The Value Add of OSGi

Dmitry Sklyut, Chariot Solutions LLC

Agenda

- Problems and Challenges
- Introducing OSGi
- OSGi Architecture
- Summary
- Q/A

Problems and Challenges

Writing solid code is only part of the challenge

We are still left with:

- Packaging
- Configuration
- Deployment

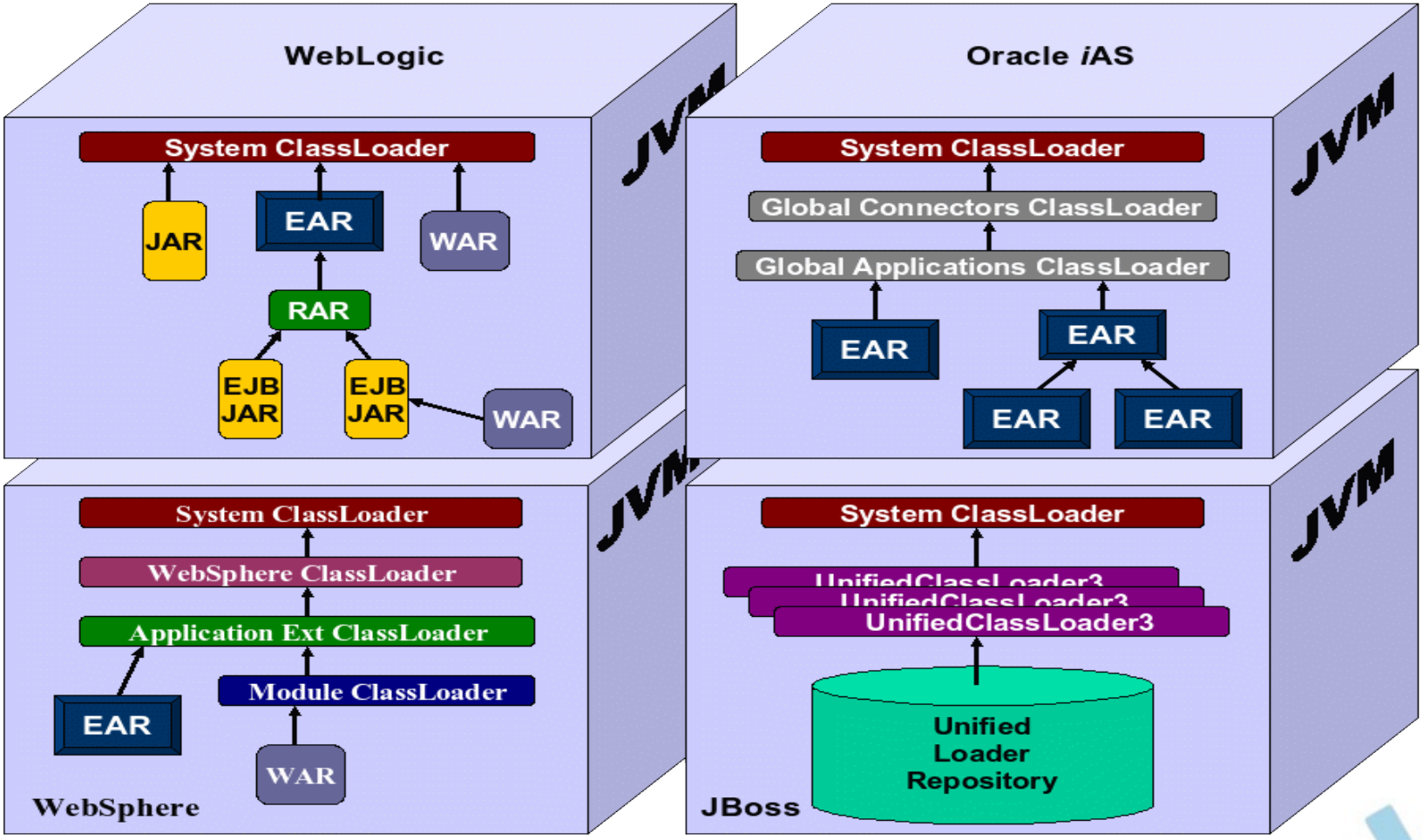
Problems and Challenges

- How to control complexity
- How to reuse existing components
- How to minimize “API abuse”
  - Published vs. Public interface dilemma
- How to ensure uptime during upgrades
- How to control multitude of configurations
- How to add features to the product without invasive rewrites

Where we are now

- Packaging – jars, wars, ears, rars, hars, etc
- Application server specific workarounds to keep classes scoped
  - or get a ClassCast/ClassNotFound Exceptions
- Internal APIs abused by client code
  - ((ServiceImpl)service).internalImplApi();
- Management – JMX, vendor provided tools
- Deployment – bring down server instance to install new version of application
  - Need redundant clusters to sustain SLAs

Where we are now...



What is OSGi?

Dynamic Module System for Java

Introducing OSGi

OSGi is Universal Middleware.

Software that you write once and can use in binary form universally: in many different platforms, many different industries, and for many different purposes.

Peter Kriens (OSGi evangelist)

OSGi Specification defines strict rules for:

- Portability
  - therefore Java Platform
- Deployment
  - bundle (JAR) as a unit of deployment
- Sharing
  - bundles run within a single JVM
- Collaboration
  - Service Oriented system design upfront
- Management
  - standard API for lifecycle management
- Security
  - built a top Java security model with fine grained constraints

Its Modular!

- Break up system into a number of modules
- Module, a.k.a “bundle” is a unit of deployment
- Strict visibility rules
- Provisioning process
  - Dependencies resolved before bundle is started
- Versioning support!

Its Dynamic!

- Modules can be
  - installed
  - updated
  - started
  - stopped
  - uninstalled

All that and no need to bounce server!

Its even Service Oriented!

- Bundles can publish services
- Service Registry allows other bundles to
  - Consume published service(s)
  - Look up service with query language
- Service lifecycle is handled by the runtime

Who is in charge?

- Sponsored by OSGi Alliance
  - http://www.osgi.org
- Started in 1999 with focus on embedded Java and networked devices
- 2003 – support for mobile devices
- 2004 – major OOS adoption
- now – moving into server-side...

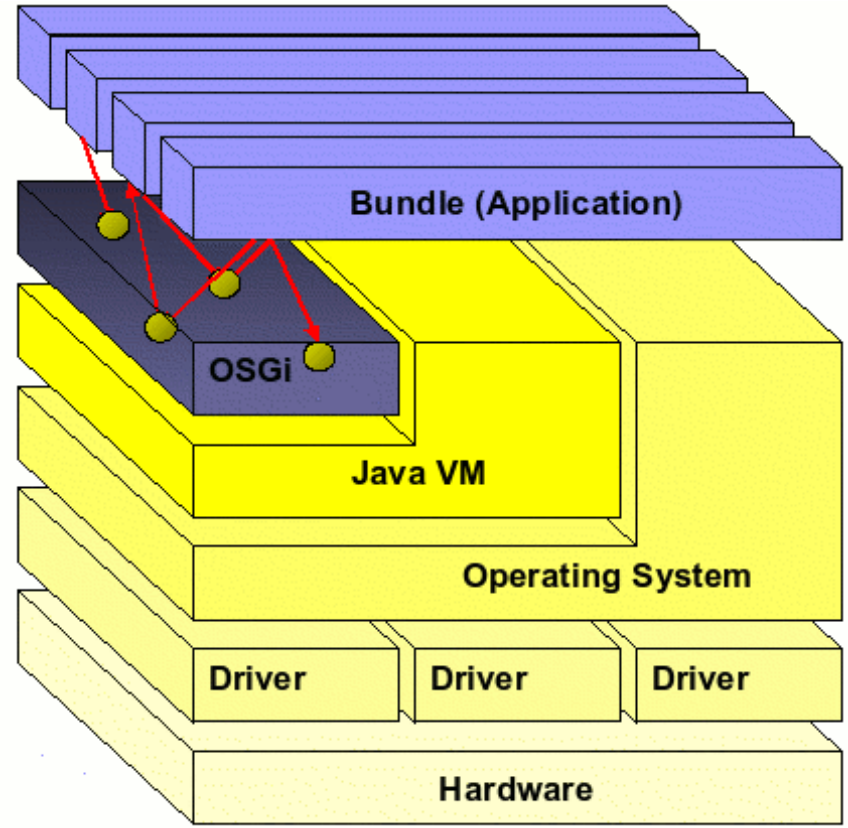
Introducing OSGi

- Open Source Implementations:
  - Equinox – Eclipse (www.eclipse.org/equinox)
  - Felix – Apache (http://cwiki.apache.org/FELIX/index.html)
  - Knoplerfish – Gatespace (http://www.knopflerfish.org)

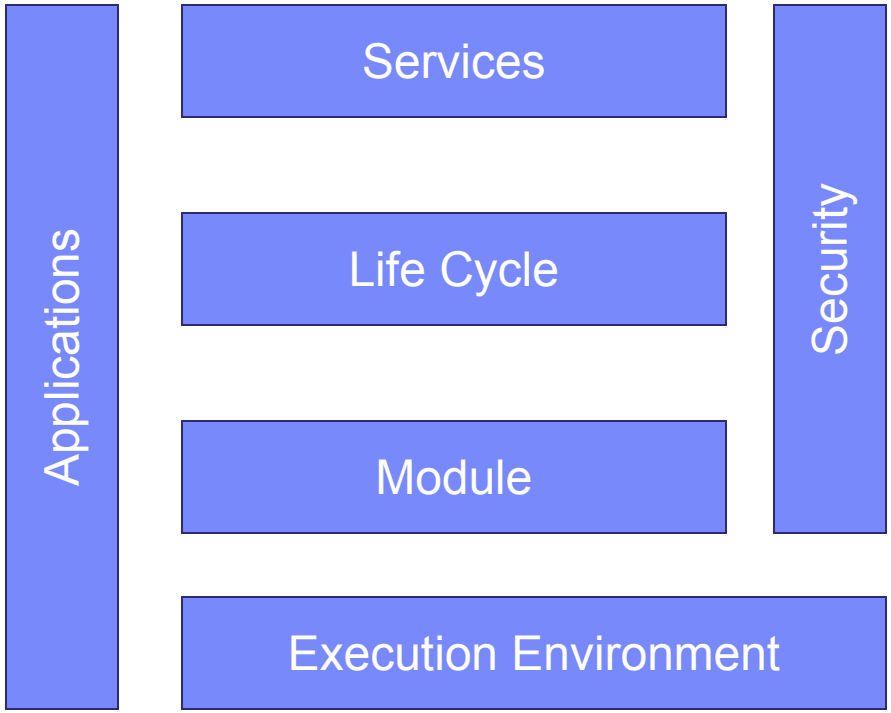
- Enterprise adoption:
  - Eclipse
  - IBM (WebSphere, Lotus, etc)
  - BEA (mSA, WebLogic, etc)
  - JOnAS
  - Oracle
  - and more...

Framework

- Thin layer on top of JVM
- Allows applications to run in shared environment
- Provides Classloading
- Life-Cycle Management
- Communication
- Collaboration
- Policy free



Framework Layers



Execution Environment

- OSGi API uses a subset of Java SE and Java ME CDC
- Implementations are free to use any type of Java platform configuration
- Security is not mandatory

Module Layer

- Defines module unit – bundle
- Controls visibility of classes within a bundle
- By default bundle is totally private
  - can't see inside with reflection or any other classloading trick
- Bundles export and import packages for use
- Supports versioning
  - Can have multiple versions of the same bundle in the runtime without conflict
- If dependencies are not satisfied bundles are not started

Life Cycle Layer

- Defines a complete API for bundle Life Cycle management
- Bundle can control other bundles life cycle
  - On demand download of required components
  - Uninstall obsolete version
  - Refresh runtime to account for newly added packages
- Bundle is always stopped before its package dependencies are changed
- Framework provides Start Level service to manage groups of bundles

Service Layer

- Dynamically links different bundles together
- Allows for composing larger system from smaller components
- Allows binding to services by interface name only
- Bundles can:
  - Register objects with Service Registry
  - Search Service Registry for objects
  - Receive notification when services registered or removed
- Services are automatically unregistered when bundle is stopped
- SOA in a Java VM! Look ma no “net”!

Provided Services

- OSGi provides number of standard services
  - Package Admin – provides information and can refresh current package sharing state of bundle
  - Permission Admin – manipulates permission of bundles
  - Start Level – ordering and grouping of bundles on startup
  - URL Handlers – allows dynamic contribution of new scheme or content handlers to URL class
  - Log Service – generic logging interface
  - Configuration Admin – allows bundles to be configured during runtime
  - User Admin – authentication and authorization service (not JAAS)
  - Preference Service – similar to Java Preferences class
  - XML Parser – locate a JAXP compatible parser
  - HTTP Service – Servlet 2.1 compatible runtime

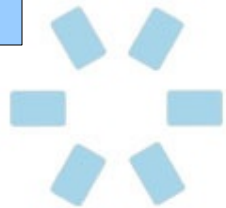
Security Layer

- Based on Java 2 security model
- Fine grained
- Permission Admin and Conditional Permission Admin services can be used to add permissions at runtime
- **Optional**

So what is this bundle thing?

- Nothing more than a jar with a custom manifest headers!

Bundle-ManifestVersion: 2
Bundle-Name: Service Client Bundle
Bundle-SymbolicName: service.client
Bundle-Version: 1.0.0
Bundle-Activator: demo.client.internal.Activator
Export-Package: demo.client
Import-Package: demo.service;version="[1.0.0,3.0.0)",
org.osgi.framework;version="1.0.0",
org.osgi.util.tracker



Export-Package

- Export-Package header is used to export packages for other bundles to use
  - passive contribution – bundles must be refreshed to see changes
- If package is not exported it is not visible outside of the bundle
- Separate published interface and internal implementation into separate packages
  - should be done anyway
  - limit visibility of client into internal implementation details
- Package can be exported with a version:
  - Export-Package: a.b.c.service;version=”1.0.0”

Import-Package

- Import-Package is used to set-up bundles classpath from external contributions
- Import-Package: a.b.c
  - will use the most current version available
- Import-Package: a.b.c;version:=”1.0.0”
  - use at least 1.0.0
- Import-Package: a.b.c;version:=”2.1.0-build56”;resolution:mandatory
  - If mandatory packages cannot be resolved, then the bundle fail to resolve.

Other ways to control classpath

- Import-Package: a.b.c;version=1.2.3;**resolution:=optional**
  - universe is not going to collapse if not present
- **Bundle-Classpath**: lib/optioanal.jar
  - is used to reference jar/directories files from within the bundle
- **DynamicImport-Package**: a.b.*
  - resolves packages at the point bundle tries to access that package
  - use as last resource
- **Require-Bundle: bundle.symbolic.name;version=...**
  - bind to all the exports of another bundle
  - Implort-Package takes precedence
- Bundle can also contain native code and other non Java resouces

Creating Bundles

- Vi, Notepad, Eclipse, Ant task, Maven plugin
  - Pick your poison
- Eclipse has very nice tooling support

Overview

General Information

This section describes general information about this plug-in.

ID: service.client

Version: 1.0.0

Name: Service Client Bundle

Provider:

Activator: demo.client.internal.ClientActivator Browse...

☐ Activate this plug-in when one of its classes is loaded

Plug-in Content

The content of the plug-in is made up of two sections:

Dependencies: lists all the plug-ins required on this plug-in's classpath to compile and run.

Runtime: lists the libraries that make up this plug-in's runtime.

Extensions

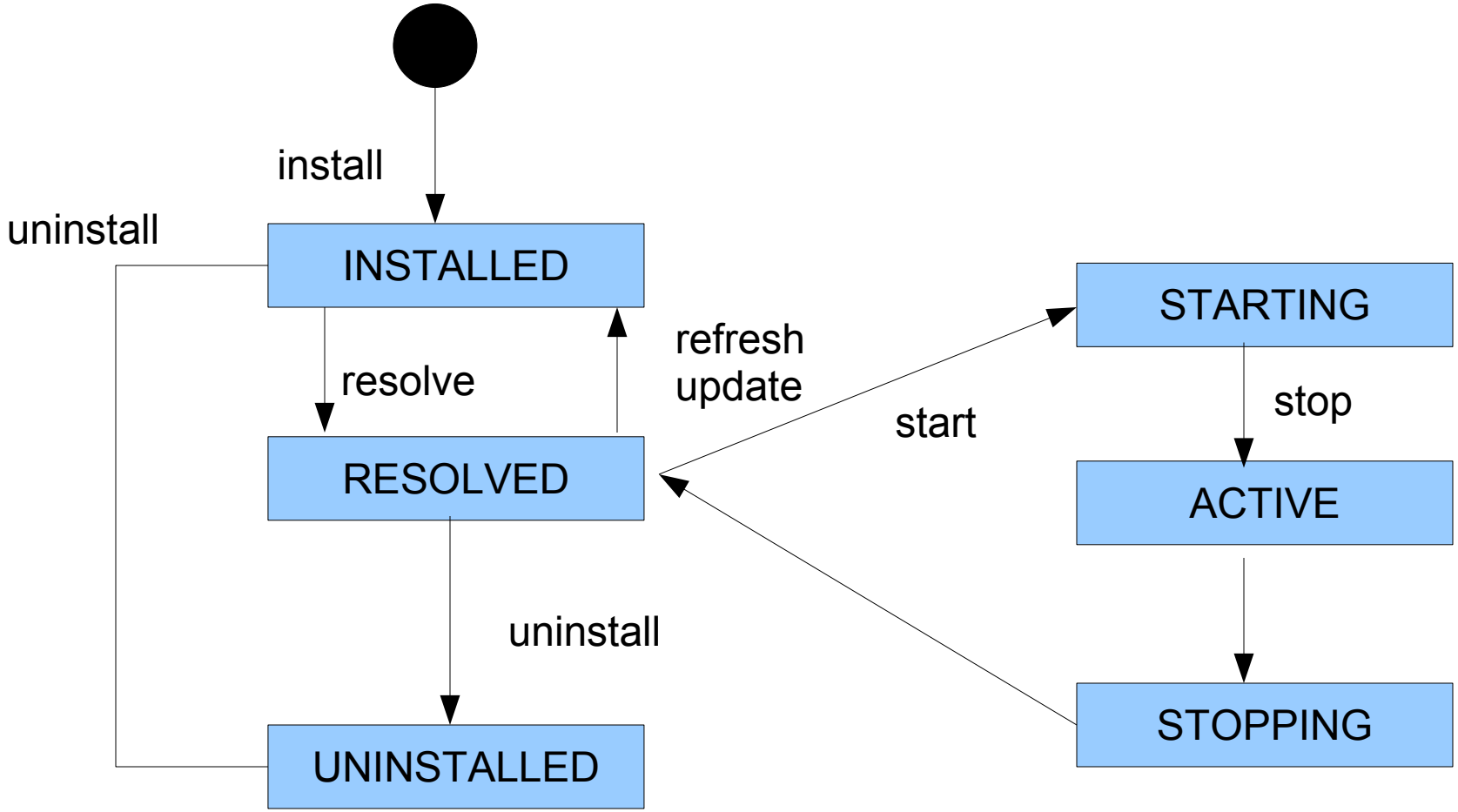
This plug-in may define extensions and extension points:

Extensions: declares contributions this plug-in makes to the platform.

Extension Points: declares new function points this plug-

Overview | Dependencies | Runtime | Build | MANIFEST.MF | build.properties

Bundle Life Cycle



Bundle Life Cycle

- Bundle is started by BundleActivator class
  - Specified by the header in the META-INF/MANIFEST.MF
- Bundle-Activator: a.b.c.internal.Activator
  - Main class for bundle
- Activator must implement
  - org.osgi.framework.BundleActivator
- Initialize bundle in
  - void start(BundleContext context) throws Exception;
- Clean up in
  - void stop(BundleContext context) throws Exception;

Services Registry Overview

- Service publishers should register/unregister service in BundleActivator.start/stop methods
  - BundleContext.registerService
  - BundleContext.unregisterService
- Service consumers follow the same paradigm
  - BundleContext.getServiceReference
  - BundleContext.getService(ServiceReference)
  - BundleContext.ungetService(ServiceReference)
- Active contribution – bundles see changes in the registered services immediately

Service Registry Overview

- Can use ServiceTracker utility
  - abstracts a lot of boiler plate code
  - still have resource acquisition issue
- or user Declarative Services (DS)
  - OSGi answer to IoC/DI
  - still not a POJO model but closer
  - only works with exported services

- Look for upcoming Spring OSGi project to provide pure POJO injection.

So how does it help?

- Packaging – Bundle
  - reduces coupling between components
  - enables independent development and testing
  - easier to maintain
  - reduces time to market
- Configuration – compose application from bundles
  - provide unique/custom features without polluting core
- Deployment – fine grained control of component lifecycle
  - install/update/etc single component
  - no server restart!
  - monitor with OSGi Console and JMX

What about ...

- JSR 277
  - http://www.osgi.org/blog/2006/10/jsr-277-review.html
  - Only address static module system similar to Require-Bundle header
  - “The Expert Group took a simplistic module loading model and ignored many of the lessons that we learned over the past 8 years. No built in consistency, no unloading, no package based sharing.”
- JSR 294
  - Allow module support at the VM layer
- JSR 291
  - OSGi r4.1 specification
- JPF - http://jpf.sourceforge.net/
- there are other contenders

Summary

- Reduces software complexity
- Software components are smaller
  - Easier to write, test and deploy
- Dependencies are known upfront without examining all of the code base
- Maximizes re-use of existing components
- Removes deployment platform problem
- Service Oriented – reduces coupling to implementation

Questions???