



**ORACLE**

## **OSGi and Java EE: Friends or Foes?**

**Mike Keith**  
Oracle Corporation

### **About Me**

---

- Member of OSGi Alliance Enterprise Expert Group (EEG)
- Member of Java EE 6 (JSR 316) and various other JCP expert groups
- Java and Persistence Architect for Oracle
- Almost 20 years experience in distributed, server-side and persistence implementations
- Presenter at numerous conferences and events

**ORACLE**

## About You

---

- ❖ How many people are using or have used Java EE?
- ❖ How many people have some knowledge of what Java EE is?
- ❖ How many people have used OSGi?
- ❖ How many people have some idea about what OSGi is?
- ❖ How many people think that OSGi is going to play some role in your future?

ORACLE

## What is Java EE?

---

- Java EE = Java Platform Enterprise Edition (current release is Java EE 5)
- J2EE initially created in 1999
  - Java split into Standard and Enterprise Editions
- Suite of technologies that can be used to create complex enterprise systems
- Complete stack from message queue to mail API
  - “Full meal deal”
- Detailed specifications to achieve portability, interoperability, and compatibility

ORACLE

## Java EE Component Technologies

---

- JSP, EL, JSTL, JSF, Servlets
- EJB, JPA, JTA, JCA
- JAF, JMS, JavaMail
- WS Metadata, JAX-WS,
- JAXB, JAXR, StAX, SAAJ
- JMX Management, Deployment
- JACC

ORACLE

## What is OSGi?

---

- **OSGi = Open Services Gateway initiative**
- **OSGi Alliance** formed in 1999 and is now composed of over 30 member companies using and advancing OSGi
- Initial mandate to create open standards for delivery of services to networks of devices
- Focus was to provision and manage home automation services on devices connected through a gateway
- Flexible deployment model assumes that services can come and go over time in the system -> a dynamic system
- Currently limited to Java - no plans to change

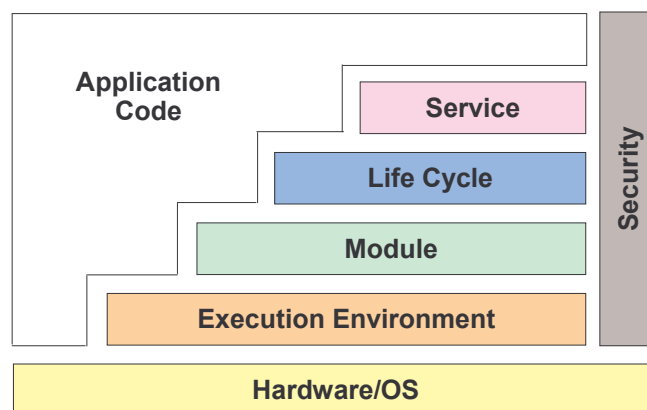
ORACLE

## Specification Development

- Three main OSGi specifications:
  - **Core Specification** – kernel framework contains basic layers of functionality intrinsic to the OSGi model
  - **Service Compendium** – Additional services that are largely independent of each other and implemented/used when available
  - **Mobile Specification** – Set of services suited to mobile embedded devices and platforms
- Specifications developed in 5 expert groups:
  - Core Platform Expert Group (CEPG)
  - Enterprise Expert Group (EEG)
  - Mobile Expert Group (MEG)
  - Residential Expert Group (REG)
  - Vehicle Expert Group (VEG)

ORACLE

## Core Services



ORACLE

## Compendium Services

- Multitude of services that may be separately deployed, registered and used

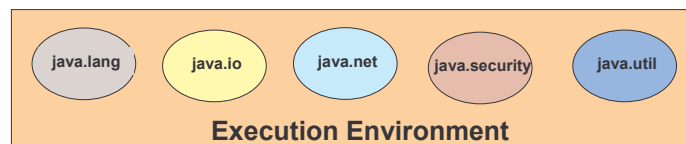
Examples:

- Http Service
- Log Service
- Configuration Admin Service
- XML Parser
- User Admin Service
- Metatype Service

ORACLE

## Execution Environment

- OSGi defines a minimal execution environment (EE) for running the Core services (ee.minimum) and a fuller version (ee.foundation) that adds some more classes
- Subset of Java ME profiles
- More may be used, depending upon the requirements of the application, but is not required



ORACLE

## Module Layer

---

- Packaging and deployment unit called a **"bundle"**
- Simple artifact that is really just a JAR file containing classes, resources and a manifest file
- Manifest metadata describes the bundle, including:
  - **Activator** class – a class to be invoked after the bundle is started
  - **Imported packages** – packages that this module depends on and invokes, but that are in some other (unknown) module(s)
  - **Exported packages** - packages that this module is allowing other modules to see and invoke

ORACLE

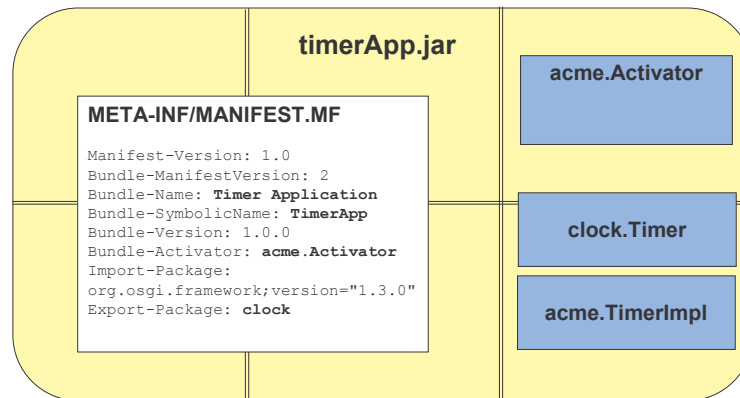
## Module Layer

---

- Each bundle has its own classloader
- Class visibility is determined by the dependencies listed in the bundle manifest
- Imported classes are version-specific (even if version numbers are not specified)
- Once a bundle is installed then all of its dependencies must be fully **resolved** before it can run
- Resolution means that all of the imported packages must be **wired** to available exported ones
- Resolution is performed using an algorithm that is documented and standardized

ORACLE

## Module Layer



ORACLE

## Activator Class

```
package acme;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;

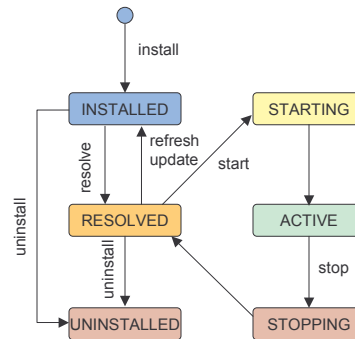
public class Activator implements BundleActivator {
    BundleContext bc;

    public void start(BundleContext context)
        throws Exception {
        // init code
    }
    public void stop(BundleContext context)
        throws Exception {
        // Cleanup code
    }
}
```

ORACLE

## Life Cycle Layer

- Install – Load bundle into the framework
- Resolve – Wire bundle dependencies to exporting classes
- Start – Invoke bundle activator on its own thread
- Stop – Terminate bundle activator thread
- Uninstall – Unload a bundle from the framework



ORACLE

## Life Cycle Layer

- Life cycle events provide a series of triggers at two levels
  1. Framework events – indicate framework state change
    - Examples: ERROR, STARTED, PACKAGES\_REFRESHED
  2. Bundle events – indicate changes to bundle state
    - Examples : INSTALLED, RESOLVED, STARTED, STOPPED
- Application and service bundles can register event listeners to act upon state changes of any bundle
- Versioning built right into the system
  - Update and Refresh provide for changing versions of a bundle or one of its dependent bundles

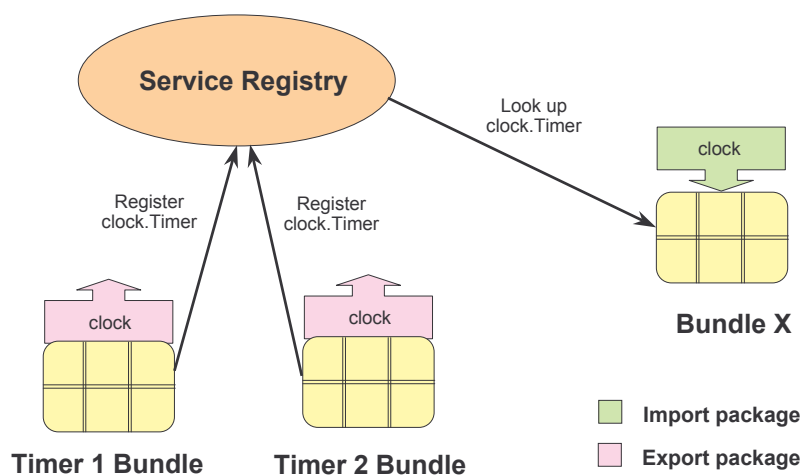
ORACLE

## Service Layer

- OSGi is a “Service Framework”
- Service layer provides much of the basic infrastructure for services in the framework
- Clean separation between interface and implementation
- One or more bundles implementing a service register it in the “Service Registry”
- Other bundles discover or are notified of the service and bind to a particular implementation
- Services may come online or go offline at any time

ORACLE

## Service Layer



ORACLE

## Registering a Service

```
public void start(BundleContext context)
    throws Exception {
    bc = context;
    context.registerService(
        clock.Timer.class.getName(),
        new acme.TimerImpl(),
        new Hashtable());
}
```

ORACLE

## Obtaining a Service

- Can use a ServiceTracker in Bundle X to track the service
- Bundle X Activator method:

```
public void start(BundleContext context)
    throws Exception {

    customizer = new TimerServiceTracker(context);
    tracker = new ServiceTracker(
        context,
        Timer.class.getName(),
        customizer);

    tracker.open();
}
```

ORACLE

## ServiceTracker Customizer

```
public class TimerServiceTracker
    implements ServiceTrackerCustomizer {
    BundleContext bc;
    Timer timer;

    public TimerServiceTracker(BundleContext bc) { this.bc = bc;}

    public Object addingService(ServiceReference reference) {
        Timer service = (Timer) bc.getService(reference);
        if (timer == null) {
            timer = service;
        }
        return service;
    }
    public void modifiedService(...) { ... }
    public void removedService(...) { ... }
}
```

ORACLE

## Security Layer

- Security can be optionally disabled or fully enforced
- Cross-cuts the other layers
- Makes use of existing Java 2 notions of security
  - permissions
  - jar signing
- Entire bundle must be signed (no partial bundle signing)

ORACLE

## Java EE and OSGi

Java EE	OSGi
Logical component-level dependencies	Package-level code dependencies
Units of deployment are WAR's, JAR's, and EAR's	Unit of deployment is bundle
Interface-based services as local/remote EJB components	Interface-based services as classes in bundles
Component life cycle callback hooks	Bundle and framework life cycle callback notification
JNDI under container control	Service registry shared bundle access
Application code strictly controlled	No real application coding constraints

ORACLE

## Service Locator Pattern

- Traditional problem in Java EE when multiple implementations of a service exist
- How does an application locate an implementation?

Example:

JDBC DriverManager

- Each driver registers with the DriverManager class
- When passed a URL, sequentially ask each one
- No standard, correct or clean way to achieve this
  - components each solve it differently

ORACLE

## OSGi hosting Java EE

---

- Current thrust in EEG is to integrate Java EE technologies
  - Application server architectures
  - Java EE profiles
  - “A la carte” technology stack
  - Versioned infrastructure libraries
- Functional requirements for Java EE applications to run on top of OSGi framework
  - Java EE applications using OSGi events
  - OSGi “violations” (context classloader, etc.)

ORACLE

## Java EE using OSGi

---

- Java is missing a module system!
- JSR 277 (Java Module System)
- JSR 291
- JSR 294
- Glassfish v3  
(Java EE 6 Reference Implementation)
- Uncertainty...

ORACLE

## Summary

---

- ✓ OSGi is more about modularity, dependency management and service flexibility
- ✓ Java EE is a complete API suite for development of enterprise applications
- ✓ OSGi is becoming more and more suitable for use in the enterprise
- ✓ Java EE and OSGi have complimentary strengths
- ✓ OSGi provides an ideal runtime and deployment framework for Java EE applications

ORACLE

## Links and Resources

---

- OSGi standards and specifications  
<http://www.osgi.org>
- Java EE Integration in OSGi (RFP 98)
- Glassfish v3 (Java EE 6 RI)  
<http://glassfish.dev.java.net>
- OSGi R4.1  
<http://www.osgi.org/Download/Release4V41>
- OSGi implementations
  - Equinox - <http://www.eclipse.org/equinox>
  - Felix - <http://felix.apache.org>
  - Knopflerfish – <http://www.knopflerfish.org>

ORACLE