

## 06.3 Products Eclipsed

Experiences in Adopting and Developing

# Who am I?

- John Graham, Staff Software Engineer at Sybase
- Working with Eclipse since late 2001 (version 1)
- Technical lead for application development:
  - On the team that evaluated & recommended Eclipse
  - Internal Eclipse evangelism and education
  - Medium sized product in Eclipse (Unwired Orchestrator)
  - Served on Eclipse Executive Committee (now defunct)
  - Planning for large scale product integration into Eclipse
  - Currently assigned to Sybase® WorkSpace product

## Initial Product Team

- Approximately 10 developers
- Two people had some experience developing Eclipse plug-ins
- Another had experience with Swing
- A couple others had some Java experience
- The rest were C/C++ server developers

# Adopting Eclipse

- We needed a highly extensible platform
- Preferred Java
- Evaluated a number of options:
  - Visual Studio, NetBeans
  - Existing internal candidates
  - Build a new platform
- And chose Eclipse:
  - Flexible
  - Open source
  - Feature rich

# Eclipse is Flexible

- Eclipse looks most like an IDE
- With RCP, Eclipse could look like any other application type
- Before the RCP, the notion of a platform “for everything and nothing in particular” was a bit of a stretch

# Open Source

- In The Success of Open Source Steven Weber points out that traditionally property is configured around the rights to exclude usage of it. In open source, property is configured around the right to control distribution of it.
- “Free speech, not free beer”
- There is a range of opinion about what “open source” means
- From a company’s perspective, the license attached to a particular open source offering is effectively what “open source” means in that case
- For developers, the ecosystem around a particular open source offering is what “open source” means in that case

# Open Source Characteristics

- The source code is freely available
- The source can be modified & redistributed (depending on license)
- Benefits Claimed:
  - No vendor lock-in
  - Quality easier to ascertain
  - “Many eyeballs” theory
- Possible Risks:
  - Leveling of the playing field
  - IP concerns
  - Many throats to choke

# Eclipse is Open Source

- Eclipse source code is available
- The community encourages reuse of source code
- Visible project plans, designs, bug databases
- Ability to have a voice through the Foundation and newsgroups
- Liberal license (EPL)
- Many individuals and organizations committed to furthering Eclipse

# Eclipse Maturity

- Bernard Golden's Succeeding with Open Source presents the "Open Source Maturity Model" (OSMM)\*
- OSMM evaluates:
  - Product Software
  - Support
  - Documentation
  - Training
  - Product Integrations
  - Professional Services
- How mature is Eclipse in this model?

\* OSMM is a service trademark of Navica.

# Eclipse is Feature Rich

- Full Java development environment
- Full plug-in development environment
- Team support with CVS by default
- Many plug-ins available
- People will want to use Eclipse for these features, so why not add your product's feature in as well?

# Eclipse Ecosystem

- Eclipse Foundation fostering:
  - Individual and company involvement
  - Conference participation
  - Publications
  - Marketing initiatives
  
- Grass roots movement:
  - Eclipse is becoming the IDE of choice for Java development
  - Eclipse often finds its way into organizations at the grass-roots level
  - Eclipse is poised to expand its scope

# Developing for Eclipse

- Resources for ramping teams up
- Initial hurdles for teams new to Eclipse
- Policies for using Eclipse, and avoiding abusing Eclipse
- Specific pitfalls that seem so obvious now....

# Resources for Eclipse Development

- Eclipse site
- Eclipse newgroups
- Various magazine articles
- Books (next slide)
- The Eclipse source code

# Books about Eclipse Plug-in Development

- Numerous books about Eclipse available
- Many of these tend to concentrate wholly or mostly on using Eclipse, rather than developing Eclipse plug-ins
- Even fewer cover developing products in Eclipse
- Gamma & Beck's Contributing to Eclipse is useful for the types of things we are discussing in this presentation
- Among the books that cover plug-in development, a common set of topics is provided. More advanced topics/techniques remain in the minds of developers and their products

# Extensibility Environment

- Eclipse is not completely foreign to those who have worked with other extensible environments
- It does, of course, have its own way of doing things
- There are many ways to extend Eclipse, and this can be overwhelming at first
- Eclipse terminology: views, editors, actions....
- PDE makes it easy to build, test, and package plug-ins
- There are some potential pitfalls (discussed later)

# SWT & JFace

- SWT isn't just like AWT
- SWT isn't just like Swing
- It is UI library, and it does take time to learn
- Skills from AWT, Swing, MFC, etc. are, of course, useful
- For SWT and JFace, as well as Eclipse in general, knowledge of design patterns (especially GOF) helps to understand the design motivations

# Use Eclipse for Plug-in Development

- This sounds obvious but...
- Developers tend to have strong feelings about development environments and especially code editors
- Learn/absorb how Eclipse acts
- After several years, I still find people who try to work in other environments

## Install Platform “Ownership”

- Install your plug-ins and Eclipse as one unit
- Install your plug-ins into any existing Eclipse platform
- Do both
- What about product exposure through Eclipse re-branding?
- Owning the platform simplifies things, but users might want to use existing installs, add new plug-ins, and so on

# Level of Eclipse Integration

- There's a number of choices here
- “Levels of Integration” article by Jim Amsden on Eclipse site
- Need to make this decision early
- Re-hosting existing Java or COM-based (Windows) applications is not trivial, but often a better choice than re-writing in SWT/JFace
- The effort required for thorough integration compared to re-hosting should not be underestimated
- There is no simple “port to Eclipse” for any substantial application, if you want it done correctly

## Good Use of Eclipse

- Working well within Eclipse takes sensitivity about how Eclipse does things and effort to do the same
- Product ports tend to under-utilize Eclipse, making them feel foreign to some extent
- New product development tends to over-utilize Eclipse, resulting in too many perspectives, views, actions, default outlines, and property values.

# Problematic Extenders

- Eclipse is highly extensible
- And new plug-ins typically like to be extensible as well
- This opens your plug-in up to bad behavior from extenders
- These extenders, however, look like they are part of your plug-in
- Be careful about extender errors that can crash the extended plug-in
- The .log file is a great source of information in these cases

# Performance Considerations

- Eclipse performance is reasonable, but plug-ins can cause it to quickly degrade
- Limit the number of builders
- Watch out for too many change listeners
- Complex graphical operations require caution
- See <http://www.eclipse.org/eclipse/development/performance/bloopers.html> for more details

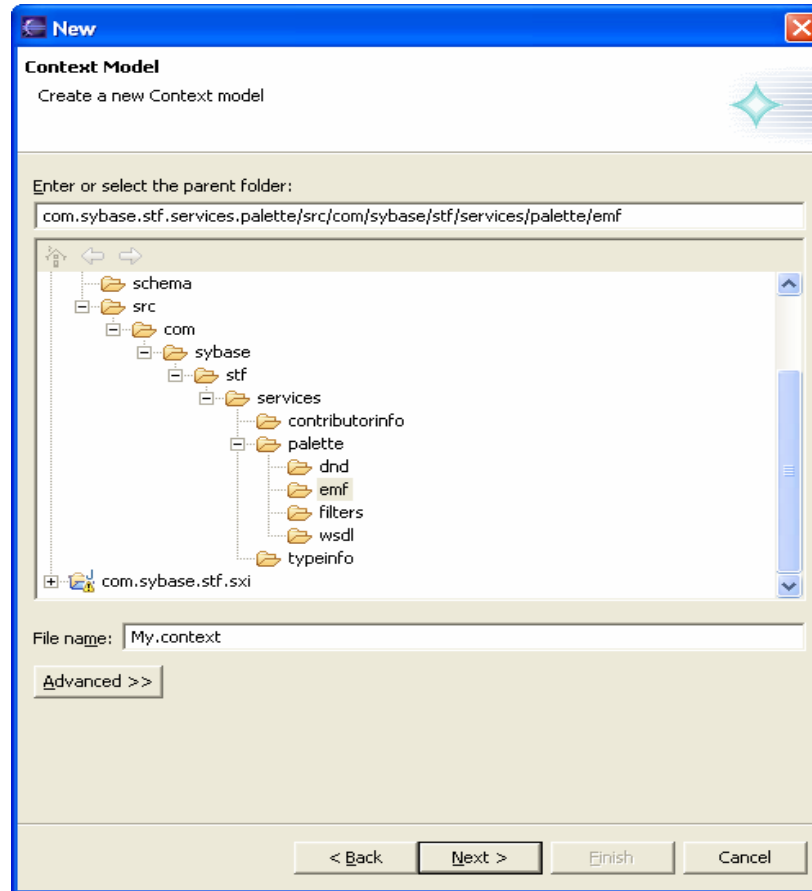
# Eclipse API Usage

- Eclipse has “internal” API
- You can use these, but strictly that is not recommended
- See “How to Use Eclipse API” by Jim des Rivieres on Eclipse site
- In the plug-in community, opinions on whether to use internal API or not is split
- If you do use internal API, take care to track the usage for potential later breakage

# Changing the Eclipse Source

- There's a great temptation to change the Eclipse source and do a "custom" build
- This seems to be more prevalent in development teams not used to working with open source
- For obvious reasons, any decision to change Eclipse source should be seriously considered
- At least work with the Eclipse Foundation first to try to find an alternative

# A Possible Platform Change



# Crossing Platforms

- Eclipse runs on a number of platforms
- Your plug-ins could too
- There are subtle UI behavior differences between platforms
- This requires extensive testing on all chosen target platforms
- Coding to a common quality level is very tricky to achieve

# Plug-in Dependencies

- Cycles in the plug-in dependency graph will disable all plug-ins involved
- For a few plug-ins, this is not hard to avoid
- For a medium/large set of plug-ins, it quickly can become complex
- Create a plug-ins dependency diagram and keep it up to date

# Case Sensitivity

- Some file systems are case-sensitive, others are not
- Users of a particular file system typically expect applications run on top of a file system to conform
- Eclipse `getFile(...)` and `findMember(...)` are case-sensitive, even on Windows ®
- This causes confusion when users expect two files with different case to be the same and they are not

## Two Class Paths

- One is in the .classpath file
- The other is in the plugin.xml file
- It is a common mistake to add libraries in the .classpath file but not in the plugin.xml file
- Especially in the case of libraries that depend on other libraries, this can be hard to diagnose

# Summary

- Eclipse is great product hosting environment
- You have to buy into the Eclipse way of doing things at several levels
- Your users also have to buy into the Eclipse way of doing things
- Leverage the Eclipse ecosystem for support
- Leverage the Eclipse grass-roots movement

# Wrapping Up

- Questions?
  
- Contacting me:
  - John Graham
  - [john.graham@sybase.com](mailto:john.graham@sybase.com)