OSGi on the Server

Martin Lippert (it-agile GmbH)

lippert@acm.org

 | October 6th, 2009

Overview

- OSGi in 5 minutes
- Apps on the server (today and tomorrow)

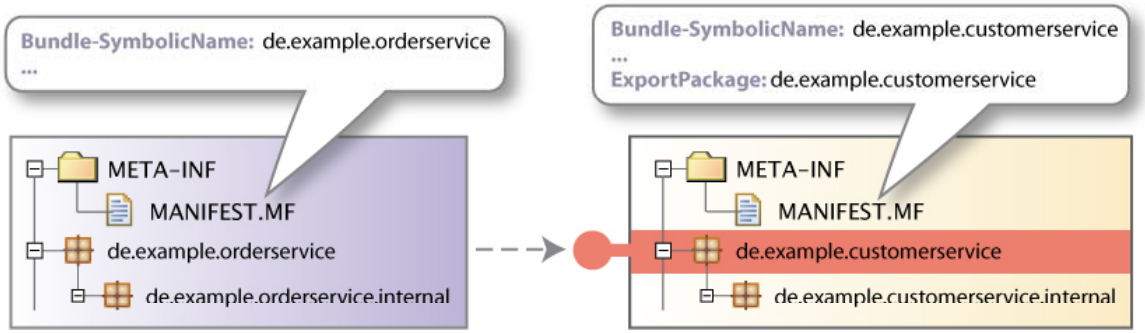
- Dynamics in OSGi
- Dynamics on the Server

OSG – What?

- OSGi™:
  - **„A dynamic module system for Java“**

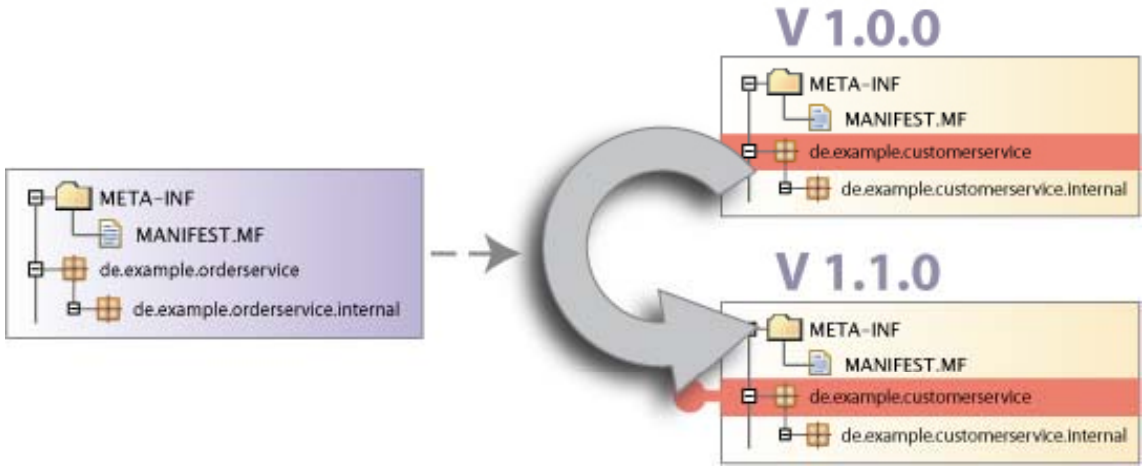
OSGi is …

- … a module system for Java that allows the definition of …
  - **Modules** (called „bundles“),
  - **Visibility** of the bundle contents (public-API vs. private-API)
  - **Dependencies** between modules
  - **Versions** of modules



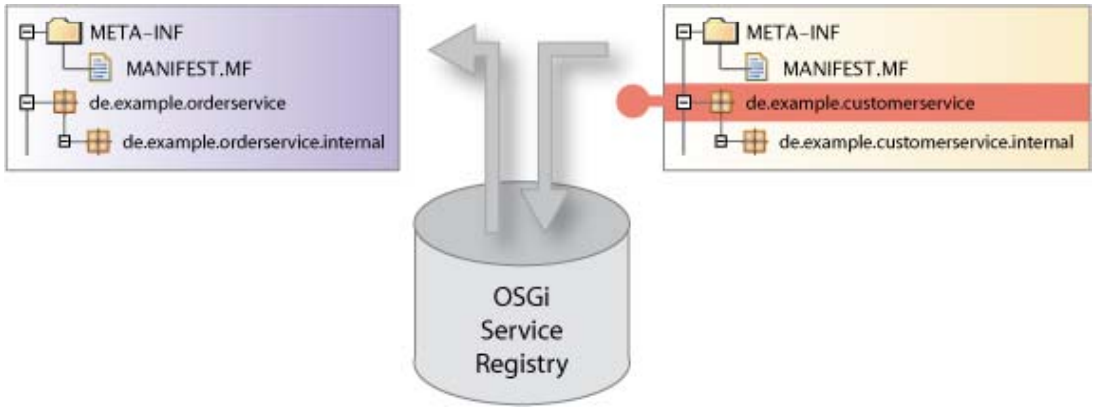
OSGi is …

- … dynamic
  - Bundles can be installed, started, stopped, uninstalled and updated at runtime



OSGi is …

- … service oriented
  - Bundles can publish services (dynamically)
  - Bundles can find and bind to services through a service registry
  - The runtime allows services to appear and disappear at runtime



What does OSGi look like? (Low Level)

Identification

```
Bundle-SymbolicName: org.eclipse.equinox.registry
Bundle-Version: 3.2.100.v20060918
Bundle-Name: Eclipse Extension Registry
Bundle-Vendor: Eclipse.org
```

Classpath

```
Bundle-ClassPath: ., someOtherJar.jar
```

Lifecycle

```
Bundle-Activator: org.eclipse.core.internal.registry.osgi.Activator
```

Dependencies

```
Import-Package: javax.xml.parsers,
 org.xml.sax,
 org.osgi.framework;version=1.3
Require-Bundle: org.eclipse.equinox.common;bundle-version="[3.2.0,4.0.0)"
Bundle-RequiredExecutionEnvironment: CDC-1.0/Foundation-1.0,J2SE-1.3
```

Exports

```
Export-Package: org.eclipse.equinox.registry
```

Implementations

- Open source implementations
  - Eclipse Equinox (http://www.eclipse.org/equinox/)
  - Apache Felix (http://cwiki.apache.org/FELIX/index.html)
  - Knopflerfish (http://www.knopflerfish.org/)
  - ProSyst mBedded Server Equinox Edition (http://www.prosyst.com/products/osgi_se_equi_ed.html)
- Commercial implementations
  - ProSyst (http://www.prosyst.com/)
  - Knopflerfish Pro (http://www.gatespacetelematics.com/)

(not necessarily complete)

OSGi in Action

- Eclipse
  - SDK, RCP, RT, …
- Desktop
  - RCP-Apps, widely adopted throughout the industry
  - Swing-based enterprise apps
- Mobile
  - Starting to gain interest (again)
  - Spring Titan platform (mobile phones)
- Server?
  - Hm…

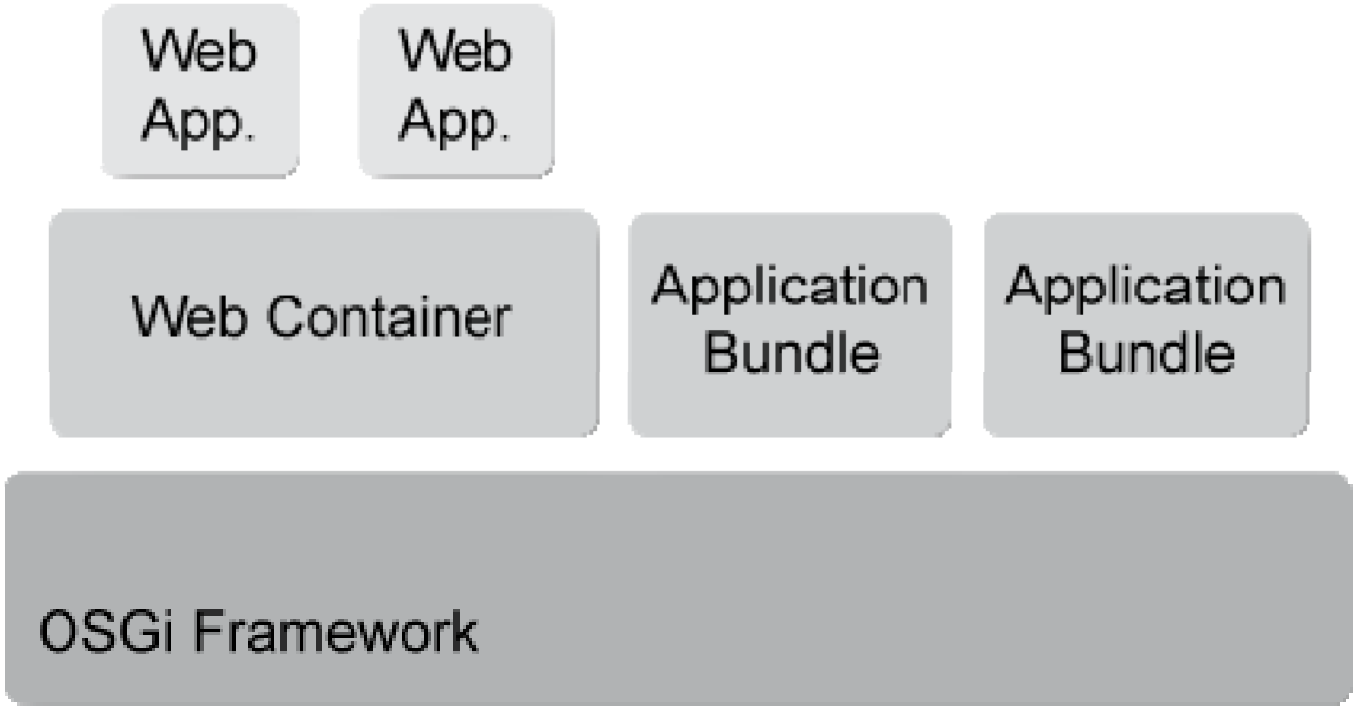
Server app settings

- No UI
  - Running standalone or inside an app server
  - Often managed environment (container)
  - Serves as back-end
- Web UI
  - Running inside a web server
  - Many different frameworks and languages
    - GWT, Grails, Spring WebFlow, RAP, JSF, Lift, JavaScript, REST, …

Managed environment and OSGi?

- The web- or app-server manages the environment **and has a very special view on the deployable app**
  - Servet/JSP API, WAR files, …
  - EJB spec, EAR files, …
  - Runtime environment
- How does OSGi conform to this?
  - JAR'd modules (bundles)
  - Runtime environment
- **Who is the boss?**

The OSGi Way


Web App.
Web App.
Web Container
Application Bundle
Application Bundle
OSGi Framework

Ready to use

- Choose an OSGi runtime
  - Equinox, Felix, …
- Choose a web container
  - Jetty, Tomcat
- Choose an extender mechanism
  - PAX Web Extender
  - Spring DM

- **Go!**

Plain old WAR files?

- Add an OSGi manifest to the WAR file
  - And it becomes an OSGi bundle
- Deploy the WAR file into the runtime
  - It’s a bundle like all other bundles from the runtime perspective
- Infrastructure takes care of connecting the WAR bundle with the running container
  - Extender pattern

Modularity

- Reduce the scope of the WAR files
  - Just the UI parts

- Extract functionality into separate bundles
- Extract libs into separate bundles

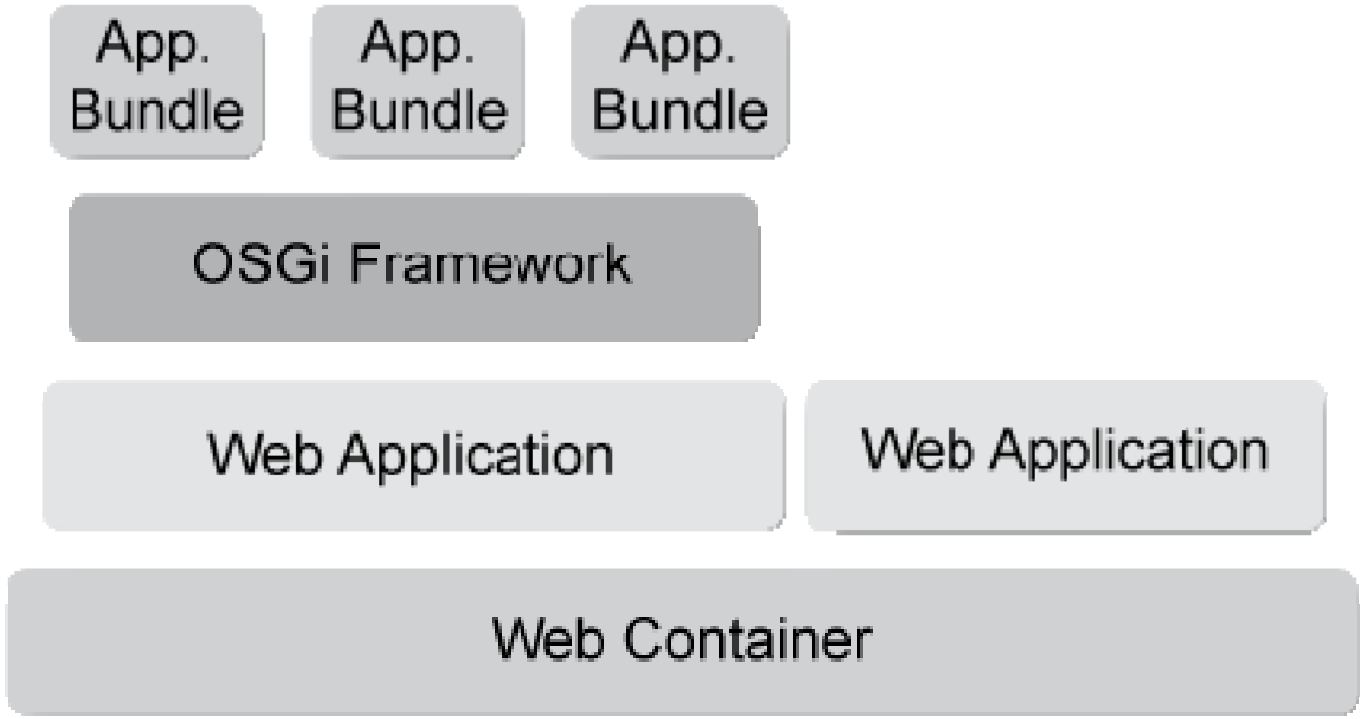
Isolation

- If all bundles live in the same space, what about application isolation?
  - Can be good
  - Can be bad
- We need an additional abstraction for isoloation
  - SpringSource dm Server introduces proprietary construct
  - OSGi spec will introduce something in the future (called Composite Bundles?)

Existing servers

- Interesting: Most app servers are built on top of OSGi
  - WebSphere, Glassfish, SpringSource dm Server, etc.
  - But not all of them expose this to the app
  - Implementation detail only
- The trend: Moving towards more OSGi options
  - Having the server built on top of OSGi
  - **Letting apps be deployed as OSGi bundles**
  - SpringSource dm Server is the most advanced product in this area

The Migration Way



Too many limitations

- Bridge is tiny
- OSGi HTTP Service is old
  - Servlet API 2.1
  - No filters, no listeners, …
- Need to do JSP compiling from inside

- **Just for the migration phase**

Conclusions for now…

- Most app servers don't support OSGi app deployment directly
  - SpringSource dm Server and Jonas are the glory exceptions
  - The other servers are still working on it
- The future belongs to:
  - OSGi Web Service (RFC 66)
  - OSGi JEE Bindings
- Server side OSGi is still a bit bleeding edge, but possible and promising

OSGi is dynamic

Wouldn‘t it be cool for server apps…

- To have real modularity?

- To update only what you really changed?
- To update only small parts, not the whole app?
- To update without downtime?

Dynamic OSGi applications

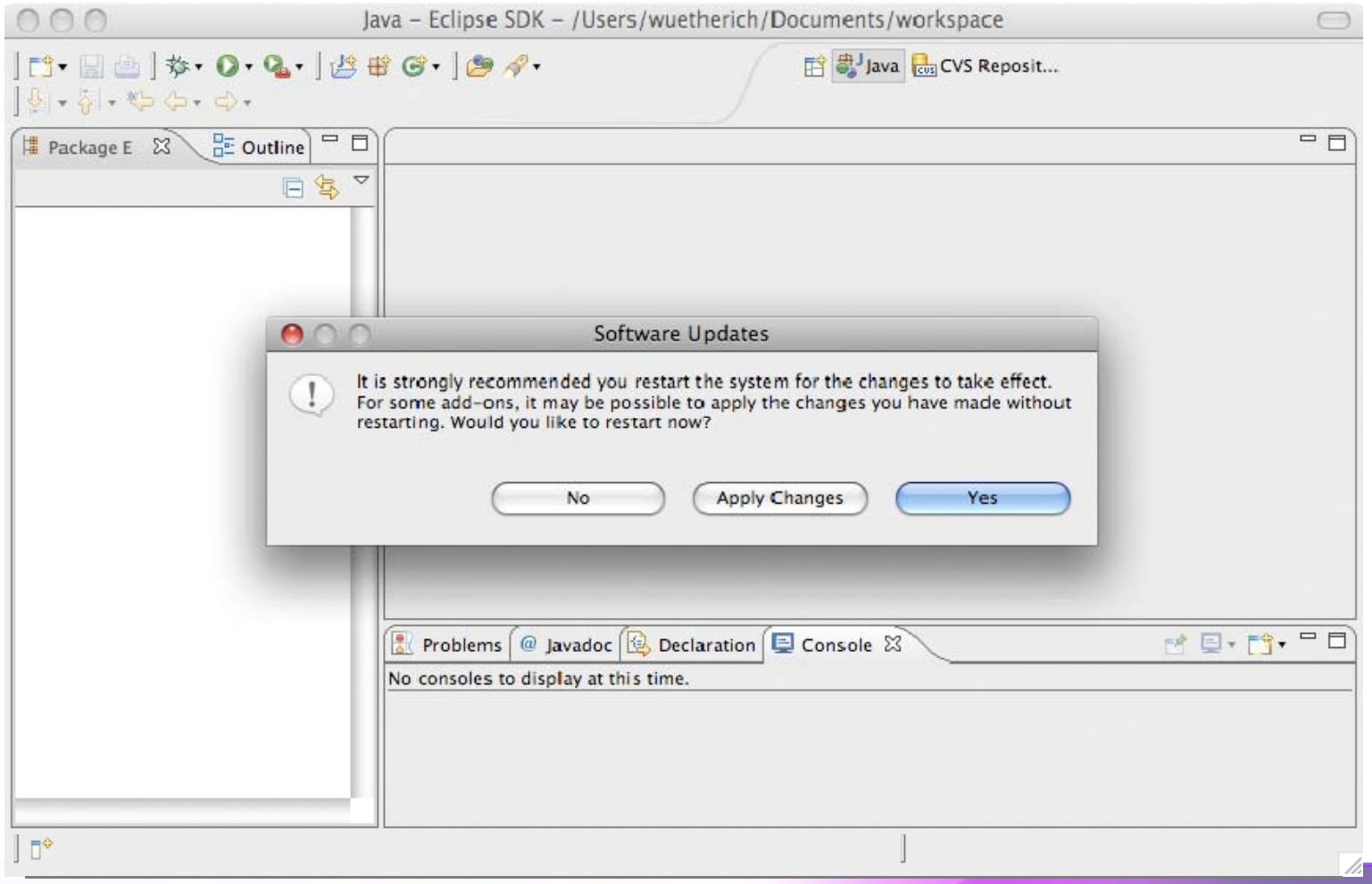
- Deployment unit:
  - Bundle = JAR + additional manifest headers
- Supports dynamic scenarios (during runtime)
  - Update
  - Installation
  - Deinstallation

The first impressions

- "Wow - OSGi does dynamic install, uninstall and update of bundles, this is cool…"
  - I don’t need to take care of dynamics anymore
  - I don’t need to think about this at all
  - Everything is done automatically under the hood
  - Objects are changed/migrated and references to objects are managed all automatically
  - Huge bulk of magic

- **This is all wrong!!!**

If its all magic, why this?



The basic idea

- OSGi controls the lifecycle of bundles
  - It allows you to install, uninstall and update bundles at runtime
  - It gives you feedback on all those actions
  - But it does not change any objects or references for you
    - "No magic"

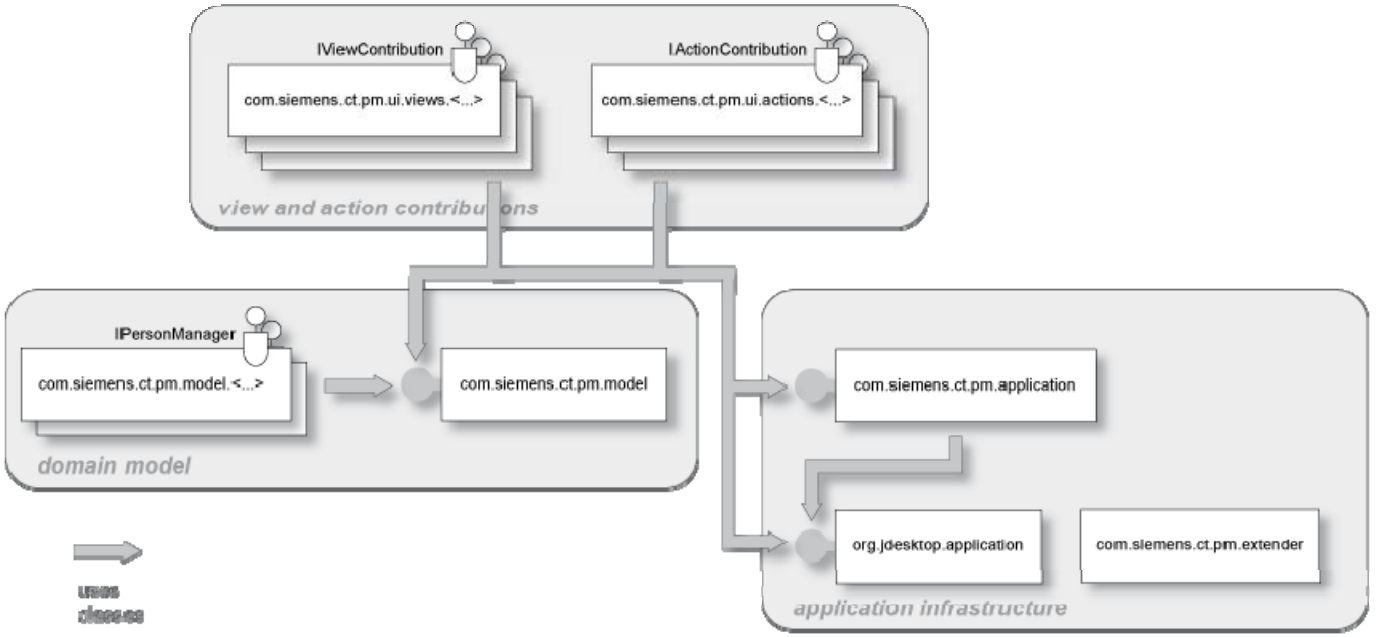
- **OSGi gives you the power to implement dynamic applications**
- **How you use this power is up to you**

What is the problem?

- Bundles have dependencies
  - e.g. package or service dependencies

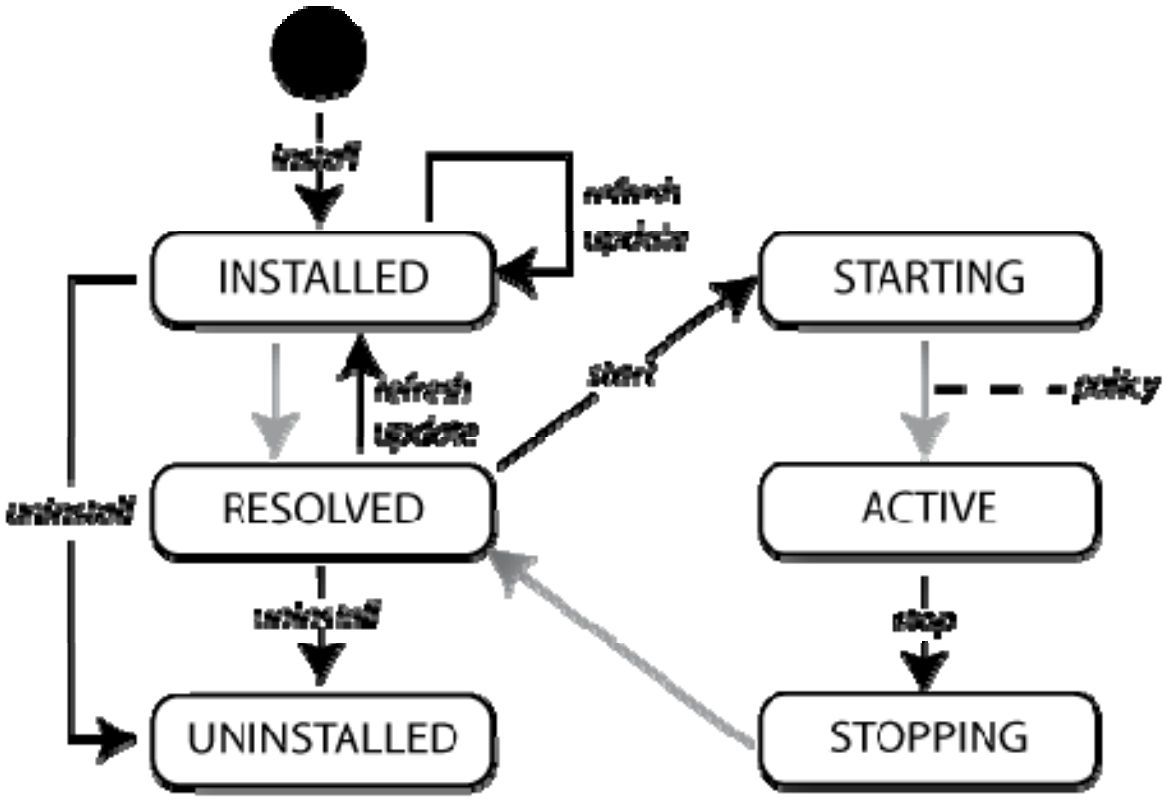
- **Dependencies have to be handled with respect to the dynamic behavior!**

Package Dependencies

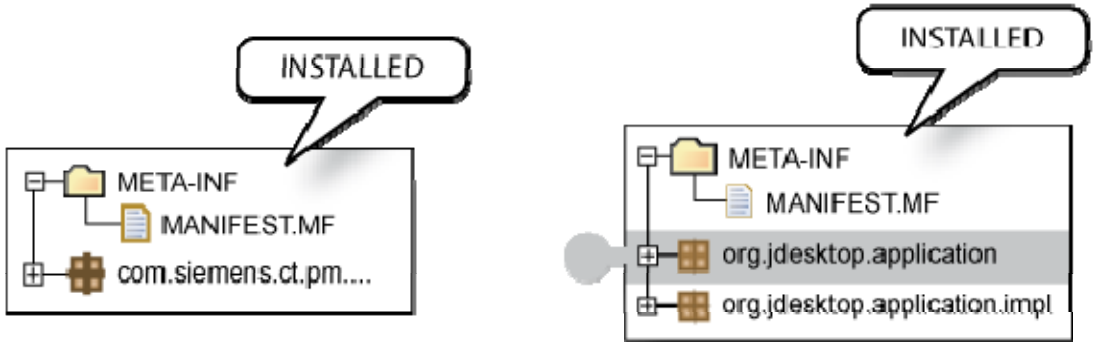


- Export of packages with **Export-Package**
- Import of packages via **Import-Package** or **Require-Bundle**

Bundle-Lifecycle

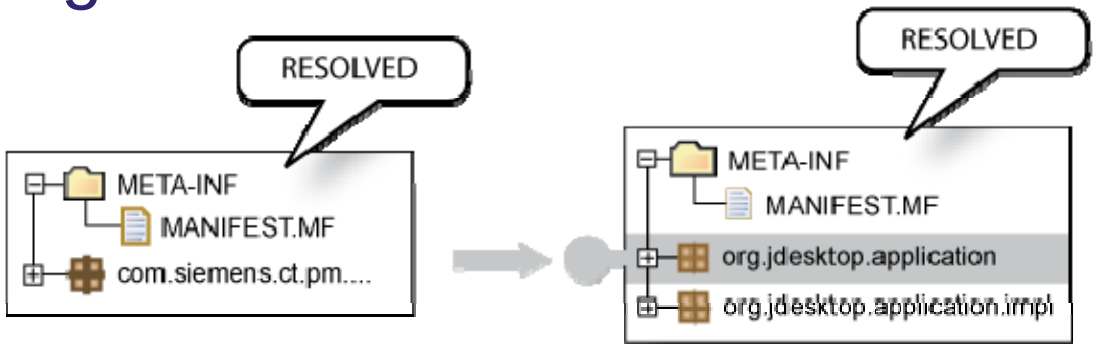


Installing



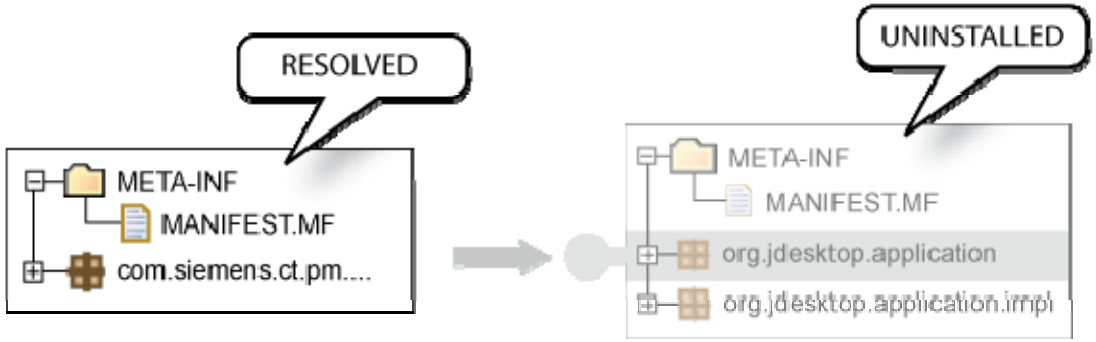
- Makes a Bundle persistently available in the OSGi Framework
  - The Bundle is assigned a unique Bundle identifier (long)
  - The Bundle State is set to INSTALLED
  - The Bundle will remain in the OSGi Framework until explicitly uninstalled

Resolving



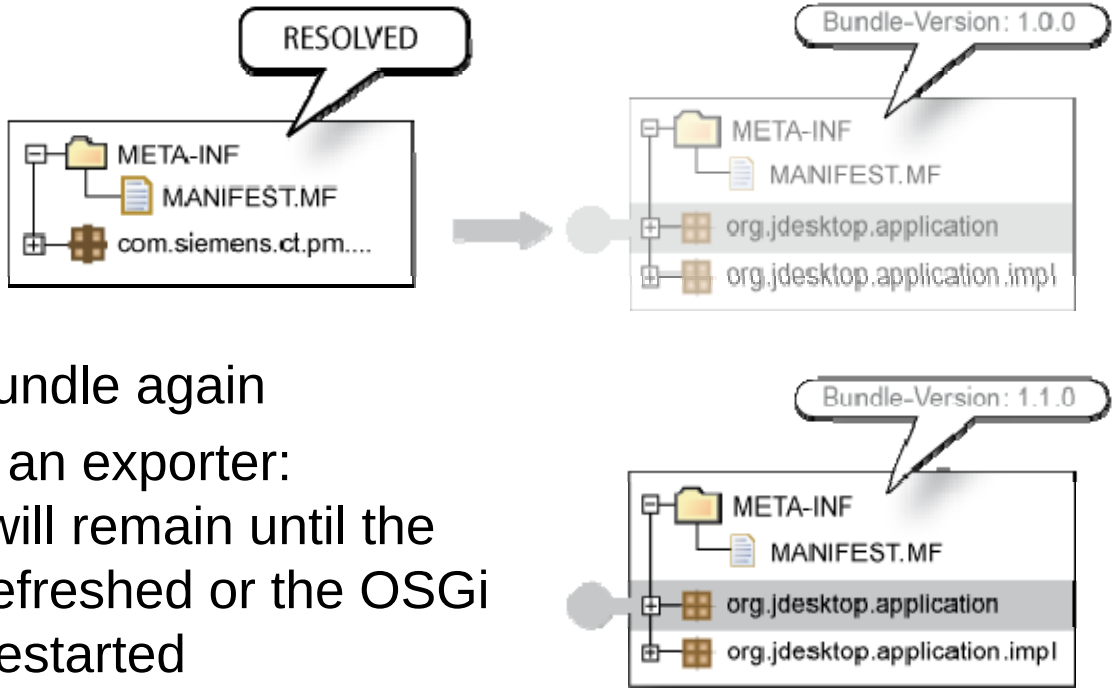
- Wires bundles by matching imports to exports
- Resolving may occur eagerly (after installation) or lazily
- There is no API for resolving
- After resolving -> Bundle is in state RESOLVED

Uninstall



- … removes a Bundle from the OSGi Framework
- The Bundle State is set to UNINSTALLED
- If the Bundle is an exporter: Existing wires will remain until
  - the importers are refreshed or
  - the OSGi Framework is restarted

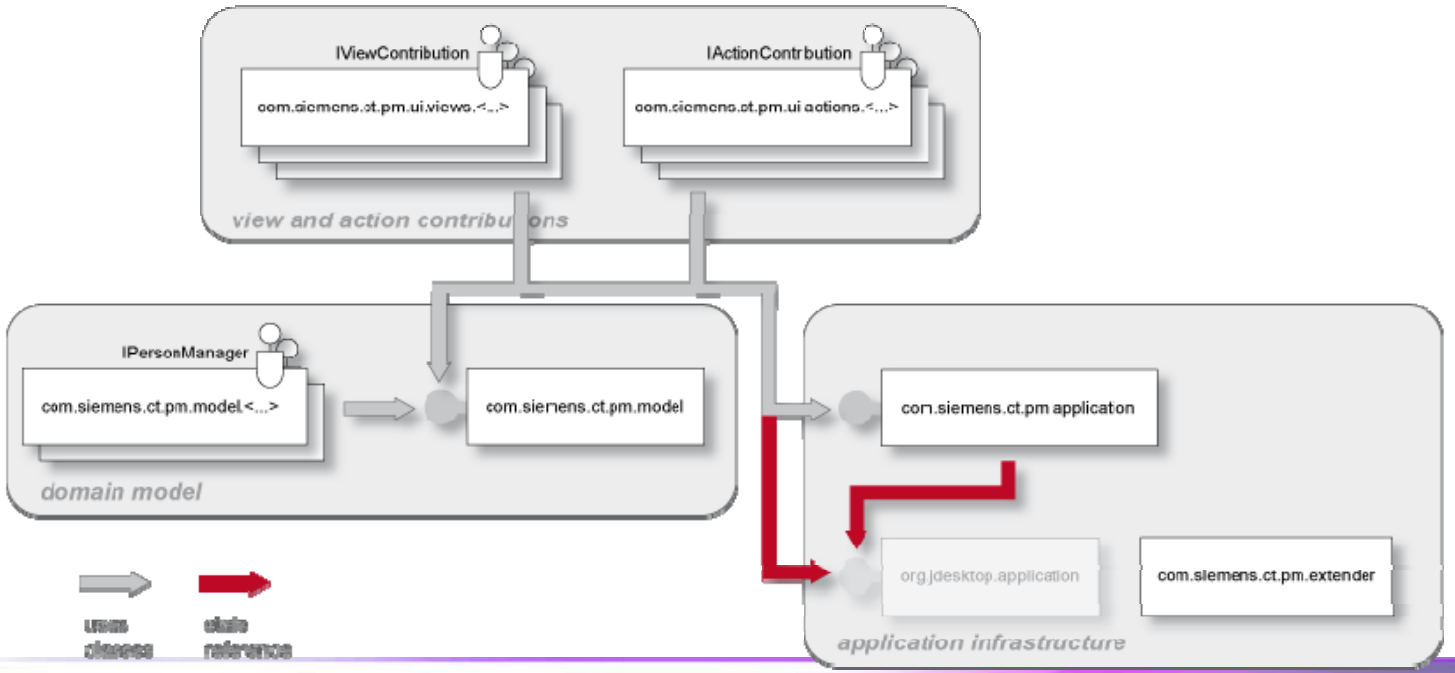
Update and Refresh



- Update:
  - Reads in the Bundle again
  - If the Bundle is an exporter: Existing wires will remain until the importers are refreshed or the OSGi Framework is restarted
- Refresh:
  - All the bundle dependencies will be resolved again

What does this mean?

- Update or uninstall of bundles can lead to stale package references
- Refresh -> restart of the bundles



We need to re-think designs

- Just modularizing into bundles with clearly defined package dependencies is not enough!

- **Think about dynamics while building the system**
- **Think even more about dependencies**

Good Practices: Less Dependencies

- Only import packages that are really used/needed
- Use Import-Package rather Require-Bundle
- Only use Require-Bundle when it comes to split-packages
  - This is the unfortunately the case in many bundles of the Eclipse platform!

- **-> Reduce coupling**

Good Practice: OSGi Services

- **The way to deal with dynamics**

- Reduce coupling:
  - Split between interface and implementation
  - Lookup implementation at runtime
  - Dependency inversion
- **And always keep in mind: Services can come and go at any time**
  - You need to program against this from the beginning
  - Respect dynamics

ServiceListener / ServiceTracker

- But be careful:
  - If you lookup a service implementation, you get the direct reference to that object
  - If the implementing bundle goes away, you need to be careful not to keep this object referenced
- ServiceListener / ServiceTracker help you
  - ServiceListener: calls you back if something changes
  - ServiceTracker: listens to service listener events for you (less code than using service listeners manually)

Declarative Approaches

- **Declarative Services**
  - Part of the OSGi specification, declarative description of services with XML
- **Spring Dynamic Modules / Blueprint Service**
  - Spring goes dynamic with help of OSGi http://www.springframework.org/osgi
- **iPojo**
  - “Original” DI framework for OSGi
  - http://ipojo.org
- **Guice - Peaberry**
  - Guice: Performant, lightweight DI Framework
  - Peaberry: Extension of Guice for OSGi
  - http://code.google.com/p/peaberry/
  - http://code.google.com/p/google-guice/

Good Practices: Using Services

- Use a ServiceTracker
  - Don’t do all the service getting manually
  - Service tracker help you with dynamically coming and going services
- Better: Use declarative approaches!
  - Either DS or Spring DM
  - Both help you with service dependencies and dependency injection

Dynamics on the Server…

- Not much different to general OSGi dynamics, right?
  - Install, uninstall, update bundles
  - The less package dependencies the better
  - Use services to deal with dynamics
- Harder to program, but additional abstractions help
  - Declarative approaches
- Sounds nice!!!

- **But…**

Challenges

- Works mostly fine for things without state
  - stateless services
- Does that mean stateless web apps?
  - Interesting… ;-)
- What about…
  - Long-living transactions?
  - Session state?
  - Caches?

Session state

- The case:
  - Put an object into a session
  - Update the bundle that provided the type
  - Retrieve the object from the session
- What happens?

Ough…

ClassCastException !!!

- Types are not compatible across bundle updates
- Leaking classloaders

- **Only primitives in sessions**

Conclusions

- **Modularity is good**
  - You can implement real sustainable and flexible architectures
- **Dynamics is good**
  - Allows you fine-grained updates while you keep going
- **Both are not for free**
  - New structures, new designs, new challenges

Conclusions cont.

- **OSGi on the server is not without pain**
  - Leading edge technology
- **OSGi as the base infrastructure is the way to go**
  - Many promising solutions
  - You are lucky when you can control your setting
- **The programming model of the future**
  - from my point of view… :-)

Thank you for your attention!

- Questions and feedback welcome!

- Martin Lippert:
  lippert@acm.org
  www.martinlippert.org
  twitter.com/martinlippert