

# Turning UML-Subsystems into Architectural Units

Harald Störrle

Ludwig-Maximilians-Universität München

Oettingenstr. 67

81369 München

Germany

++49-89-2178-2134

stoerrle@informatik.uni-muenchen.de

## ABSTRACT

In this paper, I show how the subsystem concept of UML may be adapted for architectural modeling. Previous approaches used stereotypes of UML's class-concept instead. For greater conceptual clarity, I pursue an explicit metamodeling approach. However, the modifications I propose could be reformulated using only UML's standard extension mechanisms, if that were desired. The present approach is inspired by ROOM and the IEEE's P1471 standard. It is a follow-up work of my recent PhD-thesis.

## Keywords

UML, Subsystem, Metamodel, Architectural Modeling, ROOM, IEEE P1471

## 1 INTRODUCTION

### Background

This position paper summarizes our experiences in trying to integrate architectural modeling concepts like those of Real Time Object Oriented Modeling (ROOM) into the Unified Modeling Language (UML). In contrast to existing approaches like [8, 3, 11, 6], our approach starts from the UML, and offers a deep rather than a shallow integration. Also, our approach is based on a modification of `Subsystem` concept rather than on a stereotype of `Class`.<sup>1</sup>

This approach originates from a CASE-tool development effort in a non-industrial setting. The experiences have since been confirmed by other practical projects in industrial and academic settings. This position paper addresses mainly the second question of the Call For Papers (concerning UML extensions to cater for architectural issues), but also touches on the questions 3 through 6.

### Approach

Underlying this work is the belief in the importance in abstractly modeling software architectures. Since the 1980ies,

there have been many results in this area, e.g. Architecture Description Languages, but also industrial methods like ROOM. None of these has established itself as generally accepted, however. But since facilitating communication is one of the prime purpose of software architectures, it is vital that the medium of communication is universally understood. In the foreseeable future, this is only possible with the UML, “*the lingua franca of the software engineering community*” (cf. [15, p. v]). Against this background, all deficiencies the UML still does have are irrelevant, though in need of amelioration, which is exactly the point of this paper.

And indeed, there are some such deficiencies—see e.g. [5] for an interesting discussion for the area of architectural description. I shall focus on one particularly grave problem, and that is the lack of an adequate concept for what I like to call “architectural units” (abbreviated to AU). The extensions and modifications I shall propose are strongly influenced by ROOM [14] and IEEE P1471 [7].

## 2 CONCEPTS FOR ARCHITECTURAL MODELING

In this position paper, I shall deal only with the concept of AUs and those directly related to it, that is, *views* to structure them, *ports* to define their interfaces, and *connectors* to glue them together. A synopsis of these concepts and their relationships is provided in Figure 1.

A more detailed discussion is found in [16].

### Requirements for Architectural Modeling

In this section, we enumerate a few basic requirements that motivate why and how I define the concepts for architectural modeling the way I do.

1. First of all, observe that a system architecture should be one of the first artifacts created in the course of a development project. In order to be able to visualize an architecture, reason about it, communicate with it and treat it in other ways, there must be an abstract, design-level representation of AUs, that is, following the earlier argument, an UML representation.
2. It is also clear, that there should be no technological bias: many architectures involve legacies, which are not object-oriented, and these must be dealt with, too. So,

<sup>1</sup>Whenever we are referring to elements of the UML metamodel, we put them in the sans-serif font.

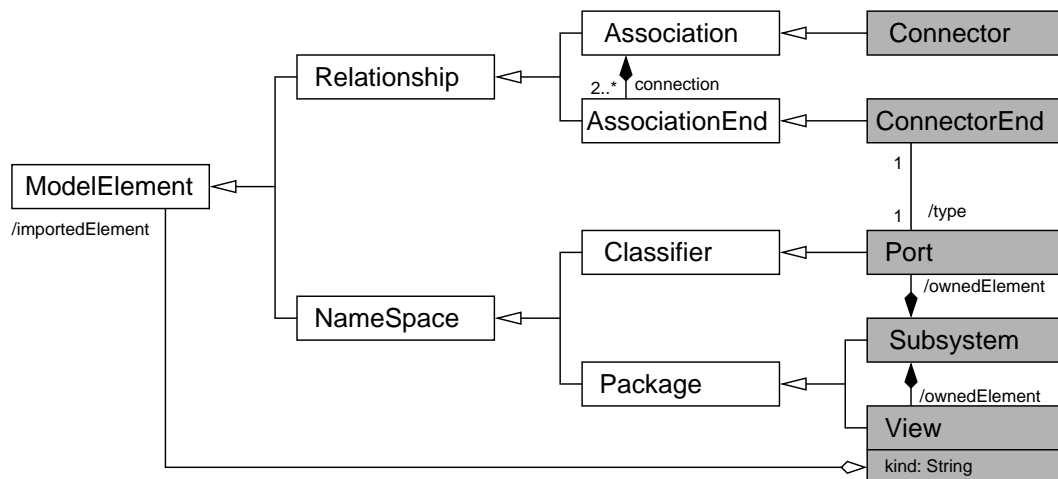


Figure 1: The concepts for architectural modeling defined here and their relationships as a UML class diagram on the metalevel  $M_2$ . Shaded boxes represent metaclasses modified or introduced in this paper. For the sake of the presentation, we have omitted some details from the UML metamodel, such as `GeneralizableElement`.

we should avoid anything specific to OO systems.

3. Then, the building blocks of an architecture, its AUs, need internal structuring, both for the separation of concerns, and as the basis for tool support. Intuition, practical experience, and empirical surveys show that this is best achieved using multiple views.
4. Next, one of the fundamental objectives of dealing with software architecture is to foster system evolution and integration of prefabricated parts. In other words, one of the important issues is that of reuse. Since reuse is the more efficient, the coarser the reused entity is, an AU must be—among other things—a conceptual entity rather than (only) a piece of code: an AU must represent an identifiable functionality that is relevant and non-trivial in the problem domain.
5. As a variation of the reuse leitmotiv, AUs have to be hardened against the ripple-effect, that is, the impact that changes to some part of a system must not affect other AUs. I propose to counter this by a particularly strong mechanism of encapsulation, that controls not only the stimuli *to* the AU, but also those *from* it.
6. Finally, applying Occam’s razor, it is clear that the modifications and extensions necessary to better support architectural modeling in the UML should be as limited as possible, and that as much as possible of the syntax and concepts of the UML should remain unchanged. This is also the best way to increase the acceptance of such modifications.

Let’s now implement these requirements by turning `Subsystems` into AUs within the conceptual framework of the UML metamodel.

### Subsystems as Architectural Units

In this situation, there is always a trade-off between compliance with the existing standard and conceptual clarity. Since this workshop is intended to generate change requests for the UML 2.0, I shall concentrate on the latter, that is, I will liberally propose changes to the subsystem-concept as it is in the UML 1.3.

It is obvious, that some concept for AUs is needed in architectural modeling, and there are some candidates already present in the UML: `Class`, `Component`, `Package`, and `Subsystem`. Let’s now consider these in turn.

- `Class` represents classes of OO-programming languages, that is, `Class` has an unwanted technological bias and represents entities of too small granularity.<sup>2</sup>
- A `Component` “*represents a physical piece of implementation of a system*” (cf. [12, p. 2-29]). A `Component`, in other words, is a piece of the implementation, not one of the design. So it lives on the wrong level of abstraction for an AU.
- A `Package`, on the other hand, may be an element of the design: it may contain any kind `ModelElement`. It is completely unstructures, however, and has only interfaces in the sense of UML, that is, it offers only weak protection against the ripple-effect.
- Finally, there is the concept of `Subsystem`, which is a subclass of `Package`. A `Subsystem` has three partitions for `Operations`, and “specification” and “implementation elements”, respectively, but no concept of view, or port (other than UML-Interfaces, that is).

<sup>2</sup>Alternatively, `Class` may represent an analysis-level type—but that is even further away from AUs.

Also, **Subsystems** may not appear in deployment diagrams, they may be mixed with objects, classes, and packages in static structure diagrams, mingled, that is, with entities of different granularity and purpose. On the other hand, they may not appear in deployment diagrams, though, this is frequently an important architectural issue. Their semantics and pragmatics is currently not adequately defined.

So, I propose to modify **Subsystem** such that the three partitions are replaced by a view-mechanism. Instead of a mere set of **Operations**, I propose to endow **Systems** with a non-empty set of **Ports**.

### **View and Viewpoint**

The IEEE P1471 distinguishes between viewpoints (generic “kinds” of views) and views (concrete “instances” thereof). Both of these are realized as metaclasses in the P1471 conceptual model. In my opinion, it is indeed important to identify and distinguish the two of these, but they live on different levels of the metamodelling hierarchy: Consider the definition of the relationship between view and viewpoint: it is said to be, somehow, an instance-relationship. But “instance” in what sense? Looking at the UML, there are at least two such notions: the kind of relationship between the metaclasses **Class** and **Object**, say (or **UseCase** and **UseCaseInstance** and so on), and on the other hand, the relationship between a class in an object-oriented programming language and the UML metaclass **Class** (or the latter and the MOF concept of **Metaclass**). For simplicity, I call these two kinds of instance-relationships within and between metamodelling levels as horizontal and vertical, respectively.

Now, I suggest that the instance-relationship between view and viewpoint is vertical rather than horizontal. So, if view is to become a concept in the UML metamodel, viewpoint should correspond to what will be called UML profile or preface - though these two concepts are not yet defined entirely satisfactorily. In other words, viewpoint should not (as the P1471 suggests) be a concept in the metamodel.

Just which ensemble of viewpoints is relevant for some development project is specific and characteristic for that project. Determining some set of viewpoints (i.e. declaring it, and only it, as relevant) is instrumental and indispensable in the requirements elicitation phase. Making it explicit is, in my experience, equally revealing for the solution domain as defining use-cases is for the problem domain.

However, there are some views that are likely to occur in very similar shapes under almost all circumstances. For these, particularly rich support in terms of tools and techniques can be provided. So, there is a tradeoff between the availability of tools, knowledge, training, experience and so on, versus the degree of fitness to the respective project. In other words, it is not (only) the problem domain that determines the ensemble of viewpoints, but also the solution domain, and to a

quite considerable degree.

So, I propose to introduce into the UML metamodel a subclass of **Package** called **View** with a string-valued attribute kind. Different **Views** present only projections of the set of **ModelElements** populating the **Subsystem**, that is, **Views** do not own **ModelElements** exclusively.

### **Port and Connector**

Concerning ports, there are the requirements to (a) shield AUs from the ripple-effect, and (b) to avoid technological bias. In UML **Subsystems**, there is no concept of port, and neither of these requirements are satisfied: the points-of-interaction of a UML subsystem are a set of **Operations**, that is, specifications of method-calls. Thus, an object oriented bias, asynchrony Also, many architectures are distributed, so that a synchronous calling mechanism (which is the way, **Operation** is defined in the UML) is inadequate. Finally, there is no provision to control the calls emanating from the AU.

So, I propose to define a new metaclass **Port** that is specified by two sets of incoming and outgoing **Signals** or **Operations**, that are further specified by a **StateMachine** of their behavior. Again, there is the question of how and where to introduce this concept within the UML metamodel. Obviously, a **Port** would be a **NameSpace**, even a **Classifier** since **Ports** are to be connected by **Connectors**. The type of a **ConnectorEnd** must be a **Port**, and **Connectors** have as their connection only **ConnectorEnds**, so that, all in all, **Ports** are only connected by **Connectors**.

## **3 Perspectives**

Based on this approach, there are at least three further issues to be considered. These refer to questions 3 through 6 in the CFP of the workshop.

### **Analysis of Models**

Finally, coming back to our initial motivation, the main purpose of using models (of software architecture) in the first place is to be able to conduct certain operations on them easier, faster or cheaper than on the real thing (that is, the concrete software architecture: the code). In particular, it is vital to support automated support for (a) checking the consistency of a large system of partial specifications, (b) analyzing performance aspects of systems, and (c) generate prototypes or conduct interactive simulations. All of these depend crucially on the definition of a generally accepted, mathematically precise definition of the semantics of UML and the desired system properties. In this area, there have been promising advances over the last years, see e.g. [13, 16, 9].

### **Reuse**

The issue of reusing architectures (e.g. in product line architectures, COTS components etc.) and of evolution of systems is directly affected by the AU concept used in an approach. So far, little field experience has been gained on this aspect in relation with UML.

## Process

Finally, there is a pressing need for process support. Since architectural units (frequently) are *reusable* units, there have to be special processes for the activities related to reuse, like finding preparing units for reuse, finding a unit matching a given profile, assessing the fitness and so on. On the other hand, the overall structure of the development process is also affected: if a system is built from components, it has a recursive, that is, fractal structure. In order to facilitate the mapping between product and process, the development process should have such a fractal structure, too. This can be achieved by process patterns (cf. [2, 17, 1]). The important bit is to precisely define the relationships between different patterns: when are they applicable, what are their results, and so on. Again, formal semantics are essential.

## REFERENCES

- [1] Scott W. Ambler. *More Process Patterns: Delivering Large-Scale Systems Using Object Technology*. Cambridge University Press, 1998.
- [2] Desmond Francis D'Souza and Alan Cameron Wills. *Objects, Components and Frameworks with UML. The Catalysis Approach*. Addison-Wesley, 1999.
- [3] Alexander Egyed and Nenad Medvidovic. Extending Architectural Representation in UML with View Integration. In France and Rumpe [4], pages 2–16.
- [4] Robert France and Bernhard Rumpe, editors. *Proc. 2<sup>nd</sup> Intl. Conf. on the Unified Modeling Language (<<UML>> 1999). Beyond the Standard*. Number 1723 in LNCS. Springer Verlag, 1999.
- [5] David Garlan and Andrew J. Kompanek. Reconciling the Needs of Architectural Description with Object-Modeling Notations. In Selic et al. [15], pages 498–512.
- [6] Christine Hofmeister, Robert L. Nord, and Dilip Soni. Describing Software Architecture in UML. pages 145–159. Kluwer Academic Publishers, 1999.
- [7] The IEEE Architecture Working Group. Draft Recommended Practice for Architectural Description, IEEE P1471/D5.1. Technical report, IEEE, October 1999. Available at [www.pitecanthropus.com/~awg](http://www.pitecanthropus.com/~awg).
- [8] Mohamed Mancona Kandé and Alfred Strohmeier. Towards a UML Profile for Software Architecture Descriptions. In Selic et al. [15], pages 513–527.
- [9] Johan Lilius and Ivàn Porres Paltor. Formalising UML State Machines for Model Checking. In France and Rumpe [4], pages 430–445. Short version of [10].
- [10] Johan Lilius and Ivàn Porres Paltor. The Semantics of UML State Machines. Technical Report TUCS Technical Report No 273, Turku Centre for Computer Science, May 1999.
- [11] Nenad Medvidovic. *Architecture-Based Specification-Time Software-Evolution*. PhD thesis, University of California, Irvine, 1999.
- [12] OMG Unified Modeling Language Specification (version 1.3). Technical report, Object Management Group, June 1998. Available at [uml.shl.com](http://uml.shl.com).
- [13] Gianna Reggio, Alexander Knapp, Bernhard Rumpe, Bran Selic, and Roel Wieringa, editors. *Dynamic Behavior in UML Models: Semantic Questions. Workshop Proceedings*, Oktober 2000. Also appeared as Technical Report No. 0006 of the Ludwig-Maximilians-Universität, München, Fakultät für Informatik, October 2000.
- [14] Bran Selic, Garth Gullekson, and Paul T. Ward. *Real Time Object Oriented Modeling*. John Wiley & Sons, 1994.
- [15] Bran Selic, Stuart Kent, and Andy Evans, editors. *Proc. 3<sup>rd</sup> Intl. Conf. <<UML>> 2000—Advancing the Standard*, number 1939 in LNCS. Springer Verlag, October 2000.
- [16] Harald Störrle. *Models of Software Architecture. Design and Analysis with UML and Petri-nets*. PhD thesis, Ludwig-Maximilians-Universität München, Institut für Informatik, December 2000. to be published, ISBN 3-8311-1330-0.
- [17] Harald Störrle. Describing Process Patterns with UML. In Giovanni A. Cignioni, editor, *Proc. Eur. Ws. Software Process Technology*, number N.N. in LNCS. Springer Verlag, 2001. accepted for publication.