NDIA 9th Annual Systems Engineering Conference

"Pattern Library for use in Weapons System Engineering"

October, 2006

Frank Salvatore
High Performance Technologies, inc.
3159 Schrader Road
Dover NJ, 07801
(973) 442-6436 ext 249
fsalvatore@hpti.com

Outline

- ❑Purpose/Objective
- ❑Background
- ❑Describe Patterns
- ❑Give Example
- ❑Open Discussion
- ❑Discussion Summary
- ❑Pattern References

Purpose/Objective

- The purpose of this talk is to communicate to industry an Idea about developing a Weapon Systems Engineering Pattern Library

- The objective is to stimulate thought and get industry input on this idea, expand on it and reach an agreement as to weather or not to pursue such effort.

Background

- 1960’s early 70’s Christopher Alexander Started the movement and wrote several books dealing with architecture patterns
- 1987 Ward Cunningham and Kent Beck experimented with patterns
- 1990-1994 Erich Gamma, Rich Helm, Ralph Johnson, John Vlissides (Gang of Four) Authored the book of design patterns that was published in 1994
- 2006 Software Developers are using Patterns.

Patterns for system architecting are very much in their infancy. **Today, the pattern discipline is supported by several small conferences, by a broad spectrum of activities at established software engineering conferences, and by a growing body of literature.**

What is a Pattern?

A "pattern" has been defined as: "an idea that has been useful in one practical context and will probably be useful in others" [Analysis Patterns - Reusable Object Models].

Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution. (Alexander)

As an element of language, a pattern is an instruction, which shows how a solution can be used, over and over again, to resolve the given system of forces, wherever the context makes it relevant.

In The Open Group Architecture Framework (TOGAF), patterns are considered to be a way of putting building blocks into context, to describe a re-usable solution to a problem.

Etc….

What is a Pattern Used For?

Building blocks are what you use: patterns can tell you how you use them, when, why, and what trade-offs you have to make in doing so.

As an element in the world, each pattern is a relationship between a certain context, a certain system of forces which occurs repeatedly in that context, and a certain spatial configuration which allows these forces to resolve themselves.

What do patterns contain?

Name: A meaningful and memorable way to refer to the pattern, typically a single word or short phrase.

Problem: A description of the problem indicating the intent in applying the pattern - the intended goals and objectives to be reached within the context and forces described below (perhaps with some indication of their priorities).

Context: The preconditions under which the pattern is applicable - a description of the initial state before the pattern is applied.

Forces : A description of the relevant forces and constraints, and how they interact/conflict with each other and with the intended goals and objectives. The description should clarify the intricacies of the problem and make explicit the kinds of trade-offs that must be considered. (The need for such trade-offs is typically what makes the problem difficult, and generates the need for the pattern in the first place.) The notion of "forces" equates in many ways to the "qualities" that architects seek to optimize, and the concerns they seek to address, in designing architectures.   For example: Security, robustness, reliability, fault-tolerance, Efficiency, performance, throughput, space utilization ,Scalability, Extensibility, maintainability, modularity, re-usability, composability, portability, Completeness and correctness, Ease-of-construction, etc….

Creates a syntax for communicating problems in a context and the conditions for which a solution exists.

What do patterns contain? (cont.)

Solution: A description, (text and/or graphics), of how to achieve the intended goals and objectives. Should identify both the solution's static structure and its dynamic behavior - the people and computing actors, and their collaborations. May include guidelines for implementing the solution. Variants or specializations of the solution may also be described.

Resulting Context: The post-conditions after the pattern has been applied. Implementing the solution normally requires trade-offs among competing forces. This element describes which forces have been resolved and how, and which remain unresolved. It may also indicate other patterns that may be applicable in the new context. (A pattern may be one step in accomplishing some larger goal.) Any such other patterns will be described in detail under Related Patterns.

Examples: One or more sample applications of the pattern which illustrate each of the other elements: a specific problem, context, and set of forces; how the pattern is applied; and the resulting context.

Rationale: An explanation/justification of the pattern as a whole, or of individual components within it, indicating how the pattern actually works, and why - how it resolves the forces to achieve the desired goals and objectives, and why this is "good". The Solution element of a pattern describes the external structure and behavior of the solution: the Rationale provides insight into its internal workings.

Provides a solution w/examples & rational to a problem within a context for a given set of conditions

What do patterns contain? (cont.)

Related Patterns; The relationships between this pattern and others. These may be predecessor patterns, whose resulting contexts correspond to the initial context of this one; or successor patterns, whose initial contexts correspond to the resulting context of this one; or alternative patterns, which describe a different solution to the same problem, but under different forces; or co-dependent patterns, which may/must be applied along with this pattern.

Known Uses; Known applications of the pattern within existing systems, verifying that the pattern does indeed describe a proven solution to a recurring problem. Known Uses can also serve as Examples.

Patterns may also begin with an Abstract providing an overview of the pattern and indicating the types of problems it addresses. The Abstract may also identify the target audience and what assumptions are made of the reader.

Provides reference to other patterns with similar context and know uses wereby forming the basis for a searchable and well referenced repository of solutions

Anti-Patterns

An AntiPattern is a pattern that tells how to go from a problem to a bad solution. (Contrast to an AmeliorationPattern, which is a pattern that tells how to go from a bad solution to a good solution.)

A good AntiPattern tells why a bad solution looks attractive (e.g. it actually works in some narrow context), why it turns out to be bad, and what positive patterns are applicable in its stead.

Identifying bad practices can be as valuable as identifying good

Good Pattern Characteristics

***Solves a problem*:** Patterns capture solutions, not just abstract principles or strategies.

***It is a proven concept*:** Patterns capture solutions with a track record, not theories or speculation.

***The solution isn't obvious*:** Many problem-solving techniques (such as software design paradigms or methods) try to derive solutions from first principles. The best patterns *generate* a solution to a problem indirectly--a necessary approach for the most difficult problems of design.

***It describes a relationship*:** Patterns don't just describe modules, but describe deeper system structures and mechanisms.

The pattern has a significant human component (minimize human intervention). All software serves human comfort or quality of life; the best patterns explicitly appeal to aesthetics and utility.

A ***pattern language*** defines a collection of patterns and the rules to combine them into an architectural style. Pattern languages describe software frameworks or families of related systems.

What is the Value of Patterns?

Useful for Capturing & Communicating

- Best Practices
- Lessons Learned
- Design Guidance
- Knowledge
- etc….

Provides Instruction

Increase Productivity

Reduce Risk

The Value Proposition

A Weapon System Pattern Library will enable retrieval of information in a form were it can be used by others to successfully solve problems .

Pattern Example

Name: Buy the First Round, Kevlin Henney, April 2001, revised June 2001

Context: Meeting colleagues for a few drinks in a bar or pub.

Problem: How do you maximize your drinking, whilst both minimizing expenditure and maintaining good will with your drinking colleagues?

Forces: You have limited money, or at least limited desire to spend it. The drinks are not free. You want to drink (possibly lots). Your colleagues will not all turn up on time, but there will be quite a few of them eventually. You do not want your colleagues to think (or perhaps notice) that you are being tight.

Solution: Buy the first round of drinks before all your colleagues have arrived.

Consequences: You need to get to the bar early, preferably first and with a couple of colleagues in tow. Volunteering to buy the first drink shows good nature and enthusiasm, and spreads good will. You get free drinks for the rest of the evening, assuming a reasonable increase in the number of colleagues and a steady rotation of round buying. If there is enough drinking, others will not notice your use of this pattern or recall any other known uses. This supports repeatability in future as only your generous nature will be remembered.

Pattern used in Lucent telecommunication products such as the Switching System® (extracted informally from *Adams, 1996*).: Copied from Hillside Group Website. http://hillside.net/patterns/definition.html

Pattern Example

Name: Try All Hardware Combos

Problem: The control complex of a fault-tolerant system can arrange its subsystems in many different configurations. There are many possible paths through the subsystems. How do you select a workable configuration when there is a faulty subsystem?

Context: The processing complex has several duplicated subsystems including a CPU, static and dynamic memory, and several busses. Standby units increase system reliability. 16 possible configurations (64 in the 4 ESS) of these subsystems give fully duplicated sparing in the 5ESS. Each such configuration is called a configuration state.

Forces: You want to catch and remedy single, isolated errors. You also want to catch errors that aren't easily detected in isolation but result from interaction between modules. You sometimes must catch multiple concurrent errors. The CPU can't sequence subsystems through configurations since it may itself be faulty. The machine should recover by itself without human intervention, many high-availability system failures come from operator errors, not primary system errors. We want to reserve human expertise for problems requiring only the deepest insights.

Pattern Example (cont.)

Solution: Maintain a 16-state counter in hardware called the configuration counter. There is a table that maps that counter onto a configuration state. The 5ESS switch tries all side 0 units (a complete failure group), then all side 1 units (the other failure group), seeking an isolated failure. When a reboot fails, the state increments and the system tries to reboot again. The subsequent counting states look for multiple concurrent failures caused by interactions between system modules.

Resulting Context: Sometimes the fault isn't detected during the reboot because latent diagnostic tasks elicit the errors. The pattern ***Fool Me Once*** solves this problem. And sometimes going through all the counter states isn't enough; see the patterns ***Don't Trust Anyone*** and ***Analog Timer***.

Rationale: The design is based on hardware module design failure rates (in Failures in a trillion (FITs)) of the hardware modules. The pattern recalls the extreme caution of first-generation developers of stored program control switching systems.

Open Discussion

Let's Discuss and Brainstorm the Application of patterns to the Domain of doing systems engineering.

Is this something that we need to do?

What patterns should we write?

Should they be domain specific?

Should they be generic?

What does Industry think about developing a systems engineering pattern language?

Discussion Summary

Did we decide that patterns could be useful in our industry?

Did we identify any Patterns that should be written?

Who should develop such a repository?

Some Pattern Resources

Christopher Alexander Website

www.patterlanguage.com

The OpenGroup

http://www.opengroup.org/architecture/togaf8-doc/arch/chap28.html

Pattern-Oriented Software Architecture: A System of Patterns

http://www-128.ibm.com/developerworks/patterns/

Hillside Group

www.hillside.net

Portland's Pattern Repository

http://www.c2.com/cgi/wiki?HistoryOfPatterns

Design Patterns: Elements of Reusable Object Oriented Software", Eric Gamma, Richard Helm, Ralph Johnson, John Vlissides