UML as an Architecture Description Language

B.Bharathi¹, Dr.D.Sridharan²
¹Research Scholar, Sathyabama University, Chennai, India
E-mail: bharathi_cse@sathyabamauniv.ac.in
²Department of VLSI, Anna University, Chennai, India
E-mail: sridhar@annauniv.edu

Abstract. Architecture Description Languages (ADLs) are specialized formal languages supporting modeling and reasoning on software architectures. Although number of ADLs counts in the tens, their popularity and usage by practitioners is very low. The object-oriented Unified Modeling Language (UML), which has become the OMG standard, offers a great variety of concepts for the definition of the structure and the expected behavior of a software system. Unified Modeling Language is de facto industrial standard, however not fully qualified ADL. It has the potential to replace many previously used software architecture description language. Compared with other ADLs, UML has the main drawback that its module concept is continuously changing from version to version without reaching a well-defined state. It is the purpose of this contribution to revisit the development of the UML module concept, to criticize its current form, and to present a compact and precise definition of its visibility rules.

I. INTRODUCTION

Software architecture research is directed at reducing costs of developing applications and increasing the potential for commonality between different members of a closely related product family. Software development based on common architectural idioms has its focus shifted from lines-of-code to coarser-grained architectural elements (software component and connectors) and their overall interconnection structure. To support architecture-based development, formal modeling notations and analysis and development tools that operate on architectural specifications are needed. Architecture Description Languages (ADLs) and their accompanying toolsets have been proposed as the answer. However, as well as there is no common single definition of software architecture, there is no consensus upon what an ADL is. Loosely defined, “an ADL for software application focuses on the high-level structure of the overall application rather than the implementation details of any specific source module”.

In this paper, we would like to briefly summarize current considerations on ADLs and provide overview on recent approximations of UML to ADL.

This paper is organized as follows. Section 2 introduces ADLs; Section 3 discusses UML as language capable to describe and model software architecture; Section 4 deals with contributions for the development of UML as a powerful tool for modeling of and reasoning software architectures emphasizing profiling UML; and finally, Section 5 concludes the paper.

II. ARCHITECTURE DESCRIPTION LANGUAGE

To obtain the benefits of an explicit architectural focus, software architecture must be provided with its own body of specification languages and analysis techniques. Languages are needed to define and analyze properties of a system upstream in its development, thus minimizing the costs of detecting and removing errors.

Architecture Description Language (ADL) is defined as “a language (graphical, textual, or both) for describing a software system in terms of its architectural elements and the relationship among them”.

In other words, ADL is a language enabling formalization, description, specification, modeling and reasoning on software architectures. Each of these features should be fulfilled by a language that is proclaimed to be ADL. A good ADL must provide abstractions that are adequate for modeling a large system. Each ADL embodies a particular
approach to the specification and evolution of architecture.

III. UML

Unified Modeling Language (UML)\(^{[5]}\) is formal graphical language considered a de facto industrial stand. Although the language has been created as graphical language firstly to support object-oriented software analysis and design, the language has been revised couple of times and today, it is a general formal language capable to describe a software system. The UML has well defined formal syntax and semantics and can be machine checked and processed. UML includes a set of graphical notation techniques to create abstract models of specific systems.

A. Development toward UML 2.0

UML has matured significantly since UML 1.1. Several minor revisions (UML 1.3, 1.4, and 1.5) fixed shortcomings and bugs with the first version of UML, followed by the UML 2.0 major revision that was adopted by the OMG in 2005. There are four parts to the UML 2.x specification: the Superstructure that defines the notation and semantics for diagrams and their model elements; the Infrastructure that defines the core metamodel on which the Superstructure is based; the Object Constraint Language (OCL) for defining rules for model elements; and the UML Diagram Interchange that defines how UML 2 diagram layouts are exchanged. The current versions of these standards follow: UML Superstructure version 2.1.2, UML Infrastructure version 2.1.2, OCL version 2.0, and UML Diagram Interchange version 1.0.

IV. UML AND SOFTWARE ARCHITECTURES

The significance of UML as widespread language supported by industry has been attracting research attention for a long time\(^{[3],[4] and [1]}\). However, plain UML has never been identified strong enough to be considered ADL. The position of plain UML is fairly the position of documenting language.

The expressive power of Architectures by UML is more than any ADL. UML provides large, useful and extensible set of predefined constructs, is semiformally defined, has the potential for substantial tool support, is based on experience with mainstream development methods. UML explicitly enumerates the domain modeling specification of an architecture unlike other ADLs. Requirements for evaluating suitability of UML as ADL are

- UML clearly states the configuration and topology and therefore well suited to model structural concerns.
- Captures a variety of stylistic issues (explicitly or implicitly). Provides design vocabulary for recurring topologies and generic system behaviour.
- Model software connectors and component interaction paradigms.
- Capture constraints arising from systems’ structure, behaviour, interaction and styles.

<table>
<thead>
<tr>
<th>Types</th>
<th>Instance</th>
<th>Using diagram notation</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>Objects</td>
<td>Class diagram</td>
<td>The most intuitive OO mapping</td>
</tr>
<tr>
<td>Stereotypes</td>
<td>Classes</td>
<td>Class diagram</td>
<td>Stereotyping</td>
</tr>
<tr>
<td>Classes</td>
<td>Classes</td>
<td>Class diagram</td>
<td>Homogeneous view</td>
</tr>
<tr>
<td>Stereotypes</td>
<td>Components</td>
<td>Component diagram</td>
<td>The ADL approximation</td>
</tr>
<tr>
<td>Subsystems/ packages</td>
<td>packages</td>
<td>Component or deployment diagram</td>
<td></td>
</tr>
</tbody>
</table>

C. UML as is

Omitting formalization brings problems to semantics of graphical notation. Figure 1. which shows a set of classes drawn from a system that manages the assignment of students and instructors to courses in a university. The most important problem is the usage of arrows. However, using arrows in association is strongly intended to be about navigability in the model. Hence there is no general semantics on responsibilities, time dependencies or dataflow. An association specifies a semantic relationship that can occur between typed instances. Any attempt to use hidden semantics in connectors slightly represented by associations is misusing UML but prior formal definition. Fig 1: semantics of association

Courseschedule ----> course
Add(c:Course) ----> Iterator
D. UML constrained – UML + OCL

Object Constraint Language (OCL)[6] is a formal language based on Metaobject Facility (MOF), which is an OMG model for accessing UML model structure, first-order logic and set theory. OCL is integral part of UML. However, the most industrial UML tools still does not support OCL more than allowing putting down a text. Embedding OCL with UML provides extra syntactical and semantic limitations making the model clearer and more complicated. OCL is a way of definition of stereotypes. A stereotype extends the vocabulary of UML allowing you to create new kinds of building blocks that are derived from existing ones but that are specific to the problem. Fig 2. models exceptions of models as classes.

D. Profiling UML

The UML profile for scheduling, performance, and time specification was described in [1] has been adopted as an official OMG standard in March 2002. In general, UML profile defines a domain-specific interpretation of UML; it might be viewed as a package of specializations of general UML concepts that capture domain-specific variations and usage patterns. Additional semantic constraints introduced by the UML profile must conform to standard UML semantics. To specify a profile, UML extensibility mechanisms (i.e., stereotypes, tagged values, constraints) are used [2].

The main aims of the UML profile for scheduling, Performance and Time (Real-time UML standard) are to identify the requirements for enabling performance and scheduling analysis of UML models. It defines standard methods to model physical time, timing specifications, timing services and logical and physical resources, concurrency and scheduling, software and hardware infrastructure, and their mapping. Hence, it provides the ability to specify quantitative information directly in UML models allowing quantitative analysis and predictive modeling. This profile has been defined to facilitate the use of analysis methods and to automate the generation of analysis models and of the analysis process itself.

V. CONCLUSIONS

UML is widely accepted language by practitioners. UML is capable to describe and model even software architectures. The most promising way of mapping software architectures to UML is using profiles on UML such that those profiles are derived as mappings of ADLs. This approach leads to fully sound modeling language capable to cover all requirements on ADLs. Recent versions of UML, in addition, reduce the problem area of profiles. UML2.0 and later versions are now almost software architectures conformant.

UML profile for schedulability, performance and time specification defines a domain specific interpretation of UML. It does not give the advantage of direct mapping to a performance model for architecture evaluation. It has to be increased to another level of extended UML diagrams annotated with performance information out of purely functional oriented UML diagrams.

REFERENCES