eclipse CON 2004

Inside the RCP Runtime

Dynamic Plug-ins and Beyond

Jeff McAffer
IBM Ottawa Lab

What is the Eclipse Runtime?

- Java component (*plug-in*) model
  - Dependency management
  - Classloaders
- Extension registry
  - Extensions
  - Extension points
- Various utility classes
  - IStatus
  - IPath/Path
  - …

Small, simple, flexible

What is a plug-in?

- Basic unit of modularity/install
  - ID and version
  - Set of elements to put on the plug-in's classpath
  - Set of prerequisites
  - Extension and Extension point declarations
  - Additional files
- Each plug-in gets its own classloader
  - Allows code independence
  - Used as a trigger for plug-in activation

Plug-in == component

Why a New Runtime?

- RCP usecases are different than IDE usecases
  - User skill/knowledge level
  - User expectations
  - Security requirements
  - Install-base (100s vs 100,000s machines)
- Main 3.0 runtime themes
  - Dynamic plug-ins
  - Standards based
  - Varying configuration management
  - Backward compatibility

Expand where Eclipse is used

Dynamic Plug-ins

- Change plug-in configuration without restarting
- Add – Easy! (well sort of)
- Remove – Easy for runtime, hard for everyone else
- Update – Remove+Add and retain state
- No magic: dependent plug-ins must be stopped or restarted
- Usecases
  - User attempts to use new function not yet installed
  - Auto-update subscription services
  - Better startup behaviour

Reduce need to restart

Are You Dynamic? (A plug-in self test)

1. Do you notice when a friend
   a) Suddenly bloats up
   b) Arrives or leaves the party
   c) Starts depending on others
2. When friend goes away do you
   a) Forget all about them
   b) Keep a memento in case they come back
   c) Irrationally cling to any/all traces of them
3. When you are ready do you
   a) Assume everyone else is
   b) Sit and wait for others
   c) Make the best of the time and do something else
4. When you are done do you
   a) Assume the lights will go out
   b) Kick everyone else out
   c) Leave the room
5. When getting ready for the day do you
   a) Clean the house and cook a turkey in case you forgot it is Christmas
   b) Open your eyes
   c) Kick back, sip a mocha coconut frappacino and read the obituaries

Answers: 1.b, 2.a, 3.c, 4.c, 5.b

On Being Dynamic

- *Dynamic aware* plug-ins must listen/react to other plug-ins coming and going
  - Updates to the registry (update structures and/or destroy objects)
  - Drop references to objects from other plug-ins (hard)
- *Dynamic enabled* plug-ins must be more aware of the environment
  - Do not assume other the plug-ins/services are available
  - Cleanup on stop (don't assume Eclipse is exiting)
  - Do less on plug-in activation

Similar challenges as being concurrent

Facilitating Dynamic Plug-ins

- APIs for
  - Install – BundleContext.installBundle()
  - Uninstall – Bundle.uninstall()
  - Update – Bundle.update()
- Plug-in lifecycle events
  - BundleEvent
  - Event types: Installed, Resolved, Started, Stopped, Unresolved, Uninstalled, Updated
- Extension registry lifecycle events
  - IRegistryChangeEvent
  - IExtensionDelta
- Eclipse 3.0 RCP related plug-ins use these APIs

Demo: Dynamic Plug-ins

Professional Eclipse hacker. Closed configuration. Do not attempt. Obey posted coding conventions. Always code responsibly.

OSGi

- Open Services Gateway initiative
- Consortium includes:
  - Nokia, NTT, Motorola, Philips, Siemens, Oracle
- Component platform for devices (e.g., set-top box)
- Same vintage as Eclipse
- OSGi framework features:
  - Dynamic install/uninstall/update of bundles
  - Service architecture
  - Security (based on Java 2)
  - Remote configuration API

Terminology

- *Plug-in* vs. *Bundle*
  - No difference
  - Use the terms interchangeably
  - Manifest.mf vs. plugin.xml
- Eclipse *Runtime* sits on top of OSGi *Framework*

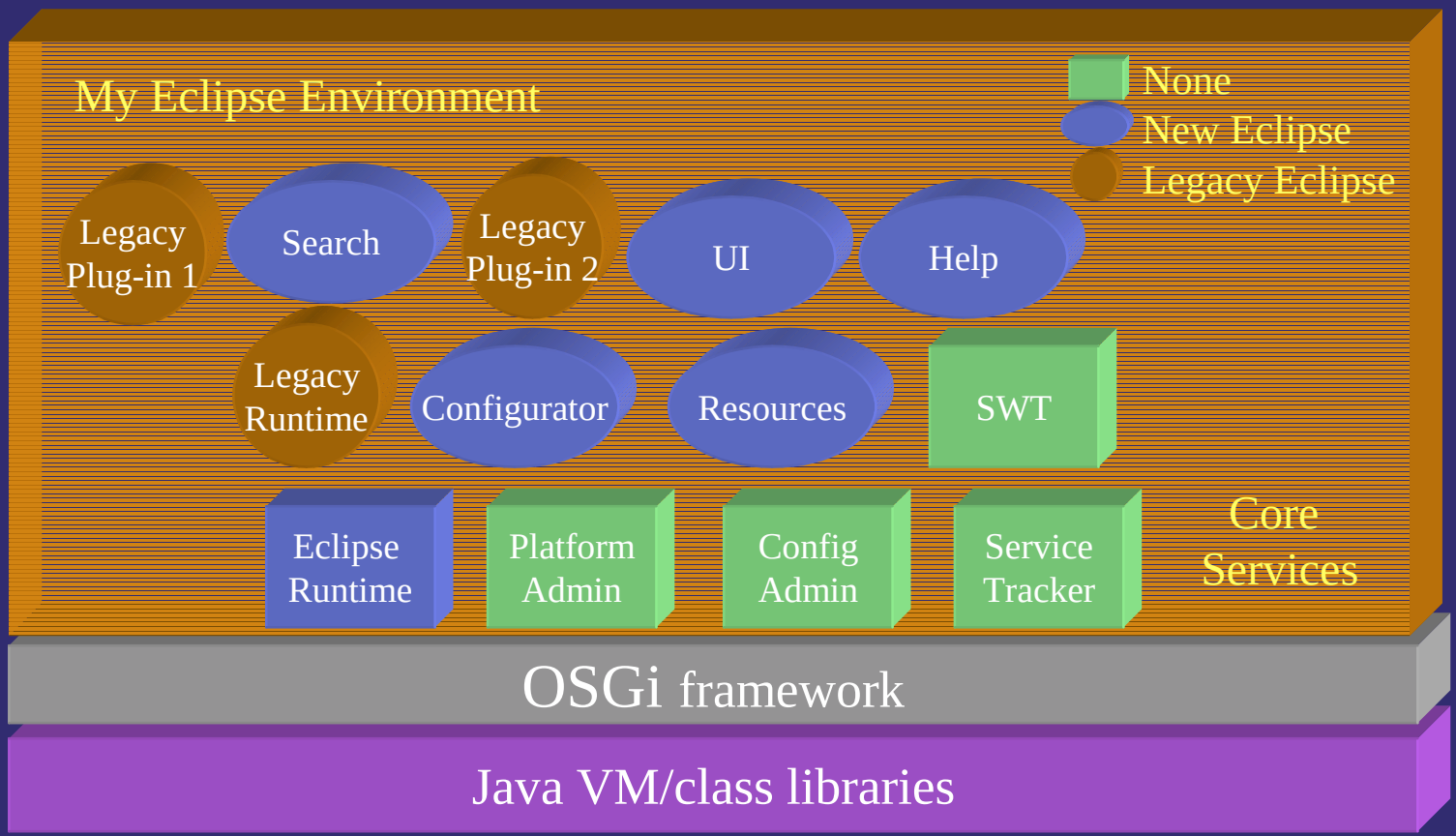
Plug-in == Bundle

OSGi Challenges

- Different usage/install scenarios
  - Less user management
- Scale (1,000s of bundles rather than 10s or 100s)
- Missing
  - Fragments
  - Bundles as modules
  - Miscellaneous utilities (file access methods)
- Working with OSGi community to extend specification
  - They are tackling many of the same issues

Eclipse moving down, OSGi moving up

Eclipse 3.0 Runtime API Use



Compatibility

- Complete runtime was replaced but…
- Eclipse 3.0 Runtime is 99% binary backward compatible
- Compatibility layer maintains original Eclipse API
  - org.eclipse.core.runtime.compatibility
- API breakages almost entirely fringe cases

- Eclipse 2.1 style API is the official Eclipse 3.0 API
- Additional API is *experimental*
  - Paints the way for the future

Change required only if exploiting new function

Porting: API changes

- ILibrary
  - Library info is organized differently in manifest.mf
  - Cannot be reverse engineered
  - Reconsider usecase or use OSGi API to get required info
- org.eclipse.core.runtime.urlHandlers extension point removed
  - Move to the OSGi mechanism to enable dynamic behaviour
  - Java design contains only protected classes so cannot wrapper
  - Easy porting steps
- BootLoader runtime management methods removed
  - Boot structure/sequence changed
  - No longer relevant to start and stop the runtime as before

Porting: API changes (cont'd)

- Plug-in parsing method moved
  - Moving away from the PluginModel class structure
  - Model and related methods moved to the compatibility layer
- Class load order
  - Prerequisites take precedence over self
  - More inline with standard Java approach
- Default package cannot be shared

Porting: Practice changes

- Classloader protection domain not set
  - Only set if using a SecurityManager
- PluginModel casting
  - Plugin*Model used to implement IPlugin*
- URL changes
  - Cannot assume the shape of URLs
- Re-exporting runtime API
  - Update plug-ins which re-expose changed runtime API
- Build scripts
  - Names and locations of JARs changed

Best Compliment to a Core Guy

- I didn’t even notice that things changed…

Configuration Management

- Make the Platform agnostic wrt update/install
- Framework exposes API to install/update/uninstall plug-ins
- Framework remembers installed plug-ins
  - Previously done by Install/Update
- *Configurators* reconcile external plug-in list with framework list
  - Needed if external picture can change without the runtime's knowledge
  - Eclispe 3.0 includes an Install/Update configurator
- Ship preconfigured Eclipse installs
  - No configurator needed
  - Install/Update mechanism needed only if function required

Alternative Install/Update mechanism possible

Other Cool Features

- Console
- Start levels
- Running from JARs
- System properties vs. command line args
- Enable workspace selection

Future Directions

- Not all of Eclipse 3.0 dynamic
- Embrace and exploit OSGi mechanisms
  - More dynamic behaviour
  - More function available as Services
- OSGi API is relatively low-level
  - Provide mechanisms and infrastructure for common usecases
- Sub and (possible) future themes
  - Varying execution environments
  - Alternative implementations
  - Security

We have only scratched the surface

Eclipse Technology Projects

- This work is largely the output of the Equinox Technology project
- Technology projects can be used as incubators
- Equinox deliverables were prototypes suitable for use in Eclipse
- Very lightweight process
- Very effective for addressing particular aspects
- Good for attracting others to learn more and contribute
- Resulted in several new Eclipse committers

Summary

- New runtime based on OSGi
- Enables many RCP scenarios
- 99% backward compatible
- Example of a successful technology incubator
- Unlocks huge potential
- Opens platform for more contributions
  - Standards
  - Optional services

