Software Architecture—A Rational Metamodel

Philippe Kruchten

Rational Software Corp.
240-10711 Cambie Road
Vancouver, B.C., V6X 3G5
Canada
pkruchten@rational.com

Abstract

The purpose of this position paper is to define the terminology used at Rational to speak about software architecture and to put in perspective the various concepts involved.

Introduction

It seems fashionable lately to claim that software architecture is a concept too complex or too vague to be defined. However this position does not help the practitioners who have to take a stand, even a modest one, in their daily work. Especially in a consulting practice, the software architect brought in to help can rarely wave her hands in the void, exhibit 25 different—although slightly overlapping—definitions, pull out a mathematical model full of Greek letters ("Let an architecture be defined as the tuple {A, Ψ, Ω, ... } where A is the set of ..."), or bring on the table an evolving prototype of yet another language, understood by only a handful of initiates.

And even if the following definitions are incomplete, imperfect, or unsatisfactory, they have allowed us in the last few years to make some progress in capturing and communicating the architecture of software-intensive systems.

This model has formed the basis of software architecture practice at Rational. Although it is the current step on which we stand, it is by no means an end, and we continue to exploit what researchers may bring to this field to incorporate it and evolve it, while still making it practical and understandable.

Software Architecture Description

The main focus is *software architecture.* However software architecture is a concept hard to define precisely, and therefore the only thing we can really speak and reason about is an architecture which has been articulated in a *software architecture description.*

SIGSOFT 96 Workshop, San Francisco CA USA
© 1996 ACM 0-89791-867-3/96/10 ..$3.50

Views

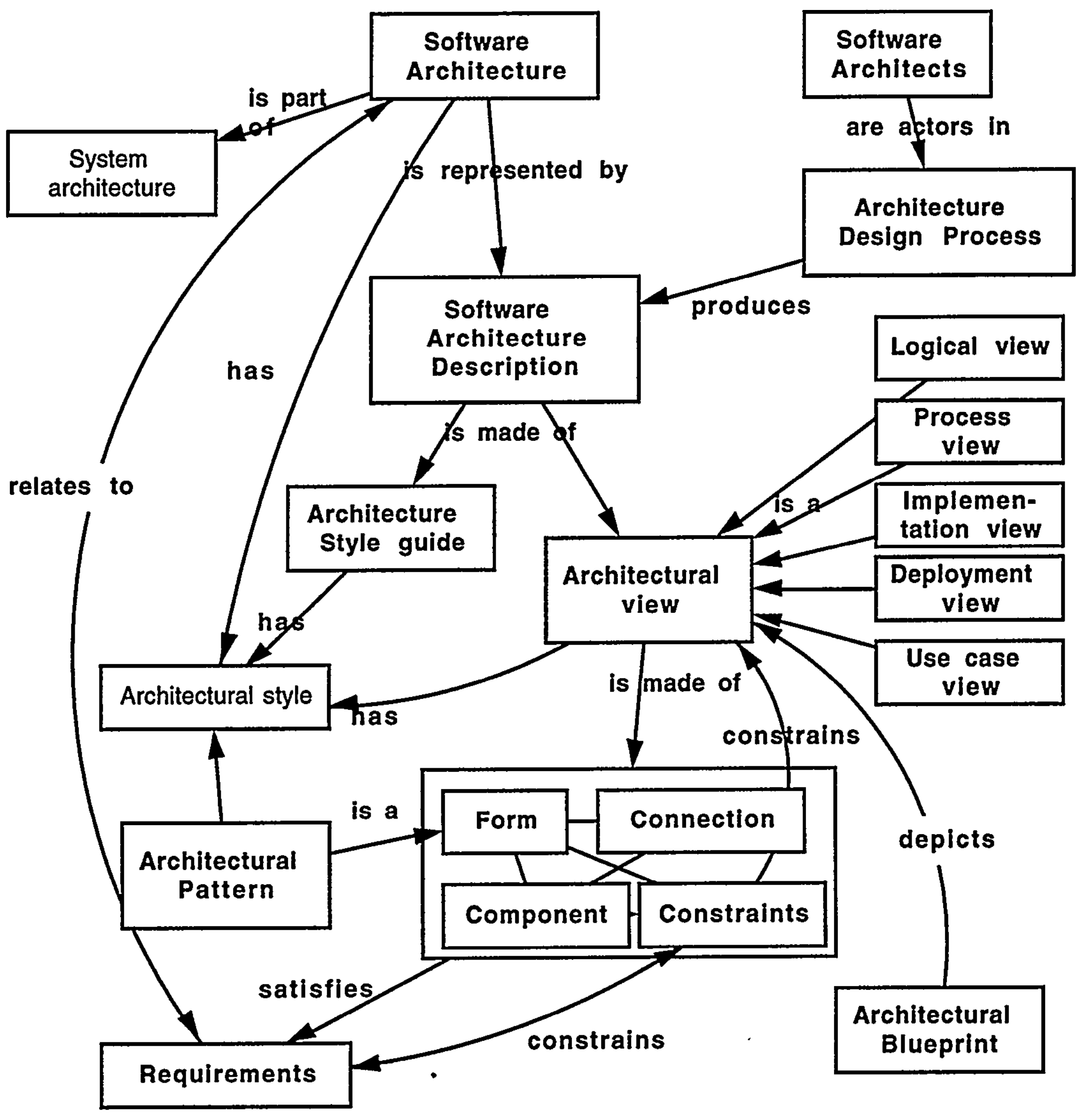
We have chosen to represent software architecture by multiple *architectural views.* Each architectural view addresses some specific set of concerns, of use by some specific stakeholders. [5, 7]

The views capture the major structural design decisions by showing how the software architecture is decomposed into *components*, how components are connected by *connectors* to produce useful *forms* [4]. These design choices must be tied to the *requirements*: functional, non functional or other constraints. But in turn these choices put further *constraints* on the requirements and on future design decisions at a lower level [9].

We often start from a typical set of views, called the 4+1 view model [5], which comprises:

- A *logical view*, where the decomposition is expressed in terms of objects, classes, class categories, and where the connectors are inheritance, association, containment, etc.
- A *process view* where the components are processes and threads, the connectors are inter-process communication and synchronization mechanisms.
- An *implementation view*, where the components are the modules, the subsystem, and where the major connectors are compilation/construction dependencies.
- A *deployment view* which describes the various processors (physical nodes), their interconnections (bus, LANs, WANs) and the mapping of processes and threads (logical nodes) onto this topology.
- a *use case view* which describes the system as complete sets of transactions from the point of view of external actors, and puts the architecture into its context. [8]

Additional views can be envisaged to express different special concerns: user-interface view, security view, data view, etc. [3] or conversely, some views may be omitted for simple systems.



Process

The *architecture design process* (or architecting process) is the process whose main outcome is the architecture description, and the main actors are the *software architects*. This process is one of the process elements of the Objectory process [8].

Patterns, framework, style, etc.

Architectural patterns are ready-made forms which solve recurring architectural problems [2]. An *architectural framework* or an *architectural infrastructure* (middleware) are sets of components based on which a specific architecture can be constructed. Many of the major architectural difficulties have been resolved in the framework or in the infrastructure, usually for a specific domain.

A software architecture, or an architectural view, may have an attribute called *architectural style*, which reduces the set of possible forms to choose from, and imposes a certain degree of uniformity to the architecture [6]. The style may be defined by a set of patterns. For a given system, some of the style can be captured as part of the architectural description in an *architecture style guide.*

Blueprints

The graphical depiction of an architectural view is called an *architectural blueprint.* For the various views described above, the blueprints are composed of the UML diagrams [1]:

- logical view: class diagrams, sequence diagrams and collaboration diagrams
- process view: class diagrams and collaboration diagrams encompassing processes
- implementation view: component diagrams
- deployment view: deployment diagrams
- use case view: use case diagrams and associated collaboration diagrams

and the accompanying documentation templates in Rational Rose and SoDA.

Software architecture in context

Software architecture is related to *system architecture* in many ways that are beyond the scope of this memo, but it is expected that most of the decoupling is done via the platform view.

Finally, through its representation, the software architecture relates the user's needs to the system to be built.

Conclusion

This set of definitions have allowed us to do some progress in capturing, communicating and discussing software architecture, over a wide range of real-life projects all around the world. The concepts and techniques they cover may not be breaking any new ground, but they have been used on numerous real-life projects.

We are working to evolve this model, taking the best of what the research can bring forward, and trying new concepts on real projects. Some of our current efforts focus on better capturing the architecting process and its integration with other software life-cycle processes [8].

The model we present here is seems relatively in line with the directions taken by the newly formed IEEE Working group on system architecture [10].

Acknowledgements

I would like to thank my friends and colleagues at Rational who helped bring this view to life along several internal workshops and tried the ideas, and to Patrik Jonsson, Ivar Jacobson, Walker Royce, Grady Booch, and Tom Merenyi who helped me with this paper.

References.

1. Grady Booch, Ivar Jacobson and James Rumbaugh, *Unified Modeling Language*, version 0.8, White paper, Rational Software Corp., Santa Clara, Ca., 1995, (and Addendum 0.9, 1996.)
2. Frank Buschmann, Régine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal, *Pattern-Oriented Software Architecture - A System of Patterns,* Wiley and Sons, 1996.
3. David Emery, Richard Hilliard, and Timothy Rice, "Experiences Applying a Practical Architectural Method," Alfred Strohmeier (ed.), *Reliable Software Technologies—Ada-Europe'96 proceedings*, Montreux, June 10-14, 1996, LNCS #1088, 1996, Springer-Verlag, pp.471-484
4. Dewayne E. Perry and Alexander L. Wolf, "Foundations for the Study of Software Architecture," *ACM Software Engineering Notes,* 17 (4), Oct. 1992, pp.40-52
5. Philippe Kruchten, "The 4+1 View Model of Architecture," *IEEE Software,* 12 (6), November 1995, IEEE, pp.42-50.
6. Mary Shaw and David Garlan, *Software Architecture—Perspectives on an Emerging Discipline,* Prentice-Hall, Upper Saddle River, NJ, 1996, 242p.
7. Dilip Soni, Robert L. Nord, Liang Hsu, and Paul J. Drongowski, "Many Faces of Software Architecture," David A. Lamb and Sandra Crocker (eds.), *Proc. of workshop on studies in software design*, Baltimore, May 1993, Springer Verlag.
8. *Objectory —User Guide and Reference Manual,* version 4.0, Rational Software Corp., Santa Clara, Ca., 1996.
9. Alan Burns and Andrew M. Lister, "A Framework for Building Dependable Systems," *The Computer Journal,* 34 (2), April 1991, pp.173-181
10. W. Ellis, et. al, "Towards a recommended practice for architectural description," To be presented at the IEEE ICECCS'96 conference, Montréal, QC, October 1996.