Patterns of Practical Architecting*

Rich Hilliard†

2 November 1999

Introduction

Since 1994, we[1] have been developing and applying an approach to architecting large, software-intensive systems. The approach has developed, not from theory, but practice wherein we have had the opportunity to take architectural responsibility for several significant command and control systems (e.g., [7, 4, 2]).

Based on the experiences gained through these opportunities, we have had the further opportunity to reflect upon what worked, and what did not work. We have written down our approach for others to review, critique and apply. Some of our lessons learned are documented in [3].

Since then, we have had the opportunity to apply the approach to other systems, in areas as diverse as Internet-based commerce, and distance learning. To the extent that we have documented our approach, it has taken the form of what others would call a *process*. We have been uncomfortable with calling it a process; because it seems a fixation on process in software engineering has come to overshadow what we believe to be the critical aspects of the design, engineering and architecting: judgement, decision-making, and decision-capture (for others). In teaching what we've learned to others, we found that students tended to be overly attentive to the steps in the process, and missed the crucial aspects. Being somewhat process skeptics, we have downplayed the process aspects of our approach, in contrast to the progression of understanding on the part of the architect in achieving, and documenting, that understanding in terms of various artifacts, *architectural descriptions*.

Last year, we starting looking at patterns as a medium for documenting our approach. This is currently work in progress. In this paper, we use it as the foundation to address the concerns raised by the *Call for Participation*, related to the role of development process in architecting.

Our current system of patterns focuses fairly narrowly on getting to an architectural description—only one small part of the architect's job. Future versions

*Position paper at the *OOPSLA'99 Workshop on Patterns in Software Architecture: the Development Process*. The version revised with up-to-date email contact and bibliographic reference information (2006), but with no changes to content.

†r.hilliard@computer.org

[1] See acknowledgements.

of the system of patterns will focus on other parts, but we felt it essential to begin with architectural description; motivated by concreteness and also by parallel work on architectural description, in particular IEEE 1471, *Recommended Practice on Architectural Description of Software-intensive Systems* [5].

Context: Practical Architecting

Upon discovering G. Meszaros' "Archi-Patterns" [6], it occurred to us that documenting our approach as an system of patterns might offer the right degree of flexibility, capture the essence of making architectural judgements, and avoid specifying an overly rigid process that others would fixate on.

The goal of our work thus far has been to create a foundation on which to elaborate our own work as well as incorporating the insights found in [6], [1] and others.

At first, we considered trying to build our approach atop Meszaros' framework, but found that would not work, for a couple of reasons: First, although Meszaros codifies many useful patterns, some of the underlying assumptions of his framework did not accord with our own. Second, Meszaros' pattern language mixes three things: architecting, architectural description, and architectures. Third, his patterns pertaining to architectural description presume a fixed set of viewpoints. What we were led to was a more "agnostic" framework, that permits the inclusion of Meszaros' patterns, as well.

The patterns we have developed (so far) fall into several regions:

- Initiation,
- Stakeholder Analysis,
- Framing the Architecture, and
- Architecture Integration.

These address the early, constructive stages of architecting. Future versions of our system of patterns will address the Construction and Maintenance stages, as well. In the figures below, individual patterns are depicted as ovals. Resulting contexts (new patterns to apply) are marked with arrows.

Initiation

In this stage, the patterns pertain to deciding whether a system warrants an architectural effort, bounding the context of the effort relative to the system's environment, and identifying the key system stakeholder of the architecture:

Stakeholder Analysis

This stage focuses the architect's attention on understanding the problem to be solved, in terms of the specific concerns and needs of the system stakeholders:

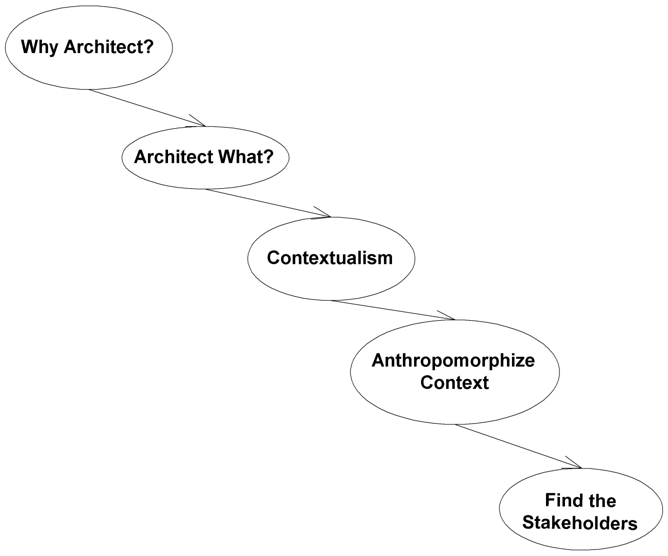

Figure 1: Initiation of Architecting

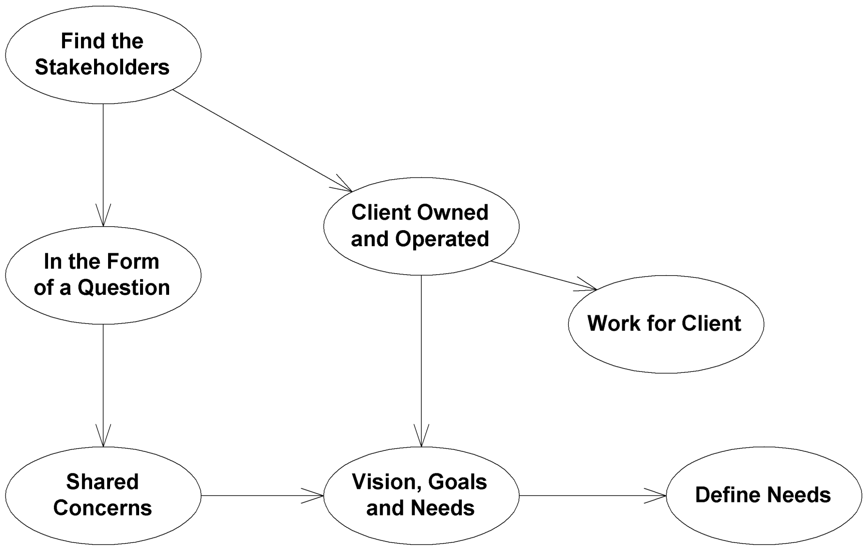

Figure 2: Stakeholder Analysis

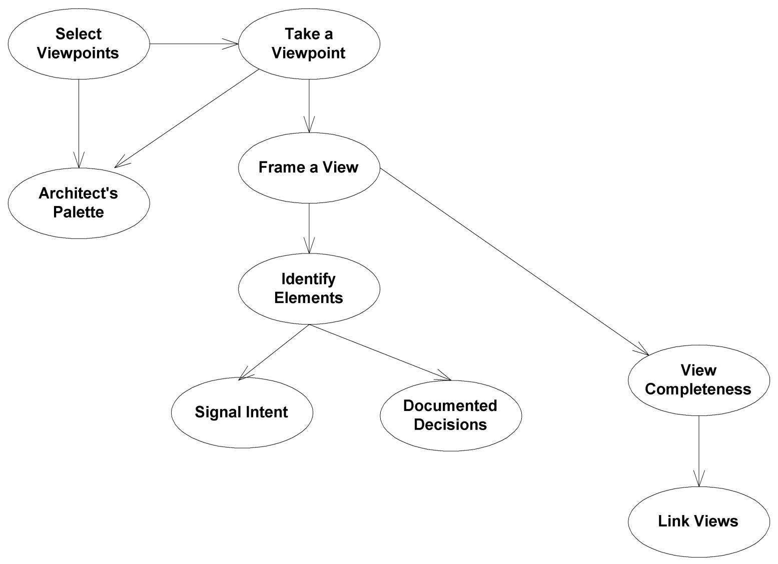

Figure 3: View Modeling and Integration

Framing the Architecture

The main effort of the architect at this stage is formulating an architecture for the system which meets the stakeholders' diverse needs. This is predominantly an exploration and modeling effort, leading to an architectural description consisting of a number of views. Each view addresses the concerns of one or more stakeholders:

Development Forces

There are always development concerns; in many cases, these may be satisfied by ordinary development approaches (existing processes, tools, methods of an organization)—without recourse to architectural consideration. However, there are cases where development *is* an architectural consideration.

In almost all of our efforts, we have resorted to directly addressing development issues at the architectural level. Whereas the more "structure-oriented" views guide developers in *what to build*, a "developers view" guides the developer, and vendors, and maintainers in *how to build* the system. This is frequently needed for large projects where developer teams are separated by time or space, such as in:

- product lines
- frameworks with separated application developers
- large migrations of existing systems, etc.

In our experience, the need for explicit architectural attention to development is usually identified from the stakeholders. In the first architecture we did, we defined a Developer-Maintainer view because the developers were located in industrial contractor shops, and the maintenance would be accomplished by a government organization. For a component-based development, allowing both CORBA and COM, it was necessary to separate out the rules for component producers and component integrators [2].

Conclusion

The system of patterns above is offered as a starting point for discussions of the role of development in architecting because it has been created to be process- and method-independent. It is hoped this will allow participants a way to focus on key development forces without the details of particular processes, etc.

Acknowledgements

A number of people have contributed to the Practical Architecture Method: the "core" participants have been David Emery (MITRE), Rich Hilliard and Tim Rice (ISIS); other contributors have been Rich Baldwin (MITRE), and Stephen Schwarm (EMC).

References

[1] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-oriented software architecture: a system of patterns.* Wiley, 1996.

[2] DII–AF Chief Architects' Office. *The Air Force's Command and Control System Target Architecture version 1.0*, 1998.

[3] David E. Emery, Rich Hilliard, and Timothy B. Rice. Experiences applying a practical architectural method. In Alfred Strohmeier, editor, *Reliable Software Technologies – Ada-Europe '96*, number 1088 in Lecture Notes in Computer Science. Springer, 1996.

[4] R. Hilliard, J. R. Knisley, and R. A. Ringdahl. Reserve component automation system (RCAS) final assessment. Technical Report MITRE Working Note 95W0000177, The MITRE Corporation, September 1995.

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

[6] Gerard Meszaros. Archi-patterns: a process pattern language for defining architectures. In *PLoP'97*, 1997.

[7] Timothy B. Rice et al. Sustaining base information services (SBIS) software architecture. Technical Report W114-HU-161, The MITRE Corporation, Ft. Huachuca, AZ, September 1994.