Patterns Conference (PLoP"94)

Author: Mark Windholtz.
Date: 7 Aug 1994

There are a number of ways to describe what happened at PLoP"94. First, in order to provide the motivation and history of patterns work, this paper will present an introduction to patterns. Second, the paper will explore three aspects of the PLoP"94 conference. The three aspects are the community building, the writers workshops, and the soon to be published book, "Design Patterns: Micro-architectures for OOD", by Erich Gamma, et al. Lastly, the paper will consider the future of Patterns.

Introduction to Patterns

Computer Science has often borrowed from other intellectual disciplines. The fruit borne by this cross-pollination include algorithms from mathematics, large scale project organization from engineering, object oriented classification from biology, and most recently patterns from architecture.

Patterns were first proposed by the architect Christopher Alexander as a way to describe and design what he called "the quality without a name." What I think he was after (risking over-simplification) is "livability". He developed patterns to describe features that could be combined to design a structure. Some well read computer scientists noticed Alexander's work and have been busy applying it to the description and design of logical structures.

Before one can understand an answer, its important to understand the problem. What is the problem that patterns solve? In most pursuits, we tend to repeat certain steps and certain thought processes. In software design certain thought processes also tend to repeat themselves. What I've described so far, sounds like algorithms. Some of the most wide spread algorithms have been documented by Kunth. Learning and applying algorithms is effective for procedural implementations that have previously been used like, FORTRAN, COBOL, Pascal, 'C', etc.

So here is the problem, "How do we describe and discuss the repeated thought processes in the age of Objects?" Algorithms are better at describing sequential steps than at describing the relationships and responsibilities of Objects. Yet those building object oriented software have come across recognizable constellations of Objects that when understood can be applied to other similar software development problems. These are called "Patterns". Patterns describe what problem they are trying to solve, prescribe a solution, and discuss the consequences and side-effects of the solution. Each pattern has

a name. The names of a collection of patterns then become the vocabulary of a pattern language. Instead of looking to the sky and directing "Look to the left of those stars all in a row with the box shape at the end ...", you can say, "look to the left of the Big Dipper". This is the concise and powerful vocabulary that pattern languages give us.

Philosophers have concluded that one cannot think of things that are indescribable in ones available language. That's why we have domain specific languages in mathematics (such as Trigonometry, Calculus, etc.). We do not attempt to resolve mathematical problems in English. A new language of patterns gives us the tool to apply higher level solutions to our increasingly complex problems. In less than 2 years the pattern names "*Observer*", "*Bridge*", and "*Iterator*" will be as common to software developers as "BinarySort" and "LinkedList" are now.

Pattern Basics

While the exact format varies, the basics of patterns are deceptively simple. A pattern contains:

1. Name.
2. Problem to be solved.
3. Context of the problem.
4. Solution to the problem.

When patterns are built in a way that they reference each other they may be considered a pattern language. Otherwise, if they stand alone, we consider them universal patterns. Lets look at the patterns that appear in our project already.

Patterns we've seen

1. Frame uses registry to notify components of events such as sign-off or change Dynamic Port. This in essence is the *Observer* Pattern. An observer (in our case an component) asks the *Subject* (in our case the Frame) to notify it in case of change. The component observes the Frame for changes. The "Design Patterns" book contains a lengthy discussion of the trade-offs with using such a strategy.
2. Each component has a procedural interface to other components. If a component happens to be object oriented then this translation to procedural is described in the *Facade* Pattern.
3. Some components use the Rogue Wave class library. To get items out of a collection one uses an iterator. This also is described as a pattern...The *Iterator* Pattern.

We shouldn't get too carried away with looking for the book's patterns in our project. Normally, patterns are based on Objects and Classes. The design trade-offs and discussions are mostly based on OO as a starting point. Applying patterns to a procedural system is more difficult since it is missing a layer of abstraction normally provided by OO.

Example
Lets look closer at the *Iterator* Pattern. The following serves as a short example of a pattern. The *Iterator* pattern in the book required 12 pages of explanation. The following uses only 15 lines of text. It is neither as clear nor as complete as the presentation in the book.

Pattern Name: **Iterator**
Problem: How to access items in a collection.
Context: A collection has a structure for holding a number of items. The internal structure of the collection could change or be optimized as development progresses. It is important that objects using the collection not be affected if the implementation of the collection changes. It is also important that the internals of a collection not be damaged by an object's misuse of the collection.
Solution: Use one object to manage the collection, and a second object, an Iterator object, to provide access to the iteration mechanism. A method of the collection should serve to create and return the iterator object, such as:

```
iteratorObject CollectionClass::makeIterator(void)
```

The iterator object should provide a public interface for First(), Next(), and isDone() functions.

PLoP"94

Now we move to the conference itself. There are three aspects of the PLoP"94 conference that I would like to discuss. The three aspects are (1) the community building, (2) the writers workshops, and (3) the "Design Patterns" book. Lastly, I'll discuss the future of Patterns.

(1) The Patterns Community

Discussions about patterns originated in small groups and over an internet mailing list. Never before this conference did all the people interested in patterns have a chance to meet each other in person. Work was taking place at Siemens AG, at AT&T Bell Labs, at IBM, and at the Univ. of Illinois.. Therefore, it became important to synchronize the scattered efforts. One goal of the conference was to establish a community for patterns discussion and exploration. Afterall, if you are interested in developing a language you need a community to validate it, adopt it, and expand it.

The patterns community was developing over the internet, but the personal contact was missing. Much community building took place at the conference. The daily schedules went from 7:30 in the morning to 10:00 in the evening. The constant work and constant contact helped foster our sense of community. We shared meals and slept in dorm rooms. But no group of insiders or cliques developed because of the constant shifting of individuals from one workshop to another. Stability was accomplished by assigning participants to "home groups". The groups meet once or twice a day to share insights, raise questions, and launch initiatives like special topic discussions. There was a Home group newspaper that helped home groups keep in touch with what other home groups were doing.

(2) The Writers Workshops

Another aspect of PLoP"94 was the process of the writers workshops. The workshops were the central events of the conference. Pattern writers sat in a circle and discussed one paper after another in a series of steps including listening to the text aloud, summarizing the submission, discussing the likes and dislikes of the submission, and answering questions from the author. Non-writers were permitted to watch and, after the first day, to participate. The workshops were modeled after poetry workshops used in the literary community. The inspiration for this format came from Richard Gabriel, a regular contributor to JOOP and serious poet. The purpose was to help the community develop a style or elegance in writing patterns. To be a good language, patterns have to provide more than just information. They must embody a naturalness of expression, a smooth rhythm and level of comfort. The emphasis was on creating a body of literature, not just a pile of documentation. To raise to that level of art, the writers workshop technique provided the author with an immediate source of feedback. Every author going through the workshop process expressed an appreciation for the process. The only problem with the technique is that a large enough community must exist so that review groups can be formed more geographically. As it now stands, there are only 80 people experienced in the technique and process...worldwide!

(3) The Design Patterns Book

The upcoming book, "Design patterns: Micro-architectures for Object Oriented Design", by Erich Gamma et al. describes a number of universal patterns. The book's patterns provided a point of reference for the discussion of other patterns. I went to the conference expecting that we would talk exclusively about the book's patterns. Fortunately, there was much broader discussion. The book's patterns were, however, constantly referred to. Comments were made such as "your pattern is similar to *Proxy*, but different in the following ways...". This made me realize how far along we already are in establishing a vocabulary of pattern languages.

It is nearly impossible to meaningfully overview the book's patterns. The best approach to them is either by reading the available excerpts, taking a class from Ralph Johnson at

Univ. of Illinois., or waiting for the book's publication in October 94. On a very high level, the patterns in the book are classified in three ways. They are: Structural, Behavioral, or Creation Patterns. They concern themselves with the connections of objects in a system, the interactions of objects in a system, and the life-cycle management of objects in a system. The book describes over 20 patterns. The conference submissions described roughly 100 patterns.

The Future of Patterns

Questions and discussions on the future of patterns continued throughout the conference. There was the absolute confidence, however, that this is the start of a big movement. There are two foundations upon which patterns are built. One is the high level of abstraction provided by object orientation. The other is the body of experience provided by the last 20 years of software engineering. It is now possible to examine that experience, abstract it in terms of objects, and express it in terms of patterns. Where patterns will lead us is still not entirely certain. Many of the conference participants were in the inner circles as OO began its development. They drew parallels to those times ten years ago and cautioned of the destructive affect of the early OO "hype". There was a discussion on how to contain the hype that is about to explode around patterns. Unfortunately, we reached no conclusion other than that we should be honest in communicating the primitive state of patterns development. The fear is that some books already being published misrepresent patterns. Those books are by Peter Coad and Jiri Soukup.

One of the participants suggested that since patterns are primarily about human to human communication, areas outside of computer science will also be able to benefit from the formalization of the pattern languages method of communication. This is an exciting prospect.

Conclusion

I have attended 5 professional conferences so far in my career. Each of them has broadened my perspective and given me new insights. But, I have never before attended an event with such a high density of industry leaders or an event with the aura of a historical moment. The conference did not succeed in providing the exact definition for "pattern". The conference did succeed in building the patterns community, in introducing the culture of writers workshops to the community, and in establishing the "Design Patterns" book as a common vocabulary for the community.