Trends for Separation of Concerns using Aspect Oriented Programming

Sushil K Bajracharya
sbajrach@ics.uci.edu

Motivation

- Techniques to manage complexity
  - Abstraction, Encapsulation, Functional decomposition …
    - Object Oriented Programming did a good job.
- OOP is great but what next ?
  - Dealing with *tangling* and *scattering* code.
  - Cross-cutting issues.
- A new view on *Separation of Concerns*

Overview

- Looks into the techniques that have been devised for dealing with cross-cutting issues in software development
- Presents AOP as a new paradigm for SOC
- Looks in the following aspects of AOP
  - Origin
  - Current state and alternate approaches
  - Future directions

Before AOP

- Separation of Concerns (SOC)
  - Only think about few things at a time… so *divide*
  - Design decisions not program flow should govern decomposition
  - You can *divide to conquer* but have to *reunite to rule*
- Extensions to OOP to achieve SOC
  - Adaptive Programming
  - Meta-level Programming
  - Composition Filters
  - Subject Oriented Programming

The birth of AOP

- AOP conceived at Xerox PARC
  - [KLM97] seminal paper gave a framework for AOP
    - Aspect Code + Component Code |–[*weaver*]→ Program
  - Goals:
    - Program Understanding
      - Separate cross-cutting issues from code
    - Efficient code
      - Optimizations (domain specific) result in tangled code
      - AOP gives untangled and yet optimized program
- AspectJ emerges as first generic AOP framework

[KLM97] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. *11th Europeen Conf. Object Oriented Programming,* 1997.

Inside AspectJ

- AspectJ components (v 1.1.0)
  - Aspect Language
    - Join Points
      - *well-defined point in the program flow*
    - Point Cuts
      - *select certain join points and values at those points*
    - Advices
      - *defines code that is executed when a pointcut is reached*
    - Introductions
      - *how AspectJ modifies a program's static structure, namely, the members of its classes and the relationship between classes*
    - Aspects
      - *defined in terms of pointcuts, advice and introduction*
  - Weaver
    - Ajc compiler

Besides AspectJ

- JBoss AOP
  - XML as aspect language
    - advices = interceptors = .java
      - intercept method invocations, constructor invocations, and field access
    - Introductions, metadata, pointcuts = XML
  - Bytecode manipulation to attach interceptors, uses own class loader
- Aspectwerkz
  - Aspect language
    - aspects, advices, pointcuts, join points and introductions
    - JavaDoc tags or XML
  - runtime weaving
    - can modify aspect at runtime, basing on the running states of your codes
- Others
  - Hyper/J
    - Different from AOP, based on n-dimensional separation of concerns principle, IBM
    - Hyperspace(Hypermodules(Hyperslices(Units)))
  - … many more
    - Jiazi, JAC, AspectC etc

Research Directions

- Limitations
  - AOP → implementation issue
  - AOP in the whole SW Engg. Life Cycle
- Aspect Oriented Architectures
- Modeling/Formalizing AOP
- Metrics for AOP
- Fluid AOP
  - "..Fluid AOP involves the ability to temporarily shift a program (or other software model) to a different structure to do some piece of work with it, and then shift it back"
- Beyond AOP
  - Naturalistic programming [LDLL03]

[LDLL03] Cristina Videira Lopes, Paul Dourish, David H. Lorenz, and Karl Lieberherr. Beyond aop: toward naturalistic programming. In *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 198–207. ACM Press, 2003.

Conclusions

- Outcome of this survey paper
  - Got familiar with different SOC techniques
  - Understood the fundamentals of AOP
  - Realized the evolution of AOP
  - Discovered interesting things to work on…
  - But the survey is still cursory
    - more to explore
- Questions ?