Advanced Object-Oriented Systems

Karl Lieberherr

Themes

- Design Patterns and UML
- Generic Programming
  - STL
  - GraphLog
- OO Software Architecture and OO Software Components
- Aspect-Oriented Programming

Something wrong with OO

Soren Lauesen,
Real-Life Object-Oriented Systems,
IEEE Software, 1998, March/April,
PAGES 76-83.

http://www.cbs.dk/~slauesen/OOcaseStudies/

It was like "reading a road map through a soda straw".

He means that
the soda straw is pointing to one class at
a time and you have to figure
out what the collaborations are.

Another argument for separating high-level operations from the objects
derives from GTE's experience with large OO systems.
GTE did not succeed until it put *control flow* and *business processes* -- high-level operations --
outside class behavior. If built into the classes involved, it
was impossible to get an overview of the control flow.
It was like "reading a road map through a soda straw".

TAPOS Vol. 2/Number 1
Special Issue: Patterns

- Understanding and Using Patterns in Software Development (Riehle, Zuellighofen)

Abstract

- Patterns: effective means of capturing and communicating software design experience
- What are the crucial aspects of patterns?
- Pattern types, forms
- Pattern handbooks

Outline

- 2: definition of pattern
- 3: different pattern types
- 4: pattern presentation forms
- 5: experiences with pattern sets
- 6: pattern handbook
- 7: related work
- 8: conclusions

Pattern Definitions and Characteristics

- **Def: A pattern is the abstraction from a concrete form which keeps recurring in specific contexts.**
- GOF: A pattern is a solution to a recurring problem in a context
- Alexander: Each pattern is a 3 part rule which expresses a relation between a context, a problem, and a solution.

Pattern Definitions

- Alexander (a building architect): The timeless way of building, Oxford University Press
- Pattern: a relationship between forces that keep recurring in a specific context and a configuration that resolves these forces
- Coad: A pattern is a template of objects with stereotypical responsibilities and interactions

Pattern Definitions: Form and Context

- **The form of a pattern consists of a finite number of visible and distinguishable components and their relationships.**
- Example: When analyzing an application domain we identify those objects of the domain which can be interpreted as either *tools* or *materials*. (2 types of components; tools working on appropriate materials. Pattern of the distinction of Tools and Materials.)

Pattern Definitions: Form and Context

- **A pattern is used to create, identify and compare instances of the pattern.**
- Instance of Distinction of Tools and Materials:
  - tool = pencil, material = form (analysis pattern helping to understand an application domain)
  - tool = methods, material = object graph

Pattern Definitions: Form and Context

- **Pattern instances appear only in specific contexts which raise and constrain the forces that give birth to a concrete form**
- A pattern is a form that appears in a context
- Both form and context of a pattern are abstractions
- Forces of use context have to fit the form of the pattern

Pattern Definitions: Form and Context

- **The form of a pattern is finite, but the form of its instances need not be finite. The context is potentially infinite.**
- Example: Chain of Responsibility pattern. Chain can be arbitrarily long.

Patterns and Models

- Relationship of patterns to models
- Three models
  - application domain model
    - Conceptual patterns
  - software design model
    - Design patterns
  - implementation model
    - Programming patterns

Patterns and Models
Conceptual Patterns

- **A conceptual pattern is a pattern whose form is described by means of the terms and concepts from an application domain**
- Guide perception of an application domain
- Future systems are constructed from conceptual patterns

Patterns and Models
Conceptual Patterns

- Conceptual patterns do not serve a general purpose
- Balance between too abstract and too concrete
- Too abstract: too general to guide design
  - "active collaborating object"
- Too specific: only usable in one project

Patterns and Models
Conceptual Patterns

- Examples of conceptual patterns
  - Distinction of Tools and Materials
  - Agents

Patterns and Models
Design Patterns

- **A design pattern is a pattern whose form is described by means of software design constructs, for example objects, classes, inheritance, aggregation and use relationships**
- Important to have as little semantic difference between conceptual model and software design model as possible

Patterns and Models
Design Patterns

- **Design patterns should fit or complement the conceptual space opened by conceptual patterns**
- Relate design patterns to conceptual patterns

Patterns and Models
Programming Patterns

- **A programming pattern is a pattern whose form is described by means of programming language constructs**
- Used to implement a software design
- Sometimes called idioms or cliches.

Model and Pattern Interrelationships

Goal:
little semantic difference between models

- application domain model
  - Conceptual patterns
- software design model
  - Design patterns: fit context set by conceptual patterns
- implementation model
  - Programming patterns: fit context set by design patterns

Pattern Description Forms

- Best way of description depends on intended use.
  - Alexandrian Form
    - Problem, Context, Solution
  - Design Pattern Catalog Form
    - More descriptive than generative
  - A General Form
    - Context and Pattern

Pattern Description Forms

- Alexandrian Form: The intended use of this pattern form is to guide users to generate solutions
  - Problem: concise description
  - Context: describes situations where problem occurs, as well as arising forces and constraints
  - Solution: describes how to resolve forces in context

Pattern Description Forms

- Alexandrian Form: generative
  - Analogy: Patterns can be used to derive architectures much as a mathematical theorem can be proved from a set of axioms

Pattern Description Forms

- Alexandrian Form
  - Problem, Context, Solution
- Design Pattern Catalog Form
  - More descriptive than generative
- A General Form
  - Context and Pattern

Pattern Description Forms

- Design Pattern Catalog Form: also intended to help users create solutions to problems. Focus is less on when to apply the pattern, but more on the actual structure and dynamics of the pattern itself.
- Distinguish pattern from pattern form

Pattern Description Forms

- A General Form: To discuss the structure and dynamics of the recurring form and its context without promoting a specific way of using the pattern

Pattern Description Forms: Comparison

- Not one form superior than the others: different intent
- Design Pattern Catalog form good for OO design patterns
- Alexandrian form: good for problem solving
- General form: good for general presentation

Conclusions

- Summarized pattern experiences
- Open problems: How to write pattern handbooks. How to classify patterns

Design Patterns and Language Design

- Gil/Lorenz: IEEE Computer March 1998
- Similarities/differences between design patterns and programming language mechanisms
- Classify patterns by how far they are from becoming actual language features … helps demystify them

Design Patterns and Language Design

- A systematic approach to patterns can be achieved only if we disregard intent -- a much too glorified component of patterns

Design Patterns and Language Design: Abstraction

- Abstraction: identify and capture commonalities
- Two aspects to abstraction: process and mechanism (means of expression)
- Process: the way we identify abstractions is not well understood
- Mechanisms: may be subjected to an analytic approach

Design Patterns and Language Design: Abstraction

- Mathematics: good analogy for distinction:
  - no mathematical theory to deal with the intellectual process of coming up with a new theorem
  - but theorems and proofs are expressible using well understood mathematical theory

Design Patterns and Language Design: Abstraction in Languages

- Each language: a toolbox of abstraction mechanisms
  - Example: Object, class, inheritance are fundamental abstraction mechanisms of oo languages

Design Patterns and Language Design: Abstraction in Languages

- An exact definition of a mechanism or a feature must cover many different details
- Language design is an art as much as it is a delicate and extremely difficult engineering task.

Design Patterns and Language Design: Abstraction in Languages

- Designer picks most useful, powerful and definable mechanisms. Tries to strike balance between
  - utility
  - complexity

Design Patterns and Language Design: Abstraction in Languages

- Abstraction mechanisms in current oo languages: low level
- Higher level lingual abstraction mechanisms - those that specify simultaneous interaction of several classes: are a rarity
  - too costly
  - too specialized to justify the price

Design Patterns and Language Design: Design Patterns

- Enhance the power of oo mechanisms
- Does not have to be a closed solution
  - must not work in all circumstances
  - and with all other mechanisms
- May be less precise. Are applied by an adaptive human, not by a rigid compiler

Design Patterns and Language Design: Design Patterns

- Offer a pay-per-use approach: what is saved is the up-front expense of a well-integrated set of general-purpose mechanisms
- Design patterns could and sometimes should grow to be language features

Design Patterns and Language Design: Idioms

- "If we assumed procedural languages, we might have included patterns called inheritance, encapsulation and polymorphism."
- In multi-method languages there is less of a need for the visitor design pattern

Design Patterns and Language Design: Unbundling the Intent

- Patterns have so far defied all attempts at analytic description
  - no pattern taxonomy
- Reason behind failure: bundling - within a pattern - both the pattern's internal working and its intent
- A systematic approach can be achieved only if intent is disregarded.

Design Patterns and Language Design: Unbundling the Intent

- Intent should be broad, open to variation and never inscribed into the pattern itself
- “If we are not careful, progress in thinking about patterns becomes hard to recognize or appreciate. If we are not careful, progress in thinking about patterns might be illusory as progress in the world of fashion.

Dotplot Patterns: A Literal Look at Pattern Languages

- Jonathan Helfman, AT&T Research
- Abstract: