Editor: Ronald D. Williams, University of Virginia, Thornton Hall/Electrical Engineering, Charlottesville, VA 22903; phone (804) 924-7960; fax (804) 924-8818; Internet, rdw@virginia.edu

Finding solutions through pattern languages

Steve Berczuk, MIT Center for Space Research

Interest in patterns and pattern languages has been on the upswing, fueled by the realization among software developers that they must simplify the process of building increasingly large and complex systems. Patterns are forms for describing architectural constructs in a manner that emphasizes these constructs' potential for reuse. They provide a way to document and share design expertise in an application-independent fashion. As evidence of this growing interest, early in August more than 70 software practitioners gathered to discuss patterns and pattern languages at the first annual conference on Pattern Languages of Programs.

The idea of using patterns and pattern languages is borrowed from work done in building architecture to describe qualities for good architectural designs. In the seventies, the architect Christopher Alexander started using pattern languages to describe the events and forms that appeared in cities, towns, and buildings in the world at large. Alexander's pattern language is "a system which allows its users to create an infinite variety of those . . . combinations of patterns which we call buildings, gardens, and towns."[1] Alexander defines a pattern as "a rule which describes what you have to do to generate the entity which it defines."[1] A pattern describes a solution to a problem in an environment "in such a way that you can use this solution a million times over, without ever doing it the same way twice."[2] Alexander documents patterns that exist all around us; for example, each building is unique, yet all buildings share many features.

One of Alexander's patterns is called "Master and Apprentices." It describes how to arrange workspaces so that new employees can learn by being in proximity to their mentors and use day-to-day experiences as a training mechanism. Although the advantages of having trainees learn from the daily work environment may seem obvious, in many organizations the office setup does not encourage this. Documenting this pattern, and referring to it when designing offices, helps a less experienced architect build a quality workplace.

Software development presents an analogous situation. Independently developed software systems often share common elements of an architectural structure. An example of a low-level pattern in C++ is checking for a nonnull pointer after allocating an object with *new* (this could also be called an "idiom"[3]). Most programs do this, and ones that don't are likely to run into problems. An example of a higher level pattern is the use of callbacks to initiate an operation when an event happens. Higher still are patterns of structure in software development organizations.[4] These patterns are discovered by experience. By documenting these patterns and their relationships, we can develop a set of languages to guide developers in building new systems.

The connection between Alexander's patterns and software architecture has led many in the software community to argue for a higher-level organizing principle in software than that of objects. Much recent discussion logically centers on object-oriented design, where it is natural to discuss interactions between entities. Yet patterns have uses in other paradigms, and people are beginning to propose patterns that apply to shell scripts and other procedural systems.

An Alexandrian pattern consists of the following components:[2]

- A name, which describes briefly what the pattern accomplishes within certain larger patterns.
- A concise problem statement.
- The body of the problem, including the motivation for the pattern and

To learn more

Alexander explains the motivation behind a pattern language and provides an example of a well-developed one.[1,2] Lea provides historical perspective on pattern languages, as well as a description of how they relate to architecture. Coad gives guidelines for finding patterns for object-oriented analysis and design. Gabriel discusses how we can use patterns to begin understanding what quailty is in software.[5]

Current applications of patterns include Coplien's book on C++ idioms and the forthcoming design patterns by Gamma et al.[7] Siemens is cataloging patterns for possible reuse in architectures.[8]

To learn more about patterns, you can subscribe to the patterns mailing list (send e-mail to patterns-request@cs.uiuc.edu).

References

1. C. Alexander, *The Timeless Way of Building*, Oxford University Press, New York, 1979.
2. C. Alexander et al., "*A Pattern Language: Towns, Buildings, Construction*," Oxford University Press, New York, 1977.
3. D. Lea, "Christopher Alexander: An Introduction for Object-Oriented Designers," *ACM Software Eng. Notes*, Vol. 19, No. 1, Jan. 1994, pp. 39-46.
4. P. Coad, "Object-Oriented Patterns," *Comm. ACM*, Vol. 35, No. 9, Sept. 1992, pp. 152–159.
5. R. P. Gabriel, "The Quality Without a Name," *J. Object-Oriented Programming*, Vol. 6, No. 5, Sept. 1993, pp. 86-88.
6. J. Coplien, *Advanced C++ Programming Styles and Idioms*, Addison Wesley, Reading, Mass., 1992.
7. E. Gamma et al., *Design Patterns: Elements of Reusable Object-Oriented Software Architecture*, Addison Wesley, Reading, Mass., 1994.
8. F. Buschmann and R. Meuiner, "A System of Patterns," *Proc. First Annual Conf. Pattern Languages of Programs*, Addison Wesley, Reading, Mass., to be published May 1995.

the forces involved in resolving the problem.
- A solution, preferably stated in the form of an instruction.
- A discussion of how the pattern relates to other patterns in the language.

We can create a similar form using patterns to document software frameworks[5] and architectures. A pattern language is a set of patterns that guide an architect through a design. Each pattern is a description of a solution to a problem using other patterns that occur in the system. The details of the form vary, but the essential elements are context, problem, and solution.

Figure 1 contains a simple example of a pattern. Notice that the callback mechanism described here is similar to the callback mechanism used by window managers to connect events to user events. This illustrates the power of patterns. They describe the static and dynamic structures that occur in a variety of software systems in a manner that emphasizes common aspects that make the pattern applicable across domains. (The *Proceedings of the First Conference on Pattern Languages*[6] contains more involved examples of patterns.)

Alexander's pattern language contains over 250 patterns, organized from high level to low level. The goal in documenting patterns that exist in software architectures is to arrive at a similar system, but this will take time. When we begin to document patterns, smaller and larger ones will be discovered, so the context cannot always immediately be specified entirely in terms of existing patterns. But we can ultimately specify context by discussing the situation that surrounds the problem.

Pattern languages are a useful medium for documenting software architectures. Unlike other ways of describing the design, a pattern by definition describes the motivation surrounding the decision to use a particular solution, including the context and forces influencing the design. Patterns are often independent of the implementation language and can be used to describe connections between components.

Why use patterns? The pattern form is well suited to documenting design techniques. Unlike a design document, a pattern reflects something that has been used in a number of situations and thus has some generality. It has a context, which explains the intent of the pattern and suggests how it is to be used. Patterns also express solutions in ways that allow for some variation depending on the details of a circumstance. Finally, patterns can express architectural considerations independent of language and design methodology.

A designer wishing to use patterns can take a number of approaches. One is to compile patterns from a domain into a book and hand it to system architects. Another is to develop a system to catalog these patterns and use a tool to extract a pattern appropriate to the problem at hand. While a system of patterns (a language) is the ultimate goal, there are many stand-alone patterns that can and should be documented.

Although patterns are often discovered during design, and using a pattern language will aid design, writing patterns is not part of a design methodology. Patterns are discovered from experience. Writing the patterns found in an application helps future developers of similar applications integrate the key architectural components.

Patterns exist in our software. When they are documented, design wisdom can be leveraged by other projects in your company.

Name: Callback & Handler

Context: A system in which processing operations need to be assigned to events dynamically.

Problem statement: In a software system it is sometimes necessary to specify an action to occur in response to an event. The event-to-operation mapping may need to be specified by the user rather than by being hard-coded.

Problem Description: Consider a system that provides a facility to read in text documents from a stream and classify the documents according to their format (plain text, PostScript, X bitmap image, etc.). When the application sees a document of a certain type, the document is displayed. The subsystem that parses the document does not know the details needed to display the various document formats, and there is a requirement that the application user be able to specify an external application to view the document.

Solution: Use an event callback/handler mechanism. Provide a facility where the interpretation subsystem dispatches documents of a specific type to a view application for documents of that type. Provide the application with a set of common default view applications to reduce the need for the user to do extensive setup.

Participants:
Event Generator (Text Interpreter): Parses input text and creates documents of a specific class.
Event to be handled (Document): A subclass for each type of document format; provide a facility for setting the appropriate viewer to display.
Handler (Viewer): A representation of the display application used to display the documents.

Figure 1. An example of a pattern.

References

1. C. Alexander, *The Timeless Way of Building*, Oxford University Press, New York, 1979.
2. C. Alexander et al., *A Pattern Language: Towns, Buildings, Construction*, Oxford University Press, New York, 1977.
3. J. Coplien, *Advanced C++ Programming Styles and Idioms*, Addison Wesley, Reading, Mass., 1992.
4. J. Coplien, "A Development Process Generative Pattern Language," *Proc. First Annual Conf. Pattern Languages of Programs*, Addison Wesley, Reading, Mass., to be published May 1995.
5. R. Johnson, "Documenting Frameworks Using Patterns," *Proc. OOPSLA*, ACM Press, New York, 1992.
6. *Proc. First Annual Conf. Pattern Languages of Programs*, Addison Wesley, Reading, Mass., to be published May 1995.

Steve Berczuk is a software engineer at the MIT Center for Space Research. He can be contacted at the MIT Center for Space Research, Room NE80-6015, 77 Massachusetts Avenue, Cambridge, MA 02193; e-mail berczuk@mit.edu.