eclipse CON 2005

Addressing UI Scalability in Eclipse

What to Do When Your Application is Too Complicated

Kimberly Horne
Eclipse Platform UI Team
kim_horne@ca.ibm.com

 Wednesday March 2nd, 2005

Agenda

- What is the problem?
- What we already have
  - Activities (aka Capabilities)
  - Contexts
- What are we lacking?
- What's next?
- Alternative uses for Activities

 |

The Problem: Users Perspective

- Some products ship with many distinct features
  - getting started can be hard for a new user
  - user.brain.OutOfMemoryError
- Most users require only the functionality specific to their role(s)
- Some functionality is not relevant to what they are currently doing
- Needs to easily switch gears and get into the task they're trying to accomplish

 |

The Problem: Plug-In Developers Perspective

- Plug-in developers typically work in a vacuum - need to worry about their own functionality
- Shouldn't have to be aware of how their plug-in will fit into larger products
- Should respect scalability mechanisms if they provide extension points for other plug-ins to extend

 |

The Problem: Product Developers Perspective

- Responsible for making sure collections of plug-ins play nicely with one another
- Because they understand how plug-ins will be used in relation to one another they are the best candidate to determine functionality grouping
- It's a dirty job but someone has to do it

 |

Progressive Disclosure: Activities

- Activities – what the user has expressed interest in doing
- Product managers find logical groupings of functionality within their product and define activity sets
- Users can be prompted on installation for features they'd like to be visible initially
- Functionality remains hidden until breadcrumbs (trigger points) are discovered

 |

Progressive Disclosure: Activities

- Activities are bound to contributions via patterns
- UI contributions are given identifiers of plug-in ID + '/' + local ID
- Patterns follow the java.lang.regex regular expression syntax

Example:

```
11  <extension
12        point="org.eclipse.ui.activities">
13     <activity
14           description="Allows for development of Things."
15           name="Thing Development"
16           id="my.feature.thingDevelopment"/>
17     <activityPatternBinding
18           activityId="my.feature.thingDevelopment"
19           pattern="my\.thing\.plugin/.*"/>
20  </extension>
```

The above activity and pattern will match all UI contributions from the plug-in "my.thing.plugin"

 |

Progressive Disclosure: Activities

We currently filter:

- New Wizards (optionally unfiltered)
- Show View menu
- Views automatically opened in new perspectives
- Open Perspective menu (optionally unfiltered)
- Toolbar/Menu contributions
- Property/Preference pages
- Editors
- Search
- Help

Progressive Disclosure: Activities

- For eclipse, all activities are enabled by default – we're small beans
- Trigger Points:
  - New wizards
  - Import/Export wizards
  - Detection of project natures
  - Opening perspectives

 |

Progressive Disclosure: Activities

- Participating in activity filtering is made easy by helpers offered by the Workbench
  - `IPluginContribution` – simple interface that provides methods to determine originating plug-in and local identifiers
  - `WorkbenchActivityHelper` – offers static methods for determining enablement of `IPluginContributions`

Example:

```
for (Iterator i = myCollectionOfExtensions.iterator(); i.hasNext(); ) {
    IPluginContribution myExtension = (IPluginContribution) i.next();
    if ( ! WorkbenchActivityHelper.filterItem(myExtension))
        myUI.addMyExtension(myExtension);
}
```

The above code would filter out all of your extensions that are currently disabled based on activity enablement.

 |

Progressive Disclosure: Activities

- Adding a trigger point is very similar
- Before using an extension ask the workbench whether it is allowed – this may prompt the user for whether to enable activities. If they decline, the operation is cancelled.

Example:

```
public void useExtension(MyExtension extension) {
    if (WorkbenchActivityHelper.allowUseOf(extension))
        doUseExtension(extension);
}
```

Assuming that `MyExtension` implemented `IPluginContribution` the above code would check to see if it should be available based on activity enablement. If it is not, the user will be consulted.

 |

Progressive Disclosure: Contexts

- Contexts – what the user is currently doing
- Early implementation – used to solve specific problems
  - Used for command and key binding resolution
  - Used for debug view management
- No general usage as of 3.1

 |

Progressive Disclosure Examples: Java Tooling

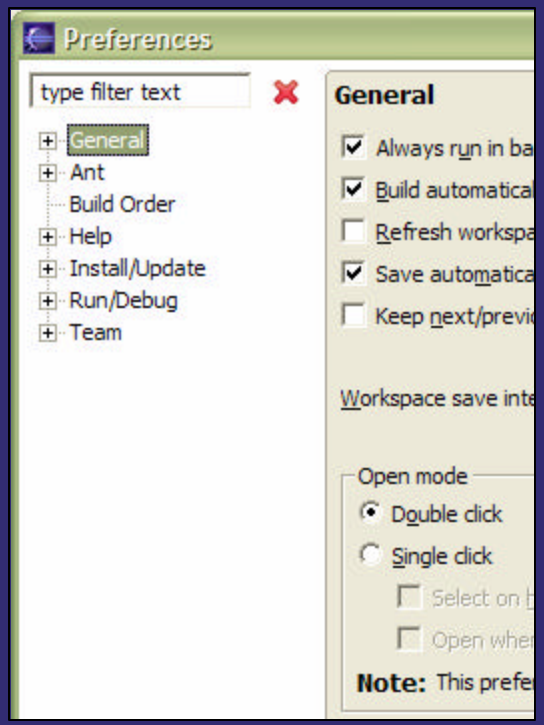
- Java functionality isn't needed until there is a Java project. Activate it when the user uses the New Java Project wizard or imports a project with the java nature.
- Certain Java debugging functionality isn't needed unless the user is currently debugging

 |

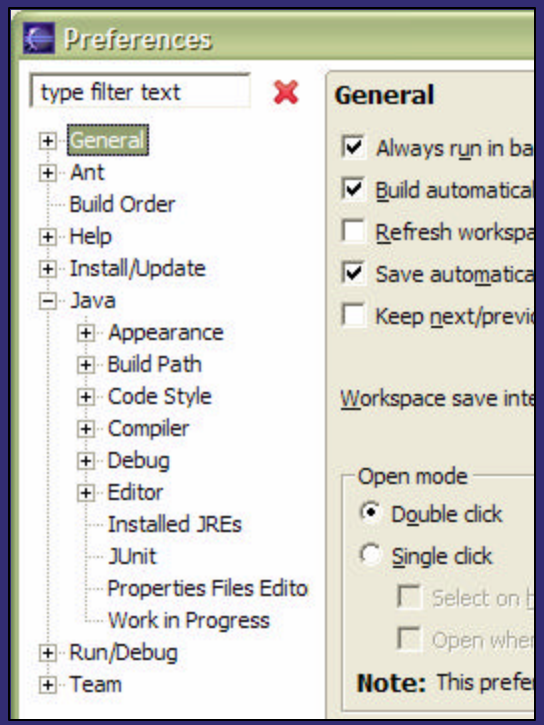
Progressive Disclosure Examples: Java Tooling

Preferences

Java Disabled



Java Enabled

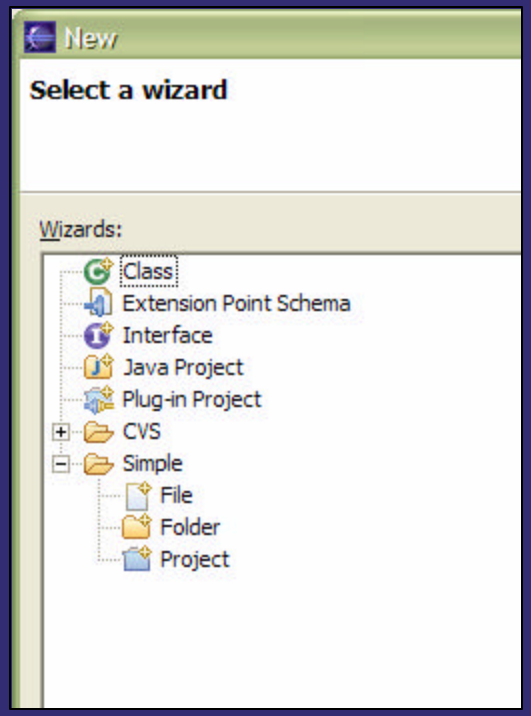


 |

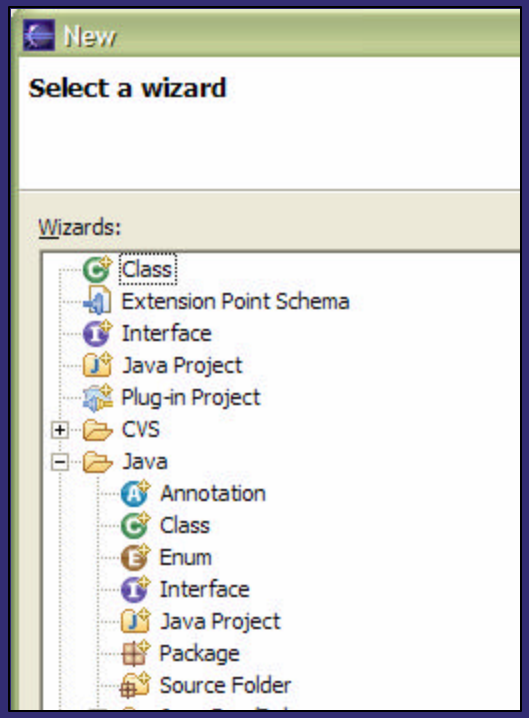
Progressive Disclosure Examples: Java Tooling

New Wizards

Java Disabled



Java Enabled



 |

Progressive Disclosure Examples: Team Functionality

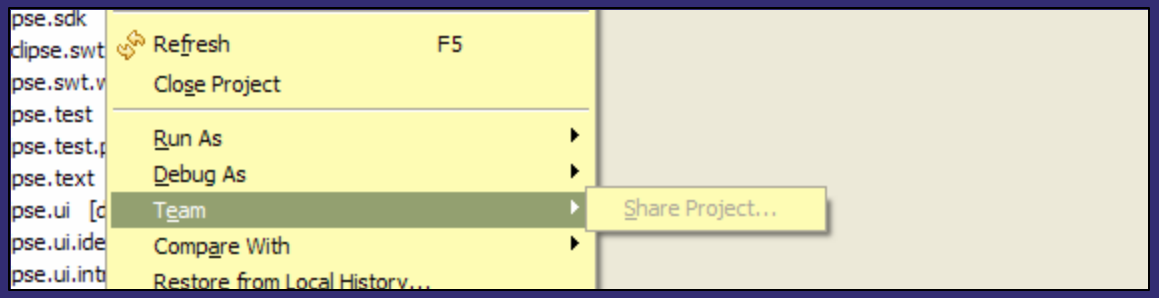
- Team CVS functionality isn't needed until the user either decides to check out a project from CVS or share a project already in their workspace.

 |

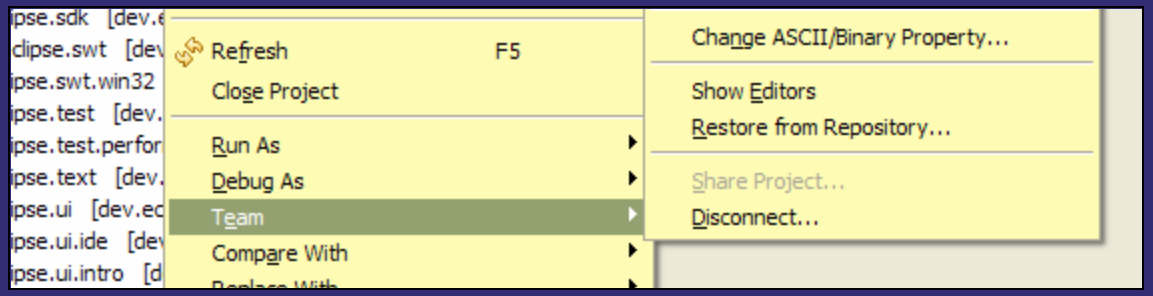
Progressive Disclosure Examples: Team Functionality

Context Menus

Team Disabled



Team Enabled



 |

In 3.0: Activities

- What's Lacking
  - Polish
  - Configurability
- What's Next
  - Can categories represent user roles effectively or is another mechanism needed?
  - Trigger point specification via plug-in XML
  - Settle nomenclature
  - Configurable trigger point handling
    - Custom dialogs
    - Custom behavior
    - Veto ability
  - Configurable preference page
  - Easier activity definitions
  - Easier 3rd party integration

 |

In 3.0: Contexts

- What's Lacking
  - Usage!
  - Performance
- What's Next
  - Automate contextual behavior within the workbench
    - View activation
    - Action visibility
    - Etc
  - Document usage and provide examples!
  - Optimize performance

 |

Alternative Uses For Activities

- Activities implementation is non-specific – just a filtering mechanism
- Activities sets need not be defined along functional lines
  - User experience level – novice, intermediate, advanced
  - User role
  - User permissions
- Examples may be found at the Platform/UI page on the Eclipse website:

  http://dev.eclipse.org/viewcvs/index.cgi/%7Echeckout%7E/platform-ui-home/index.html

 |

Other Uses: User Experience

- Best for products with a narrow focus
  - Domain-specific RCP apps
  - Language-centric IDEs
  - Teaching tools

 |

Other Uses: User Provisioning

- Only suitable for RCP apps and even then only barely
- Activities sets along user permission lines
- Disable the majority of trigger points – all capability activation becomes explicit
- Ensure lines of functionality don't cross
- 3rd party plug-ins can easily work around your safeguards
- Very poor substitute for real security – throw food at me for even suggesting this

 |