Viewpoint Modeling*

Rich Hilliard
ConsentCache Inc.
rh@ConsentCache.com

March 14, 2001

Abstract

If architecture representation is the problem (P), and the UML is the solution (S), how do we get from P to S? In this paper, I introduce the idea of *viewpoint modeling* as a part of the answer. A viewpoint, as defined in IEEE 1471, defines a set of architectural concerns and the resources needed to address those concerns. A viewpoint gives an architect the resources with which to construct an architectural view. These resouces may include notations, techniques, and guidance. Viewpoint modeling is then a kind of metamodeling to bundle up useful sets of modeling resources for the architect. The main result of this paper is a proposed template for documenting viewpoints. I then conclude with some observations relating this to the UML.

Introduction

Most modern architecting techniques [1, 2, 8, 10] start from a notion of multiple representations, i.e., *architectural views*, as a fundamental, organizing principle of *architectural descriptions* (ADs): those work products used to document the architecture of a software-intensive system.

There is a large menagerie of views out there in industry, academia and, sometimes, even in use. They are defined to varying degrees of rigor and they overlap with each other in intent (*Do I need a module view and a code view?*); if you are an architect, it may be hard to tell which to use in a given situation (*Do I need a logical view or a components-and-connectors view or both?*).

The recently published IEEE 1471, *Recommended Practice for Architectural Description of Software-Intensive Systems* [9], makes a small attempt to bring some order to this chaos. IEEE 1471 is the first formal standard to address what is an architectural description. It standardizes a set of "content requirements" on an architectural description: what an AD should include and how that content should be organized.

IEEE 1471's approach to dealing with this chaos is *not* to pick out yet another set of views and require their use to define compliance with IEEE 1471, but to introduce a level of indirection.[1] IEEE 1471 introduces the notion of an *architectural viewpoint* to capture what characteristics that should be documented about any view. A viewpoint, as defined in IEEE 1471, defines a set of architectural concerns and the resources needed to address those concerns. These resouces may include notations, techniques, and guidance. A viewpoint gives an architect the resources with which to construct an architectural view.

IEEE 1471 also requires that viewpoints be "declared" before being used in a view. The philosophy is: If you give me a map; make sure you give me the legend (set of conventions for reading the map), too.

IEEE 1471 establishes only the simplest requirements on declaring viewpoints, leaving much to the architect; they take up only a paragraph in the standard.

*Submission to 1st ICSE Workshop on Describing Software Architecture with UML.

[1] It has been said, "Any problem in Computer Science can be solved with another level of indirection." (Butler W. Lampson)

The purpose of this paper is to propose a more refined approach to viewpoints which meets the requirements of IEEE 1471 and offers practical guidance to architects and other viewpoint writers. I call this *viewpoint modeling*.

Viewpoint modeling is a kind of meta-modeling, whose result is a way to make models (views) of a certain kind. I hope it can help us to more rigorously specify what we can do in addressing architectural representation for specific purposes, hence the connection to the UML (see **Discussion**).

Viewpoint modeling is not *view modeling* (nor viewpoint-based modeling)—architects do that after a viewpoint has been defined (either overtly or implicitly).

The Viewpoint Template

In this section I propose a template for recording viewpoints, which meets the requirements of IEEE 1471 and provides some further guidance. It is more fine-grained than what IEEE 1471 requires.

The template starts here. Each "slot" has a name, a brief description of its intended content, and guidance toward developing that content. A running example is briefly sketched. The example develops a Commerce Viewpoint that I have found useful in developing web-based e-commerce systems, inspired by recent work of Gordijn and van Vliet [4] and Hauswirth, Jazayeri and Schneider [5].

Name

The unique name for the viewpoint.

Example: Commerce Viewpoint (also known as the business viewpoint, or business models at an architectural level).

Abstract

A brief overview of the viewpoint and its salient features.

Example: The Commerce Viewpoint provides a way of addressing business models at the architectural level, in terms of actors (participants) and value exchanges among those actors.

Concerns

Concerns are the architectural issues which this viewpoint is capable of addressing. This is the most important information for the architect, because it helps her decide if this viewpoint will be useful for a particular system. I have found it useful to state concerns in the form of questions (like the television show *Jeopardy*!) that the resulting view should be able to answer.

Example: The Commerce Viewpoint addresses:

- What business models does the system support?
- What kinds of transactions are allowed?
- Who participates in the transactions, and what are their roles?
- What transaction-related services are to be provided?

Typical Stakeholders

Optional. Who are the usual stakeholders for this kind of view? What are the typical audiences for views prepared using this viewpoint? Within an AD, when the viewpoint has been selected for use, it is then elaborated[2]. The viewpoint is elaborated with the actual stakeholders from the system of interest who have concerns for this viewpoint. They should be recorded in the AD.

Example: Typical stakeholders for this viewpoint are: the client, business analysts, security analysts, developers, and user interface designers.

Anti-Concerns

Optional. The kinds of issues this viewpoint is **not** good for. Articulating these may be a good antidote for certain overly used notations.

Viewpoint Language

The viewpoint language is the key modeling resource that the viewpoint makes available for constructing the view models.

IEEE 1471 requires its users to specify, "The language, modeling techniques, or analytical methods to be used in constructing a view based upon the viewpoint". I find it is useful to separate these into the language (dealt with here) and treat modeling techniques and analytical methods as operations on views (below).

I have also found it useful to separate the consideration of the viewpoint language into its core concepts and their syntactic realizations. The abstract syntax, or conceptual framework of the viewpoint, I call its *ontology* and then separately describe notations that 'satisfy' that ontology.

Viewpoint Ontology

The *ontology* is the set of conceptual entities, their properties and relations that comprise the vocabulary of views constructed with this viewpoint.[3]

There are different ways of representing ontologies. For my purposes, a UML class diagram is just fine; I can use it to capture entities, properties, relationships and any constraints among them with classes, attributes and associations and constraints, respectively.

Elements, Attributes, and Relationships

What are the major sorts of entities present in this view?

What properties do elements in this view carry?

What relationships are defined among elements within this view?

Constraints

Most languages are not freely generated over the elements. There may be constraints on well-formedness, usually organized into a grammar of some kind. Constraints are introduced here to capture these well-formedness constraints.

These constraints should not be confused with architectural constraints that apply to the subject matter of the viewpoint. See example below.

[2] To coin a phrase, from Ada. Ada types are declared then elaborated before use. The same model seems useful for viewpoints. This template specifies the form in which viewpoints are declared. Once selected for use, the viewpoint must be elaborated for the particular AD. Finally, it is used to create the view.

[3] Although this is a pretentious, overused word, I use it because it really does convey the right sense of the universe of discourse behind a viewpoint language, abstracted away from any syntactic constructs. Besides, I am not the first to introduce it into architecture—ACME was there first: "Our approach to architectural representation is based on a generic ontology of seven entities: components, connectors, configurations, ports, roles, representations, and bindings."[3].

Example: The primary elements of the Commerce Viewpoint are **actors** (participants in exchanges such as buyers, sellers, customers, intermediaries); **values** (resources such as products, services, money); **value exchanges** (i.e., transactions of value among actors) and **business rules** (e.g., fraud should be prevented; this is an example of an architectural constraint that is not a well-formedness constraint. A well-formedness constraint, in the present example is: At least two actors participate in every value exchange.)

UML class diagram omitted here, in the interest of space limitations.

(Conforming) Notations

Identify or define notations that 'satisfy' the viewpoint ontology. I.e., give the Architect concrete notation(s) to use that, when used, yield views that manifest this viewpoint.

Sometimes, no "notation" may exist. This is a job for the UML's stereotypes, and/or mundane approaches like natural language.

Example: The UML's interaction diagrams could be appropriated to represent models in the Commerce Viewpoint where: actors are represented by objects; values are objects; value exchanges are represented by (pairs of) messages; and business rules are captured by constraints.

Operations on Views

The viewpoint language gives us the vocabulary for constructing a view, the Operations defines the actions which may be applied to views. IEEE 1471 lumps together methods for creating views, analytic techniques for analyzing views, and heuristics that might be applied to views. Here, I've taken an approach of organizing these operations into categories (à la the SmallTalk–80 browser): creation methods, interpretation methods, analysis methods and implementation methods.

Creation methods

Creation methods are the means by which views are constructed under this viewpoint. These operations could be in the form of process guidance (how to start, what to do next); or work product guidance (templates for views of this type). Creation techniques may also be heuristic: identifying styles, patterns, or other idioms to apply in the synthesis of the view.

Example: *Phased Style.* Hauswirth, Jazayeri and Schneider propose a "style" for constructing business models they call a phase model [5]. It classifies the actors and exchanges into phases (such as Advertising, Negotiation, Ordering, Payment, and Delivery) to facilitate certain kinds of analysis. They use this to reason about security threats to the business. To support this style, we can extend the viewpoint ontology by adding a class (Phase), such that every value exchange falls into a phase.

Interpretive methods

Interpretive methods are the means by which views to be understood as making architectural statements and commitments.

Example: Actors in the Commerce Viewpoint should be interpreted as people, other systems, or subsystems of the system of interest. Value exchanges should be interpreted as n-party transactions.

Analysis methods

Analysis methods are operations which may be used to check, reason about, transform, predict, and evaluate architectural results from this view. This could include traceability methods with respect to other viewpoints.

Example: *Fault-tree analysis.* Gordijn and van Vliet show how this viewpoint may be used to drive fault-tree analysis to detect possible kinds of fraud and their causes, that a business model may be susceptible to.

Implementation methods

Implementation techniques describe how to design/build systems from this view.

Notes

Optional. Any additional information that may be of use to architects in selecting, customizing or using this viewpoint.

Source

What is the source (author, history, literature references) for this viewpoint, if any?

Example: The Commerce Viewpoint draws heavily upon the insights of: Gordijn and van Vliet [4] and Hauswirth, Jazayeri and Schneider [5].

Discussion

The goal of IEEE 1471's notion of viewpoint is to make the conventions by which architectures are modeled clearer. This includes making: clearer to the readers of an AD (*Maps should have legends*); and clearer to architects seeking to select, or mix, or match viewpoints for use (*Help find the right tool for the job*). Viewpoints codify one kind of reusable architectural asset that should be available to all. The long-term goal is to create a palette of viewpoints for architects to draw from.

There are numerous architectural methods in the literature which could be documented as architectural viewpoints—although they are typically not called that today. Perhaps the most studied is the "structural viewpoint," built on the ontology of components and connectors cited above.

The template I have proposed is "object-oriented" as follows: the viewpoint language defines what it means to be a well-formed view; the operations on views define techniques for creating, interpreting, analyzing views. In requirements engineering, Nuseibeh, Kramer and Finkelstein capture a viewpoint in terms of an object-oriented scheme as well [11]).

Implications for the UML. Now let's return to the original question of this position paper:

> If architecture representation is the problem (P), and the UML is the solution (S), how do we get from P to S?

Now I can answer: for each representation problem (i.e., set of concerns), define a viewpoint with its viewpoint language and ontology, and associated techniques, and then craft a suitable conforming notation from the UML, using the extensibility mechanisms.

Contrast this with current thinking about that question, which seems to be to create an architectural profile for the UML. A *profile* is a specialization of the UML metamodel for a particular domain. Various profiles have been, or are being, defined. However, a specialization approach may be too restrictive for architectural representation. There is no simple subset of the UML concepts (metamodel elements?) that can be called architectural; so the idea of a profile is probably simplistic to address architectural use of UML. In such cases, it seems a profile could not be used since a profile would not permit multiple specializations of the same element.

A viewpoint is like a profile, but lighter: for a given job an architect may need to select one or more viewpoints; each of which specialise elements of the metamodel differently. I tried my hand at integrating this into the UML metamodel in earlier work [6, 7].

References

[1] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison Wesley, 1998.

[2] D. E. Emery, R. Hilliard, and T. B. Rice. Experiences applying a practical architectural method. In A. Strohmeier, editor, *Reliable Software Technologies–Ada-Europe '96*, number 1088 in Lecture Notes in Computer Science. Springer, 1996.

[3] D. Garlan, R. Allen, and J. Ockerbloom. Exploiting style in architectural design environments. In *Proceedings of SIGSOFT'94, Foundations of Software Engineering*. ACM Press, December 1994.

[4] J. Gordijn and H. van Vliet. On the interaction between business models and software architecture in electronic commerce. Case study presented at ESEC/FSE'99.

[5] M. Hauswirth, M. Jazayeri, and M. Schneider. A phase model for e-commerce business models and its application to security assessment. In *Proceedings of the Hawaii International Conference on System Sciences*, January 2001.

[6] R. Hilliard. Building blocks for extensibility in the UML. Technical report, Object Management Group, 1999. Response to Object Management Group's Request for Information for Unified Modeling Language 2.0. Available from OMG as: ad/99-12-12. See also: ad/99-12-13 and ad/99-12-14.

[7] R. Hilliard. Using the UML for architectural description. In R. France and B. Rumpe, editors, *≪UML≫'99 The Unified Modeling Language, Second International Conference*, volume 1723 of *Lecture Notes in Computer Science*, pages 32–48. Springer, 1999.

[8] C. Hofmeister, R. Nord, and D. Soni. *Applied Software Architecture*. Addison-Wesley, 1999.

[9] *IEEE Std 1471-2000 Recommended Practice for Architectural Description of Software-Intensive Systems*, October 2000.

[10] P. B. Kruchten. The 4+1 view model of architecture. *IEEE Software*, 28(11):42–50, November 1995.

[11] B. Nuseibeh, J. Kramer, and A. Finkelstein. A framework for expressing the relationships between multiple views in requirements specification. *IEEE Transactions on Software Engineering*, 20(10):760–773, 1994.