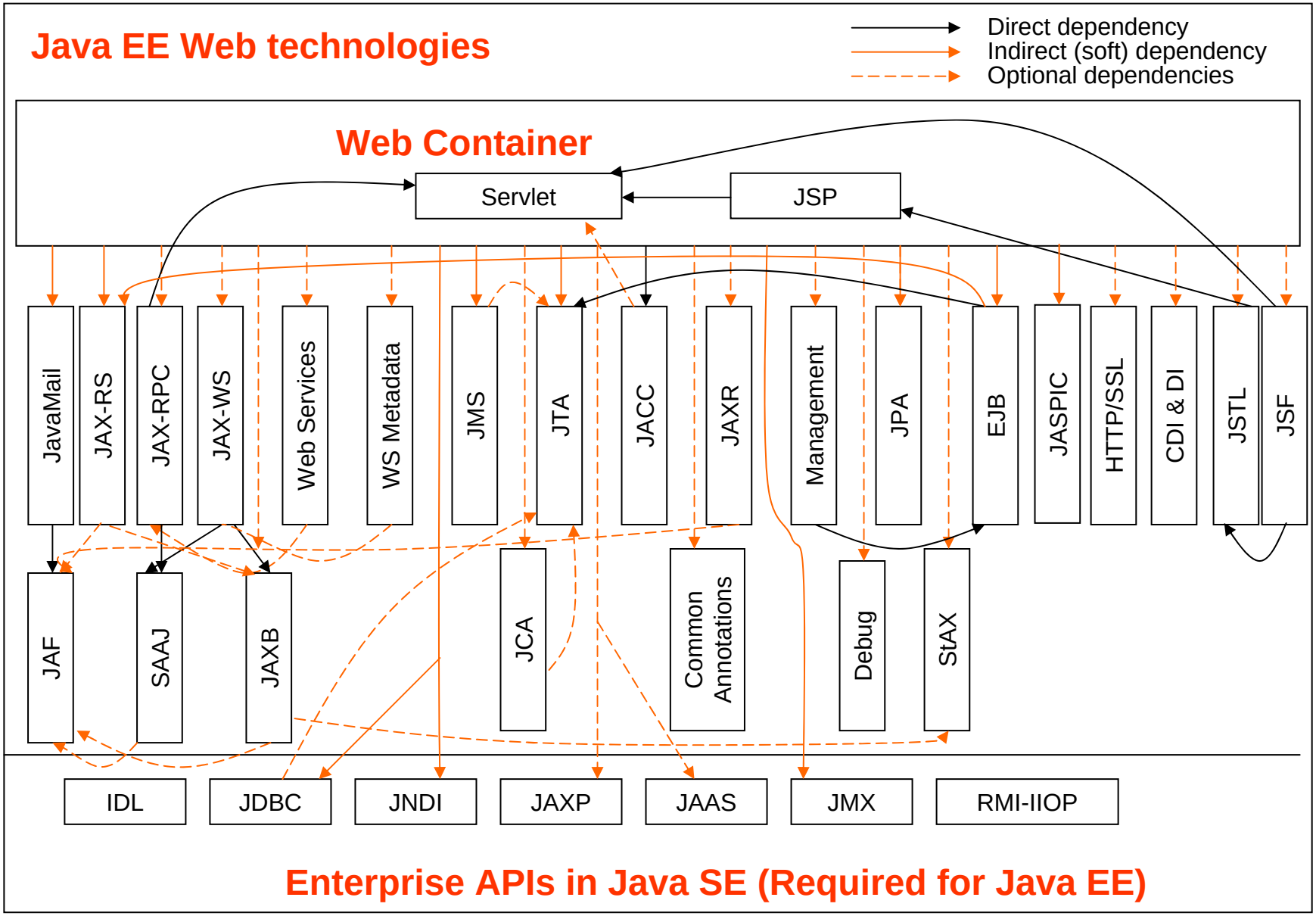
Is OSGi Ready for the Enterprise?

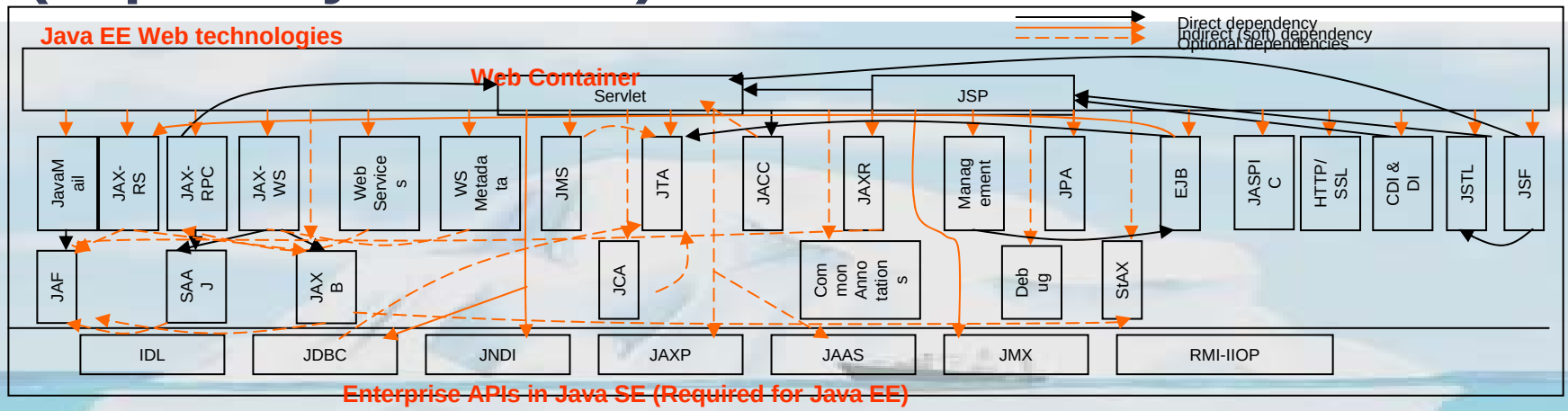
Ian Robinson
IBM Distinguished Engineer

The Questions

- Why should we care?
  - Does Enterprise Java need OSGi?
  - Does OSGi need Enterprise Java?


Java EE Web technologies
Direct dependency
Indirect (soft) dependency
Optional dependencies
Web Container
Servlet
JSP
JavaMail
JAX-RS
JAX-RPC
JAX-WS
Web Services
WS Metadata
JMS
JTA
JACC
JAXR
Management
JPA
EJB
JASPIC
HTTP/SSL
CDI & DI
JSTL
JSF
JAF
SAAJ
JAXB
JCA
Common Annotations
Debug
StAX
IDL
JDBC
JNDI
JAXP
JAAS
JMX
RMI-IIOP
Enterprise APIs in Java SE (Required for Java EE)

(capability == bloat)?



HTTP Session Replication | Connection Management | Persistence Providers | User Registries | Event Mgmt

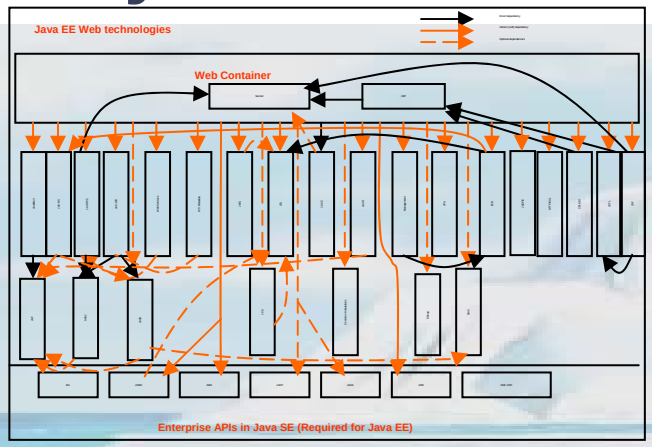
Logging and Tracing | MBean Server | Admin Console and scripting | DataSources | Messaging Engines | Security

Transport Channels | Workload Management | HTTP Serving and routing | High Availability Management | Transaction Logging and Recovery

Thread management | Classloading | Provisioning | Monitoring | Policy and SLA mgmt | Configuration

...

Beyond Java EE…



Batch

Telephony

BPM

Dynamic Scripting

SCA

Business Rules

Complex Event Processing

…

HTTP Session Replication

Connection Management

Persistence Providers

User Registries

Event Mgmt

Logging and Tracing

MBean Server

Admin Console and scripting

DataSources

Messaging Engines

Security

Transport Channels

Workload Management

HTTP Serving and routing

High Availability Management

Transaction Logging and Recovery

Thread management

Classloading

Provisioning

Monitoring

Policy and SLA mgmt

Configuration

…

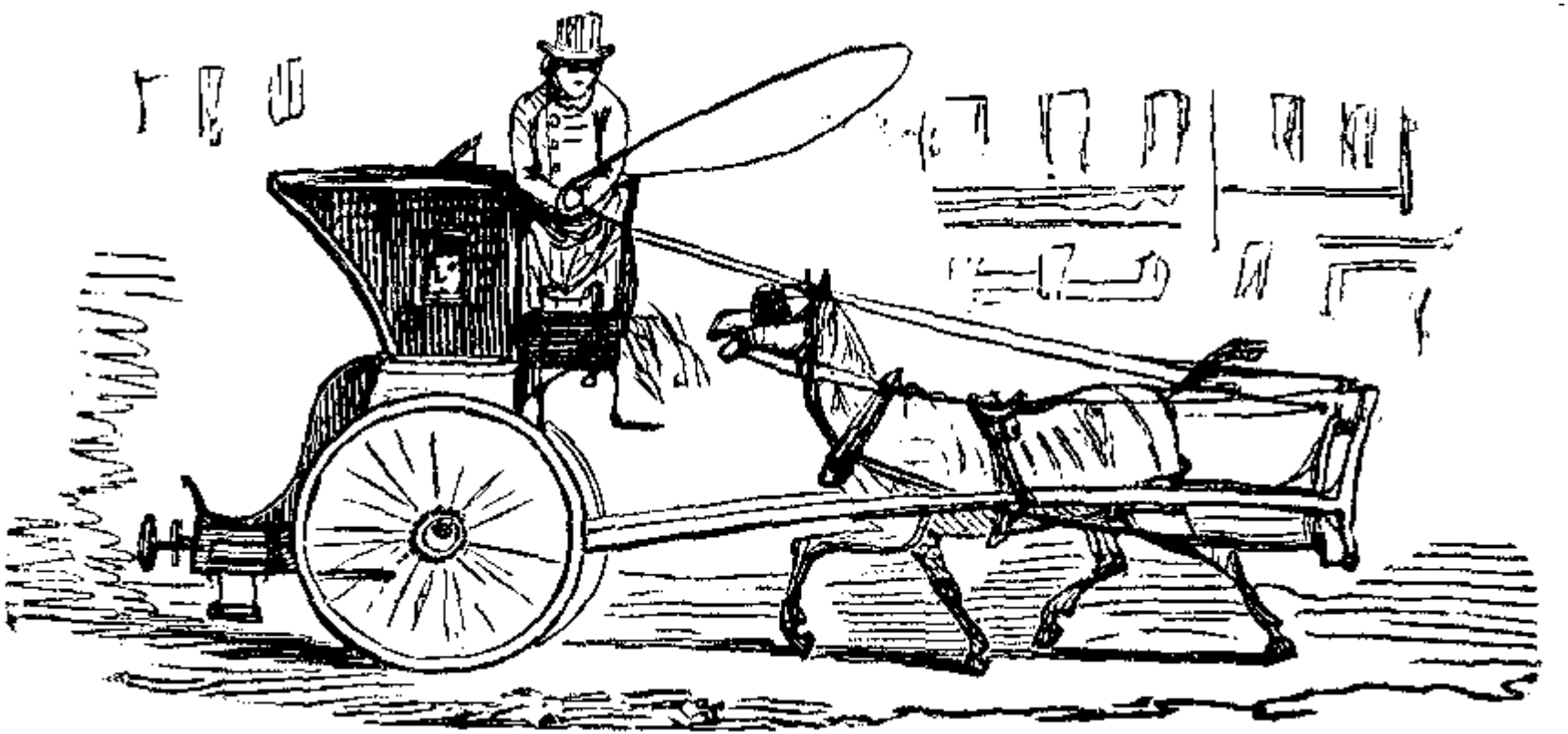
Modularity *inside* Enterprise platforms

- The major AppServer and Integration vendors and Open Source communities have all embraced OSGi for their own internal architecture
  - IBM WebSphere, Oracle WebLogic, Red Hat JBoss, Sun GlassFish, SpringSource dm Server, Paremus Service Fabric, Eclipse Platform, Apache Geronimo, (non-exhaustive list)
    - http://www.osgi.org/wiki/uploads/News/2008_09_16_worldwide_market.pdf
- These complex engineering projects required a modular approach to development as they evolved and grew
  - Breaking the problem into well-defined coherent modules enables development teams to focus on their area of concern.
  - OSGi not only *enables* but *enforces* modularity
- Does the Enterprise *runtime* need OSGi? The vendors answered with a resounding *Yes*.

What about the Applications?

- The enterprise landscape until recently has been OSGi on the inside only. Why?
- Primarily because:
  - Java EE defines an enterprise application programming model and container contracts to honour it
  - OSGi has typically defined platform specifications and services for those platforms.
    - It hasn't had much to say on application concerns such as
      - What's the component model?
      - Whets the persistence architecture?
      - How are qualities of service configured?
      - Remoting
- This has changed in OSGi v4.2.

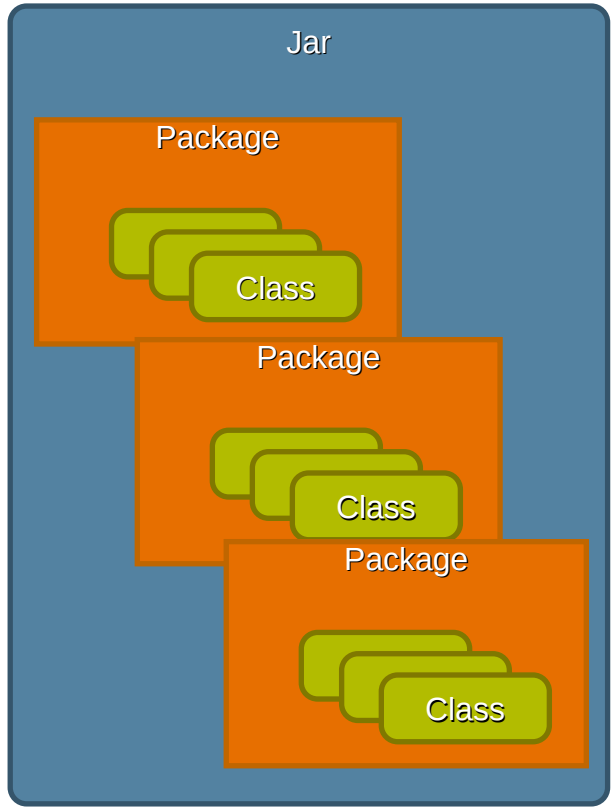
What's the Problem?



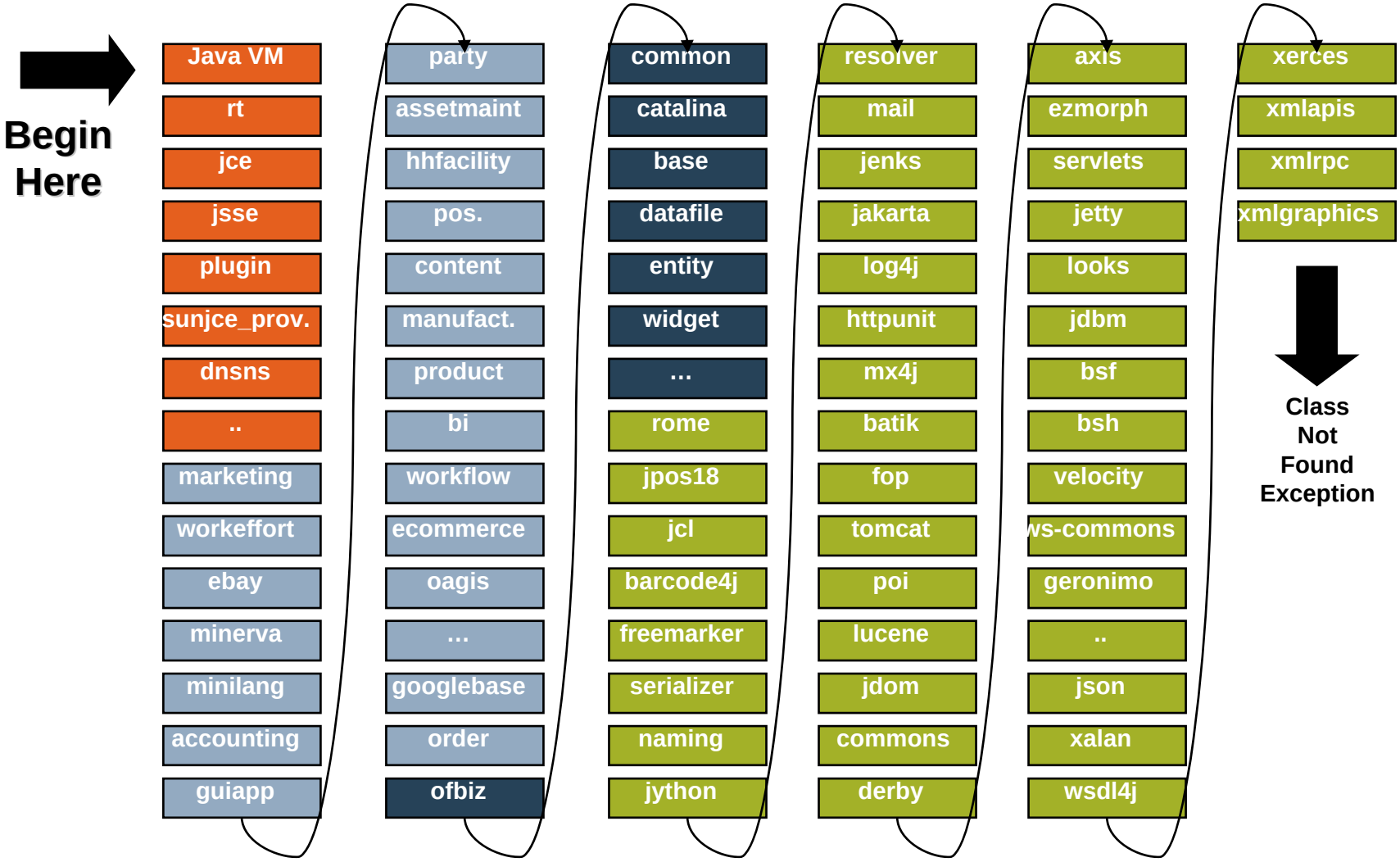
http://www.gutenberg.org

Problems with JARs

- Java Platform Modularity
  - Classes encapsulate data
  - Packages contain classes
  - Jars contain packages
- Class visibility:
  - private, package private, protected, public
- No “jar scoped” access modifiers.
- No means for a jar to declare its dependencies.
- No versioning.
- Jars have no modularization characteristics
  - At runtime there is just a collection of classes on a global classpath



The Global Classpath

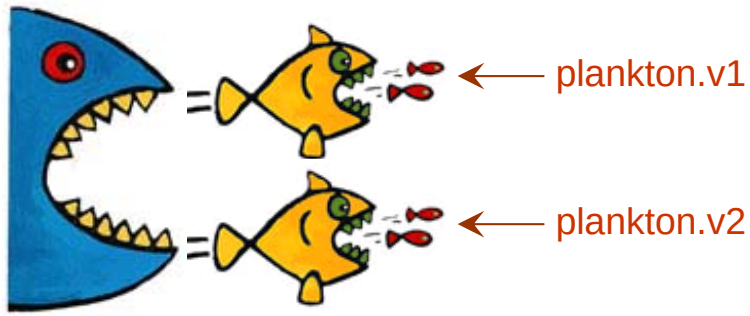


Problems with EARs/WARs

Enterprise Apps have isolated classpaths but…

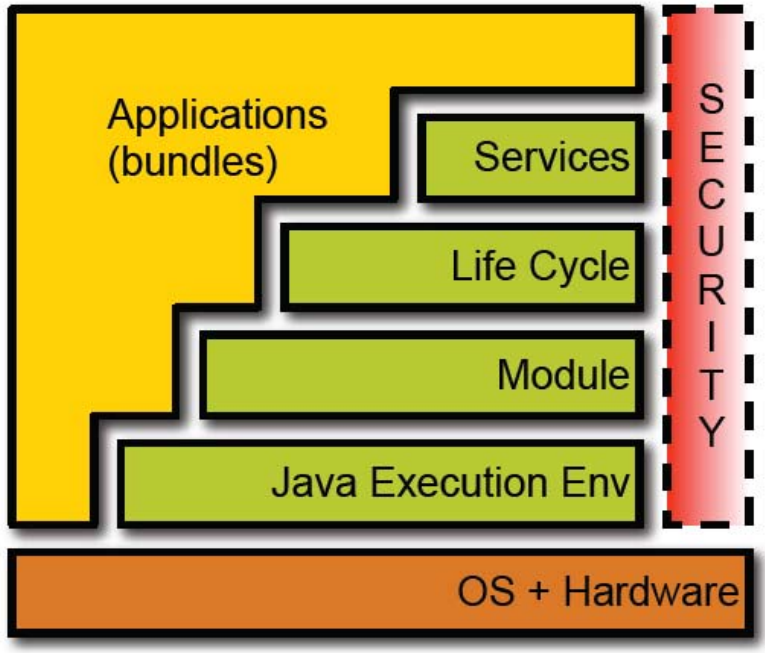
- *Across apps* - each archive typically contains all the libraries required by the application
  - Common libraries/frameworks get installed with each application
  - Multiple copies of libraries in memory

- *Within apps* - 3rd party libraries consume other 3rd party libraries leading to version conflicts



OSGi – The Dynamic Module System for Java

- The OSGi Service Platform specifies a modular architecture for dynamic component based systems
  - Execution Environment
  - Module Layer
  - Life Cycle Layer
  - Service Layer
- Runs on a variety of standard Java profiles.
- Introduces *Bundles* as modules



Bundle Metadata

- The higher-level module is the “bundle” which is a normal jar containing:
  - Classes
  - Resources
  - Manifest

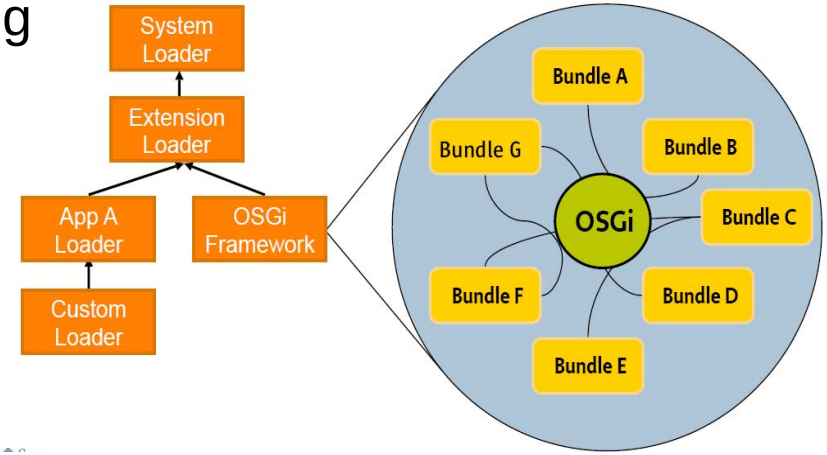
META-INF/MANIFEST.MF

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: MyService bundle
Bundle-SymbolicName: com.sample.myservice
Bundle-Version: 1.0.0
Bundle-Activator: com.sample.myservice.Activator
Import-Package:
 org.apache.commons.logging;version="1.0.4"
Export-Package:
 com.sample.myservice.api;version="1.0.0"
```

- OSGi headers in the manifest include:
  - Bundle-Version: **Multiple versions** of bundles can be active concurrently.
  - Import-Package: Explicitly declares the (package) **dependencies** on other bundles. Used by the OSGi f/w for bundle resolution.
    - Can explicitly define the required version or version range for each package.
  - Export-Package: Declares **package visibility** outside the bundle. Other packages are encapsulated by the bundle.
  - Bundle-Activator (used to notify the bundle of lifecycle changes)
- Eclipse tooling provides convenient editors for the manifest

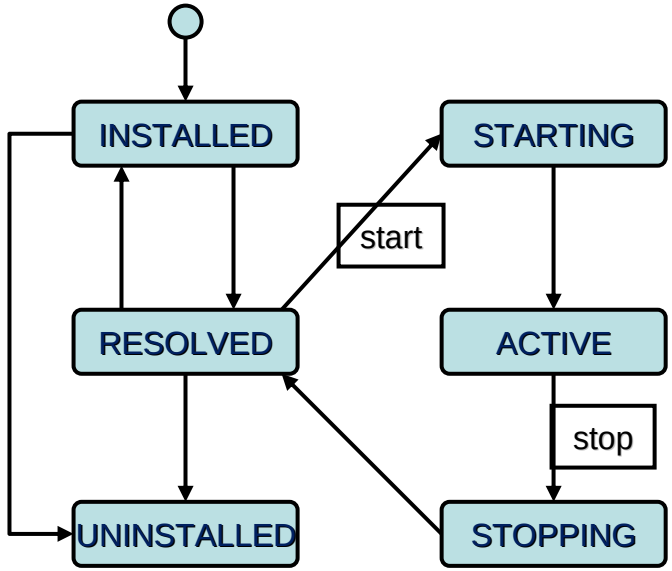
OSGi Class Loading

- No global, flat classpath to search. Each bundle has its own classloader / classpath. More efficient for large systems.
- Class sharing and visibility is based on declarative dependencies
  - Not constrained by hierarchical class loader trees built up at runtime.
  - OSGi f/w takes care of working out the dependencies.
- Multiple versions of bundles supported concurrently.



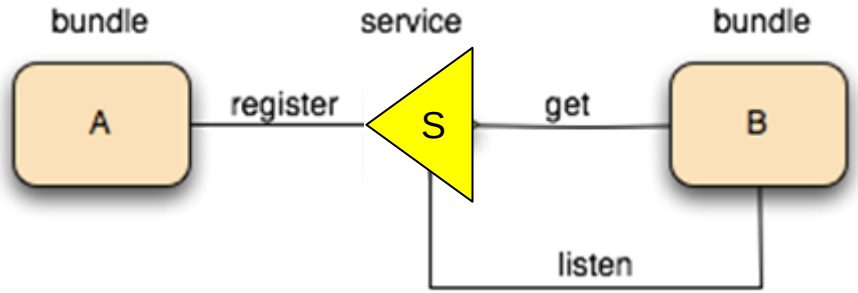
Bundle Lifecycle

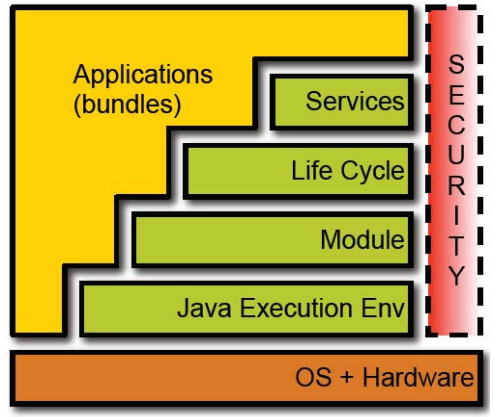
- Optional BundleActivator class has opportunity to initialize and cleanup a bundle
  - Header in Manifest refers to this class
- BundleActivator has 2 methods
  - Start: Initialize bundle
  - Stop: Cleanup



- Resolution of dependencies at deployment time. If resolution fails, bundle is not available for use.
  - No ClassNotFoundExceptions at runtime
- Bundle lifecycle is dynamic and independent of the rest of the f/w.
  - BundleListener can be registered for bundle state change events.

OSGi Service Layer





- OSGi intrinsically supports SOA through its service layer.
- The core platform includes a non-durable “Service Registry” component.
- Services are published to and discovered from this Service Registry.
  - Services are the primary means of collaboration between bundles.
- An OSGi service is a POJO published to the SR under one or more Java interface names with optional key/value pair metadata
  - Service discovery can be filtered on the key/value metadata
- Services are fully dynamic and typically have the same lifecycle as the bundle that provides them.

OSGi Enterprise Specification

- Release date – 22 March 2010
  - The product of the OSGi Enterprise Expert Group (EEG)
- Brings Enterprise technologies and OSGi together
- Using existing Java SE/EE specifications:
  - JTA, JPA, JNDI, JMX, WebApps…
- Adds Spring-derived component model and DI container

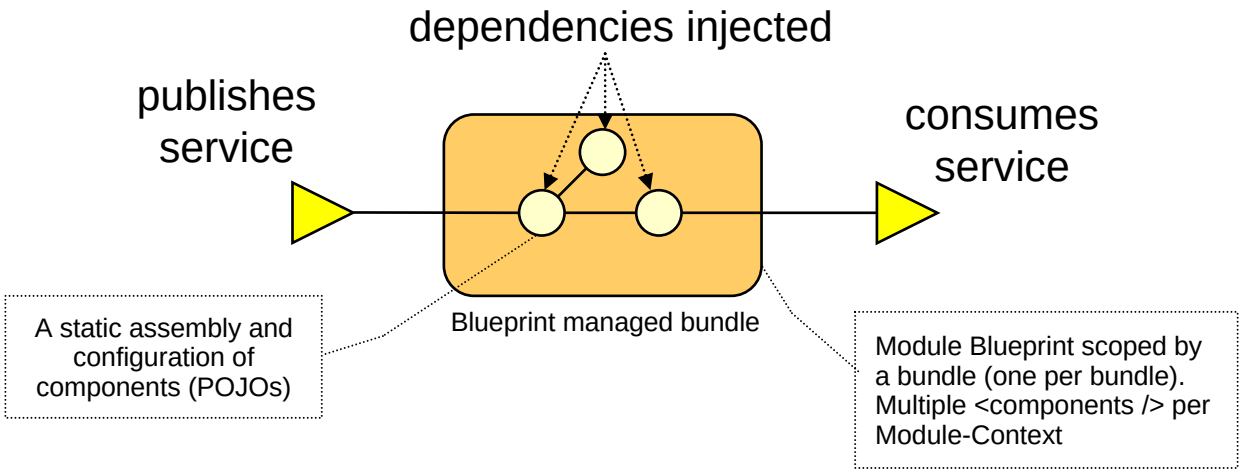
- **Java EE provides the core enterprise application programming model**
- **Deploying modules as OSGi bundles simplifies reuse between applications, provides versioning, encourages (and enforces) modular design and enables dynamic module updates.**

New Open Source Activities

- Apache “Aries” created as a new Apache incubator project in Sep 2009:
  - to provide enterprise OSGi spec implementations http://incubator.apache.org/aries/
  - to provide an environment to collaborate and experiment with new technologies to inform further EEG standardization.
    - In particular the programming model aspects of OSGi applications in an enterprise environment such as multi-bundle composites.
  - to build a broad development community to encourage implementation and adoption of EEG specs
- Eclipse Enterprise Modules (“Gemini”) - currently being provisioned.

  - Sub-projects will provide EEG spec implementations http://www.eclipse.org/proposals/gemini/

Blueprint Components and Service

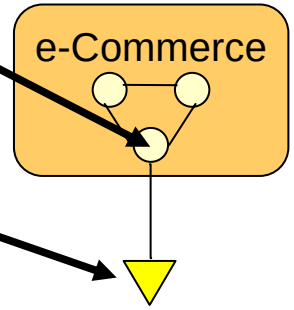


- Specifies a Dependency Injection container, standardizing established Spring conventions
- Configuration and dependencies declared in XML “module blueprint”, which is a standardization of Spring “application context” XML.
  - Extended for OSGi: publishes and consumes components as OSGi services
- Simplifies unit test outside either Java EE or OSGi r/t.
- The Blueprint DI container is a part of the server runtime (compared to Spring which is part of the application.)

Exploiting Blueprint Components and Services

e-Commerce bundle

```
<blueprint>
  <bean id="shop" class="org.example.ecomm.ShopImpl">
    <property name="billingService" ref="billingService" />
  </bean>
  <reference id="billingService"
      interface="org.example.bill.BillingService" />
</blueprint>
```



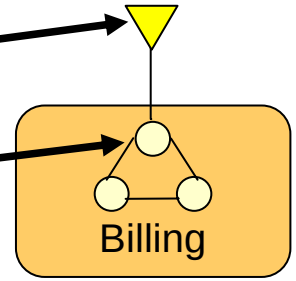

```
public class ShopImpl {

private BillingService billingService;
void setBillingService(BillingService srv) {
      billingService = srv;
      }

void process(Order o) {
      billingService.bill(o);
      }
}
```

-injected service reference
-service can change over time
-can be temporarily absent
  without the bundle caring
-managed by Blueprint container

Exploiting Blueprint Components and Services

Billing service bundle

```
<blueprint>
  <service ref="service" interface =
           "org.example.bill.BillingService" />
  <bean id="service" scope="prototype"
      class="org.example.bill.impl.BillingServiceImpl" />
</blueprint>
```



-"prototype" scope indicates a new instance is created by the container for each use.
-"singleton" scope is the default.

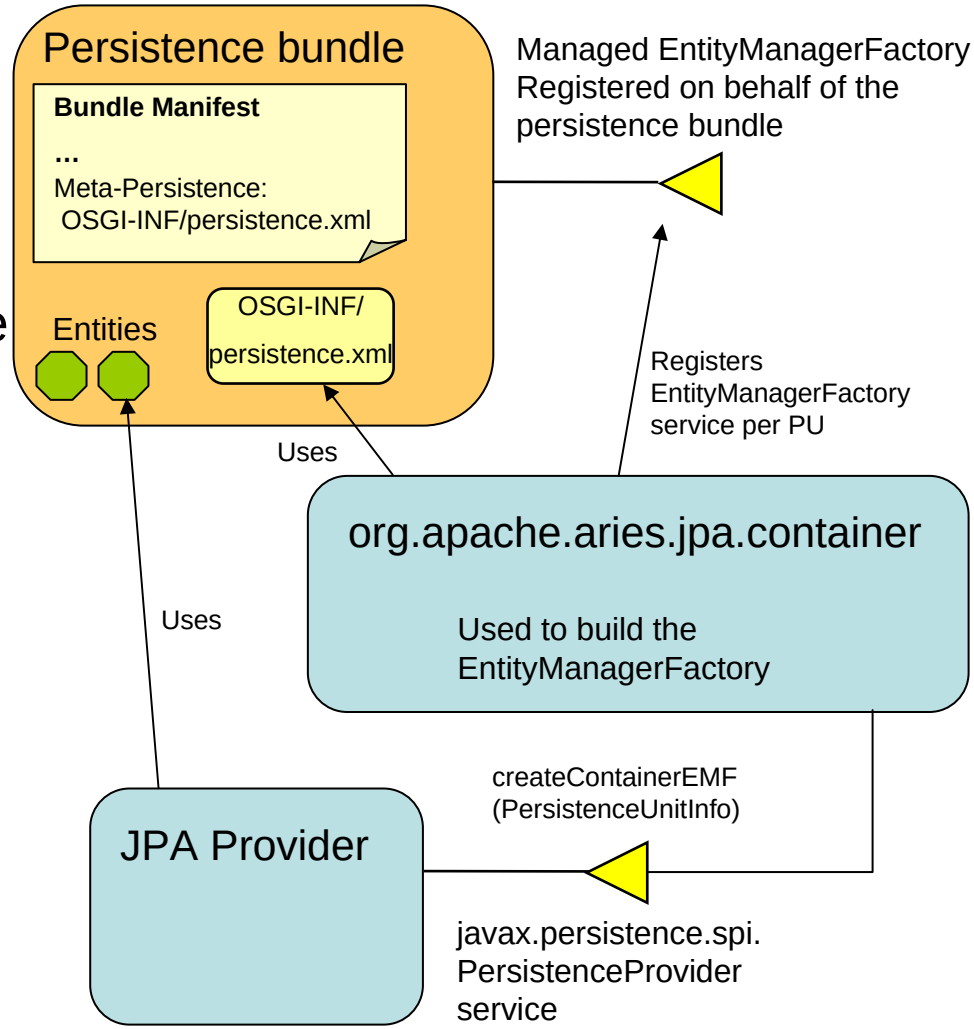
```
public interface BillingService {
      void bill(Order o);
}
```

Aries Blueprint Container

- Pre-dates Aries – moved from Apache Geronimo
- The Aries BP container implementation is highly extensible:
  - Namespace handlers supported to extend the Blueprint definitions
  - Bean interceptors can be registered by handlers
- Other Aries components contribute handlers – “jpa” and “jta” handlers.
- Some other Aries components are implemented as Blueprint bundles themselves
  - e.g. JPA container

Java Persistence and Persistence Bundles

- Each persistence bundle has its standard JPA metadata located through the Meta-Persistence bundle header.
- JPA Container locates a JPA Provider which can service the PU and registers a Provider-created EntityManagerFactory service for each PU in each persistence bundle
  - EMF service lifecyce follows the persistence bundle lifecycle
- JPA Provider gets persistence bundle classloader from PUInfo.



Aries JPA Container – Blueprint Integration

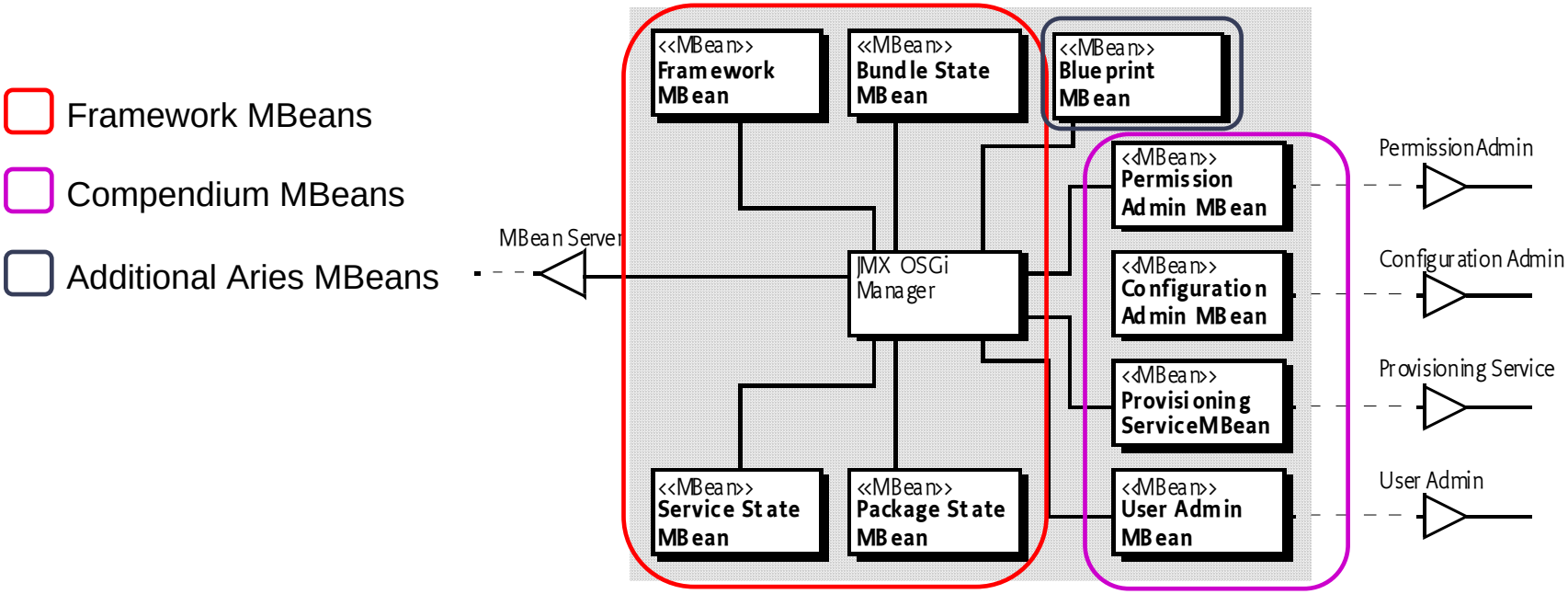
- The Aries JPA container context bundle provides a blueprint namespace for dependency injection of managed JPA resources.
- Managed persistence units (EntityManagerFactory objects) can be injected with or without a JTA Transaction Services implementation.
- Managed persistence contexts (EntityManager objects) are only available with a JTA Transaction Services implementation.
- Both managed persistence units and managed persistence contexts behave as per the JPA specification.

- Example blueprint with JPA resource injection and container-managed transactions:

```
<blueprint
xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
xmlns:jta="http://aries.apache.org/xmlns/transactions/v1.0.0"
xmlns:jpa="http://aries.apache.org/xmlns/jpa/v1.0.0">

  <bean id="appMgd" class="com.acme.AppManaged">
    <jpa:unit property="emf" unitname="myUnit" />
  </bean>

  <bean id="containerMgd" class="com.acme.Container">
    <jpa:context property="em" unitname="myUnit"/>
    <jta:transaction method="*" value="Required" />
  </bean>

</blueprint>
```

JMX Integration

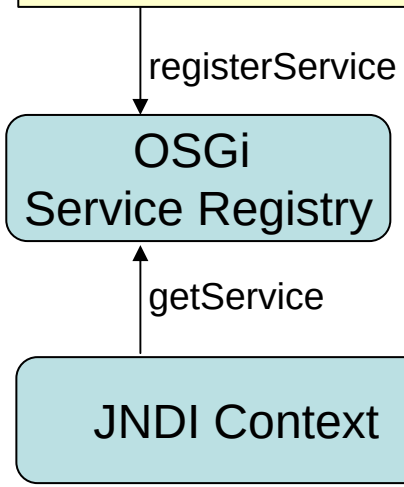
- Implementation of OSGi JMX specification.
- Aries JMX bundle automatically registers the JMX MBeans into any javax.management.MBeanServer service in the OSGi Service Registry.



JNDI integration

- Provides JNDI-based access to OSGi Service Registry

```
<blueprint xmlns=...>
   <bean id="bloggingServiceComponent"
       class="org.apache.aries.BloggingServiceImpl">
   </bean>
   <service ref="bloggingServiceComponent"
      interface="org.apache.aries.samples.blog.api.BloggingService"/>
   ...
</blueprint>
```



- A way for a Web component to access a Blueprint component

```
InitialContext ic = new InitialContext();
BloggingService blog= ic.lookup("osgi:services/"
                + BloggingService.class.getName());
```

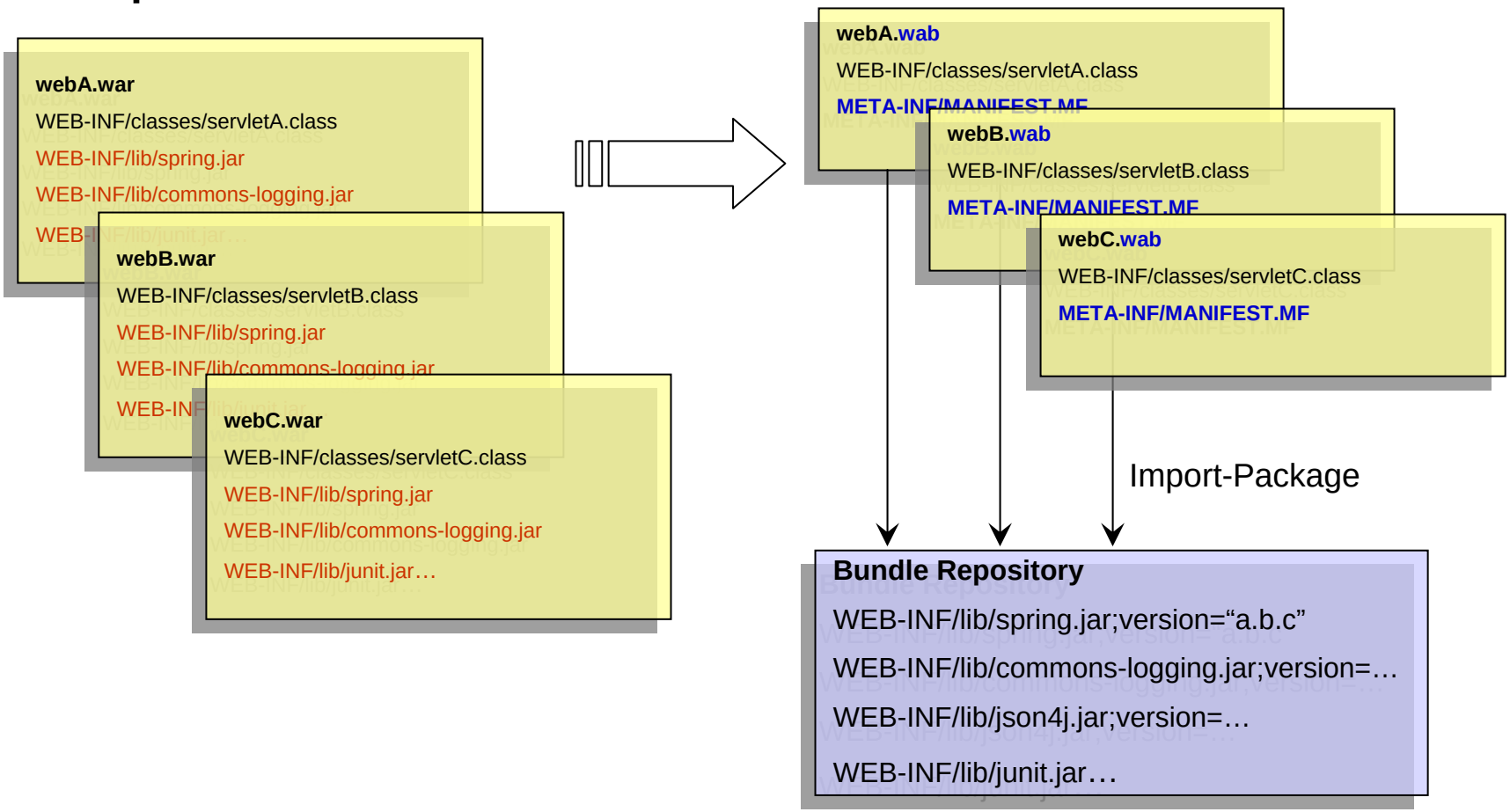
IBM WebSphere Application Server & OSGi

- OSGi used internally in WAS since 2005 (WAS 6.1 shipped in 2006)
- Application-level exploitation of OSGi introduced in Nov 2009…
- WAS V7 Feature Pack for OSGi Applications (Beta) https://www14.software.ibm.com/iwm/web/cc/earlyprograms/websphere/wasfposgiajp/support.shtml
  - Freely available – installs on top of WAS v7.0.0.7
  - Integrates Apache Aries technologies with the WAS runtime and administration.
  - Supports deployment of web applications as isolated applications consisting of one or more OSGi bundles

Module Re-Use Made Easy

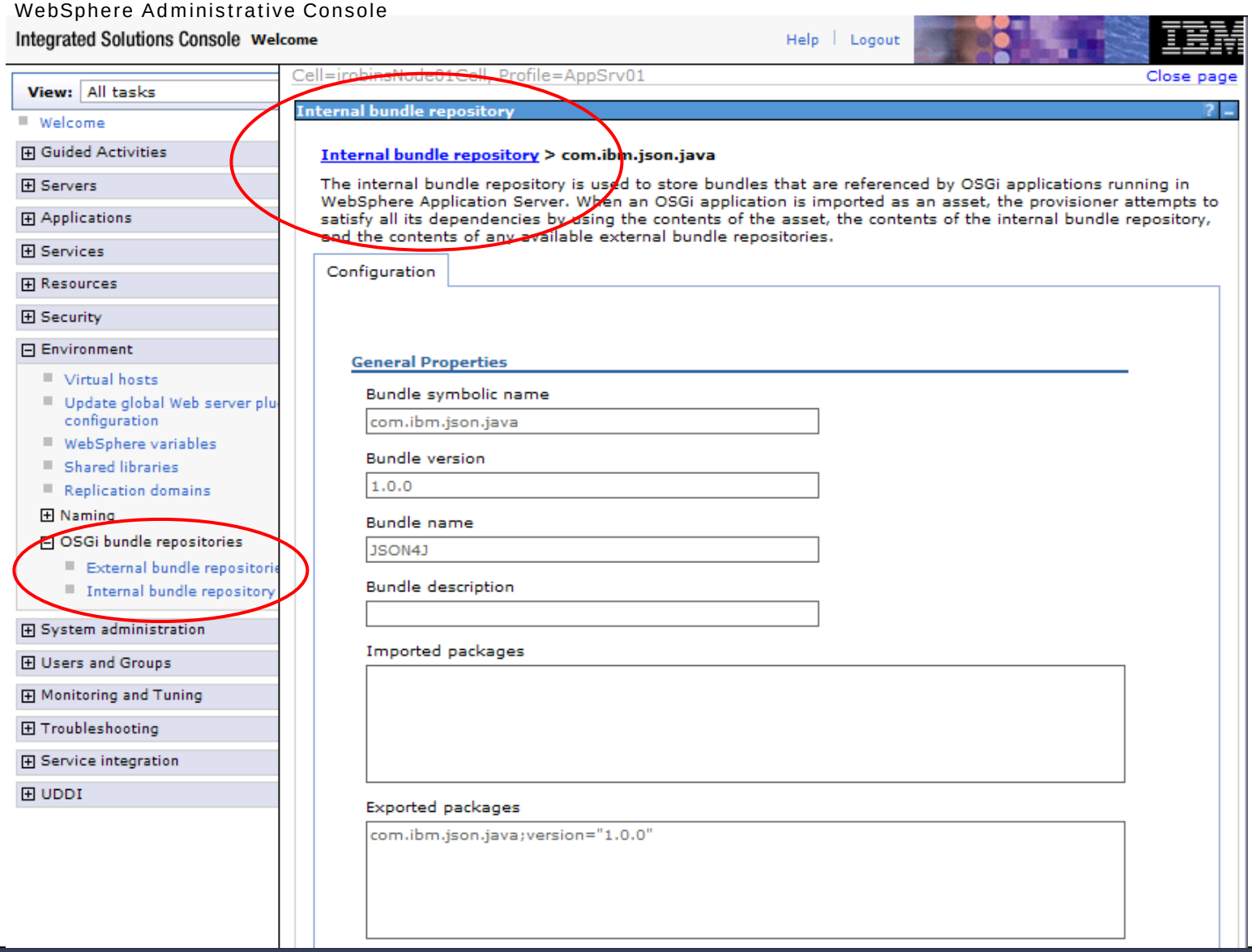
No Java code changes; war modules -> bundles

Common, bundles may be easily factored out of the WARs and used at specific versions



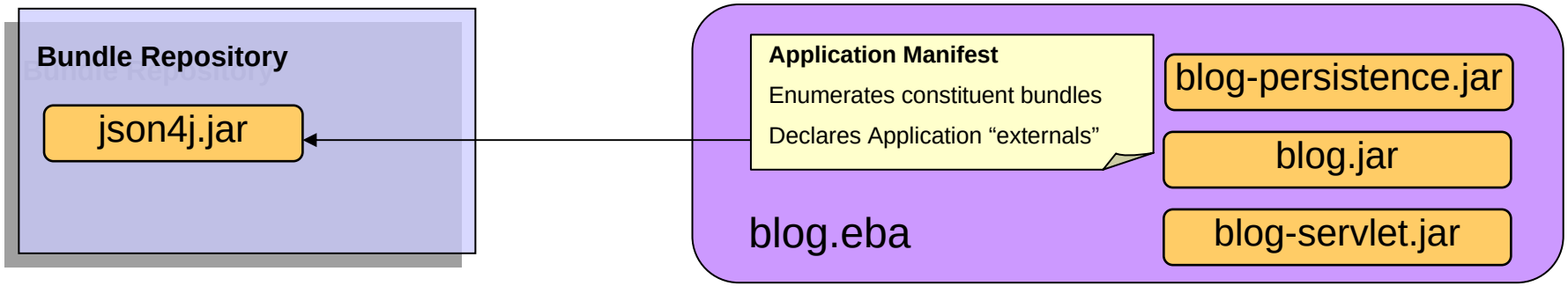
App Deploy Using OSGi Bundle Repositories

WebSphere Administrative Console



“Enterprise Bundle Archive” (EBA)

- An isolated, cohesive application consisting of a collection of bundles, is deployed as a logical unit in a “.eba” archive
  - An “OSGi Application”.
- Constituent bundles may be contained (“by-value”) or referenced from a bundle repository.
- Services provided by the application are isolated to the application unless explicitly exposed through EBA-level application manifest
- Config by exception - absence of APPLICATION.MF means:
  - application content is the set of bundles contained by-value plus any repository-hosted dependencies identified during deployment.



Resolving and “freezing” the deployment

Install (via Admin Console / wsadmin)

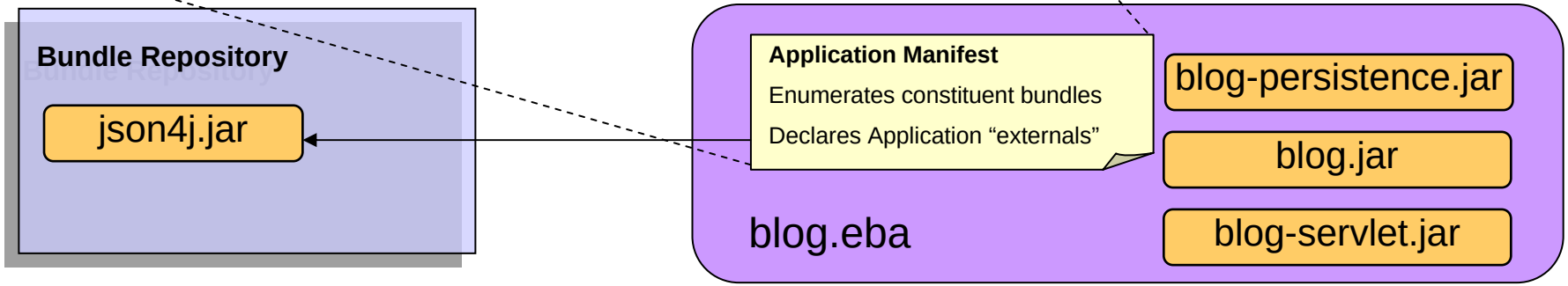
Application Manifest (developer/assembler **authored** artefact)
Enumerates constituent bundles and allowable version ranges
Declares Application “externals”

```
Manifest-Version: 1.0
Application-ManifestVersion: 1.0
Application-Name: Blog Application
Application-SymbolicName: aries.sample.blog
Application-Version: 1.0
Application-Content:
  aries.sample.blog; version="[1.0.0,1.1.0)",
  aries.sample.blog-api; version=1.0.0,
  aries.sample.blog-persistence; version=1.0.0,
  aries.sample.blog-servlet; version="[1.0.0,1.0.0]"
```

Deployed Result

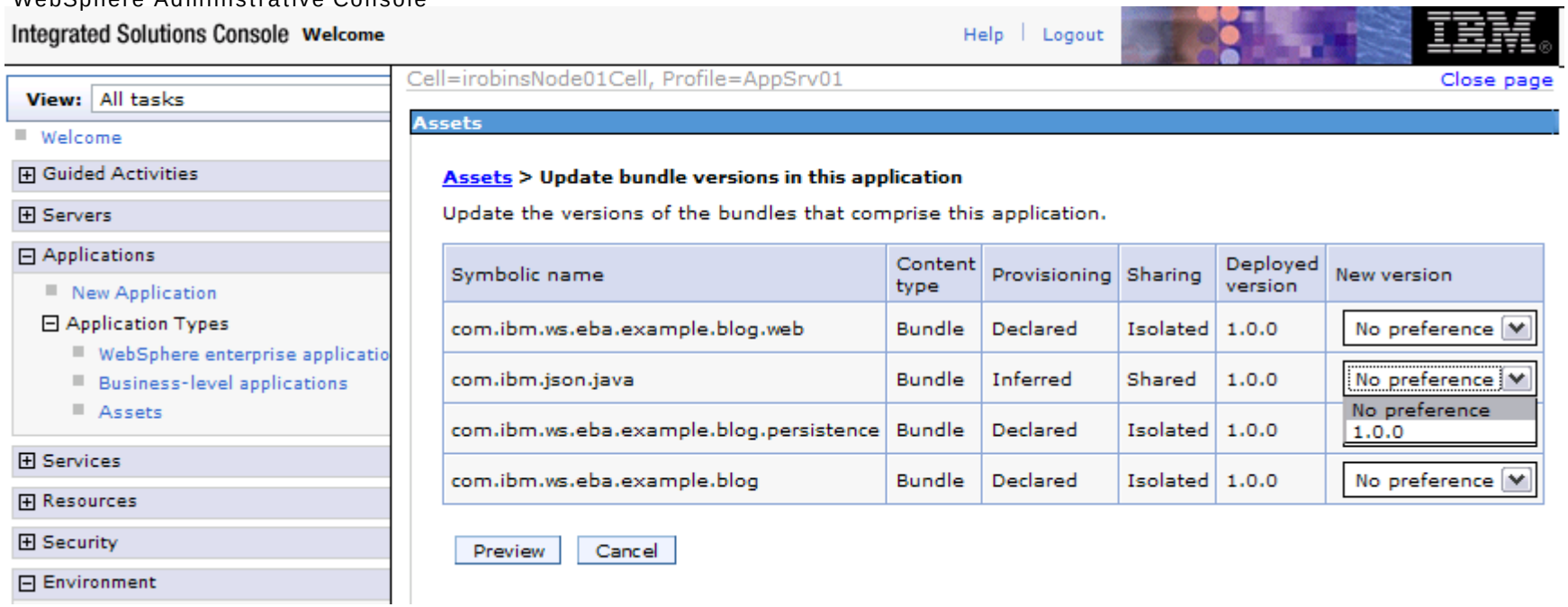
Deployment Manifest (**generated** during *createApplication*)
Transitively closed description of all bundles resolved at specific versions to “freeze-dry” the application.

```
Manifest-Version: 1.0
Deployment-ManifestVersion: 1.0
Application-Name: Blog Application
Application-SymbolicName: aries.sample.blog
Application-Version: 1.0
Deployed-Content:
  aries.sample.blog; deployed-version=1.0.0,
  aries.sample.blog-api; deployed-version=1.0.0,
  aries.sample.blog-persistence; deployed-version=1.0.0,
  aries.sample.blog-servlet; deployed-version=1.0.0,
  com.ibm.json.java; deployed-version=1.0.0
```

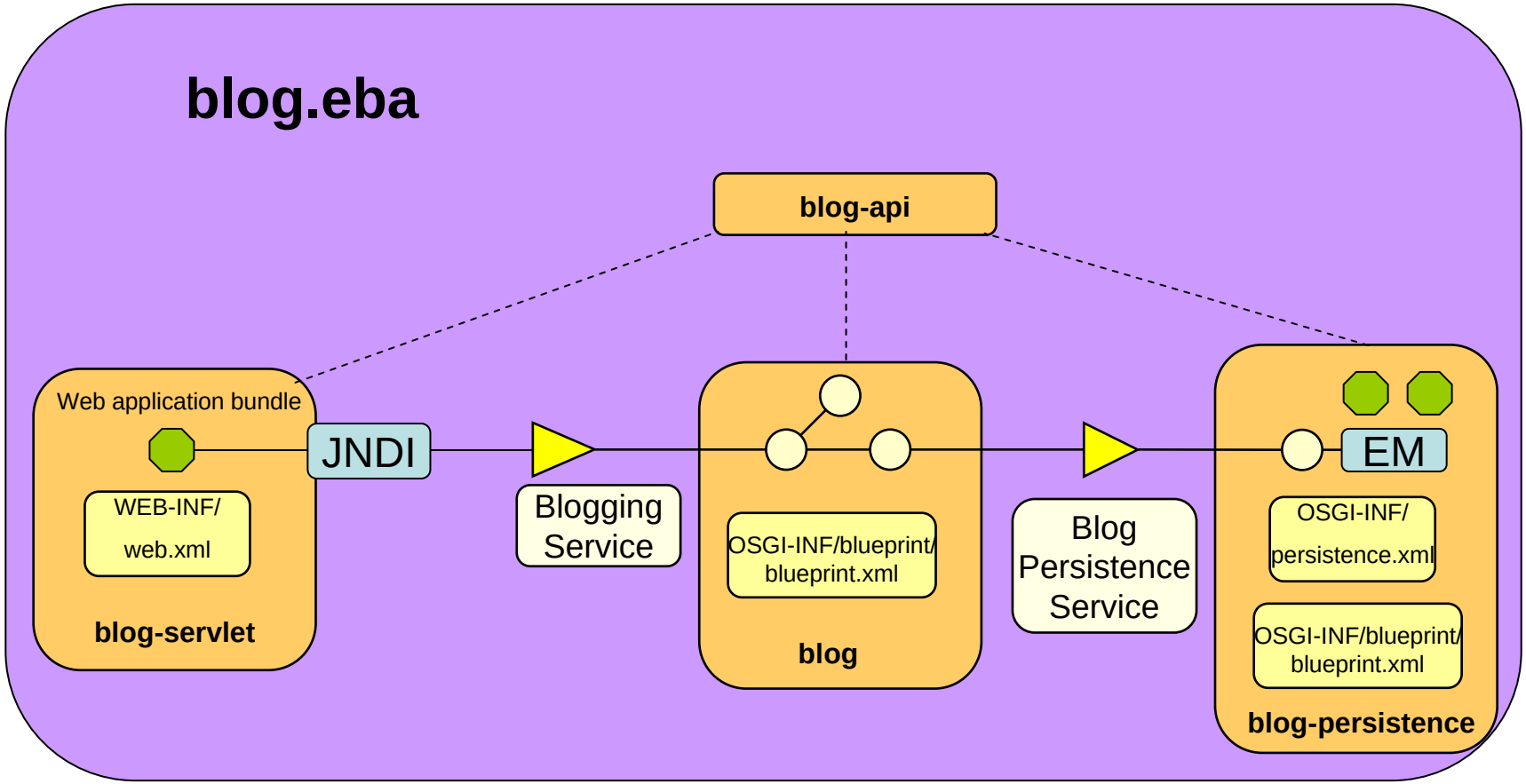


Bundle-level Management

WebSphere Administrative Console



Example “Blog” Application Architecture




In the near future…
all major Java
enterprise runtimes
will support Apps
deployed as bundles

For further information visit:
ibm.com/WebSphere/dev