

Montreal Software Analysis Talks

10/10/2007

Design Patterns Composition: Methodology, First Results, and Future Work

Simon Denier

Ptidej Team, GEODES Group, DIRO, University of Montreal
FCI/FEI & Lavoisier Grants

Outline

- Context: the Pattern Hypothesis in OO Design
- Problems with Implementation of Design Patterns
- PhD Thesis: Expression and Composition of Design Patterns with Aspects
- Future Work (Here and Now) on Pattern Detection, Composition, and Software Comprehension...

General Context

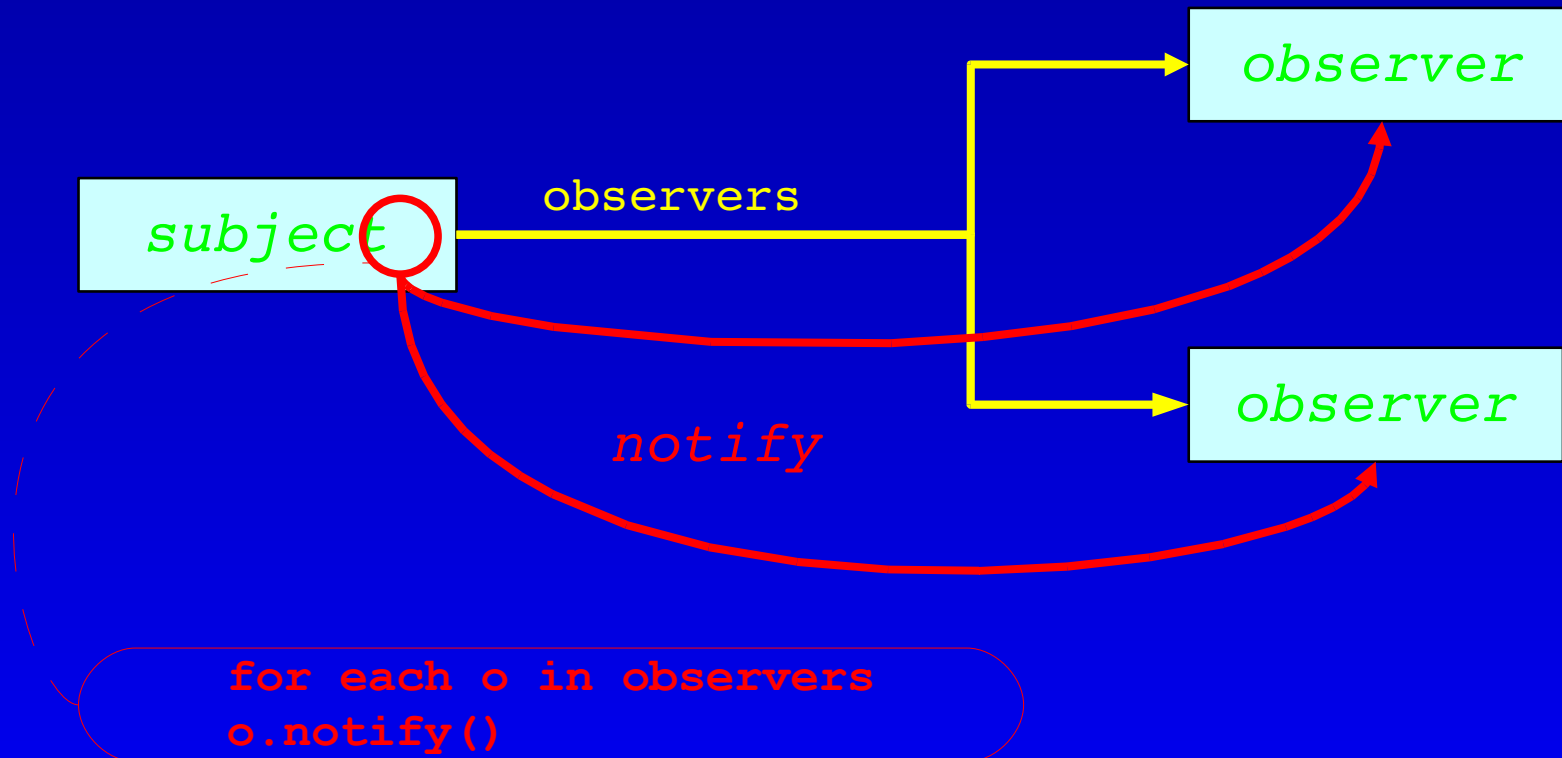
- Software maintenance
 - How does a program evolve and mature?
 - Are there structures which better stand the test of time?
 - Use these structures to ease maintenance: program comprehension, evolution
- In OOP, *design patterns* have been a success story (at least in book selling)
 - Hidden structures of OO programs?
 - Can we exploit those for maintenance?

The Pattern Hypothesis

- Some hypothesis made by the pattern community
 - 1st: design patterns arise “naturally” in the code
 - similar to “patterns are everywhere”
 - 2nd: the more “mature” a program, the more patterns there are
 - aka “high density of design patterns” [JUnit’s Cook Book]
- Application: knowing patterns *being* in the code
 - It’s easier to understand, thus maintain
 - It’s easier to evolve the program (in some directions)

Observer Pattern

- Notify change from subject to observers
- Roles, Structure, Collaborations



Problems with Patterns Implementation

- [Soukup, Bosch]
- Lack of modularity
 - Traceability: losing patterns in the code
 - Reusability: reusable occurrences, but no reusable patterns
 - Interdependences: classes dependent on each other because of pattern implementation
- Uneasy to follow and use patterns while in maintenance

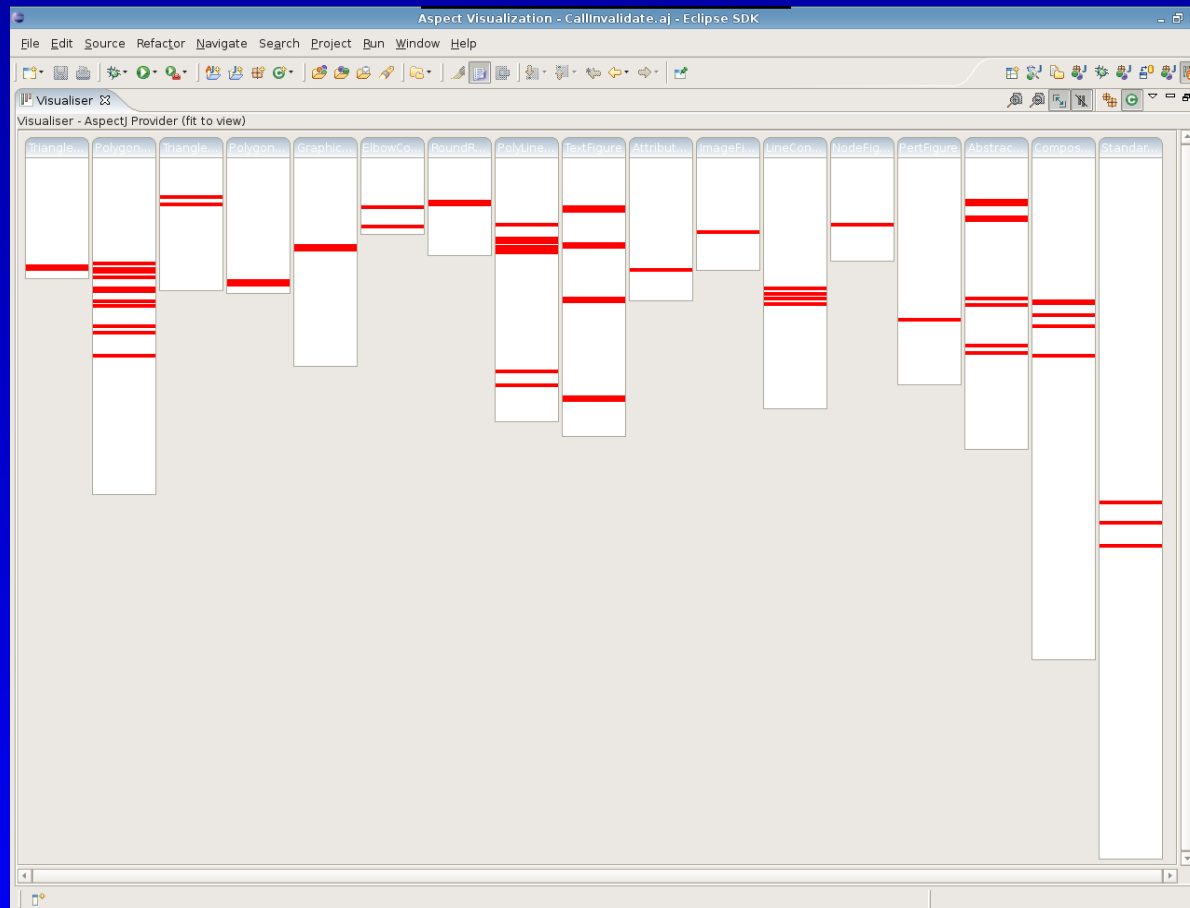
Some Solutions

- Software engineering tools
 - Detection of design patterns: traceability
 - Code generation, refactoring: reusability
- Programming languages
 - Modularity of design patterns implementation
 - Transformation of design patterns
 - New mechanisms for old problems

Observer Implementation:

Scattering and Tangling Problems

- JHotDraw : notification by subjects of Observer pattern



Aspect-Oriented Programming and Design Patterns

- Aspects can modularize scattered and tangled code
- OO design patterns modularization with AspectJ
[Hannemann and Kiczales, OOPSLA 2002]
 - Extensive study of each GoF pattern
 - Better modularization (70%), some reuse possible (50%)

PhD Thesis

- Expression and Composition of Design Patterns with Aspects
 - OBASCO team, École des Mines de Nantes
- The problem
 - What is a composition of patterns?
 - How to compose them with objects, with modularized aspects?

Contributions

- Pattern composition
 - begins with superposition of roles on the same class
- Modality = how patterns are composed
 - Categorization describing implementation of composition
 - 5 modalities: cohabitation, cooperation, use, sharing, interaction
- Implementing composition
 - With objects: problems arise when reuse is favored over decoupling; cross class modification
 - With aspects: some modular reusable composition (Composite-Visitor); reduced, modularized interaction

Contributions 2

- Programming with AspectJ
 - Programming idioms (especially for patterns)
 - Methodology: criteria for refactoring (or not!) with behavioral and structural aspects
- Case study: JHotDraw
 - Impact of patterns on objects
 - Manual refactoring with aspects and lessons learned

Future Work

(with One Year's Blinkers)

- Software Engineering: pattern detection, study of composition, and metrics
 - Interface coverage: analysis to detect misplaced factored code
- Program Visualisation and Comprehension
- Programming Languages

Study of Design Patterns

- Detection of patterns and composition
 - Categorization by modalities
 - Case study: projects, and also evolution of projects
- Can we link patterns and software quality?
 - Relate composition with quality metrics
 - Metrics for patterns and composition (density)?
 - Check (or not) the pattern hypothesis
- Application: refactoring suggestion, maintenance help with traceability of design patterns and design choices

Program Comprehension and Visualization

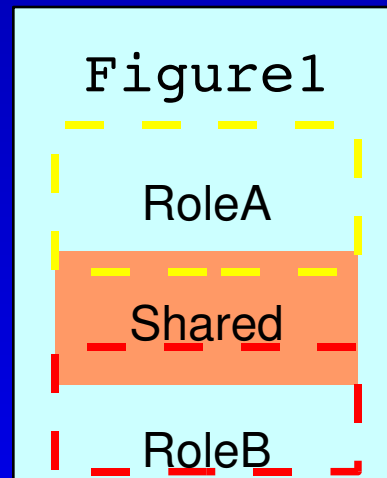
- Develop models to visualize and explain code
 - Based on patterns, composition (modalities), and UML
- Evaluate model performance with eye-tracking system on programmers

Programming Languages

- State typing in object language
 - Application to aspects pointcut language, rule-based systems
- Aspect language
 - Usability study of new Visitor pattern
 - Interaction detection with model checking

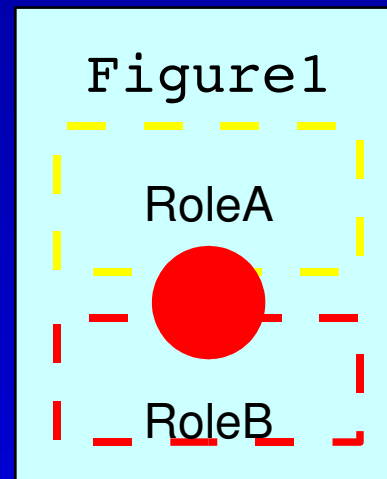
Sharing

- Goal: implementation reuse
 - Object: opportunity to reuse an existing implementation
 - Tangling of concern with new dependency
 - Structural aspect (or other reuse mechanisms) to promote modular reuse



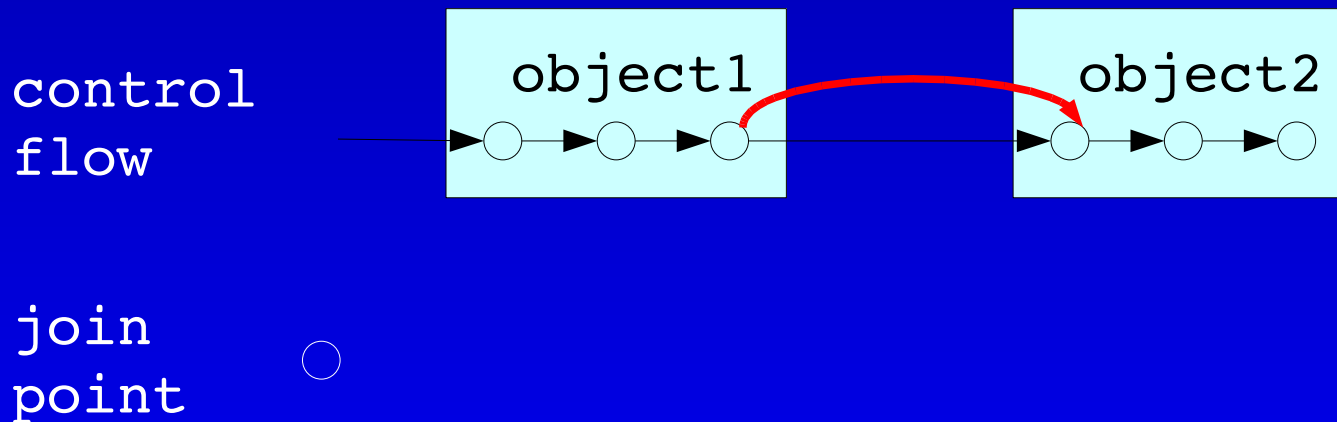
Interaction

- Patterns specifications incompatible in context
 - Invasive modification of implementation



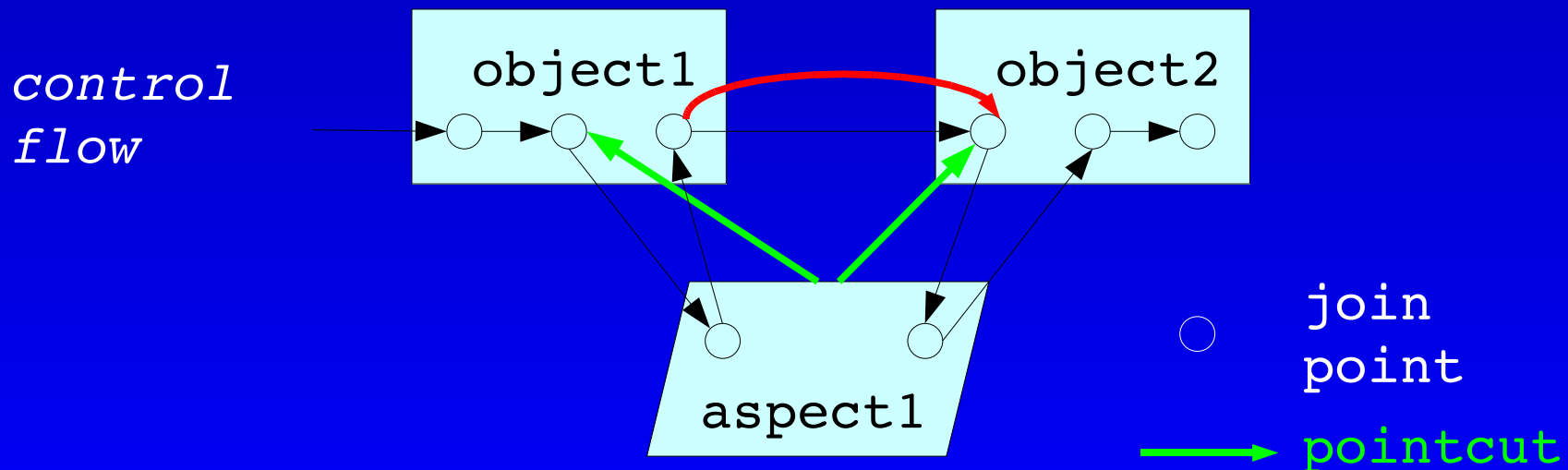
Aspect-Oriented Programming

- Modularize scattered and tangled code
- Behavioral aspect:
 - Execution = events trace = stream of join points



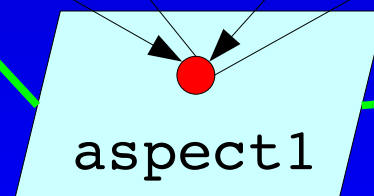
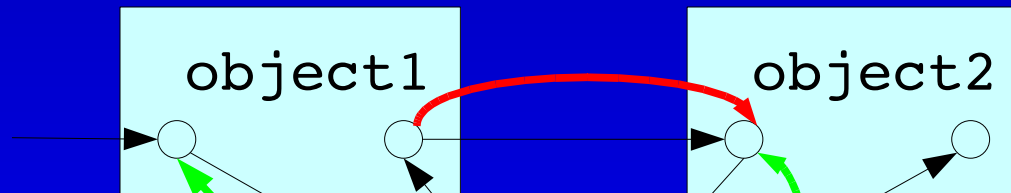
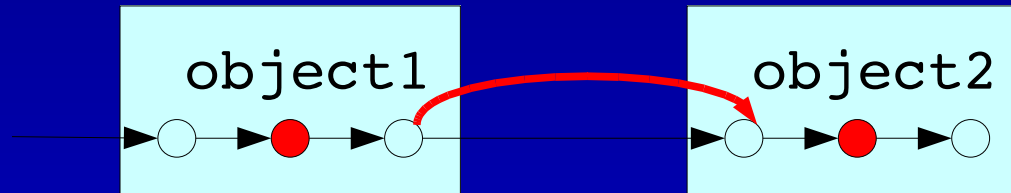
Aspect, Pointcut, and Advice

- Concepts
 - Pointcut: a declarative statement of join points to catch
 - Advice: response applied when join point captured
- Example of incremental aspect



Factoring Scattered Code

*control
flow*



○ join
point
→ pointcut