

Scala Support in Eclipse

Monkey-patching the JDT for fun and profit?

Miles Sabin, Chuusai Ltd.
<http://www.chuusai.com/>

Outline

- JVM Languages Need Java-level Tools
- The Scala IDE for Eclipse
- History
- How to Move Forward?
- AspectJ and Equinox Aspects
- Who is Using This Approach?
- Retrospect and Prospects

The Need for Java-level Tools

- Many of the new JVM Languages offer significant productivity gains over Java, but ...
- Mainstream adoption requires improvements across the entire toolchain
 - A x2 productivity gain at the language level is irrelevant if there's a /2 loss at the tool level
- Especially so for IDEs

The Need for Java-level Tools

- A subset of JVM languages are also “Java-compatible”
 - Close source and binary mapping to Java
 - Scala, Groovy, JavaFX, AspectJ
- This holds out hope that many Java tools will Just Work, or work with minor adaptation
 - eg. Java annotation driven tools generally work as expected with Scala
- Additional promise of gradual migration

The Need for Java-level Tools

- Does this carry over to IDEs?
 - On the positive side there is much common infrastructure across languages
 - This is reflected in IDE extension APIs
 - Buts it's not enough to provide orthogonal support
 - Gradual migration demands that major features of the Java tooling be aware of new language artefacts
 - The big value features are the ones which are informed by language semantics
 - Typically private to IDEs language tooling

The Scala IDE for Eclipse

- The Scala tooling for Eclipse is an attempt to answer the preceding question in the positive
- The aim of the project is to achieve tooling which approaches that of the Eclipse JDT but with vastly less effort

History 1

- Started very early on in Scala's timeline (early 2005)
- Very simple IDE plugin
- Little functionality beyond basic syntax highlighting and build invocation
- Written in Java
- There was a least one other similar offering at the time: Scaliptor

History 2

- Announced December 2005, first release February 2006 (2.1.0)
- Rewritten in Scala
- Some semantic features acquired (eg. limited auto-completion)
- Even at this early stage requests for the ability to mix Java and Scala and JDT interop were coming in

History 3

- Announced June 2007, first release February 2008
- Attempted much deeper integration with the Scala compiler,
 - Interactive error reporting
 - Semantic highlighting
 - Incremental compilation
 - Dependency management
- Many hooks added to scalac

History 4

- Start of my involvement (May 2008)
- Prompted by very generous sponsorship by EDF Trading
- Primary goals
 - Ease Java/Scala migration
 - Mixed Scala/Java compilation in scalac
 - Mixed Scala/Java projects in Eclipse
 - Improved Eclipse stability and release process

History 4

- First commit to trunk in July 2008
- Results of the work first visible in 2.7.2.RC1 in August 2008
- Final 2.7.2 release in November
- Goals somewhat met,
 - Mixed Scala/Java enabled in scalac and Eclipse
 - Release process dramatically improved
 - However JDT integration limited and stability issues remain

How to Move Forward?

- Various non-options
 - Fork Eclipse?
 - It's open source, but too big and too rapidly changing. Maintaining a patch against it would be a huge effort
 - We want users to be able to install our tools into their existing eclipse environments
 - If several JVM languages fork the JDT which one wins?

How to Move Forward?

- Various non-options
 - Lobby for extensions to the JDT?
 - Andrew Eisenberg's patch has been languishing in Eclipse Bugzilla for 3+ years
 - JDT team understandably reluctant to see private implementation exposed as public API
 - Alternative JVM languages still a minority interest, so little business motivation to make the change
- Fortunately AOP provides us with a Plan B

AspectJ and Equinox Aspects

- AspectJ is a well-known AOP extension to Java
 - Allows behaviour of existing Java classes to be modified
 - Pointcuts specify slices through execution flow
 - “Advice” is code which is executed before, after or completely replacing original specified by pointcuts

AspectJ and Equinox Aspects

- Opinions are divided on AOP
 - There are some compelling use cases
 - Logging, error handling
 - These cases are typically passive and observational
 - Aspects can also actively modify behaviour, but
 - It's a gross violation of encapsulation
 - It can obscure the flow of execution
 - Violating encapsulation is a bad thing ...
 - ... except when that's exactly what you need to do!
 - In which case AOP is an industrial strength tool

AspectJ and Equinox Aspects

- A collection of aspects is effectively a patch
- AspectJ was used to retrofit the desired extensibility features to the JDT and expose them via public API
- The key modification:
 - The JDT's CompilationUnit is the entry point to its internal model, but it assumes Java source
 - An aspect can turn its constructor into a factory method

AspectJ and Equinox Aspects

- One more piece of the puzzle
 - How to get these aspects applied to an Eclipse Installation?
 - We can't require a custom Eclipse build or we're back to square one
- Enter Equinox Aspects
 - Work of Martin Lippert
 - An OSGi framework extension supporting the weaving of aspects into binaries at load time

Who is Using This Approach

- Unsurprisingly the AspectJ tooling for Eclipse (AJDT) was the first
- The Scala tooling picked it up very soon after
- The Groovy Eclipse tooling came next
- JavaFX is the most recent addition
- There has been significant collaboration and cross-fertilization across these projects

Retrospect and Prospects

- The results are a qualified success
- But the patch was premature,
 - Experience has shown that although the initial round of JDT modifications removed the biggest hurdles,
 - There are still many scenarios where Java source assumptions are made
 - The retrofitted public API maps JVM language constructs to the JDT's model in a way which best fits Java extensions rather than new languages

Retrospect and Prospects

- More cross-language collaboration needed
 - We need to draw on the experiences of the various groups and factor out common API and functionality
 - We need to encourage other languages to follow, esp. languages not so close to Java
- We need to reconsider the fork option
 - A common framework and shared effort makes this easier

Retrospect and Prospects

- We need to build a case for this work to be rolled into Eclipse
- That will only happen if the alternative JVM languages ecosystem flourishes
- Which brings us back to where we started, but hopefully with a clearer picture of the the landscape

JVM Languages Support in Eclipse

Monkey-patching the JDT for fun and profit?

Miles Sabin, Chuusai Ltd.
<http://www.chuusai.com/>