VM-level AOP

AOSD 2005

Sam Pullara

State of the art

- When is AOP applied to classes?
  - Compile time
  - Class load time
  - Hotswap
  - Runtime OO methods
- How is AOP applied to classes?
  - Bytecode modification
  - Dynamic proxies

Compile Time

- Pros:
  - Predictable runtime performance
  - IDE friendly (AJDT)
  - Bounded set of classes and effects
  - Most compatible with current JVMs
- Cons:
  - Must have all classes available
  - Cannot dynamically change aspects
  - Can break the license of some software
  - Large up front cost

Class Load Time

- Pros:
  - Aspect decisions are deferred to deployment
  - Can be used with most hot deployment systems
  - Can apply to code unavailable at compile time
- Cons:
  - Startup times can be arbitrarily large
  - Some software incompatibilities

Hotswap

- Pros:
  - Similar to class load time initially
  - Can be changed without reloading classes
  - Deployment time configuration
- Cons:
  - Startup time
  - Some JVM incompatibilities
  - Requires native code in < 1.4
  - Complicated command lines (-Xdebug or -Xjavaagent:AOP.jar, etc)

Runtime OO methods

- Pros:
  - Very dynamic, very compatible
  - Easy to understand for OO programmers
  - Implementations are much simpler
- Cons:
  - Often slow due to reflection usage
  - Supports many fewer pointcut types
  - Startup time is affected

Bytecode modification

- Pros:
  - Can be implemented on virtually any VM
  - Mostly invisible to the user
- Cons:
  - Can change the shape of the class
  - Some side effects like serialization are exposed
  - Class level code only
  - Typically only one agent / weaver is possible

Dynamic Proxies

- Pros:
  - Standard Java facility
  - Very simple
- Cons:
  - Recursion is not easy
  - Often inefficient
  - Need to be regenerated on each run

What is natural?

- Current AOP implementations are like previous generics implementations
- Debugging the current systems are often much more difficult than plain Java
- Without standard support, portability suffers between VMs and platforms
- Many target users will be unwilling to depend on a language feature thaf isn't "supported"

VM-level AOP

- Move the "weaver" to the VM
  - It's not really weaving anymore, since no bytecode modification would be needed
  - Debugging would be simplified as the debugging APIs would support AOP
  - Performance could be tied into the VM optimizer more efficiently
  - Memory usage would be better because double analysis and storage would not be needed

Other advantages

- Technical:
  - Supporting pointcut matching on reflective invocations would come naturally
  - The VMs are becoming managed containers, AOP could become the development model
- Community:
  - Adding AOP to the VM will legitimize it in some people's eyes

Current projects

- The JRockit team has a prototype
- Research on the Jikes RJVM is underway (SteamLoom)