Extending Architectural Representation in UML with View Integration

Alexander Egyed and Nenad Medvidovic
Center for Software Engineering
University of Southern California
Los Angeles, CA 90089-0781, USA
{aegyed,neno}@sunset.usc.edu

Abstract. UML has established itself as the leading OO analysis and design methodology. Recently, it has also been increasingly used as a foundation for representing numerous (diagrammatic) views that are outside the standardized set of UML views. An example are architecture description languages. The main advantages of representing other types of views in UML are 1) a common data model and 2) a common set of tools that can be used to manipulate that model. However, attempts at representing additional views in UML usually fall short of their full integration with existing views. Integration extends representation by also describing interactions among multiple views, thus capturing the inter-view relationships. Those inter-view relationships are essential to enable automated identification of consistency and conformance mismatches. This work describes a view integration framework and demonstrates how an architecture description language, which was previously only represented in UML, can now be fully integrated into UML.

1 Introduction

Software systems are characterized by unprecedented complexity. One effective means of dealing with that complexity is to consider a system from a particular perspective, or view. Views enable software developers to reduce the amount of information they have to deal with at any given time. It has been recognized that “it is not the number of details, as such, that contributes to complexity, but the number of details of which we have to be aware at the same time.” [1].

A major drawback of describing systems as collections of views is that the software development process tends to become rather view centric (seeing views instead of the big picture). Such a view centric approach exhibits a fair amount of redundancy across different views as a side effect. That redundancy is the cause for inter-view mismatches, such as inconsistencies or incompletenesses. On top of that, views are used independently, concurrently, are subjected to different audiences (interpretations) and the manner in which model information is shared is extremely inconsistent. All this implies that information about a system must be captured multiple times and must be kept consistent.
To deal with this problem, a major emphasis needs to be placed on mismatch identification and reconciliation within and among views (view integration). We design not only because we want to build (compose) but also because we want to understand. Thus, a major focus of software development is to analyze and verify the conceptual integrity, consistency, and completeness of the model of a system.

There are numerous reasons for the lack of automated assistance in identifying view mismatches. We believe that one of the major reasons is improper integration of views on a meta-level: although both the notation and semantics of individual views may be very well defined, meta-models that integrate these different views are often inadequate or missing. The Unified Modeling Language (UML) [2] is a good example of this. UML defines a set of views (such as class diagrams, sequence diagrams, and state diagrams) and defines a meta-model for those views. However, the UML meta-model was primarily designed to deal with the issue of capturing and representing modeling elements of views in a common data model (repository).

Another problem with UML is that its views had to be designed to be generally understandable and they are therefore rather simple. The result is that UML view semantics are not well defined so as not to over-constrain the language or limit its usability. Therefore, UML becomes less suitable in domains were more precision (performance, reliability, etc.) is required. One way of addressing this deficiency is the use of additional views. Figure 1 shows an example on how a general purpose development methodology such as UML may be used together with more (domain) specific description languages such as architecture description languages (ADLs). What we mean by architecture is a course grain description of a system with high-level components, connectors, and their configuration. A design further refines the architecture by elaborating on the details of individual components/connectors as well as their interactions. In Figure 1, UML serves as a general development model, whereas more specific views can be generated by taking excursions off the main process to investigate specific concerns, e.g., deadlock detection among modeling