Introduction to Eclipse, Creating Eclipse plug-ins and the Overture editor

David Holst Møller
Engineering College of Aarhus
Agenda

- Part I – Introduction to Eclipse and Eclipse Plug-ins
- Part II – The Overture IDE
A bit of history

- Industry leaders formed the initial eclipse.org Board of Stewards in November 2001 (Borland, IBM, MERANT, QNX Software Systems, Rational Software, Red Hat, SuSE, TogetherSoft and Webgain)
- Originally an IBM project developed by OTI, the aim was to develop a platform which could be used for integrating all their tools in a common software base.
- In January 2004, the Eclipse Foundation was created.
- Annual release since 2006
Introduction to Eclipse

- An Integrated Development Environment (IDE)
- A Rich Client Platform
- Platform independent
Terms

• Workbench
  • Resources
    - Projects
    - Folders
    - Files
  • Perspectives
    - Views
    - Editors
```java
public class Main {
    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
    }
}
```
The Eclipse Java IDE

- Java development tooling (JDT)
- Competing with Netbeans (Sun) and Jdeveloper (Oracle)
Features in Java environment

- Outline
- Code Completion
- Team development (CVS - integrated)
- Refactoring
- Debugging
- Error
- Syntax
- Etc.
Eclipse Plug-in Architecture

- Designed for plug-ins
- Far superior range of plug-ins.

[Google Trends chart showing search volume for eclipse plug-in, netbeans plugin, and jdeveloper plugin from 2004 to 2009.]
Plug-in Terms 1

• A **plug-in** in Eclipse is a component that provides a certain type of service within the context of the Eclipse workbench.

• A **feature** is a way of grouping and describing different functionality that makes up a product. Grouping plug-ins into features allows the product to be installed and updated using the Eclipse update server and related support.
Plug-in Terms 2

- **Extensions** are the central mechanism for contributing behaviour to the platform.

- **Extension points** define new function points for the platform that other plug-ins can plug into.

- Except for a small kernel known as the **Platform Runtime**, all of the Eclipse Platform's functionality is located in plug-ins.
Extension loading

1. Query the registry for registered compliant extensions
2. Present extensions based on markup
3. Load classes only when the extension is needed
Example: Preference page

- Plug-ins may contribute preference pages
- All preference pages are assembled and categorized in the Preferences dialog
- How is the Preferences dialog created?
- How and when is a particular preference page created?
Create the Preferences Dialog - 1/3

- The UI plug-in provides the org.eclipse.ui.preference Pages extension point
- The UI plug-in first creates an empty Preferences dialog
- Now the dialog needs to be populated...
The UI plug-in queries the extension registry for all `org.eclipse.ui.preferencePages` extensions.

The preference page index is then generated using the XML markup only:

- Names for available preference pages are displayed in the tree using the `name` attribute.
- The `category` attribute is used to categorize the pages.
Create the Plug-in Development Preference Page (3/3)

- When the Plug-in Development preference page gets selected, the UI plug-in asks the extension registry to load and instantiate the Java class specified by the class attribute of the corresponding extension.
- The class gets loaded and the preference page gets created.
  The plug-in providing that extension (i.e. the org.eclipse.pde.ui plug-in) may then get activated, if it’s not already active.
Defining Plug-ins

- Eclipse Workbench
  - Lazy loaded on demand
  - Packaged from Source Jar
    - Java source files

- Parsed and cached in the Eclipse plug-in registry
  - Plugin.xml
    - Externalized strings are read from
    - Plugin.properties

- MANIFEST.MF
<?xml version="1.0" encoding="UTF-8"?>
<eclipse version="3.2">
<plugin>
  ...
  <extension point="org.eclipse.ui.perspectives">
    <perspective
      class="org.overturetool.ui.OverturePerspective"
      icon="icons/vdm16.png"
      id="org.overturetool.ui.OverturePerspective"
      name="%OverturePerspectiveName"/>
  </extension>
  ...
</plugin>

OverturePerspectiveName=Overture
pluginName=Dynamic Languages Toolkit Overture UI
pluginProvider=Eclipse.org
...

Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: %pluginName
Bundle-SymbolicName: org.overturetool.ui; singleton:=true
Bundle-Version: 1.0.0.qualifier
Bundle-Activator: org.overturetool.internal.ui.UIPlugin
Bundle-Vendor: %pluginProvider
Bundle-Localization: plugin
Require-Bundle: org.eclipse.ui,
  org.eclipse.core.runtime,
...
Bundle-ActivationPolicy: lazy
Export-Package: org.overturetool.internal.ui,
  org.overturetool.internal.ui.editor,
...
Bundle-RequiredExecutionEnvironment: J2SE-1.5
package org.overturetool.ui;

//IMPORTS...
public class OverturePerspective implements IPerspectiveFactory {

    public void createInitialLayout(IPageLayout layout) {
        ....
    
    // views – standard workbench
    layout.addShowViewShortcut(IPageLayout.ID_OUTLINE);
    layout.addShowViewShortcut(IPageLayout.ID_PROBLEM);
    layout.addShowViewShortcut(IConsoleConstants.ID_CONSOLE);
    layout.addShowViewShortcut(navigator);
    layout.addShowViewShortcut(IPageLayout.ID_TASK_LIST);
    layout.addShowViewShortcut(IProgressConstants.PROGRESS_VIEW_ID);

    // new actions
    layout.addNewWizardShortcut("org.overturetool.internal.ui.wizards.OvertureProjectWizard");
    layout.addNewWizardShortcut("org.overturetool.internal.ui.wizards.OvertureFileCreationWizard");
    layout.addNewWizardShortcut("org.eclipse.ui.wizards.new.folder");
    layout.addNewWizardShortcut("org.eclipse.ui.wizards.new.file");
    layout.addNewWizardShortcut("org.eclipse.ui.editors.wizards.UntitledTextFileWizard");

    }
}
The Overture IDE
VDM development today

- No integration of interpreter and editor
- Cumbersome development process
- Poor navigation
- No intellisense
- No support for refactoring
- Debugging support is good, but it could be better
- Test coverage measurement – but rather bothersome
- Support for Code Generation
- Support for UML $\rightarrow$ VDM $\rightarrow$ UML
Vision for the Overture IDE

- **Integrated** Development Environment
- The IDE should offer features known from other IDEs. Such as:
  - Syntax highlighting
  - Easy navigation
    - Files, definitions, errors, warnings
  - Refactoring
  - Advanced debugging features
- Test coverage measurement
- Code generation and UML to and from VDM
- High Extensibility
- The only tool needed for all VDM development - including all dialects
Introduction to the Overture Editor
class ATMCard is subclass of BankAccount

-- instance variables

  cardnumber : seq of digit;
  expiry : digit * digit * digit * digit;

inv (let mk_ \( m_1, m_2, y_1, y_2 \) = expiry
   in m_1 * 10 + m_2 <= 12) and
len cardnumber >= 8

operations

-- TODO sss

public

GetCardnumber : (x) \Rightarrow seq of digit
GetCardnumber () \Rightarrow
startSort () {
    data := [2, 4, 67, 70, 3, 60];
    SelectionSorter(1);
    temp := print();
}

diego := f(3);
	null

private print () : bool
print () ==
{ return io().echo("Hello world: \n");

temp := io().echo(data());

warn 5012: Recursive function has
warn 5008: '1' in 'MergeSort' /'hor
warn 5007: Duplicate definition
warn 5007: Duplicate definition
warn 5000: Definition 'l' not used
warn 5012: Recursive function has
warn 5012: Recursive function has
warn 5012: Recursive function has

Vision for the Overture IDE revisited

- **Integrated Development Environment** 😊
- The IDE should offer features known from other IDEs. Such as:
  - Syntax highlighting 😊
  - Easy navigation 😊
    - Files, definitions 😊, errors, warnings
  - Refactoring 😞
  - Advanced debugging features 😞
- Test coverage measurement 😞
- Code generation and UML to and from VDM 😞
- High Extensibility 😊
- The only tool needed for all VDM development - including all dialects 😞
Overture IDE Implementation
The project has no documentation.
Debug Protocol

- A common debugger protocol for languages and debugger UI communication
- Part of the DLTK
Extras...
OSGi

- object-oriented (instead of procedural)
- Modular (as opposed to unmodular)