Available online at www.sciencedirect.com
SCIENCE DIRECT®
The Journal of Systems and Software 79 (2006) 1180–1206
The Journal of Systems and Software
www.elsevier.com/locate/jss

AVDL: A highly adaptable architecture view description language

Jungwoo Ryoo [a], Hossein Saiedian [b,*]

[a] Information Sciences and Technology, Penn State Altoona, Altoona, PA 16601, United States
[b] Department of EECS, University of Kansas, Lawrence, KS 66045, United States

Received 1 November 2004; received in revised form 24 February 2006; accepted 25 February 2006
Available online 18 April 2006

Abstract

Architectural views are rapidly gaining a momentum as a vehicle to document and analyze software architectures. Despite their popularity, there is no dedicated language flexible enough to support the specifications of an unbound variety of views including those preexisting and needing to be newly created on demand. In this paper, we propose a novel view description language intended for specifying any arbitrary views, using a uniform set of conventions for constructing views and how to use them. The highly adaptable nature of the new language results from its built-in mechanisms to define different types of views in a systematic and repeatable manner.

Keywords: Software architecture; Views; Viewpoints; Specifications; Meta object facility; Object constraint language; Extensible markup language; Schema; Meta-data interchange

1. Introduction

Software applications implementing non-trivial requirements are often so complex that they can be incomprehensible in their entirety. The IEEE recommended practice for architectural description (IEEE, 2000) defines software architecture as "the fundamental organization of a system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution". One can adopt this notion of architecture and alleviate both perceived and real complexity of software applications by describing them in terms of less granular and more abstract concepts such as components and their relationships. However, much of the original complexity often remains even in an architectural specification. To overcome this problem, many researchers (Perry and Wolf, 1992; Kruchten, 1995; Issarny et al., 1998; IEEE, 2000; Baragry and Reed, 2001; Clements et al., 2003; Wijnstra, 2003; Purhonen et al., 2004) advocate partitioning an architectural description into more than one view, which is "a representation of a whole system from the perspective of a related set of concerns (IEEE, 2000)". For example, inspired by the field of building architecture, Perry and Wolf state that "a software architect needs a number of different views of an architecture for the various uses and users (Perry and Wolf, 1992)".

Typically, there are *many* people and organizations interested in the successful construction of a software system. These are referred to as *stakeholders*. Examples of stakeholders include customers, end users, developers, customer care organizations, etc. Views not only reduce the complexity of architectural description but are also vehicles for separation of concerns by providing a highly specialized specification especially meaningful for a certain class of stakeholders.

The idea of views is further developed by the introduction of a higher-level abstraction referred to as *viewpoint*. "A viewpoint is a specification of conventions for constructing and using a view (IEEE, 2000)". In other words, it is a *pattern* or *template* from which to develop individual views by establishing the purposes and audiences for a view and the techniques

* Corresponding author. Tel.: +1 913 897 8515; fax: +1 913 897 8682.
E-mail addresses: jryoo@psu.edu (J. Ryoo), saiedian@eecs.ku.edu (H. Saiedian).

0164-1212/$ - see front matter
doi:10.1016/j.jss.2006.02.045

for its creation and analysis. The idea of viewpoints is not new and has been used in various contexts of software engineering. For instance, in their requirements engineering research, Nuseibeh and Easterbrook (2000) use the term viewpoint to refer to "loosely coupled, locally managed, distributable objects capturing partial representational knowledge, development process knowledge, and specification knowledge about a system and its domain".

From these definitions one can conclude that (1) a view conforms to a viewpoint and that (2) a complete viewpoint specification requires, at a minimum, information about:

- the **process** governing the life cycle (creation, description, and analysis) of views instantiated from a viewpoint,
- the **style**: defining models and notations used by views belonging to a viewpoint, and
- the **domain**: describing the area of concerns addressed by a viewpoint and its instances.

Making a clear distinction between views and viewpoints is a desirable practice recommended by IEEE (2000). However, the term *viewpoint* is sometimes used interchangeably with *view* as in Kruchten (1995), Hofmeister et al. (1999), Herzum and Sims (1999). Clements et al. (2003) use the term *view style* in place of viewpoint.

Although solving the problem of complexity and satisfying the individualized concerns, the use of views poses its own set of challenges. One of the most significant among them is meeting the need for defining different types of views that may be either preexisting or needing to be created on-demand.

For instance, IEEE Std 1471-2000 (IEEE, 2000) provides basic requirements for views and viewpoints but falls short of establishing a consistent and uniform specification method.

Initially, researchers (especially, Kruchten, 1995) harbored a hope that there might exist an ultimate set of views addressing all the possible concerns of any arbitrary stakeholders. After trying these supposedly all-purpose views for real-life problems, practitioners are coming to a conclusion that no set of views are comprehensive enough to completely defy the necessity of introducing new kinds of views. The special requirements of unexplored domains prompted sporadic proposals (Purhonen et al., 2004) for a new set of highly specialized views, but the practice of describing views largely remains ad hoc.

Some of the consequences of this lack of a standardized and unified view specification mechanism include:

- difficulties in learning how to describe, discern, and refine well-known views due to the idiosyncrasies inherent to each different type of view,
- overhead associated with creating a new type of view since one must each time determine a specification method for it first, and
- incompatible view specifications that are not amenable to a formal analysis, which is due to employing disparate specification methods.

In this paper, we propose a novel language devoted to describing architectural views, which is deliberately designed to accommodate the need for defining and specifying different types of views in a systematic and repeatable manner. For the remainder of our discussion, the new language is referred to as AVDL (Architecture View Description Language). AVDL is based on an observation that a meta-relationship between any two views can be regarded as that of refinement. We show that the constraints governing the refinement relationships can be encoded into the metamodel of AVDL and hold the key to making it highly adaptable.

The organization of this paper is as follows. Section 2 lays the groundwork for our discussion of AVDL. Section 3 formally defines AVDL in terms of its metamodel and notations (both textual and graphical). Section 4 provides several concrete examples demonstrating how AVDL can be applied to a real-life problem. Section 5 discusses a case study we conducted to verify the practicality of our approach. We present our work in the context of related research in Section 6. Finally, we round out our paper with concluding remarks and further research.

2. Background

In order to understand the ensuing sections of this paper, it is critical to have some basic understanding on the languages and facilities such as UML, MOF, XML, and XMI. In this section we provide overviews on each of them.

2.1. UML

Unified modeling language (UML) (Object Management Group, 2003) is a visual modeling language and has a semi-formal syntax and semantics. It is intended to allow its users (mostly practitioners) to capture their design decisions in a group of easy-to-use diagrams when developing software applications. Each UML diagram addresses issues associated with a certain aspect of a software system under development. For example, a class diagram identifies classes (including their attributes and operations) and their relationships. A state diagram shows possible states in which a class can be