Integrating the Unified Modeling Language with an Architecture Description Language

Cécile Péraire, Robert A. Riemenschneider and Victoria Stavridou

System Design Laboratory
SRI International
Menlo Park, CA 94025
{cecile, rar, victoria}@sdl.sri.com

1. Introduction

The goal of our research is to promote the wider use of languages and tools dedicated to specification and validation of software architectures. This goal can be reached by integrating UML (Unified Modeling Language) with an Architecture Description Language (ADL). UML has the advantage of being a visual language widely used for modeling a large range of software systems, and it is supported by many tools. ADLs have the advantage of being formally defined. This theoretical foundation makes description languages unambiguous and allows the development of validation methods and tools.

The integration will encompass the advantages of both languages. Specifically, it will give formal semantics to UML constructs and thus will provide UML with a theoretical foundation for architecture modeling which is currently lacking. Furthermore, the integration will provide benefits for both ADL and UML users: it will enable ADL users to utilize general-purpose UML tools, and will enable UML users to utilize ADL validation capabilities.

The structure of this paper is the following. Sections 2 and 3 introduces respectivaly ADLs and UML and focus on their advantages and drawbacks. Section 4 presents a method for integrating UML with an ADL.

2. Software Architecture and Architecture Description Languages

Software architecture development is an approach to specify software systems of important size and complexity. It focuses on the structure of the overall system rather than on the choice of computation algorithms. An architecture is defined in terms of a collection of components and the interactions among those components. According to [Bass 98], the software architecture of a system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.

Software architecture descriptions are formalized via dedicated languages called Architecture Description Languages (ADLs). ADLs provide a rigorous description of the system. This description could be shared and understood without ambiguities by its authors, readers and tools. Ideal ADLs should support composition, abstraction, refinement, architecture styles, reusability, validation, etc. Thanks to their precise notations and rules, the creation of tools to assist the development process is made easier. However, their formal aspect makes them complex and difficult to use for non-specialists.

A large number and variety of ADLs are emerging from industrial and mostly academic research groups. An ADL can be general or dedicated to one or more particular types of application, and can provide dedicated validation tools. One ADL has been developed at the SRI SDL, called SADL (Structural Architecture Description Language, [Moriconi 97]). SADL provides precise notations for describing mainly the structural properties of the software systems. Its interesting features include support for explicit mappings between architectures, generic architectures, architecture styles, and architecture patterns allowing refinement and abstraction. In terms of semantics, a SADL architecture denotes a logical theory in an extended first-order logic that includes numerical quantifiers (ω-logic). Thanks to this theoretical foundation, facilities are provided to check the consistency between different hierarchy levels ([Moriconi 95] and [Riemensch 99]). SADL has been used to specify and analyze real-world architectures, such as the X/Open Distributed Transaction Processing reference architecture and secure extensions [Gilham 99].

3. Unified Modeling Language

Unified Modeling Language (UML, [Booch 99]) is a graphical language for modeling software systems. UML is not dedicated to a particular type of application, but remains general enough to model various kinds of applications. However, it provides mechanisms to extend the language to address particular issues in software development. UML unifies notations of several popular design methods (Booch, OMT, OOSE, Fusion, etc.), and thus represents a collection of best engineering practices. It has the advantage of being widely used by practitioners and researchers, and it is supported by many tools. A number of companies are incorporating UML as a standard into their development process and products. UML partners include Hewlett-Packard, IBM, Microsoft, Oracle, and others.

UML can be employed at the requirement, architecture and design phases of the development process. At the architecture phase, UML allows us to model the system from five different views (use cases, design, process, implementation, deployment). Each view focuses on a particular aspect of the system, and a set of UML diagrams is proposed to model the static and dynamic aspects of the view. These five views can be considered individually or in relation to one another (e.g. nodes in the deployment view hold components in the implementation view that, in turn, represent the physical realization of classes, interfaces and collaborations from the design and process views). This abundance of modeling capabilities shows the power of UML for modeling software architecture. However, this abundance could make UML confusing and difficult to use for non-specialists. This confusing aspect of UML is enforced by a lack of formal semantics. In addition, UML is less expressive than ADLs. For instance, UML does not distinguish components and connectors, and does not have a built-in notation of architecture style constraints.

The UML semantics is defined by UML meta-models, formal notations and informal English. It is admitted by many authors that the UML semantics lacks of precision. According to [Clark 98], although much of the syntax of the language has been defined, and some static semantics given, dynamic semantics are mostly described using lengthy paragraphs of often ambiguous informal English, or are missing entirely. Furthermore, little consideration has been paid to important issues such as proof, compositionality, and rigorous tool support. To overcome this semantics problem, several researches are conducted by different groups, like the *Precise UML group* (PUML) which attempts to give a formal semantics to UML concepts in Z [Evans 98]. Our works will be inspired by the PUML research, but it will take place in the context of architecture modeling.

4. Integrating UML with an ADL

The integration of UML with an ADL, for instance SADL, will be achieved in two steps. The first step will provide a two-way syntactic mapping between SADL and a subset of UML. This UML subset will be chosen to cover most of the SADL constructs. The second step will give a precise semantics to the selected UML subset. This semantics will supply validation support for the UML subset. The integration of UML with SADL and its benefits are illustrated in Figure 1.

The two-way syntactic mapping between SADL and the UML subset will be established by providing a method to translate SADL constructs into UML constructs and vice versa. For this purpose, we will take advantages of the SADL mapping capabilities. Once a mapping has been defined, patterns could be proposed to allow a syntactic translation of a given architecture style. This two-way syntactic mapping will enable SADL users to utilize UML as input and/or output notation, and to use general-purpose UML tools for syntactic treatment (e.g. code generation, reverse engineering). However, to enable UML users to utilize SADL validation capabilities (e.g. preservation of architecture correctness, deadlock detection, proofs), a precise semantics must be given to the UML subset.

A precise semantics can be given to the UML subset by describing in SADL (i.e. in ω-logic) not all the UML concepts, but the concepts encompassed in the selected UML subset (e.g. types, instances, associations, hierarchy, etc.).

The two-way syntactic mapping and the precise semantics will make possible the following process:

- Specification of a given architecture using a subset of UML.
- Translation of this architecture into a SADL architecture via the 'UML to SADL' syntactic mapping (possible specific information may be added).
- Verification of the resulting architecture using SADL tools.
- Translation of this architecture again into UML via the 'SADL to UML' syntactic mapping. Thanks to the precise UML semantics, the resulting UML architecture is guaranteed to satisfy the same properties as its corresponding SADL architecture.

This will be a new and elegant way to provide verification support for UML.

Some research has already been done by the ADL community to extend UML to have the capabilities of an ADL [Robbins 97]. However, the integration of UML with an ADL has not been satisfactorily studied yet.

5. Conclusion

This paper has presented our approach to bring languages and tools for specification and validation of software architectures into wider use. This approach is based on a two-way syntactic mapping between SADL and UML, and on a UML precise semantics. It will provide UML with a theoretical foundation for architecture modeling. It will also enable ADL users to utilize general-purpose UML tools, and will enable UML users to utilize ADL validation capabilities. We believe that the result will be a rigorous software development process which is currently lacking.

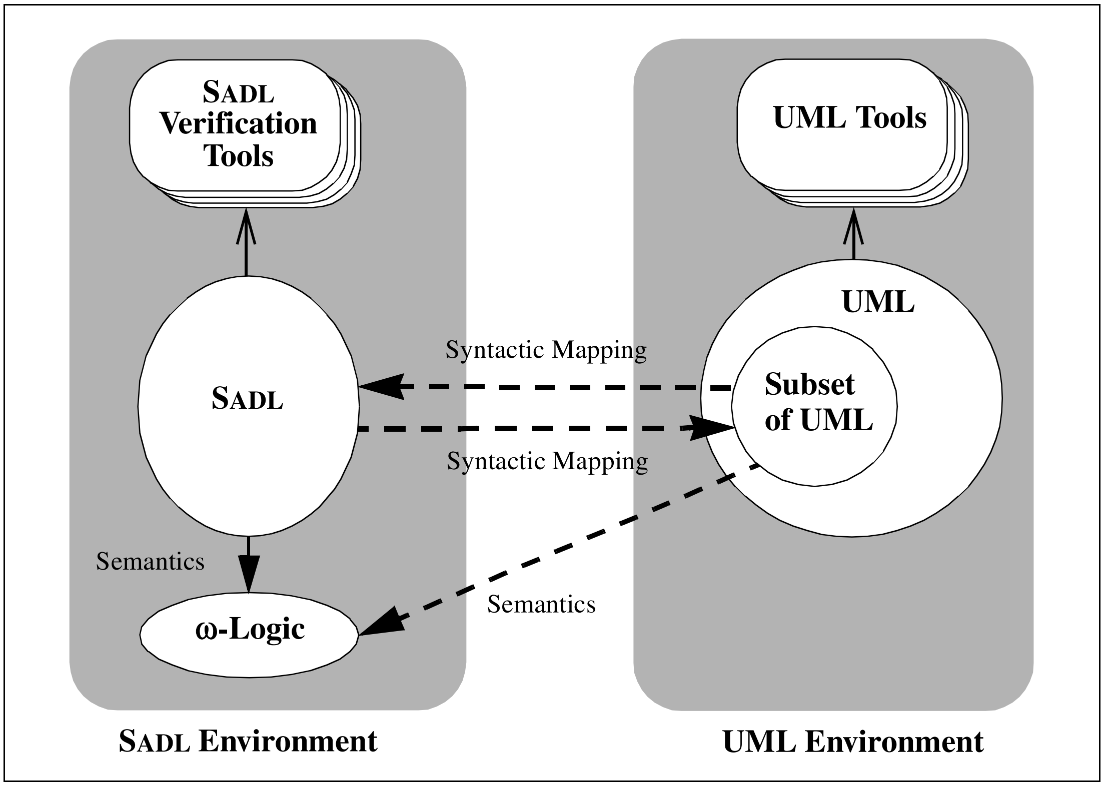

Fig. 1. Integration of UML with SADL

6. References

[Bass 98] L. Bass, P. Clements and R. Kazman. *Software Architecture in Practice*. SEI Series in Software Engineering. Addison-Wesley, 1998.

[Booch 99] G. Booch, J. Rumbaugh and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

[Clark 98] A.N. Clark and A.S. Evans. Foundations of the Unified Modeling Language. In 2nd *Northern Formal Methods Workshop, Ilkey, Electronic Workshop and Computing*, Springer-Verlag, 1998.

[Evans 98] A.S. Evans, R.B. France, K.C. Lano and B. Rumpe. The UML as a Formal Modeling Notation, In *UML'98 - Beyond the Notation*, LNCS, Springer, 1998.

[Gilham 99] F. Gilham, R.A. Riemenschneider, and V. Stavridou. Secure Interoperability of Secure Distributed Databases: An Architecture Verification Case Study, *FM'99, World Congress on Formal Methods*, Toulouse, France, September 20-24, 1999.

[Moriconi 95] M. Moriconi, X. Qian, and R. A. Riemenschneider. Correct Architecture Refinement, *IEEE Transactions on Software Engineering*, volume 21, number 4, pages 356-372, April 1995.

[Moriconi 97] M. Moriconi and R.A. Riemenschneider. Introduction to SADL 1.0. Technical Report 97-01, SRI Computer Science Laboratory, March 1997.

[Riemensch 99] R.A. Riemenschneider. Checking the Correctness of Architectural Transformation Steps via Proof-Carrying Architectures, In *Proceedings of the First Working IFIP Conference on Software Architecture*, San Antonio, Texas, February 22-24, 1999.

[Robbins 97] J.E. Robbins, N. Medvidovic, D.F. Redmiles, and D.S. Rosenblum. Integrating architecture description languages with a standard design method. 1997.
http://www.ics.uci.edu/pub/arch/uml/research/.