AJEER:
Load-Time Aspect Weaving for the Eclipse Platform

Enabling programmers to combine Eclipse plugin technology and AspectJ

Martin Lippert (lippert@acm.org) - Download: http://www.martinlippert.org/

Eclipse Plugin Technology

> Build IDEs and business applications as a set of plugins

> Plugins are compiled separately as independent units with defined dependencies

> Plugins can be installed and uninstalled from a system without recompiling the rest of the system

> Plugins offer a component level above packages, but they do not provide any new language modularization mechanisms

AspectJ and AJDT

> AspectJ offers a language extension for Java to modularize concerns with aspect-oriented techniques

> Great IDE support via AJDT

> Aspects serve as a new modularization unit orthogonal to classes, but the AspectJ compiler produces Java-compliant bytecode

> The current AspectJ implementation needs to recompile (or at least reweave) all targets that might be affected by the aspects in the system

The Vision: Using Eclipse Plugins and AspectJ Together

> Modularize cross-plugin concerns into separate plugins
> Build aspects into a platform to let other plugins follow the rules

Just let developers combine the benefits from both worlds without limitations!!!

The Problem: Separate Compilation vs. Aspect Weaving

> AspectJ needs to weave the system to let aspects work correctly
> You would need to recompile the whole system when new aspects come in, old aspects are deleted or existing aspects change
> You would not be able to plug in new pluings into an existing system without recompilation

This cuts off major features of the Eclipse plugin technology

The Solution: AJEER: An AspectJ-Enabled Eclipse Runtime

Features weaving of AspectJ aspects into existing plugins at class-loading time (including dependency management)

-> Standard Eclipse plugins can add new aspects to the system via a new extension point (aspect-promotion)
-> Already existing or new plugins do not need to be recompiled
-> Separate compilation for plugins still possible

-> Open issues: performance and footprint of the weaving process (but weaving performance improves and AJEER implements caching of woven plugins)

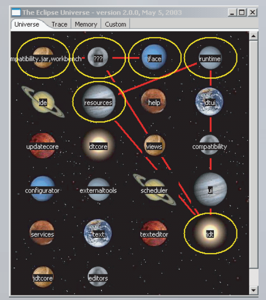
The Challenge: Dynamic Plugins and Aspects

> OSGi-based runtime allows plugins to be (un-)installed at runtime
> What happens if aspect-promoting plugins are (un-)installed at runtime???

Possible solution:
> AJEER has to take care to update active plugins at runtime (update mechanism of OSGi kernel)
> Therefore, AJEER needs to keep track of aspect dependencies and possible targets of aspects

Example 1: The Eclipse Monitor

Get a Demo!
Just ask for it !!!

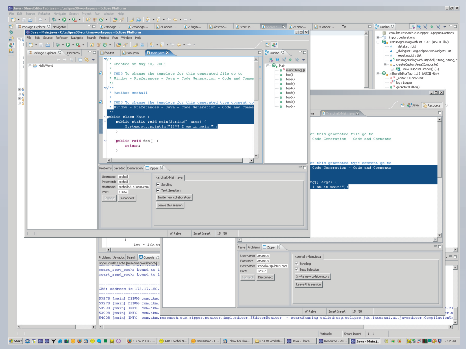


> AJEER-based reimplementation of Chris Laffra's Eclipse-Monitor

> Shows the internal behavior of Eclipse (method calls, object creations, plugin communication)

Example 2: The Zipper System for Replicated Application Sharing

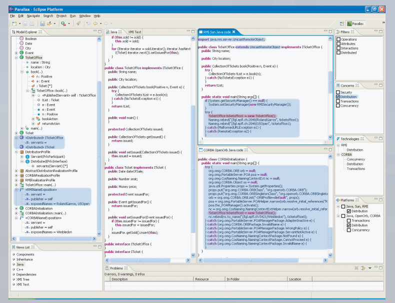
> Replicated Application Sharing for Eclipse
  - currently works with text-based editors
  - GEF in progress
> Uses Aspects for catching events, code archaeology (which events to catch?), and missing API workarounds



http://www.research.ibm.com/zipper
Steven Rohall
IBM T.J.Watson Research Center

Example 3: Parallax

> MDA Eclipse Support for Addressing *Middleware-Specific Crosscutting Concerns* Based on *Aspect-Promoting Plug-ins*



http://parallax-lgl.epfl.ch/
Raul Silaghi [rsilaghi@acm.org]
Software Engineering Laboratory
Swiss Federal Institute of Technology in Lausanne (EPFL)