How Should Patterns Influence Architecture Description Languages?

A Call for Discussion

Mary Shaw and Paul Clements

Computer Science Department and Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

July 1996

Abstract: The software architecture and the design pattern communities have overlapping interests. The software architecture community is chiefly concerned with structure and organization of large software systems; the patterns community with exposition of design information. These intersect in the exposition of design information at the system level. This essay lays out a framework for the software architecture community to consider (a) what new ADL capability is suggested by design patterns and (b) to what extent ADLs are appropriately carriers of pattern information, and how they should do so.

Keywords: software architecture, architectural styles, style classification/taxonomy, design patterns, pattern languages

1. Introduction

Two technological innovations in large-scale software design have, in the past few years, been undergoing rapid growth and finding community acceptance. In the software architecture community, *architecture description languages* (ADLs) have been created to support the creation, refinement, validation, and analysis of systems at the architectural level. Languages such as Rapide [ref], Aesop [Garlan+94], UniCon [Shaw+95], MetaH [ref], Wright [ref], Demeter [Lieberherr96], Gestalt [ref], Artek [ref], RESOLVE [ref], and others exist and are under continuing development, and provide (as a group) a significant impetus to consider an architecture as an artifact of tangible value, and, in some cases, architectural styles as reusable design rules for system structure. In the software design community, particularly the object-oriented world, *design patterns* and *pattern languages* have emerged as a powerful aid in solving recurring problems that arise in software. A design pattern is not only the design of a software solution to a particular problem, but (following Christopher Alexander's use of the term in building architecture) also the structured presentation of that solution in a template that serves both the writer of the pattern and the reader—i.e., the software designer. Both the problem statement and the solution description are rendered in language- and (largely) implementation-independent terms. Books of pattern collections are starting to emerge [GoF95, Buschman+96]

A natural question to ask is, "What is the relation between architectures and design patterns?" The consensus at the most recent OOPSLA conference, as articulated in a panel discussion [ref], was that design patterns hold discourse at a level of detail below that of architectures, but above that of programming language idioms. A more useful question, perhaps, is whether or not there are design solutions at the architectural level that are analogous to patterns. Buschman [Buschman96] has documented certain designs commonly accepted as architectural styles using patterns, and Shaw [Shaw96] has written patterns for several architectural styles. This apparently settles the question of whether an architectural style can be documented in pattern form.

What about specific architectures? In particular, what is the relationship between the pattern templates such as the one in [GoF95] and descriptions of architectures?

This paper is intended to engender a discussion within the community of ADL builders as to the role of patterns in the description of architectures and the role of ADLs in capturing the kinds of information included in patterns. In particular, we ask which part of the information in a pattern it is appropriate for ADLs to capture and to what extend the pattern medium is helpful in describing architectures.

The paper is organized as follows. Section 2 presents an introduction to patterns, suggests the source of their intuitive power and practical value, and sets the stage for discussing them in an architectural context. Section 3 discusses design activities to which both patterns and ADLs may apply. Section 4 discusses aspects of patterns that might (or might not) be directly supported by ADLs, but are still of interest to the architecture community. The paper concludes with some suggestions for further pursuing the topic. We present this as a basis for discussion, not as a worked-out prescription.

2. Why Patterns Are Such Hot Stuff

Alexander [Alexander+77] captured his knowledge of (building) architecture design in collections of design fragments that he called *patterns*; coherent collections of complementary patterns form *pattern languages*. Each pattern had a specific structure that described not only the design element itself but also the function the element served, the reasons for using or avoiding it, and other patterns with which this pattern might interact. Different collections of patterns lead to different languages; each language corresponds roughly to a style of building that is adapted to a specific climate or culture.

The idea of patterns was picked up (probably independently) by several groups in the software community, most obviously a segment of the object-oriented community. This has now grown to an identifiable community with books and workshops on the use of patterns in software design. Roughly following Alexander, software patterns writers describe design fragments in pattern style. These descriptions are highly structured: they are informal but precise. They include not only the definition of the design fragment itself, but also a consider-able amount of supporting information about how to use the pattern well. Coherent collections of complementary patterns, often for a specific domain, form pattern languages.

Pattern writers refer to a pattern as "a solution to a problem in a context." This emphasizes that the pattern presents not only the design fragment, but a substantial body of guidance about when it should be used, how it should be adapted, and what some of the side effects may be. For the most part, pattern writers do not distinguish programming patterns from architectural ones; patterns address the spectrum from very detailed scalar representation to architectural styles.

The specific structure varies somewhat from author to author, but it generally includes four essential kinds of information, plus some auxiliary information. Various authors further subdivide the information into some of the categories shown:

- *Identification:* Pattern name, other names by which the pattern is known.
- *Problem:* What problem the pattern solves, when to apply it.
    - *Context:* When the pattern applies. Elements related to the context may include
    - *Intent:* What the pattern does, its rationale. What design issue or problem it addresses.
    - *Motivation:* Scenario that illustrates design problem and how the pattern helps to solve it
    - *Applicability:* Where it applies. Where it can help avoid bad design.
    - *Example:* Motivating problem, with solution based on pattern.
- *Solution:* Elements of solution, their relations and interactions; this is cast as an abstract description or template, not a specific solution
    - *Participants:* Elements in design, their responsibilities
    - *Collaborations:* How the participants interact
    - *Structure:* Graphical depiction and/or explicit description of relation among participants and collaborations.
    - *Dynamics:* Scenarios for behavior in practice.
    - *Significant variants:*
    - *Implementation:* Pitfalls, hints; language-specific issues
    - *Sample code:* Illustrative code fragments
- *Consequences:* Results and trade-offs of applying the pattern often concerning performance trade-offs or impact on reuse, modifiability, or portability of result.
- *Additional information:* Optional.
    - *Related patterns:* What are similar patterns? How are they significantly different?
    - *Known uses:* Appearances in real systems
    - *Credits:* Where the ideas came from, who helped elaborate them.

Patterns have met with substantial success as an expository form, especially in industry. Their power appears to reside in the expository form: they explain why and how to use the design fragment rather than simply documenting what it is. Furthermore, the patterns in a collection or language are organized in the same way, which makes them particularly accessible as reference material.

3. What Can ADLs Learn from Patterns?

To stimulate ways in which ADLs might usefully be influenced by design patterns, we can proceed by:

- inquiring what ADLs currently allow their users to do, and see how much of that might apply to supporting pattern descriptions or using patterns in system-building;
- speculating about what operations might be performed on or with a design pattern.

And then we can examine the two lists for areas of possible overlap, to suggest cross-over opportunities.

ADLs have, primarily, the capacity for carrying out or assisting with:

- *representing* architectures by means of a graphically meaningful language that shows a system's components and the (usually run-time) relation among those components.
- *refining* architectures; that is, deriving versions of the architecture that are consistent (in predefined ways) with previous versions, but that specify or bind more information. This may include working with variants of an architecture as may be derived by replacing a component with another of the same or different interface.
- *viewing* architectures through any of a number of filters or views.
- *analyzing* architectures; that is, predicting qualities or properties about systems developed using an architecture, such as its performance or behavior; this may be done with the assistance of associated tools.
- *building* systems; that is, generating code from component descriptions, or generating wrappers or glue code. This may be viewed as the final product of refinement.

On the other hand, what activities might be performed on or with a design pattern? That is, what kind of automated support would you like to be able to count on when designing at the architectural level with patterns? In this case, we hypothesize the following capabilities:

- *application:* the use of a pattern to guide development of software.
- *representation*: the ability to represent the pattern in a language consistent across all patterns
- *retrieval* from a library: patterns are reusable designs that reside in a repository. A search and retrieval capability will become vital, especially as the number of patterns grows.
- *recognition*, or extraction from an application: the ability to recognize the occurrence of a pattern, given a specification of a system to a sufficient level of (implementation) detail.
- *analysis*, including checking for type-correctness
- *refinement* or customizing a pattern
- *composition*, or joining of two or more patterns
- code generation, for those parts of the pattern that are well-enough defined to allow it

Where do these facilities overlap? Representation, refinement, analysis, and system-building are all things that ADLs do that design pattern people might wish to do. Of course, it does not follow that just because an ADL contains enough architectural information to (for example) generate code for components that it would be straightforward to imbue it with the power to generate code from a design pattern. Nevertheless, these four operation types suggest candidate areas for expanding ADLs to accommodate pattern-like information.

Whether or not it is reasonable to expect a particular ADL to stretch to provide these facilities for patterns will be left to Section 5.

4. What Pattern Information Should Be Captured by ADLs?

Architecture description languages and patterns capture some of the same information, but they have different objectives. It makes sense to ask which of the information described by a pattern may appropriately be captured by an ADL. It does not, however, make sense to expect ADLs to capture all the information typically included in a pattern.

There is a significant distinction between patterns and ADLs. Patterns are intended to describe design rules, whereas ADLs have a primary objective of describing particular systems, with templates, styles, or design rules (by whatever name) a means to that end. We should therefore focus on the question of what pattern-like information can or should be captured by the portions of ADLs that are intended to define templates, styles, or design rules.

Another important distinction between patterns and ADLs is that ADLs are intended to provide definitions that are formal and precise enough to be processed mechanically, whereas patterns contain narrative and motivational material as well as the definition of the design fragment itself. This suggests that the definitional parts of patterns are the most likely candidates for direct ADL support.

Going beyond ADLs, the long-standing objective of developing a design record for software has aspirations similar to the "why" and "how" parts of patterns. There may be a happy match between patterns and the design record than with ADLs.

It would be useful to flesh out a table showing which elements of the pattern structure are supported in each of the ADLs. Some preliminary information is:

- *Identification:* All ADLs can name parts and definitions. Many support named templates, styles, or design rules.
- *Problem:* Most of this information is narrative. Property-based ADLs could be stretched to capture it in an uninterpreted value, but the utility of doing so is not clear.
    - *Context:*
    - *Intent:*
    - *Motivation:*
    - *Applicability:* Demeter carries some of this in traversal specifications.
    - *Example:*
- *Solution:* Elements of solution, their relations and interactions; this is cast as an abstract description or template, not a specific solution
    - *Participants:* For component/connector languages, these are the components.

- *Collaborations:* For component/connector languages, these are the connectors and the connection topology
- *Structure:* Many ADLs have graphical notations.
- *Dynamics:* Some
- *Significant variants:*
- *Implementation:* Pitfalls, hints; language-specific issues
- *Sample code:* ADLs often go beyond samples to provide real code -- in the instances.

- *Consequences: and other information:* Most of this information is narrative. Property-based ADLs could be stretched to capture it in an uninterpreted value, but the utility of doing so is not clear

5. Proceeding with community input

As mentioned in the introduction, this paper is a solicitation to the ADL-developing community. It is the opinion of some that the relationship between ADLs, design patterns, and the uses of each deserves consideration by the community. Towards this end, we invite the reader to consider the following questions:

1. What information provided by design patterns does your ADL already support? How is it supported (e.g., representation, refinement, analysis, code generation, etc.)? What specific features of the ADL provide support?
2. What information provided by design patterns could you reasonably add to your ADL?
3. What operational capabilities for design patterns (representation, refinement, analysis, code generation) could you reasonably add to your ADL?
4. What provisions does your ADL have for making coherent collections of definitions, as in a design language?
5. How could you use the pattern form in conjunction with your ADL to improve the presentation of abstract definitions?
6. How could you use the pattern form in conjunction with your ADL to improve the presentation of concrete (instance) definitions?
7. What automated aids for the use of design patterns at the architectural level—perhaps outside the realm of your or any ADL—do you believe would be useful to a practitioner?
8. What features or qualities of design patterns do you believe are not suitably supported by your ADL and/or ADLs in general?

Send your comments to the authors:

mary.shaw@cs.cmu.edu

clements@sei.cmu.edu

The results will be collected and distributed as appropriate.

6. Bibliography

[Alexander+77] C. Alexander, S. Ishikawa, M. Silverstein, et al. A *Pattern Language*. Oxford Univ Press 1977.

[Buschman+96] F. Buschman, R. Meunier, H. Rohnert, P. Sommerlad, M. Stahl. *Pattern-Oriented Software Architecture: A System of Patterns.* Wiley 1996.

[Garlan+94] David Garlan, Robert Allen, and John Ockerbloom. Exploiting Style in Architectural Design Environments. *Proc. Second ACM SIGSOFT Symposium on Foundations of Software Engineering*, December 1994.

[GoF95] E. Gamma, R. Helm. R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley 1995.

[Lieberherr96] Karl J. Lieberherr. *Adaptive Object-Oriented Software: The Demeter Method with Propagation Patterns*. PWS Publishing 1996.

[Shaw+95] Mary Shaw, Robert DeLine, Daniel V. Klein, Theodore L. Ross, David M. Young, Gregory Zelesnik. Abstractions for Software Architecture and Tools to Support Them. *IEEE Transactions on Software Engineering*, May 1995.

[Shaw96] Mary Shaw. Some Patterns for Software Architectures. In Vlissides, Coplien & Kerth(eds). *Pattern Languages of Program Design 2*, Addison-Wesley 1996, pp. 255-269.

7. Appendix: Pattern Structures

The description of the elements of a pattern description is a synthesis from many examples. Here are some of the specific structures.

7.1 Gang of Four [Gof95]

- *Pattern Name and Classification*
- *Intent:* What does the pattern do? What is its rationale? What design issue or problem does it address?
- *Also Known As:* Other well-known names
- *Motivation:* Scenario that illustrates design problem and how pattern helps to solve it
- *Applicability:* Where does it apply? Where can it avoid bad design?
- *Structure:* Graphical depiction
- *Participants:* Other elements in design, their responsibilities
- *Collaborations:* How the participants interact
- *Consequences:* Trade-offs and results? What aspect of system structure can you vary independently?
- *Implementation:* Pitfalls, hints; language-specific issues
- *Sample Code*
- *Known Uses*
- *Related Patterns*

7.2 Siemens Architectural Patterns [Buschmann96]

- *Pattern Name*

- *Summary*
- *Example:* Problem statement of example. Includes diagram
- *Context:* Condition of applicability
- *Problem*
- *Solution*
- *Structure:* Includes <class, responsibility, collaborators> triples and OMT diagram
- *Dynamics:* Expressed with scenarios and diagrams
- *Implementation*
- *Example resolved:* Solution to example
- *Variants*
- *Known Uses*
- *Consequences:* Benefits and drawbacks
- *See also:* References to other patterns
- *Credits*

7.3 Shaw Architectural Patterns [Shaw96]

- Name
- Problem
- Context
- Solution:
    - System Model, types of components, types of connectors, control structure
- Diagram
- Significant Variants
- Examples
- References to use of pattern