Using EJBs in Eclipse RCP

Experiences

Eclipse Finance Day Zürich
16.10.2012
Markus Hediger
Philippe Marschall
The Application

- acquiring processing back office
- manage master data
- manage business rule
- fix transaction errors
The Application II

- client-server application
- Java EE application server with EJBs
- Eclipse RCP client
- EJB remoting
The Problem

• Calling EJBs from OSGi
• EJB client library intended for use in Java EE container
• uses ThreadContextClassLoader (TCCL)
• assumes having access to all application classes
• isn’t considered sexy
Bad Solutions I

• Copy all ejb-client.jars into a single bundle
Bad Solutions II

- Buddy classloading
Old Dependencies

- central-ejb-client-bundle
- vendor-ejb-client-library
- first-client-module
- second-client-module

Dependency

Buddy

Dependency/Buddy
A Less Bad Solution
New Dependencies

central-ejb-client-bundle

vendor-ejb-client-library

first-client-module

second-client-module

Dependency

Dependency
Look Up Proxy

- ask `central-ejb-client-bundle` for service proxy
- look up client bundle
  - find JNDI name
  - get bundle class loader
- switch TCCL to bundle class loader
- do JNDI look up
Finding the Right Bundle

- Extension point to map business interface to JNDI names
- Implicitly provide client bundle
Service Call I

- after proxy look up don’t return raw proxy
- wrap with another proxy that switches TCCL before invoking
Service Call II

- Infrastructure to make service calls in Eclipse Jobs instead of GUI thread
Pros

- several small bundles
- no Dynamic-Import
- no buddy class loading
- lazy bundle activation
Cons

- no services, no OSGi remoting
- still uses TCCL
- Equinox rather than OSGi API
- client bundles depend on vendor libraries
Better Solution

- Vendor support
- OSGi specification
- OSGi remote services
Development

• connect Java EE server development and RCP client development
• quick turn arounds
• source dependencies for easy refactorings
• no budget for big tooling investments
The old way

- `ejb-client.jar` weren’t bundles
- had to be wrapped in a custom library project (~80 projects)
- building project and copying JARs was required
The new way

• turn ejb-client.jar into bundles
• source dependency from RCP projects to EJB projects
• Export-Package to hide EJBs, services, DAOs
• generate as much as possible
• only client projects are PDE projects
Generate

- META-INF/MANIFEST.MF
- plugin.properties
- plugin.xml
META-INF/MANIFEST.MF

- generate from POM
- custom Maven plugin
- generate-resources phase
- `<dependency/>` → `Require-Bundle`
- very specific rules about mapping `groupId:artifactId` to bundle symbolic name
plugin.xml

• map service interface to JNDI name
• derived from EJB
• custom annotation processor
• run by JDT when saving EJBs
• run by Maven during build
Future Improvements

• Don’t generate META-INF and plugin.xml into project root

• MANIFEST.MF is present in ejb.jar as well
• with a bit of effort cumbersome task could be automated

• patch work but works quite well