Speaking Your Language

Creating a custom DSL editor for Eclipse

Ian Graham
Background highlights

- 2D graphics, Fortran, Pascal, C
- Unix device drivers/handlers – trackball, display devices
- Smalltalk
  - Simulation
  - Sensor “data fusion” electronic warfare test-bed for army
  - CASE tools for telecommunications
- Java
  - Interactive visual tools for QC of seismic processing
  - Seismic processing DSL development tools
    - Defining parameter constraints and doc of each seismic process in XML
    - Automating help generation (plain text and HTML)
    - Smart editor that leverages DSL definition
- Now at Markit using custom language for financial risk analysis
Overview

- Introduction: Speaking Your Language
  - What is a DSL?
  - What is Eclipse?
  - Demo: Java Editor

- Eclipse architecture

- Implementing an editor
  - Implementing a simple editor
    - Demo
  - Adding language awareness to your editor
    - Demo

- Learning from Eclipse examples

- Resources
Introduction: Speaking Your Language

- **Goal**
  - illustrate power of Eclipse platform

- **Context**
  - widening use of Domain Specific Languages

- **Focus**
  - editing support for custom languages in Eclipse
Recipe Editor

Recipe Editor

# This is an example recipe
# for the "Text Editor Recipes" tutorial
# at EclipseCon2006

**Ingredients:**
- Cheese: 800 g
- White dry wine: 3 dl
- Lemon juice: 1 tbsp
- Garlic: 1 clove

**Preparation:**
1. Cut bread into pieces (approx. 1/2 inch)
2. Mix cheese with wine and lemon juice
3. Melt the cheese, stir continuously
4. Mix cherry schnaps and corn starch, add to cheese
5. Keep stirring until viscous
6. Add spices
7. Enjoy
What is a DSL?

- Domain Specific Language
  - a simplified language for specific problem domain

- Specialized programming language
  - E.g. Excel formula, SQL, seismic processing script

- Specification language
  - E.g. HTML, Regular expressions, XML, YACC grammar

- Transformation language
  - E.g. XSLT (Turing complete!)
Structured data without DSL

- XML often used
  - Advantages:
    - Many existing parsers
    - DTD/Schemas can define domain-specific constraints
    - Powerful XML editors that leverage schemas
  - Disadvantage:
    - Syntactic clutter unfriendly to humans

```
<recipe>
  <ingredients>
    <ingredient quantity="1" units="Kg">Bread</ingredient>
    <ingredient quantity="800" units="g">Cheese</ingredient>
    <ingredient quantity="300" units="ml">White wine</ingredient>
  </ingredients>
  <preparation>
    <step>Cut bread into pieces (approx. 1/2 inch)</step>
    <step>Mix cheese with wine and lemon juice</step>
  </preparation>
</recipe>
```
What is Eclipse?

- Depends on who you ask
  - State-of-the-art Java IDE
  - Multi-lingual IDE
    - Eclipse sponsored: C/C++, Javascript, FORTRAN, COBOL
    - Many others: PHP, Python, Ruby, Perl, Haskell, Scheme, Prolog, Scala, Groovy, and many more…
  - Extensible software tools platform
  - Framework for model-driven development (EMF)
  - Rich client application platform
    - IBM Lotus Expeditor and Symphony workplace products
    - NASA Maestro - Mars and Lunar robotic mission planning
Eclipse Java IDE
NASA JPL, Maestro & Ensemble

- From [Maestro Robot Interface Laboratory website](http://masrobotlab.org)
Demo: Eclipse Java Editor

- Syntax highlighting
- Hover help
- Error annotation
  - Vertical and overview bars
  - Problems View
- Code navigation
- Highlight occurrences
- Content assist (code completion)
- Refactoring
Eclipse Architecture
Platform vs. Extensible Application

- Eclipse is a platform with a small runtime kernel, which is an OSGi implementation.
Eclipse Plug-in Architecture

- **Plug-in – set of contributions**
  - Smallest unit of Eclipse functionality
  - Big example: HTML editor
  - Small example: action that creates zip files

- **Extension point** – named entity for collecting contributions
  - Example: extension point for workbench preference UI

- **Extension** – a contribution
  - Example: specific HTML editor preferences
Platform Text Architecture

- Eclipse Default Text Editor
- Eclipse Java Editor
- File Based Text Editor Framework
- Text Editor Framework
- Source Viewer Framework
- File Buffers
- Document Infrastructure
The **IDocument** Text Model

- **Sequence of characters**
  - Supports random access and replace
  - Event notifications via **IDocumentListener**

- **Sequence of lines**
  - Query by offset and line number

- **Positions**
  - Ranges that are adjusted to modifications
  - **IPositionUpdater** strategies handles overlapping changes

- **Partitions**
  - Slice the document into segments of the same **content type**
  - Language dependent – a simple semantic model
Document Partitioning

- Partitioning is always up-to-date
- Document provider ensures that the partitioning is installed
  - Documents support multiple partitionings
  - Document setup can also be managed by the file buffer manager (o.e.core.filebuffers.documentSetup)
  - File buffer document setup should only be used if the partitioning is considered of interest for non-UI clients and never contribute the default partitioning

- **SourceViewerConfiguration** needs to know the partitioning and supported partition types.
Implementing an Editor
Implementing a simple editor

1. Subclass an existing editor
   - AbstractTextEditor
     - Find/Replace, hyperlinks
   - AbstractDecoratedTextEditor
     - Adds ruler, line numbers, quick diff, configurable preferences

2. Define a document provider
   - Creates/obtains document
   - Defines partitioning

3. Define a source viewer configuration
   - Central class for customizing editor
Source Viewer Configuration

- Bundles the configuration space of a source viewer
  - Presentation reconciler (syntax coloring)
  - Content assist
  - Hovers
  - Formatter
  - ...

- Many features can be provided separately for each *partition type*
Syntax Highlighting: Damage & Repair

- PresentationReconciler
  - IPresentationDamager: define dirty region given a text change
  - IPresentationRepairer: recreate presentation for dirty region
  - DefaultDamagerRepairer does both, based on a token scanner

- ITokenScanner: parse text into a token stream
  - RuleBasedScanner uses simple rules
Set the Rules

```java
private ITokenScanner getRecipeScanner() {
    RuleBasedScanner scanner = new RuleBasedScanner();
    IRule[] rules = new IRule[4];
    rules[0] = createSectionTitleRule();
    rules[1] = createQuantityRule();
    scanner.setRules(rules);
    return scanner;
}
```

**Ingredients**
- Cheese: 800 g
- White dry wine: 2 4 fl
- Lemon juice: 1 tbsp
- Garlic: 1 clove

**Preparation:**
1. Cut bread into pieces
2. Mix cheese with wine a
3. Melt the cheese, stir
4. Mix cherry schnaps and
Demo: Simple recipe editor
Adding language awareness to your editor
Modeling your document

- Must design a model of your language
  - E.g. Recipe, IngredientsSection, Ingredient, PreparationSection, Step

- Need parser to build model from document
  - Roll your own (recipe editor does this, but out of scope of talk)
  - Generate from a grammar using parser generator such as ANTLR

- Implement a reconciler strategy that invokes parser
  - Reconciler invoked by Eclipse after a typing break
  - Runs in background thread
  - Reports errors as annotations on the document
Model Challenges

- **Scalability**
  - Probably can’t afford AST of every file, so model definitely of value for reducing scale
  - Scalability definitely an issue for general purpose languages
  - May be important even for DSL if need scope beyond file being edited
  - To scale well, model would likely require proxies and caching of model components so entire model not required to be in memory

- **Saved state vs. dirty state**
  - Lighter weight errors determined during parsing typically refer to modified in-memory document
  - Error markers typically persisted based on saved state of document
  - Files whose models reference each other need consistent way of managing error annotation and persistence
Using your model to support power editing

- Navigable Outline view of document structure
- Code folding
- Context-aware hover help
- Content-assist (templates and code completion)
  - Does not require model, but model necessary for smart context-aware code completion
- Semantic highlighting
  - Colouring member, parameter, and local variables differently
  - Occurrences of a selected variable
- Refactoring
Demo: Language-aware recipe editor
Learning from Eclipse Examples
Downloading and Installing Eclipse

- Install recent Java JRE, or preferably JDK
  - Oracle JDK or OpenJDK (GCJ Gnu Java compiler won’t work)
  - IBM JDK only for IBM machines (does BIOS check)

- Obtain Eclipse “Classic” package (eclipse.org)
  - Best starting point for plug-in developers
    - includes Platform and JDT source -- great learning resource
  - Platform specific
    - Linux/MacOS/Windows
    - 32-bit(x86)/64-bit(x64)

- Extracts to directory named eclipse
  - I rename to current release, e.g. eclipse_3.7.2
  - Create desktop shortcut or alias to eclipse executable

- Run eclipse
Eclipse Samples and Templates

- **Generating projects from Samples**
  - On Welcome screen, select **Samples**, then **Java editor**
    - Dialog will prompt to download Samples
  - Additional sample projects can be generated by
    - File → New → Project...
    - Expand Code Samples → Workbench

- **Generating projects from built-in templates**
  - File → New → Project...
  - Select Plugin Project → Next
  - Enter project name (ca.ab.cuug.xmleditor) → Next
    - Choose UI and/or RCP → Next
    - Choose template → Finish
Eclipse Java Editor – the richest example

- Import Java Editor into your workspace to browse the code easily
  1. File → Import → Plug-in Development → Plug-ins and Fragments → Next
  2. Select **Binary projects with linked content** (minimizes space)
  3. Filter by “jdt.ui” and add to “To Import” → Finish
  4. Ctrl-T JavaEditor will find it.

- Useful example is ToggleCommentAction.java
  - similar implementation could be useful in many languages

- Big and complex – not necessarily the easiest example to begin with!
Resources

- Eclipse Samples as accessed from Welcome Page
  - Simple java editor

- Eclipse plug-in templates using File → New Project…
  - Simple XML editor

- Recipe editor source from EclipseCon 2006 tutorial


- Numerous Eclipse articles, particularly corner at articles at
  - [http://eclipse.org/resources](http://eclipse.org/resources)
Resources – Excellent Current Books

  - Seminal book on developing Eclipse plug-ins, actively updated

  - Guide to developing non-IDE rich client applications on the Eclipse platform

- **The Definitive ANTL Reference: Building Domain-Specific Languages** by Terrence Parr
  - Guide to developing parsers using ANTLR 3.0, progresses from introductory use for simpler languages to deeper discussions for trickier parsing challenges
Resources – Older Books

- *Contributing to Eclipse: Principles, Patterns, and Plug-Ins* by Erich Gamma and Kent Beck
  - My favourite Eclipse book, but old (2003) and not a reference
  - People love it or hate it: if you don't enjoy philosophical aspects of software development, you'll probably hate it. If you want all the code itself to work as is
  - Primarily walks you through a simple but broad-reaching tutorial using a test-driven development approach to develop tightly integrated TDD-supporting tools on top of JUnit.
  - Presents an excellent set of Eclipse “House Rules” which you should absorb online if you don’t get the book

- *Eclipse 3.0 FAQs* by John Arthorne and Chris Laffra
  - The book was a great reference but is now outdated and all the updated material is now available on-line.
Credits

- NASA JPL Maestro screenshot taken from the Maestro Robot Interface Laboratory website
  - http://www-robotics.jpl.nasa.gov

- A number of much appreciated slides extracted from Text Editor Recipes by Tom Eicher presented at EclipseCon 2006.