Enhancing Support for Languages in Eclipse – The LDT Project

Dr. Tim Wagner
BEA Systems, Inc.
Background

- **BEA WebLogic Workshop**
  - Award-winning IDE focused on J2EE and Web Standards
  - Strong innovation in compiler technology and language definition
- **Eclipse IDE**
  - Open source platform with strong adoption and market penetration
  - Language support is generally siloed; strong interest in bridging them
- **This month**
  - BEA joins Eclipse as Strategic Developer
  - BEA joins Web tools platform leadership team
  - **BEA proposes LDT project in Eclipse...**
Why Did BEA Propose A Technology Project?

- Workshop’s compiler technology (“Javelin”)
  - Multi language plugin design
  - Focus on APIs for describing languages and language services needed for source and design views
  - Cross and nested language R&D
  - BEA relies on language technologies for its product work and ease-of-use feature set

- Product experience: used in multiple, Enterprise-quality products
  - Workshop, Integration tooling, Portal development, Liquid Data, …
WebLogic Workshop and Eclipse IDE Architectures

Eclipse

UI and Lang

Platform

Workshop

Platform

Compiler

UI

Lang
Themes

- Separate model (language) from view (editor UI)
- Break monolithic language boundary / let 3rd parties contribute
  - Java → apt/mirror (JSR 175)
  - XML → BPELJ, ant, …
- Connect languages together
  - Your language will be used in places you never dreamed of…
- Sand off the hard edges for new languages
Agenda

- Intro
- Use Cases
  - Metadata-driven development in Java ("apt/mirror")
  - Languages with nested Java: JSP (or BPELJ or …)
  - XML Schema-to-Java translation and XSD Project model
  - Lowering the barrier to new languages and language services
- LDT Project Information
@createInterface(PUBLIC)

class Foo implements IFoo {
    public void method1() {...}
    private void method2() {...}
}

public interface IFoo {
    void method1();
}

Metadata-Driven Development
Command line-based Metadata Development Tool

- 5.0 JDK introduces
  - “apt” tool – a javac-like tool that processes annotations
  - Anyone can write a 3rd party annotation factory and processor
  - com.sun.mirror: Reflection-like type system for *build-time* use
  - com.sun.apt: facilities for doing something useful with annotations once you’ve seen them

- So users can write processors, define annotations, and create new metalanguages...but what about their IDE experience?
Extending the Java IDE Experience

```java
@createInterface(PUBLIC)
public class Foo {
    private void method1() {...}
    private void method2() {...}
}
```

@createInterface cannot be used on classes that have no public methods
Interoperability with Sun’s APT

apt
apt
Processor
Processor
Processor
1.5 JDK
Eclipse IDE
Command Line
JDT
JDT
How APT Processors Work

- Build-time type system (mirror)
- Record problems, create files (apt)
Integrating Support into JDT

- Goals: direct mods to JDT, extend implementation and public API
- Mirror type wrappers around native JDT bindings
- Two dispatch paths
  - AST reconciliation
  - Builders
- Factory discovery (APT native style vs. new JDT extension point)
- Processor hierarchy (vanilla vs. Eclipse-aware)
Agenda

- Intro
- Use Cases
  - Metadata-driven development in Java ("apt/mirror")
  - Languages with nested Java: JSP (or BPELJ or …)
  - XML Schema-to-Java translation and XSD Project model
  - Lowering the barrier to new languages and language services
- LDT Project Information
Nested Languages: Enhance JSP, Enable BPELJ

- Create a framework for nested languages to communicate
  - Common data structures (token streams, ASTs)
  - Algorithms (control transfer, contextual information)
- Public APIs to mark the language for a specific region (not just a color)
  - Language → syntax coloring, *but also drives dispatch!*
- Support composition, unescaping, and completion of fragments into a translated file
- Automate migration of diagnostics and other UI experience back and forth between original source and generated material
- Customized dispatching in XML
  - BPELJ can contain schemas, WSDL, XQuery, Java, …
Common Data Structures: Token Streams

- Provides standard navigation APIs
  - Supports syntax highlighting, including nested language regions
- Standard “nested region” concept
  - Enables language-neutral discovery
Common Data Structures: ASTs

- Provides standard navigation APIs
  - Supports document structure view
  - Standard “nested region” concept
    - Enables language-neutral discovery
Thread-Safe Token Stream Implementation

- Time to create new version: $O(\lg N + M)$
- Additional space to represent new version: $O(\lg N + M)$
- Time to map offset to token: $O(\lg N)$
- Time to iterate through all tokens: $O(N)$
Technology Benefits

- Fast – new version can be produced on UI thread
- Incremental lexers can easily generate maximal reuse
- Dual use – result read by parser and UI clients
- Exploit sharing – IDE can maintain multiple versions efficiently
- Trivial lifetime management – normal Java GC handles it!
- Future: Same concept can be applied to incremental parsing / ASTs
[Nested] Language APIs

```java
public interface IAnalyzer {
    Reader getReader(InputStream input);
    lex(ITokenStreamBuilder, IOuterParseArg);
    incLex(...);
    parse(IAstBuilder, IParseEnv, IOuterParseArg);
    ICheckInfo check(IAst, IParseEnv, IOuterParseArg);
    gen(IAst, ICheckInfo, ICheckEnv);
}

public interface IParseEnv {
    markNestedRegion(ILanguage, IToken, int relStart,
                      int size, ITransducers[], IOuterParseArg);
    <include file processing APIs>
}

public interface ICheckEnv {
    createSourceFile(ILanguage, path, String content);
    ICompositeFileBuilder createSourceFile(ILanguage, path);
    addDependency(IFile);
}
```
Technologies Required

- Ability to create source files in arbitrary languages
- Generalized notion of “this file depends on this other file”
  - Language artifacts (Java method call)
  - File-to-file existence (URL target-missing warnings)
  - File-to-file content (JSP or C++ includes)
- Generalized work queue and “out of date” propagation
  - Cannot limit this to a single language’s silo
  - Inter-language cycles may exist; requires FPI stasis check
- Buffer management…
Two-way Buffer Mapping

```jsp
<% int i = "asdf";%>
<% if (i &gt; 4) ...%>
```

```java
int i = "asdf";
if (i &gt; 4) ... 
```
Eclipse Integration

- Ideal use case for LDT...
- Relationship to SSE work?
  - Can we separate presentation and language APIs for mutual benefit?
- “Wish List” for existing JSP editor...
  - Attribute analysis
  - JSP 2.0 expressions treated as a fully featured nested language
  - Full support for HTML warnings in addition to Java
  - “Live” diagnostics and completion for custom tag attributes
- …would all be facilitated by increased language awareness.
- Need for clear, clean, public APIs at this level
Agenda

- Intro
- Use Cases
  - Metadata-driven development in Java ("apt/mirror")
  - Languages with nested Java: JSP (or BPELJ or …)
  - XML Schema-to-Java translation and XSD Project model
  - Lowering the barrier to new languages and language services
- LDT Project Information
Problem: Support XSD-to-Java bindings within the IDE

- Multiple technologies (EMF, XBeans, JAXB) for mapping XSDs to Java
  - Relate the type systems of the two languages together

- Want full-fledged IDE support
  - Editable schema files (including cross-file checking)
  - Editable client Java files
  - …with a live, dynamic connection between them – no “compile” step!

- Challenges
  - Large number of generated files (unlike JSP’s 1:1 case)
  - Need schema project model that matches Java visibility rules in IDE
  - Java type provenance and refactoring/find-uses implications
What We’ve Tried that Works…

- Supporting a language-neutral project [inheritance] model
  - Manage schema path like Java class path
  - Same visibility rules are key to understandable behavior

- Providing the same QoS for XSD files that Java has
  - Same gestures for find-uses, goto-def, refactoring

- Connecting XSD definitions to XML instance doc uses *and* to Java client uses (via the generated types)
...and Challenges We’re Still Researching

- How much can we do in memory?
  - Try to make incremental schema modification extremely lightweight
  - ...but can’t keep unlimited Java source around
  - Retain type names, hashcodes; check vs. build issues

- Can we facilitate incremental schema compilation the same way that JDT supports incremental Java compilation?
  - Many schemas in a single project represent a bottleneck that harms the incremental editing experience
  - Deep project hierarchies with schemas in them represent a CPU drain
Agenda

- Intro
- Use Cases
  - Metadata-driven development in Java ("apt/mirror")
  - Languages with nested Java: JSP (or BPELJ or ...)
  - XML Schema-to-Java translation and XSD Project model
    - Lowering the barrier to new languages and language services
- LDT Project Information
[Nested] Language APIs

```java
public interface IAnalyzer {
    Reader getReader(InputStream input);
    lex(ITokenStreamBuilder, IOuterParseArg);
    incLex(...);
    parse(IAstBuilder, IParseEnv, IOuterParseArg);
    ICheckInfo check(IAst, IParseEnv, IOuterParseArg);
    gen(IAst, ICheckInfo, ICheckEnv);
}

public interface IParseEnv {
    markNestedRegion(ILanguage, IToken, int relStart, int size, ITransducers[], IOuterParseArg);
    <include file processing APIs>
}

public interface ICheckEnv {
    createSourceFile(ILanguage, path, String content);
    ICompositeFileBuilder createSourceFile(ILanguage, path);
    addDependency(IFile);
}
```
Language-level Extensibility

- Command Line
- IDE
- Analysis Tools

Language Service Level ("compiler bus")

- Java
  - JWS
  - JPF
- JSP
- XML
  - Schema
  - ant
- Fortran
Case Study: Java “.properties” Files

- Added new language to describe Java’s “.properties” files to Workshop…
- ~500 lines, including boilerplate and comments
- NO CHANGES TO THE IDE!

Progress
- Hour 1: syntax highlighting (lexer)
- Hour 2: document structure view (parser)
- Hour 3: diagnostics – duplicate warnings, etc. (checker)
What Makes this Possible?

- Language-neutral source editor
  - Can we “clone” the JDT’s UI and make it an “LDT” UI???
- Standard data structures (token streams, ASTs, problem lists)
  - Don’t recreate them for each new language; inhibits both cross-language linking *and* increases cost of new languages
- Standard APIs and dispatch mechanisms
  - lex, parse, check, etc.
- Clear definition of *language* services
  - …and clear separation between language and presentation APIs
- Reusable core compiler services
  - Multi-threaded work queue and job definition
  - Dependency matrix and analysis routines
A Litmus Test

Can we provide a batch compiler that handles all available languages without adding anything but the command line driver itself?*

*and the SWT classes aren’t needed to run it.
### Converting Existing Language Analyzers to an IDE

<table>
<thead>
<tr>
<th>External Tool</th>
<th>JDT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lex &amp; Parse &amp; Check</td>
<td>Lex &amp; Parse &amp; Check</td>
</tr>
<tr>
<td>Low High Responsiveness</td>
<td>Low Low Dev Difficulty</td>
</tr>
<tr>
<td>Low High Check</td>
<td>Low Low High High</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JDT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incremental Lex</td>
</tr>
<tr>
<td>Lex</td>
</tr>
<tr>
<td>Lex</td>
</tr>
<tr>
<td>Lex</td>
</tr>
<tr>
<td>Lex</td>
</tr>
<tr>
<td>Lex</td>
</tr>
<tr>
<td>Lex</td>
</tr>
</tbody>
</table>

Enhancing Support for Languages in Eclipse – The LDT Project
© 2005 by Tim Wagner; made available under the EPL v1.0 | March 3rd, 2005
Going Further: IDE-requested Language Analysis

- Identifier information
  - Extended description for rollover, hover, or ctrl-hover
  - Goto definition (file, offset)
  - Find uses (forward/reverse dependency views and navigation)

- Property views
- Pop-up editors for nested language regions
- Static and dynamic enclosing context
- Refactoring

…note that none of this is specific to a particular language!
How can an editor be language neutral?

**Diagram:**

- **Language-specific data structures**
  - Language Extension
  - Language Layer
  - UI Layer
  - UI Extension

- **Language-independent data structures**
“Conclusions”

- Eclipse has a well-developed IDE platform, and BEA sees the strategic benefit of adopting it.

- We also see room for increased support for language-level services, and have significant background in this area from our past.

- Would like to do this R&D in the open source community, rather than continue proprietary compiler/language infrastructure development.

- ...still more questions than answers...
Agenda

- Intro
- Use Cases
  - Metadata-driven development in Java ("apt/mirror")
  - Languages with nested Java: JSP (or BPELJ or ...)
  - XML Schema-to-Java translation and XSD Project model
  - Lowering the barrier to new languages and language services
- LDT Project Information
LDT Community

- Project proposal: http://www.eclipse.org/proposals/eclipse-ldt
- Newsgroup: eclipse.technology.ldt
- Mailing list (won’t exist until proposal acceptance): ldt-dev
- Reaching BEA people:
  - Tim Wagner, acting lead: twagner@bea.com
  - Tim Hanson, acting architect: thanson@bea.com
  - Gary Horen, program management: ghoren@bea.com