in Synergy UML 2.0 and ADLs

H. Muccini, P. Inverardi
Dipartimento di Informatica
University of L'Aquila
Via Vetoio, 1 - L'Aquila, Italy
muccini, inverard@di.univaq.it

P. Pelliccione
Software Engineering Competence Center
University of Luxembourg
6, rue R. Coudenhove-Kalergi, Luxembourg
patrizio.pelliccione@uni.lu

Abstract

Many formal languages to describe software architectures (SA) have been proposed so far, but only very few of them are still supported and used in practical contexts. Many UML profiles and extensions have been provided when UML became a standard, in order to model as much as possible architectural concepts. They permit to specify specific needs, but the effort in creating a sort of unified UML-based language for architectures is still not paying. In fact, since different communities require different information to be put into a diagram, depending on which architectural design aspects should be represented and analyzed, the idea of an unified UML language for SA seems to be not adequate. Building on these considerations, we propose **DUALLY**, *a core set of UML concepts, well suited for SA modeling, together with a framework which provides extensibility mechanisms to adapt the initial notation, in order to meet different needs.*

1 Introduction

In a few years, the SA community has observed a proliferation of architecture description languages (ADLs) [7] for rigorous and formal SA modeling and analysis, and the introduction of supporting tools.

Although several studies have shown the suitability of such formal languages for SA modeling and analysis, industries still tend to prefer model-based (semi-formal) notations. In particular, with the introduction of UML as the de-facto standard to model software systems and its widespread adoption in industrial contexts, many extensions and profiles have been proposed to "adapt" UML to model architectures (e.g., [5, 6, 9]). However, while such UML notations are well suited to modeling some aspects of software architectures, they fail in modeling others [4]. These extensions permit to reduce the gap between UML and ADLs, but they still fail in representing all aspects of ADLs [4, 7, 6]. Moreover, based on our academic and industrial experience on SA modeling for analysis, we realized that *modeling for documenting is quite different from modeling for analysis*, and *different analysis techniques usually require different notations*. There is not best formalization since it depends on which aspects of architectural design we want to represent and then analyze. Any time a new slightly different analysis is required, new modeling concepts are needed. This suggests there is neither a unique language for representing SAs, not a unique fit between UML and ADLs.

Then, can we really advocate a stronger synergy between UML and ADLs can be created? We believe this possible and we suggest to approach the problem from a different perspective: our starting point consists in i) *identifying a core set of architectural elements always required*; then, ii) *we create an UML profile able to model the core architectural elements previously identified*. iii) *We provide extensibility mechanisms to add modeling concepts needed for specific analysis*. Finally, iv) *we describe how semantic links mechanisms can be kept between different notations*.

From existing ADLs and from our experience on modeling and analysis of SA, we observe that there are some core architectural concepts commonly understood by any software architect, while other modeling elements are needed for specific analysis. We inherit from both xArch [1] and ACME [3] the idea of identifying a maximum common denominator (a core set) of architectural concepts (point i). Then, we create the **DUALLY** UML profile to model such core architectural concepts. Since we cannot expect to create a unique fit between UML modeling elements and architectural concepts, instead of providing a thorough and precise UML-SA profile, we propose minimal extensions to UML 2.0 in order to make the profile applicable in industrial contexts (point ii). Since our profile covers only core architectural concepts, **DUALLY** provides extensibility mechanisms, at different levels: the UML profile notation for core elements can be extended with additional modeling elements. New diagrams can be introduced. Existent

modeling and analysis tools can be integrated. The first two extensions are performed through UML profiling and metamodeling extensions. The latter extension is supported by a plugin framework, which permits to easily add new design or analysis tools (point iii). Finally, **D**U**AL**LY permits to specify semantic links, which allow the specification of semantic equivalences and differences among model elements (point iv).

2 **D**U**AL**LY

Two problems hamper the success of strategies based on traditional ADLs: 1) languages used by ADLs are generally formal and sophisticated, making difficult their integration in industrial life-cycles, 2) it is impossible to construct an ADL able to support every kind of analysis, since any analysis technique requires additional analysis-specific notations and models.

DU**AL**LY merges UML 2.0 and ADLs through a UML-based notation for Software Architecture descriptions. This choice is motivated by the fact that UML is the de-facto standard to model software systems and it is largely approved by the industries. The definition of the UML profile, able to model the identified core architectural elements, allows the solution of the problem 1.

To overcome problem 2 we outline an extendible framework that permits to add models and to extend existing ones in order to support the introduction of analysis techniques. Semantic relations are proposed to bind different elements of different models. Semantic links will permit to specify semantic equivalences/differences between model elements. These semantic relations allows to use the same modeling and analysis framework (i.e., **D**U**AL**LY) to perform many different analysis, by exploiting equivalences/differences among UML-SA notations. In particular, it allows to embed the feedback resulting from analysis into the models. For example, it is intuitive that upon detecting a deadlock in a software model, a critical component may be split in two components, and this refinement may heavily affect the software performance. Traditionally, we have to select an ADL which allows for deadlock analysis (P1) and an ADL which allows for performance analysis (P2). We have to provide two specifications in two different languages. We have to run the deadlock analysis tool, get the feedback, modify the SA and then modify the performance specification to take into account such changes.

With **D**U**AL**LY, instead, we create one profile for P1 and one for P2. P1 and P2 are introduced into **D**U**AL**LY and semantics relations between the profiles are defined. Then, P1 is submitted to deadlock analysis. If some core architectural elements in P1 are modified, thanks to the semantic relations, this change is propagated to the P2 profile. If changes in P1 affect only P1-specific elements, than such changes do not affect P2.

3 Conclusions and Future Work

DU**AL**LY is an ongoing work which provides an extensible UML-based notation for SA modeling and analysis. **D**U**AL**LY is composed by a UML profile for SA modeling, which permits to easily specify SA concepts slightly extending the UML 2.0 notation. **D**U**AL**LY provides extensibility mechanisms and semantic rules which allow the introduction and integration of UML-based notations and tools.

In future work, we will validate this initial work by extending the **D**U**AL**LY profile with our UML-based notations for SA-based model-checking and testing [8, 2].

References

[1] xArch. http://www.isr.uci.edu/architecture/xarch/.

[2] CHARMY Project. Charmy Web Site. http://www.di.univaq.it/charmy, 2004.

[3] Acme. http://www-2.cs.cmu.edu/∼acme/, Since: 1998. Carnegie Mellon University.

[4] E. M. Dashofy, A. van der Hoek, and R. N. Taylor. An infrastructure for the rapid development of xml-based architecture description languages. In *ICSE '02: Proceedings of the 24th Int. Conf. on Software Eng.*, pages 266–276, New York, NY, USA, 2002. ACM Press.

[5] D. Garlan and A. Kompanek. Reconciling the needs of architectural description with object-modeling notations. In *Proc. of the Third Int. Conf. on the Unified Modeling Language*, 2000.

[6] N. Medvidovic, D. S. Rosenblum, D. F. Redmiles, and J. E. Robbins. Modeling Software Architectures in the Unified Modeling Language. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(1), January 2002.

[7] N. Medvidovic and R. N. Taylor. A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactions on Software Engineering*, 26(1), January 2000.

[8] H. Muccini, A. Bertolino, and P. Inverardi. Using Software Architecture for Code Testing. *IEEE Trans. on Software Engineering*, 30(3):160–171, March 2003.

[9] B. Selic. On modeling architectural structures with uml. In *Proc. of the Workshop on Describing Software Architecture with UML, in ICSE 2001*, Toronto, Canada, 2001.