Introduction to OSGi and Modularity

InfoSphere MDM, Version 11.x

Dany Drouin, Senior Software Engineer MDM

Please Note:

IBM’s statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM’s sole discretion.

Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.

The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract. The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

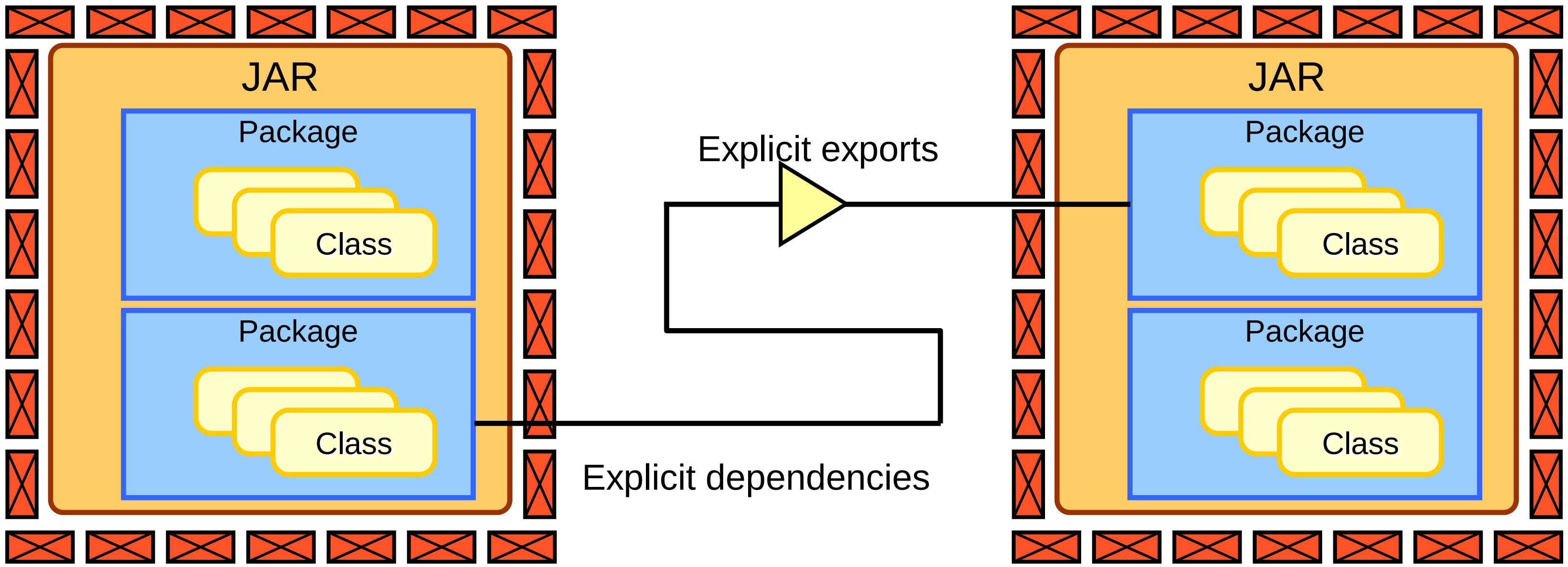
Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user’s job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

Agenda

- Part 1: Introduction to OSGi
- Part 2: MDM and OSGi
- Part 3: Customizing and extending MDM using OSGi
- Part 4: Demo
- Summary

What is OSGi and what is Modularity?

- OSGi stands for "Open Services Gateway initiative". *Which doesn't have much to do with what it's about.*
- "The dynamic module system for Java"
  - Mature 10-year old technology
  - Governed by OSGi Alliance: http://www.osgi.org
  - Used *inside* just about *all* Java-based middleware
    - IBM WebSphere, Oracle WebLogic, Red Hat JBoss, Sun GlassFish, Paremus Service Fabric, Eclipse Platform, Apache Geronimo, (non-exhaustive list) http://www.osgi.org/wiki/uploads/News/2008_09_16_worldwide_market.pdf



Components of OSGi We’ll Discuss

- OSGi bundles
- OSGi Applications
- OSGi Composite Bundles
- OSGi Bundle Repositories (OBR)
- OSGi Services

OSGI building blocks

- Bundle
  - A jar containing, classes,d resources and manifest.
  - Manifest specifies Bundle-SymbolicName, Bundle-Version , Import-Package and Export-Package
- Enterprise Bundle Archive (EBA)
  - EBA to OSGI is what an EAR is to JEE.
  - EBA's APPLICATION.MF specifies all its assembled OSGI bundles and their versions
  - Application manifest specifies values for Application-ManifestVersion, Manifest-Version, Application-Name, Application-SymbolicName, Application-Version, Application-Content, etc.
- Composite Bundle Archive (CBA)
  - Has the characteristics of an EBA (as it contains bundles) and a Bundle (as it provides import and export the packages/services it needs)
  - Composite Bundle Archive (CBA) is used to attach customized logic to an EBA
- Bundle Repository
  - Allows putting sharable OSGi bundles amongst applications
- OSGI Blueprint
  - Blueprint is an XML based definition that describes OSGi service(s) or OSGI service listener(s).
  - Each bundle may have one or more blueprint files

OSGi Bundles

```
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:
com.something.i.need;version=1.1.2
Export-Package: com.myservice.api;version=1.0.0
```

- OSGi Bundle – A JAR containing:
  - Classes and resources.
  - OSGi Bundle manifest.
- What's in the manifest:
  - Bundle-SymbolicName: this is the name that matters.
    - It's the name the OSGi container uses to refer to the bundle.
    - It's the name to which the version number (Bundle-Version, below) is attached.
  - Bundle-Version: Multiple versions of bundles can live concurrently.
  - Import-Package: What packages from other bundles does this bundle depend upon?
  - Export-Package: What packages from this bundle are visible and usable outside of the bundle?

Enterprise Bundle Archives (EBAs)

- EBAs represent OSGi Applications. An EBA is to OSGi what an EAR is to JEE.
- Bundles are assembled into Enterprise Bundle Archives.
- EBAs isolate the bundles in them from bundles outside the EBA
- Bundles are specified in the APPLICATION.MF of the EBA.
- Bundles can be included in the EBA itself or not. If they are not they must be provisioned from a repository.

```
Application-ManifestVersion: 1.0
Manifest-Version: 1.0
Application-Name: InfoSphere Master Data Management
Application-SymbolicName: com.ibm.mdm.hub.server
Application-Version: 11.0.0
Manifest-Version: 1.0
Application-Content:
com.ibm.mdm.server.commonutils;version=11.0.0,
 com.ibm.mdm.server.referencemodels;version=1.0.0,
 com.ibm.mdm.server.adminservices;version=11.0.0,
 com.ibm.mdm.server.batch.management;version=11.0.0,
 com.ibm.mdm.server.bizservices;version=11.0.0,
 com.ibm.mdm.server.btm;version=11.0.0,
 com.ibm.mdm.server.codetypes;version=11.0.0,
 com.ibm.mdm.server.config.client;version=11.0.0,
 com.ibm.mdm.server.config.repo;version=11.0.0,
 com.ibm.mdm.server.coreutilities;version=11.0.0,
 com.ibm.mdm.server.dwlbizservices;version=11.0.0,
 com.ibm.mdm.server.dwlcommonservices;version=11.0.0,
 com.ibm.mdm.server.extrules.common;version=11.0.0,
 com.ibm.mdm.server.linguistics;version=11.0.0,
 com.ibm.mdm.server.logging;version=11.0.0,
 com.ibm.mdm.server.metadata.common;version=11.0.0
...
```

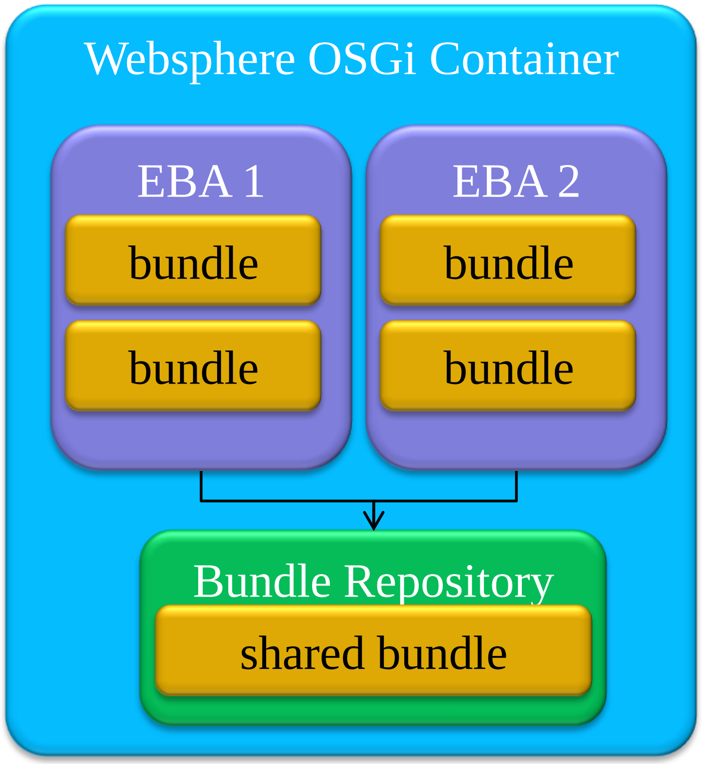
Composite Bundle

- Composite Bundle has the characteristics of an EBA and a Bundle.
- It's like an EBA because
  - It contains bundles
  - It provides a level of isolation to the bundles
  - It exports the services it provides
- It's also a bundle because
  - It imports and exports the packages it needs and provides
- We use composite bundles to attach customizations by clients

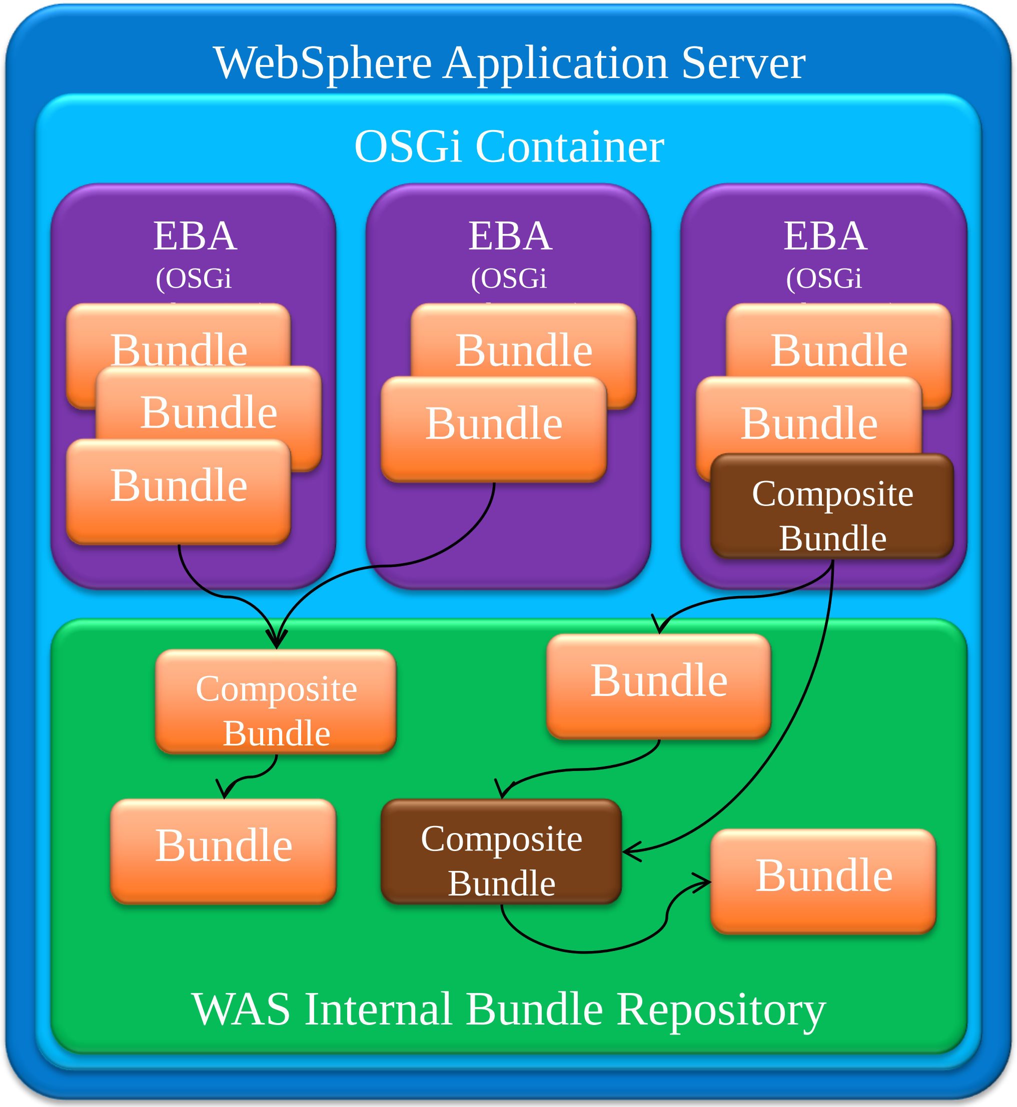
```
Bundle-Name: com.ibm.mdm.server.compositeBundle
Bundle-SymbolicName:
com.ibm.mdm.server.compositeBundle
Bundle-Version: 1.0.0
CompositeBundle-ManifestVersion: 1
CompositeBundle-Content:
com.ibm.mdm.server.bundle1,
 Com.ibm.mdm.server.bundle2
Manifest-Version: 1.0
Import-Package: com.dwl.base,
 Com.dwl.base.xml
Export-Package: com.ibm.server.extensions,
 com.ibm.mdm.server.extensions2
CompositeBundle-ExportService:
com.ibm.mdm.z.api.MyInterface
```

OSGi Bundle Repositories - OBR

- Bundle repositories are places to put OSGi bundles that are shared amongst applications.
- Bundle repositories can be used to share third party jars amongst application.
- Bundle repositories are used in WAS during runtime to provision shared bundles amongst applications.

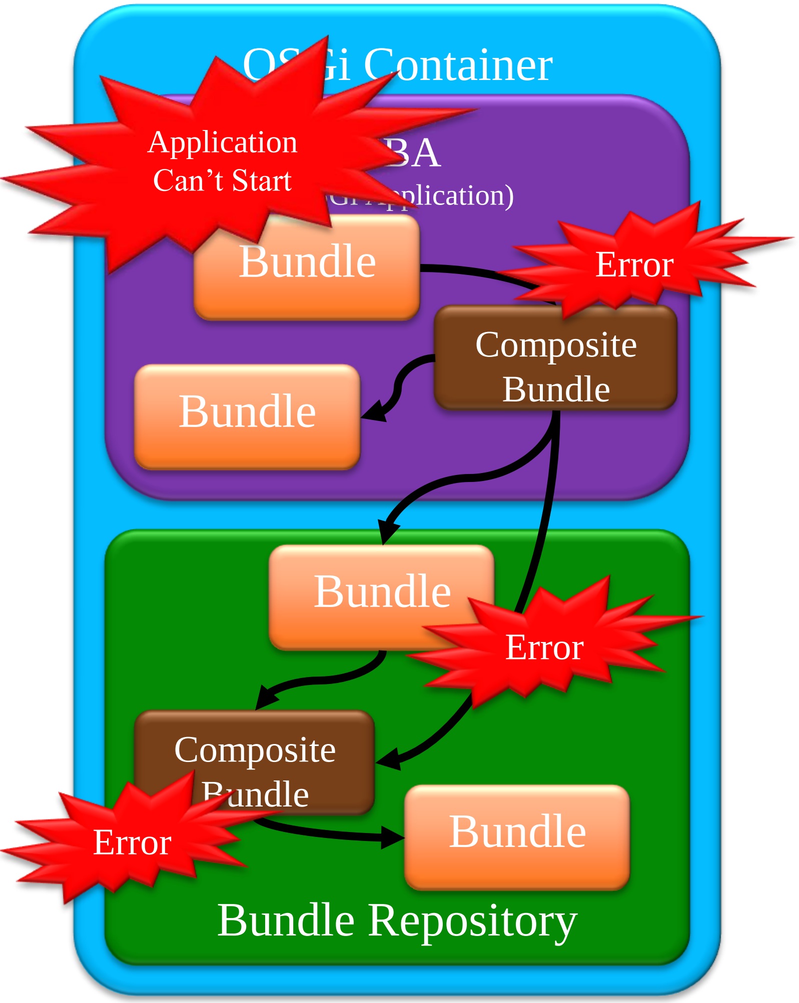


How it all Fits together



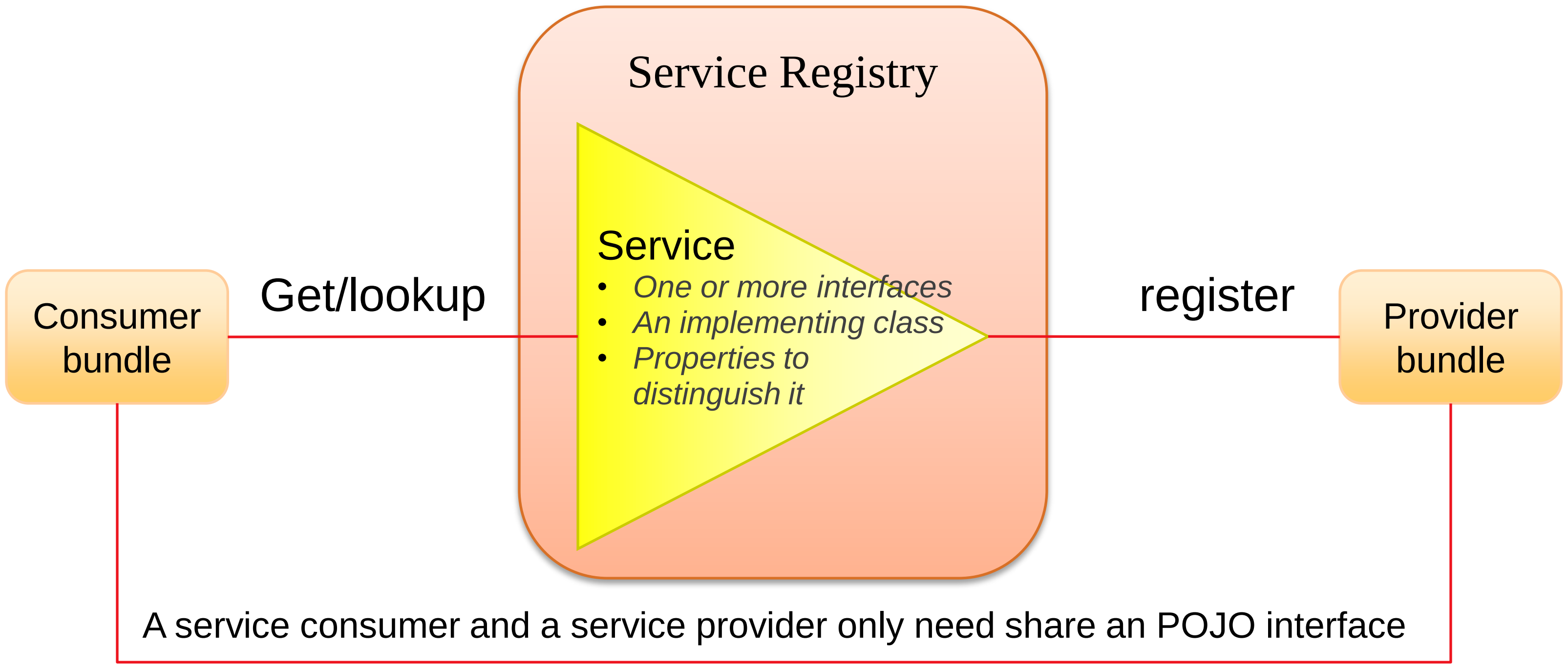
- Bundles can exist in several places in an OSGi container. The two we're concerned with are:
    - Within an OSGi Application.
    - In WAS's internal bundle repository.
        - You can load composite bundles or bundles into a WAS bundle repository.
        - These bundles exist in a repository ready to be used by any application that needs them.

Provisioning



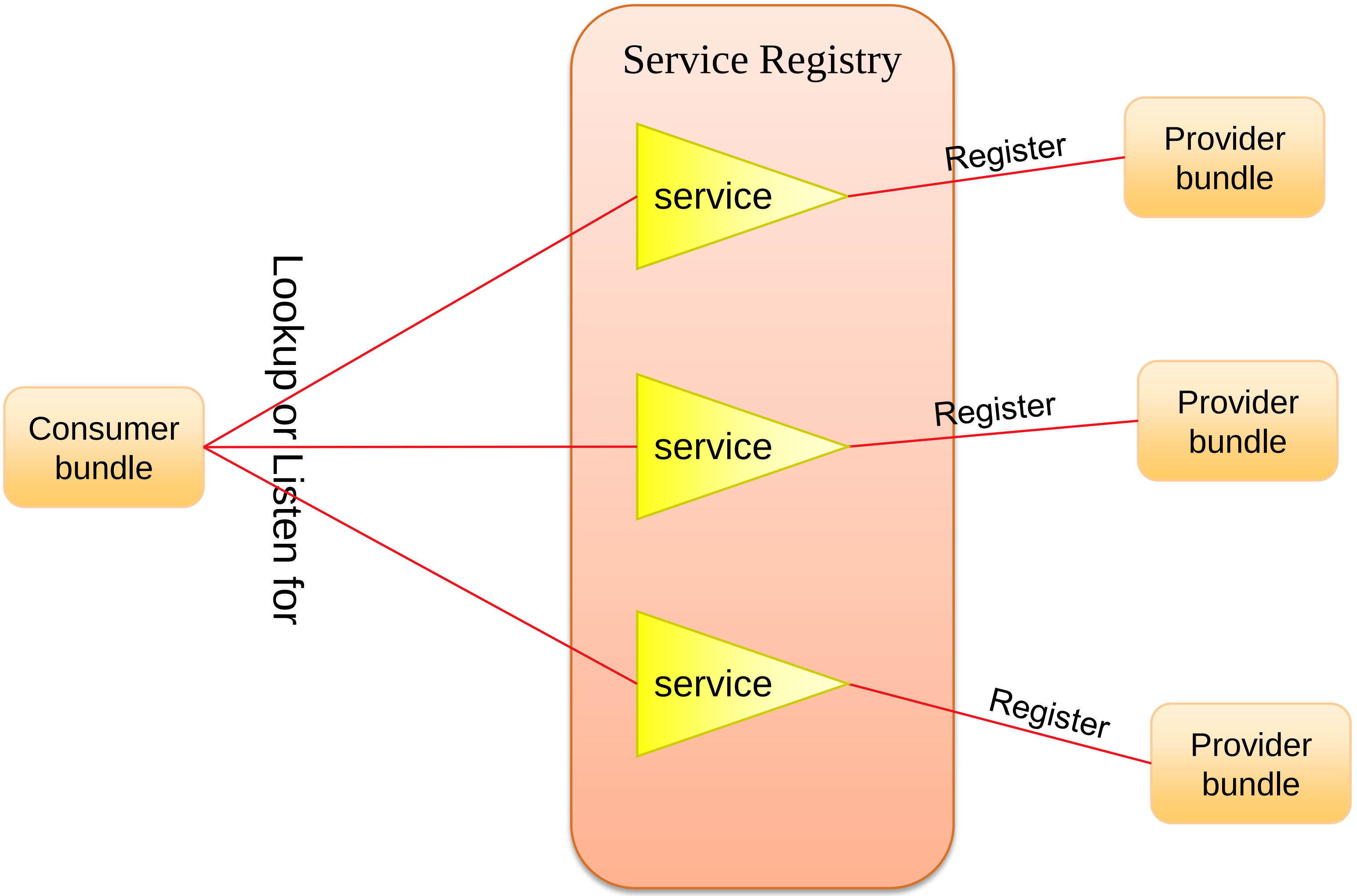
- When an application is deployed/installed or started
- The OSGi container provisions the application by examining each bundle it contains
- It looks at which packages a bundle imports and goes off and finds the bundles that provide those packages.
- It will look inside the EBA first.
- It will then look in the bundle repository.
- Bundles in the bundle repository may themselves have dependencies on other bundles in the bundle repository.
- If one of the necessary bundles isn't there, the application can't be provisioned and won't be able to start.

OSGi Services



- Publish/find/bind service model
  - Fully dynamic
  - Local
  - Non-durable
- A service is a POJO* advertised with properties and/or interface and/or class
- Primary mechanism for bundle collaboration

Services are Dynamic



Defining Services using Blueprint

- All the services us use in MDM are defined using blueprint.
- Blueprint is an XML based languages that describes OSGi services.
- Each bundle in MDM has one or more blueprint files, each file defining one or more services or service listeners.
- The following pages illustrate several of the ways that we use blueprint to define our services.
- Any feature or capability in MDM where you can plug in a new implementation or where you can extend MDM with new entities, is defined as a services.
- Recall that services are the means by which we can provide loose couplings between bundles that consume features and bundles that provide them. Services are the natural means by which we can enable the addition of new capabilities.

Blueprint Examples: Defining Services

```
<service id="BusinessProxy.getComparativeMultipleParties"
  interface="com.ibm.mdm.common.servicefactory.api.BusinessProxyFactory">
  <service-properties>
    <entry key="transaction.name" value="getComparativeMultipleParties" />
  </service-properties>
  <bean class="com.ibm.mdm.common.servicefactory.BusinessProxyFactoryImpl" >
    <argument type="java.lang.Class"
              value="com.dwl.tcrm.coreParty.bp.TCRMGetComparativeMultiplePartiesBP" />
    <argument ref="blueprintBundle"/>
  </bean>
</service>
```

The ID of the service can be any name In this case it describes the service as being for a business proxy and the transaction name it supports

A service can be defined by one or more interfaces. In this case the interface is for a business proxy factory. Our business proxies are not services themselves but the factories that create them are. So any bundle containing a business proxy, must also define a service that creates the business proxy.

This service has a service property that distinguishes it from other business proxy factories. The service property is called “transaction.name” whose value is the name the transaction this particular business proxy supports. If this proxy supported more than o transaction, there would be an etry for each transaction name.

This is the name of the factory class. It implements the factory interface.

Blueprint has the ability to define arguments to services. In this the argument is the class of the actual business proxy. This is the BP that the factory will produce when called upon. This is a type of injection. The blueprint container will inject into the factory the classes it is to create.

Blueprint Example: Defining Service Listeners

```
<reference-list availability="optional" id="BusinessProxyBroker"
  interface="com.ibm.mdm.common.servicefactory.api.BusinessProxyFactory"
  member-type="service-object">
  <reference-listene  bind-method="registerBusinessProxy"
    unbind-method="u  egisterBusines sProxy">
    <bean class="com   om.mdm.common  bro kers.BusinessProxyBroker"
          factory-me   d="getBroke  nst  ce"/>
    </reference-list    r>
</reference-list>
```

This listener lists for the registration of any service having this interface. It the same interface as in the previous slide where the service is defined.

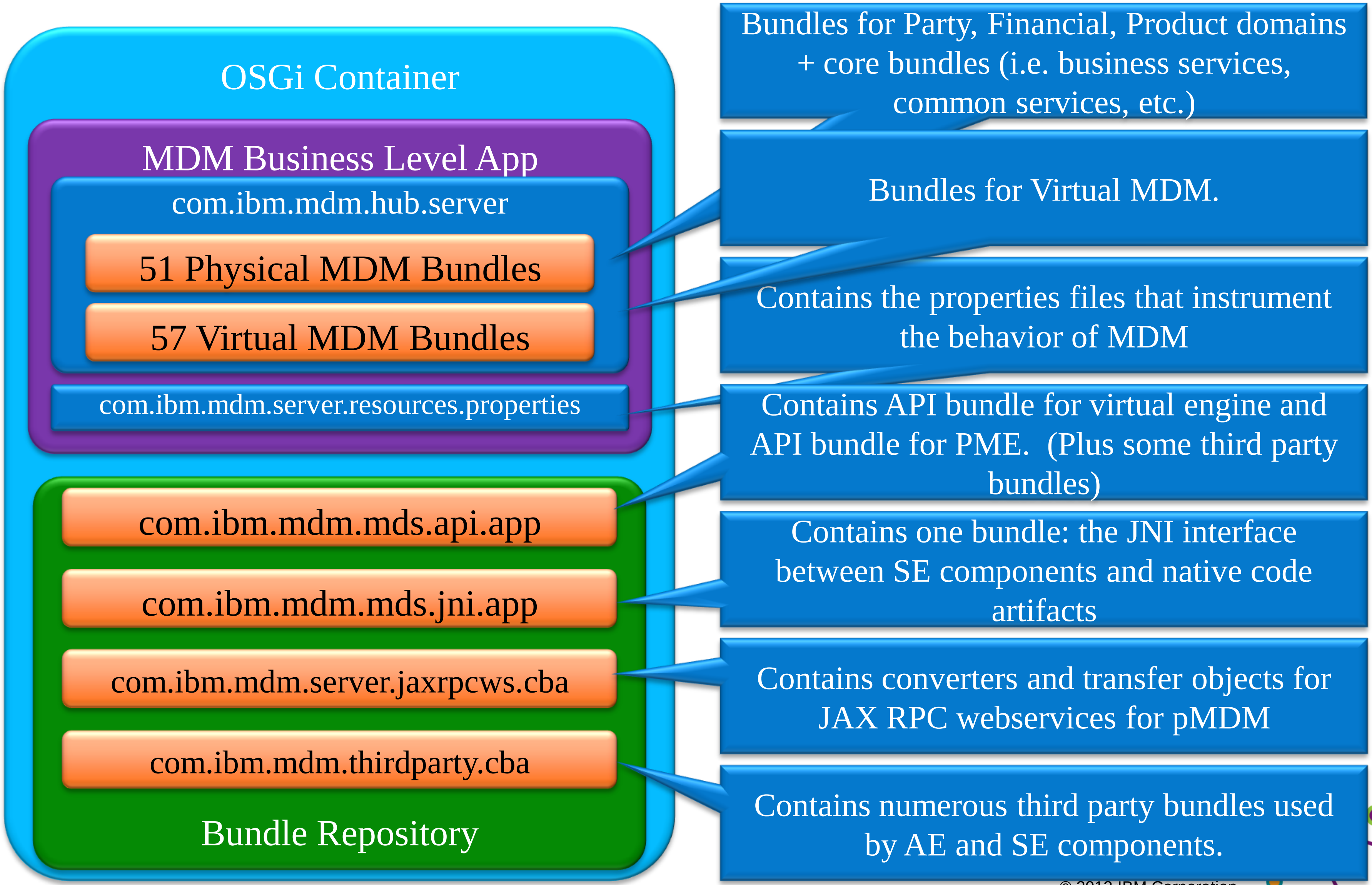
The unbind-method ins the blueprint container which metho cute whenever a service having this inte unregistered.

The bind method, instructs the blueprint container which method to execute each time a service having this interface is registered, from any bundle. This is how bundles that consume this service can be made aware of its existence.

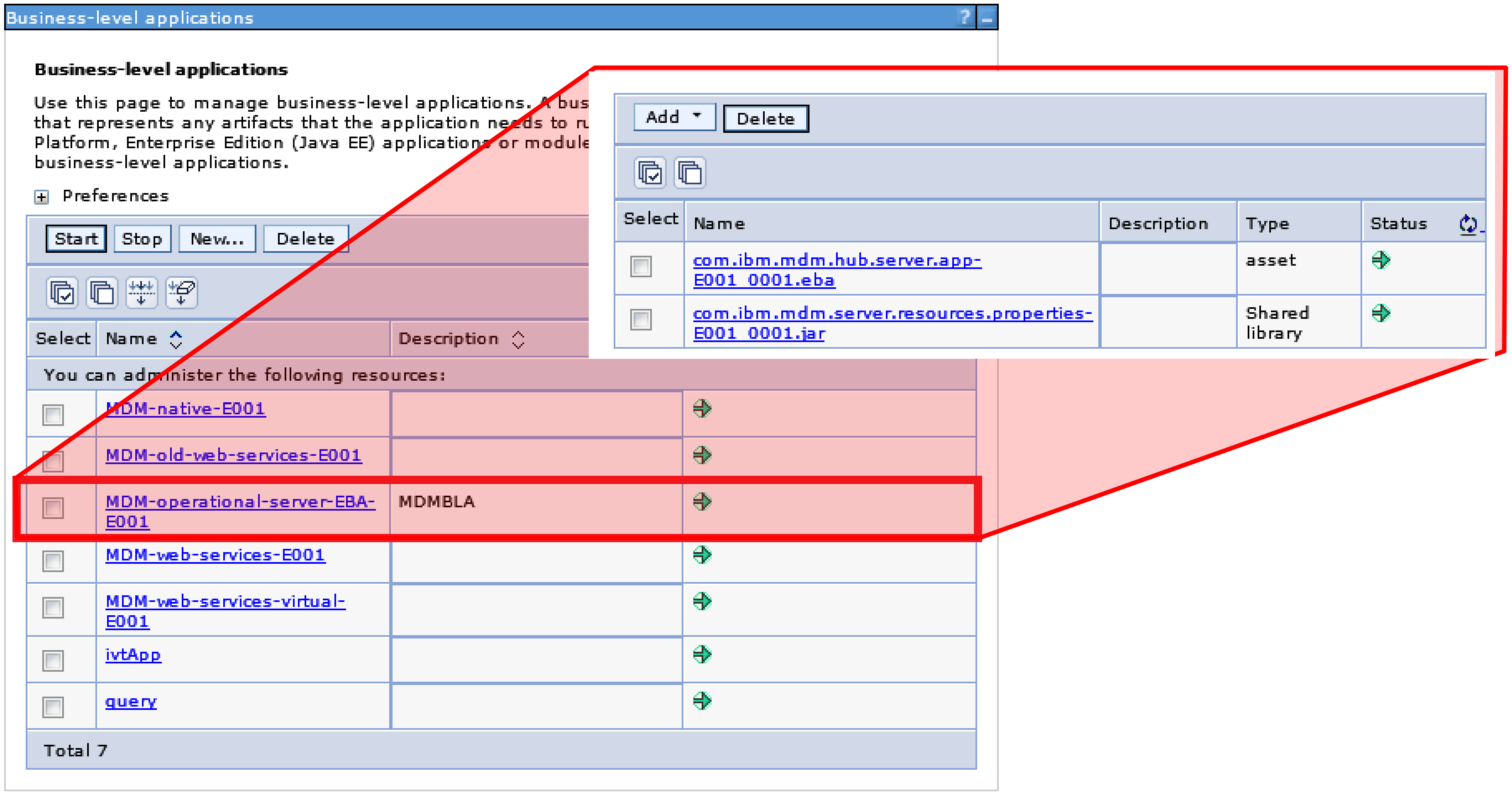
This is the class that the blueprint container will call upon to bind or unbind each time a service having this interface is registered. We call such classes brokers because they act as intermediaries between the objects that use business proxies and the BPs themselves. They keep track of which BPs are active and where they are so the can be used when called upon.

How MDM makes use of OSGi

What makes up MDM Server Standard Edition and Advanced Edition - OSGi


OSGi Container
MDM Business Level App
com.ibm.mdm.hub.server
51 Physical MDM Bundles
57 Virtual MDM Bundles
com.ibm.mdm.server.resources.properties
com.ibm.mdm.mds.api.app
com.ibm.mdm.mds.jni.app
com.ibm.mdm.server.jaxrpcws.cba
com.ibm.mdm.thirdparty.cba
Bundle Repository
Bundles for Party, Financial, Product domains + core bundles (i.e. business services, common services, etc.)
Bundles for Virtual MDM.
Contains the properties files that instrument the behavior of MDM
Contains API bundle for virtual engine and API bundle for PME. (Plus some third party bundles)
Contains one bundle: the JNI interface between SE components and native code artifacts
Contains converters and transfer objects for JAX RPC webservices for pMDM
Contains numerous third party bundles used by AE and SE components.

What makes up MDM Server Standard Edition and Advanced Edition – OSGi EBA



Business-level applications

Business-level applications

Use this page to manage business-level applications. A bus
that represents any artifacts that the application needs to ru
Platform, Enterprise Edition (Java EE) applications or module
business-level applications.

⊞ Preferences

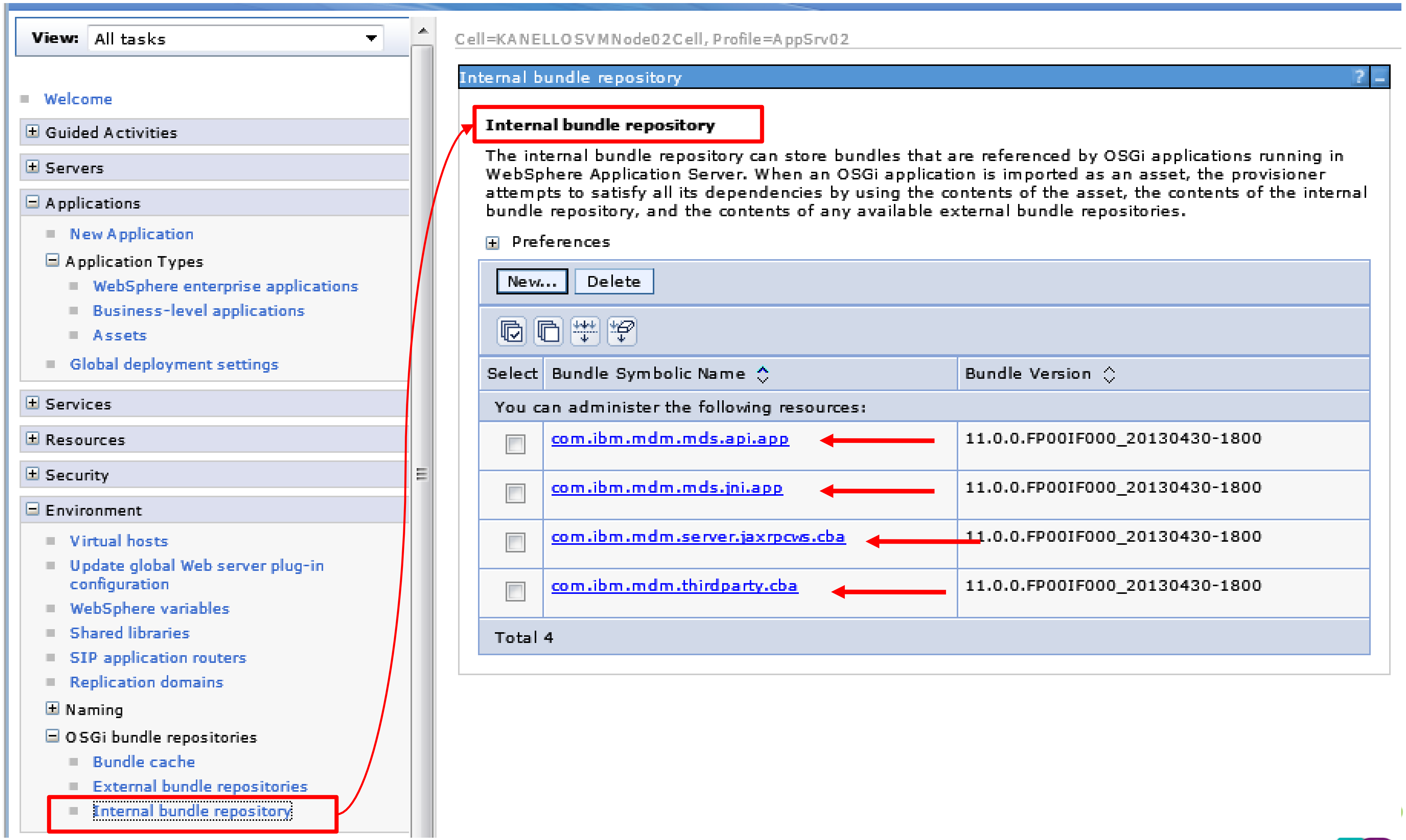
Start | Stop | New... | Delete

Select	Name	Description	
You can administer the following resources:			
☐	MDM-native-E001		➜
☐	MDM-old-web-services-E001		➜
☐	MDM-operational-server-EBA-E001	MDMBLA	➜
☐	MDM-web-services-E001		➜
☐	MDM-web-services-virtual-E001		➜
☐	ivtApp		➜
☐	query		➜
Total 7			

Add ▾ | Delete

Select	Name	Description	Type	Status
☐	com.ibm.mdm.hub.server.app-E001_0001.eba		asset	➜
☐	com.ibm.mdm.server.resources.properties-E001_0001.jar		Shared library	➜

MDM Server in the OSGi Bundle Repository



View: All tasks

- Welcome
- Guided Activities
- Servers
- Applications
  - New Application
  - Application Types
    - WebSphere enterprise applications
    - Business-level applications
    - Assets
  - Global deployment settings
- Services
- Resources
- Security
- Environment
  - Virtual hosts
  - Update global Web server plug-in configuration
  - WebSphere variables
  - Shared libraries
  - SIP application routers
  - Replication domains
  - Naming
  - OSGi bundle repositories
    - Bundle cache
    - External bundle repositories
    - Internal bundle repository

Cell=KANELLOSVMNode02Cell, Profile=AppSrv02

Internal bundle repository

Internal bundle repository

The internal bundle repository can store bundles that are referenced by OSGi applications running in WebSphere Application Server. When an OSGi application is imported as an asset, the provisioner attempts to satisfy all its dependencies by using the contents of the asset, the contents of the internal bundle repository, and the contents of any available external bundle repositories.

Preferences

New... Delete

Select	Bundle Symbolic Name	Bundle Version
You can administer the following resources:		
☐	com.ibm.mdm.mds.api.app	11.0.0.FP00IF000_20130430-1800
☐	com.ibm.mdm.mds.jni.app	11.0.0.FP00IF000_20130430-1800
☐	com.ibm.mdm.server.jaxrpcws.cba	11.0.0.FP00IF000_20130430-1800
☐	com.ibm.mdm.thirdparty.cba	11.0.0.FP00IF000_20130430-1800
Total 4		

Where we employ OSGi services in MDM

It’s pluggable and customizable, there’s a service definition for that:

- Parsers, Response Constructors, Business Proxies, Suspect Processors
- Business Objects, Components
- Controllers
- Rules, Behavior Extensions, External Validation, ASI transformation definitions

For a full set of service definitions and the before/after OSGi comparison:

See MDM OSGI Migration @

http://pic.dhe.ibm.com/infocenter/mdm/v11r0/topic/com.ibm.mdmhs.dev.platform.doc/concepts/c_Overview_OSGi_Migration.html

Sample OSGi Service

A Service is identified by its interface

```
<service id="Controller.AdditionSamplesTxn"
interface="com.ibm.mdm.commonentity.samples.interfaces.AdditionSamplesTxn">
        <service-properties>
            <entry key="osgi.jndi.service.name">
                <list>
                    <value>addNote</value>
                    <value>updateNote</value>
                    <value>addReminder</value>
                    <value>updateReminder</value>
                </list>
            </entry>
        </service-properties>
        <bean
class="com.ibm.mdm.commonentity.samples.controller.AdditionSamplesTxnBean"
/>
    </service>
```

A service can be decorated by service properties, so that you can distinguish different implementations of the same service.

A service has an implementing class that's a bean

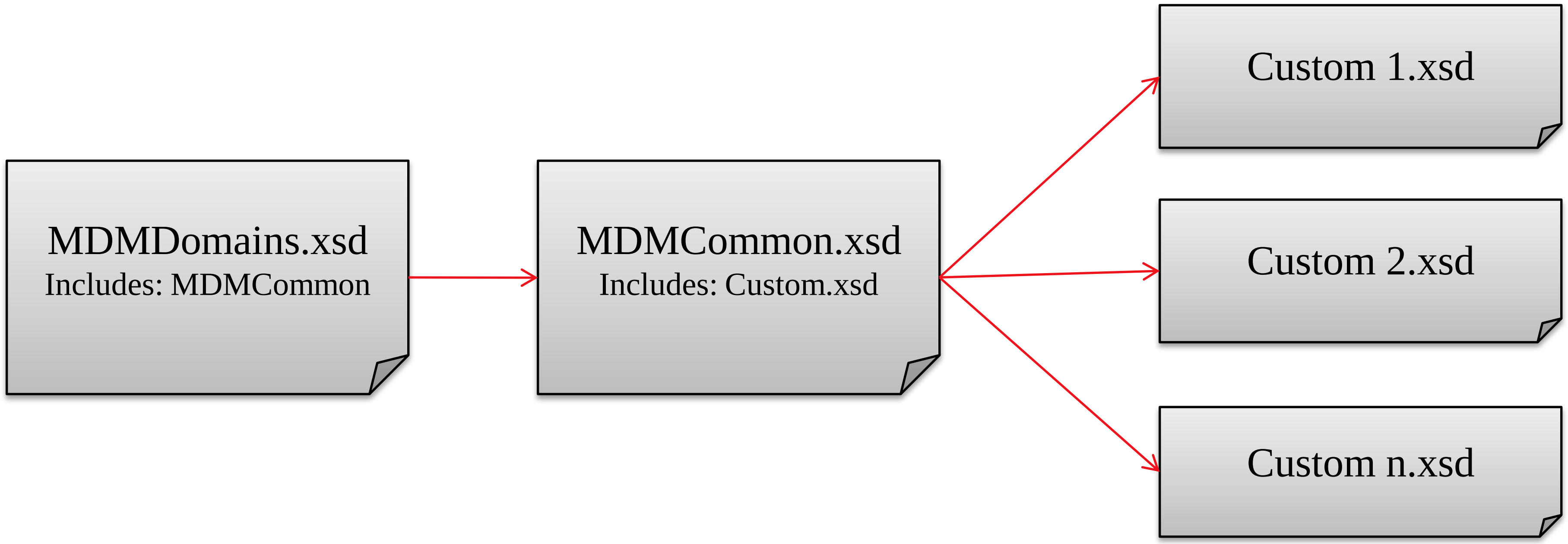
OSGi Services – lookup APIs

- Services can be looked up using JNDIservice lookup pattern:
  - `MyObject = Context.lookup("osgi:service/<service interface name>/<filter>");`
- Service locator
  - We've modified the ServiceLocator class so that it can get OSGi services for you:
    `MyObject = ServiceLocator.getOSGiService(String lookupPattern);`
- Using Brokers (found under *com.ibm.mdm.common.brokers* package)
  - BObjBroker : Used to return an instance of a BOBJ
  - BusinessProxyBroker : Used to return a BP instance
  - ComponentBroker : returns a MDM Component instance
  - ControllerBroker : returns a MDM Controller instance

  …

Schemas

- Customer Schemas are now complete schemas, not snippets
- They are self-contained fragments
- Custom schemas are pulled together at runtime.



Customizing and extending MDM

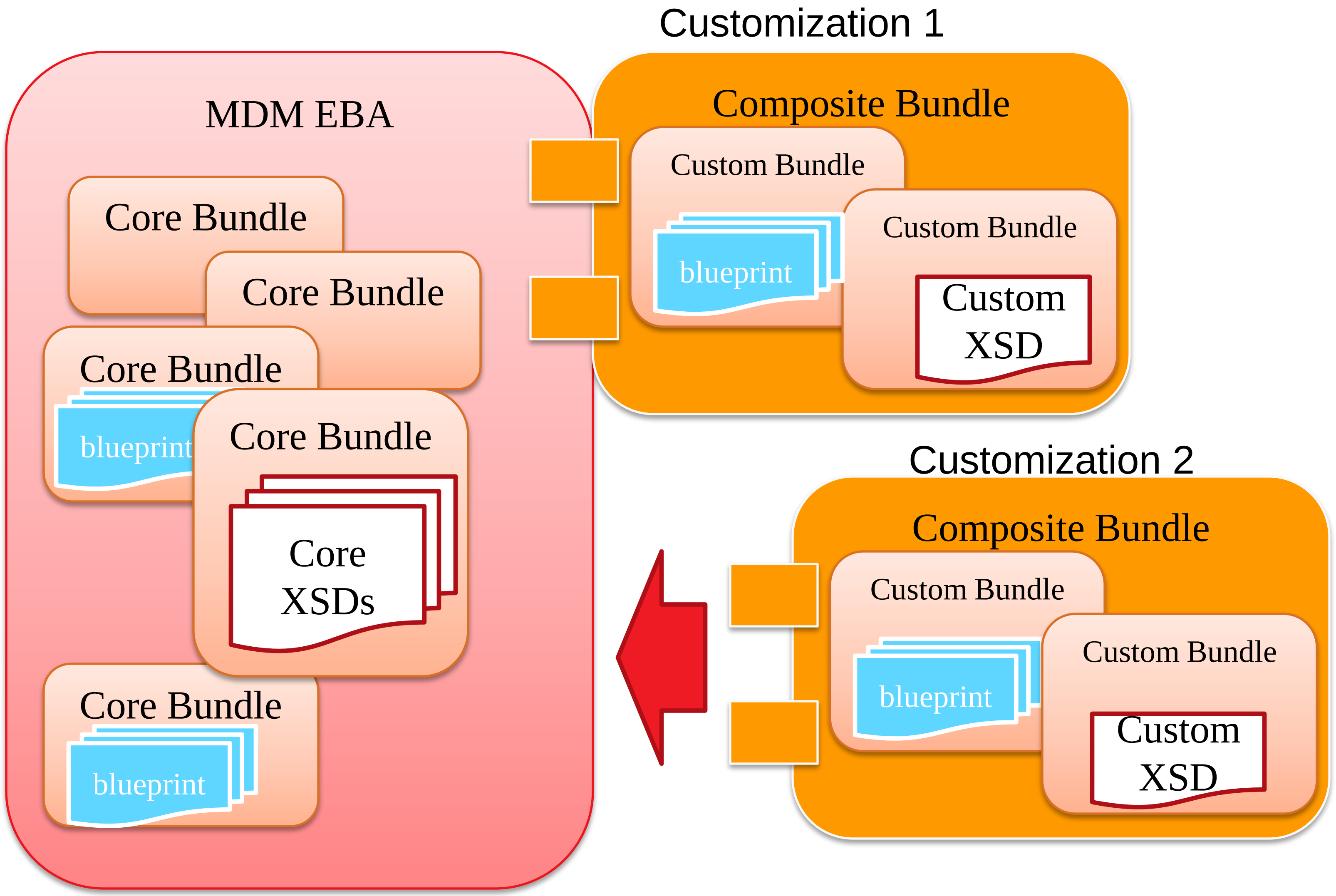
Pre-OSGi in MDM

- In the past, the method was to use the information to look up the implementation for a capability in a properties file (e.g. TCRM.properties, DWLCommon.properties, etc). Then java reflection (i.e. Class.forName) was used to create the object of the implementation. That worked when there was one class loader for the entire MDM application. It doesn't work in OSGi where the implementation can be in any bundle having its own class loader
- Also in the past, when you customized MDM, you incorporated your customization directly into MDM's own Enterprise Archive (EAR file). And you redeployed your own custom MDM EAR

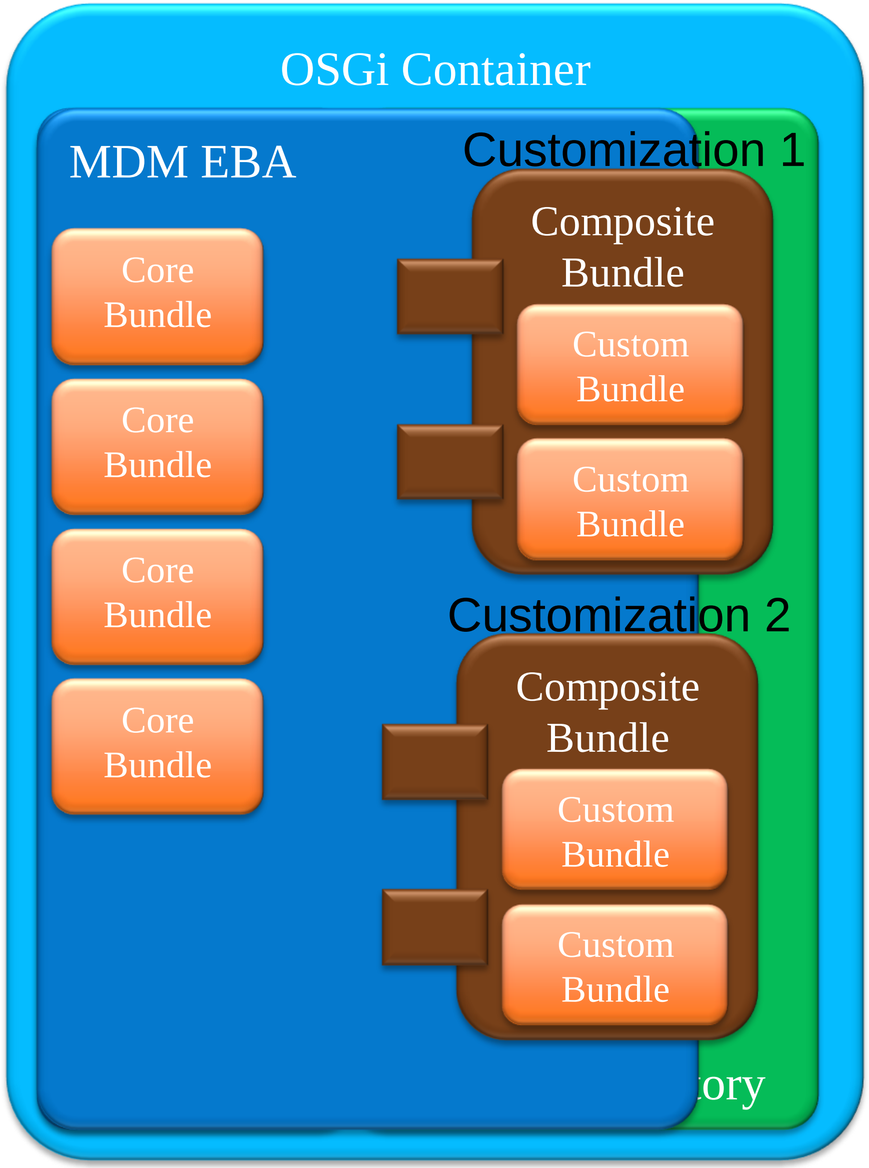
MDM OSGi

- MDM no longer is an enterprise archive file (EAR). With OSGi it is packaged in the form of an Enterprise Business Archive (EBA). With OSGi, no longer will you redeploy MDM with your customizations within it. In other words, you will never need to open MDM's EBA to include your customizations. Instead, you will generate your customizations in the form of OSGi bundles. You will package these bundles into Composite Bundle Archives (CBA) which are essentially bundles of bundles, and you will affix them to MDM as it is running.

Customizations using Composite Bundles

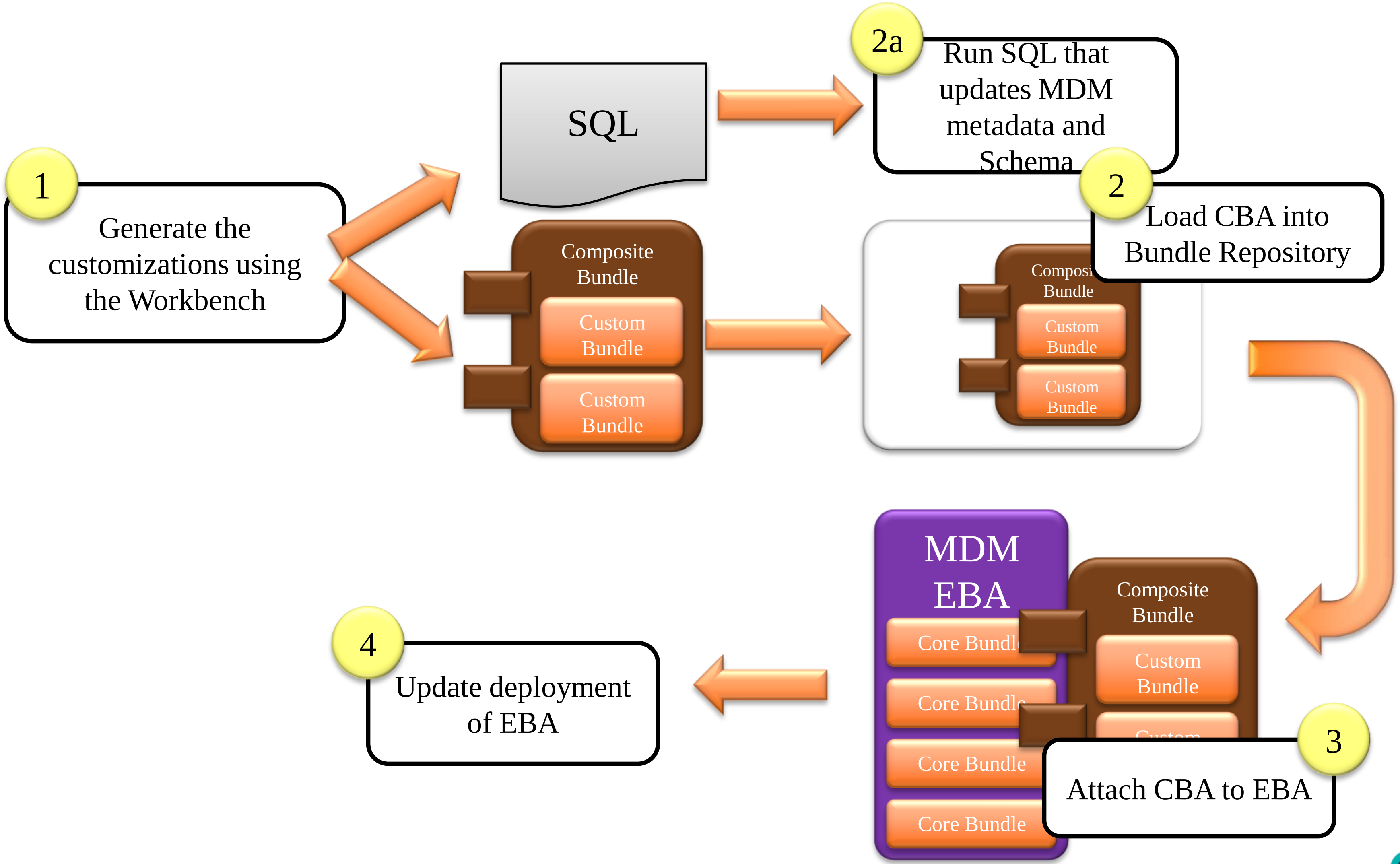


Extending Applications using Composite Bundles

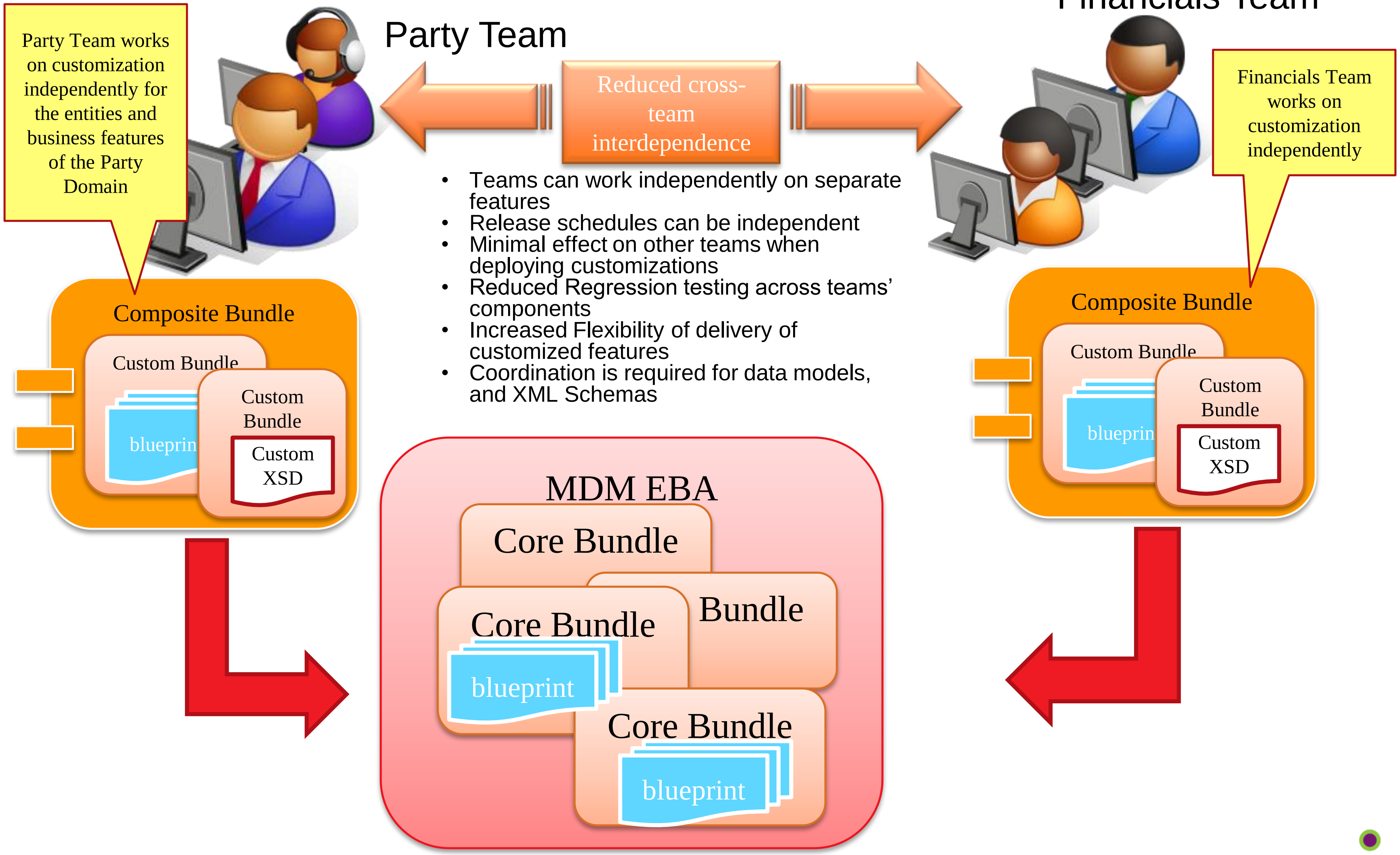


- You can customize MDM by adding features through Composite Bundles. These Features become part of the application itself.
- Here's how you can do it.
  - Load the CBAs (i.e. the composite bundles) into the WebSphere's OSGi Bundle Repository.
  - Bolt the CBAs – you can bolt on as many as you like – onto the EBA (the technical term is " composition unit extension").
- The EBA has now been extended to incorporate the Composite Bundles.

Steps to Extending MDM Server with Workbench Customizations



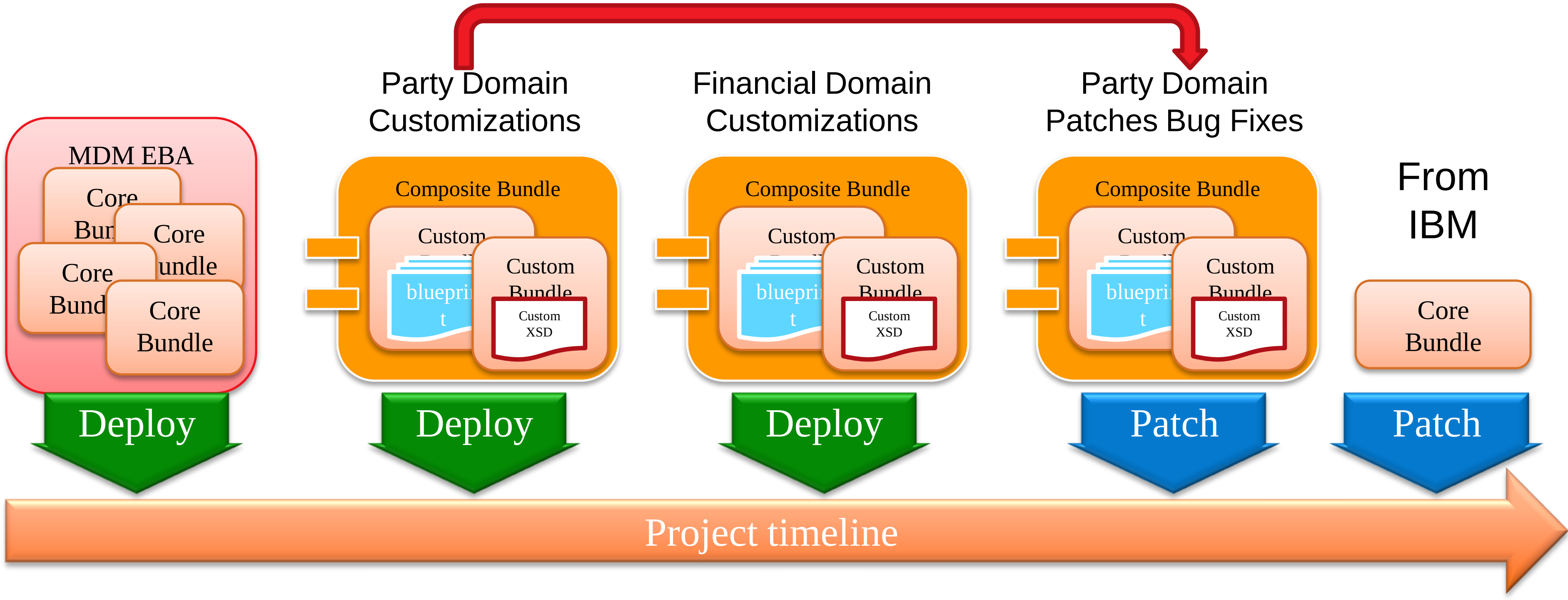
Project Management Benefits to OSGi



- Teams can work independently on separate features
- Release schedules can be independent
- Minimal effect on other teams when deploying customizations
- Reduced Regression testing across teams' components
- Increased Flexibility of delivery of customized features
- Coordination is required for data models, and XML Schemas

Hypothetical Scenario

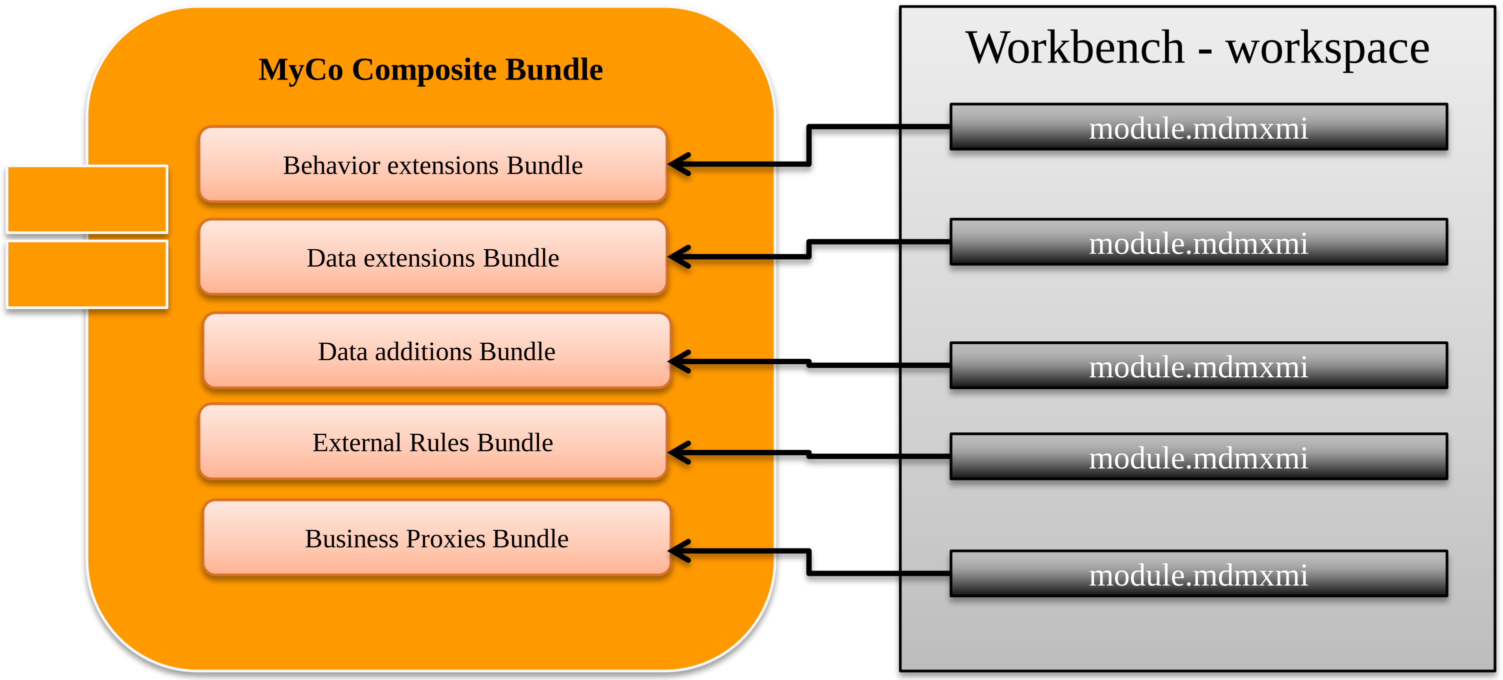
- Teams can deploy their customizations separately without affecting the customizations or the core modules installed beforehand.
- Patches only affect the patched modules, not the surrounding features.



Componentization / Modularization Scenario I

Multi-module – 1 CBA deployment

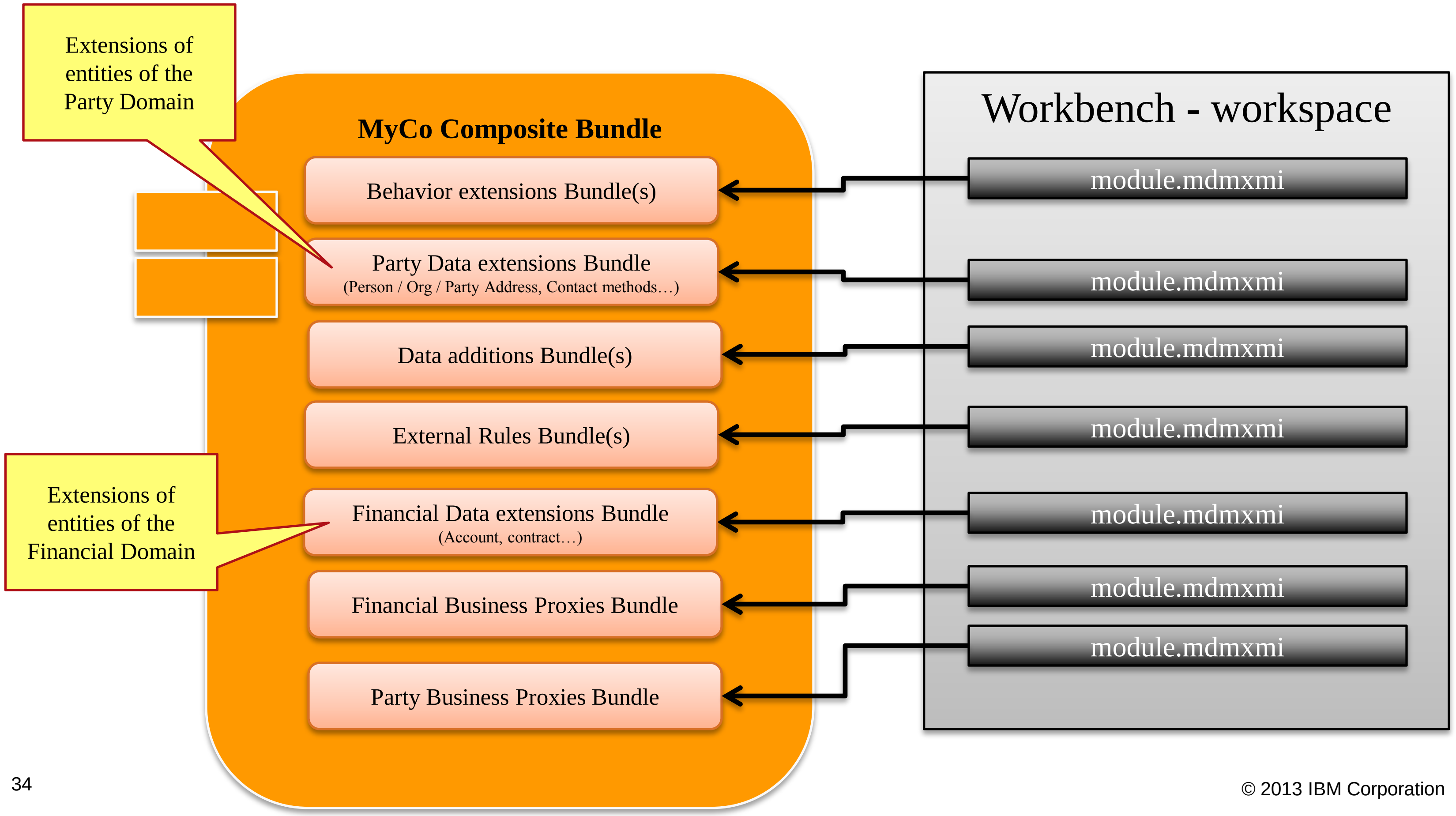
- Multi-module and bundle projects with separation of data extension, data additions, behavior extensions, external rules, business proxies etc.
- Simpler to deploy and maintain.



Componentization / Modularization Scenario II

Multi-module grouped – 1 CBA deployment

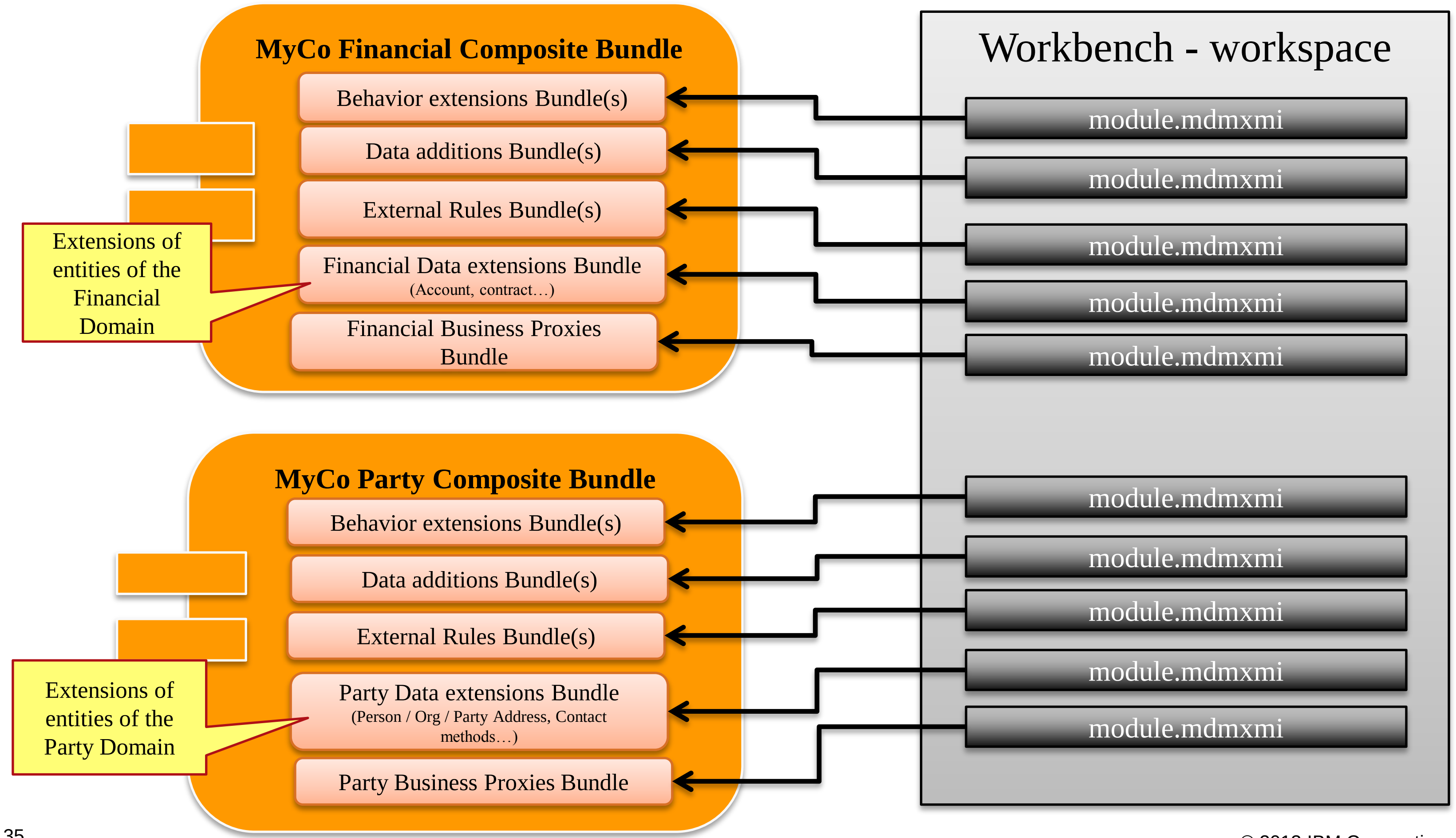
- Similar to scenario #1, but componentizing a step further and grouping by domains. Party module, financial module, etc...


Extensions of entities of the Party Domain
MyCo Composite Bundle
Behavior extensions Bundle(s)
Party Data extensions Bundle (Person / Org / Party Address, Contact methods…)
Data additions Bundle(s)
External Rules Bundle(s)
Financial Data extensions Bundle (Account, contract…)
Financial Business Proxies Bundle
Party Business Proxies Bundle
Extensions of entities of the Financial Domain
Workbench - workspace
module.mdmxmi
module.mdmxmi
module.mdmxmi
module.mdmxmi
module.mdmxmi
module.mdmxmi
module.mdmxmi

Componentization / Modularization Scenario III

Multi-module with multiple CBA deployment

- This scenario assumes there are no cross-dependencies, otherwise a single CBA is ideal to ease updates to CU extensions in WAS.

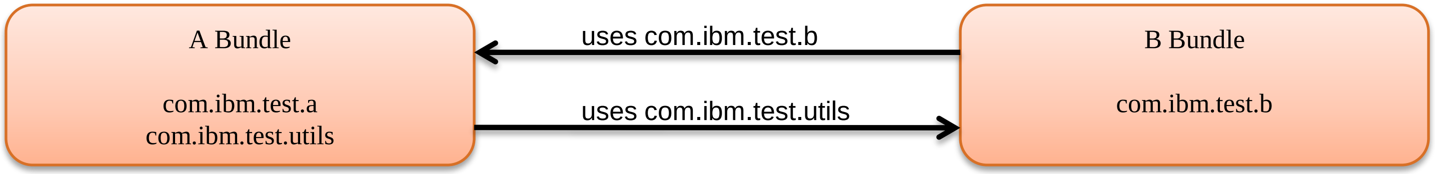


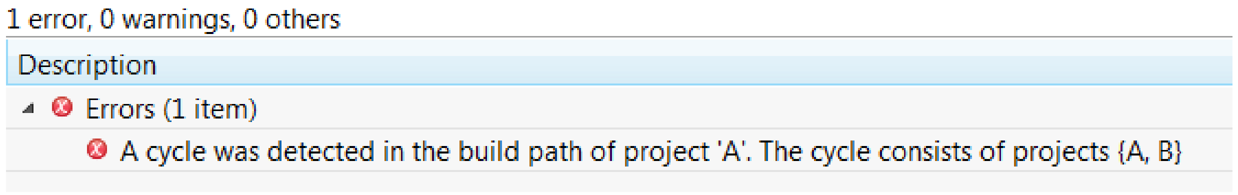
MyCo Financial Composite Bundle
Behavior extensions Bundle(s)
Data additions Bundle(s)
External Rules Bundle(s)
Financial Data extensions Bundle (Account, contract…)
Financial Business Proxies Bundle
Extensions of entities of the Financial Domain
MyCo Party Composite Bundle
Behavior extensions Bundle(s)
Data additions Bundle(s)
External Rules Bundle(s)
Party Data extensions Bundle (Person / Org / Party Address, Contact methods…)
Party Business Proxies Bundle
Extensions of entities of the Party Domain
Workbench - workspace
module.mdmxmi
module.mdmxmi
module.mdmxmi
module.mdmxmi
module.mdmxmi
module.mdmxmi
module.mdmxmi
module.mdmxmi
module.mdmxmi
module.mdmxmi

Componentization / Modularization – Best Practices

Don’ts

- Avoid componentization which would introduce cyclical dependencies.
  A class in Bundle A depends on a class from Bundle B which also depends on a class from Bundle A





1 error, 0 warnings, 0 others

Description
Errors (1 item)
A cycle was detected in the build path of project 'A'. The cycle consists of projects {A, B}

- Avoid componentization which breakup domain modules.
  Ie. Don’t extend person entity in 1 module, and Org entity in another module. The Query/Persistence framework wont tolerate this at least not without major refactoring!

Componentization / Modularization – Best Practices

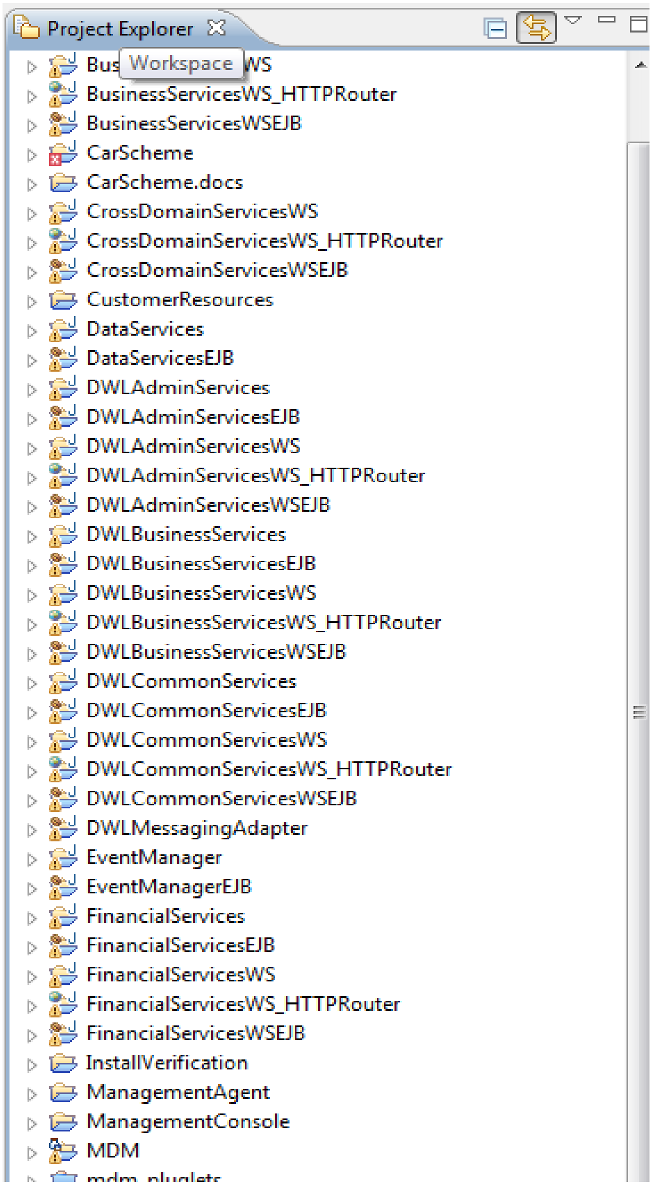
Do's

- Keep things simple!
  Rely on the workbench generated structure to avoid complexity of packaging and deployment.
- Don't over componentize; do not componentize every data extension individually. Componentize by domain/module instead.

Best practices and troubleshooting article:

http://www.ibm.com/developerworks/data/library/techarticle/dm-1409mdm11-osgi/index.html

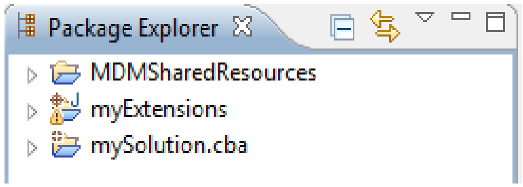
pre-v11 - Workspace



Project Explorer

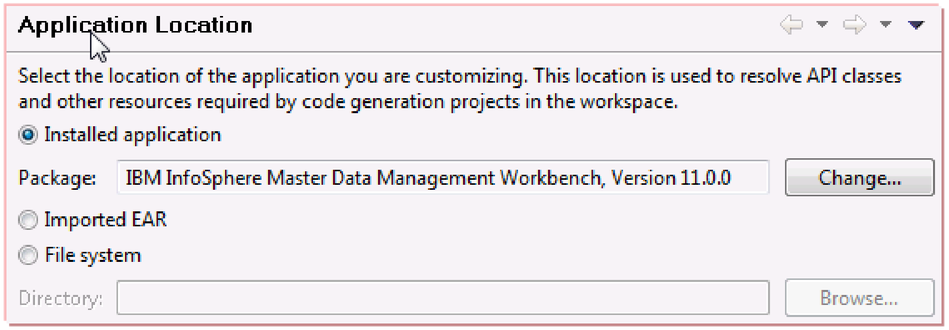
Bus Workspace WS
BusinessServicesWS_HTTPRouter
BusinessServicesWSEJB
CarScheme
CarScheme.docs
CrossDomainServicesWS
CrossDomainServicesWS_HTTPRouter
CrossDomainServicesWSEJB
CustomerResources
DataServices
DataServicesEJB
DWLAdminServices
DWLAdminServicesEJB
DWLAdminServicesWS
DWLAdminServicesWS_HTTPRouter
DWLAdminServicesWSEJB
DWLBusinessServices
DWLBusinessServicesEJB
DWLBusinessServicesWS
DWLBusinessServicesWS_HTTPRouter
DWLBusinessServicesWSEJB
DWLCommonServices
DWLCommonServicesEJB
DWLCommonServicesWS
DWLCommonServicesWS_HTTPRouter
DWLCommonServicesWSEJB
DWLMessagingAdapter
EventManager
EventManagerEJB
FinancialServices
FinancialServicesEJB
FinancialServicesWS
FinancialServicesWS_HTTPRouter
FinancialServicesWSEJB
InstallVerification
ManagementAgent
ManagementConsole
MDM
mdm_pluglets

v11 - Workspace



Package Explorer

MDMSharedResources
myExtensions
mySolution.cba



Application Location

Select the location of the application you are customizing. This location is used to resolve API classes and other resources required by code generation projects in the workspace.

[✓ Installed application]

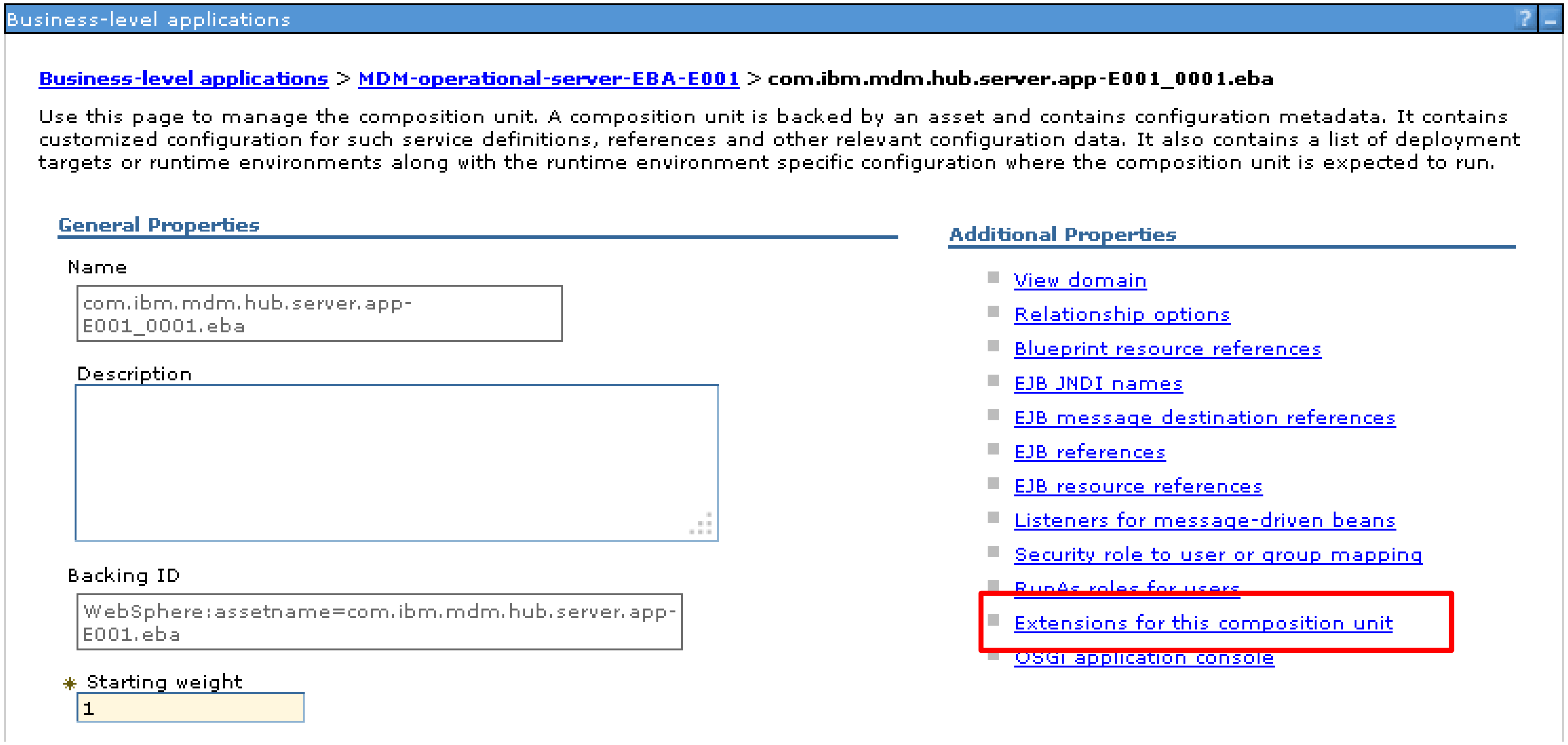
Package: IBM InfoSphere Master Data Management Workbench, Version 11.0.0 Change...

Imported EAR

File system

Directory: Browse...

Extending MDM EBA with a CBA



Business-level applications

Business-level applications > MDM-operational-server-EBA-E001 > com.ibm.mdm.hub.server.app-E001_0001.eba

Use this page to manage the composition unit. A composition unit is backed by an asset and contains configuration metadata. It contains customized configuration for such service definitions, references and other relevant configuration data. It also contains a list of deployment targets or runtime environments along with the runtime environment specific configuration where the composition unit is expected to run.

General Properties

Name: com.ibm.mdm.hub.server.app-E001_0001.eba

Description:

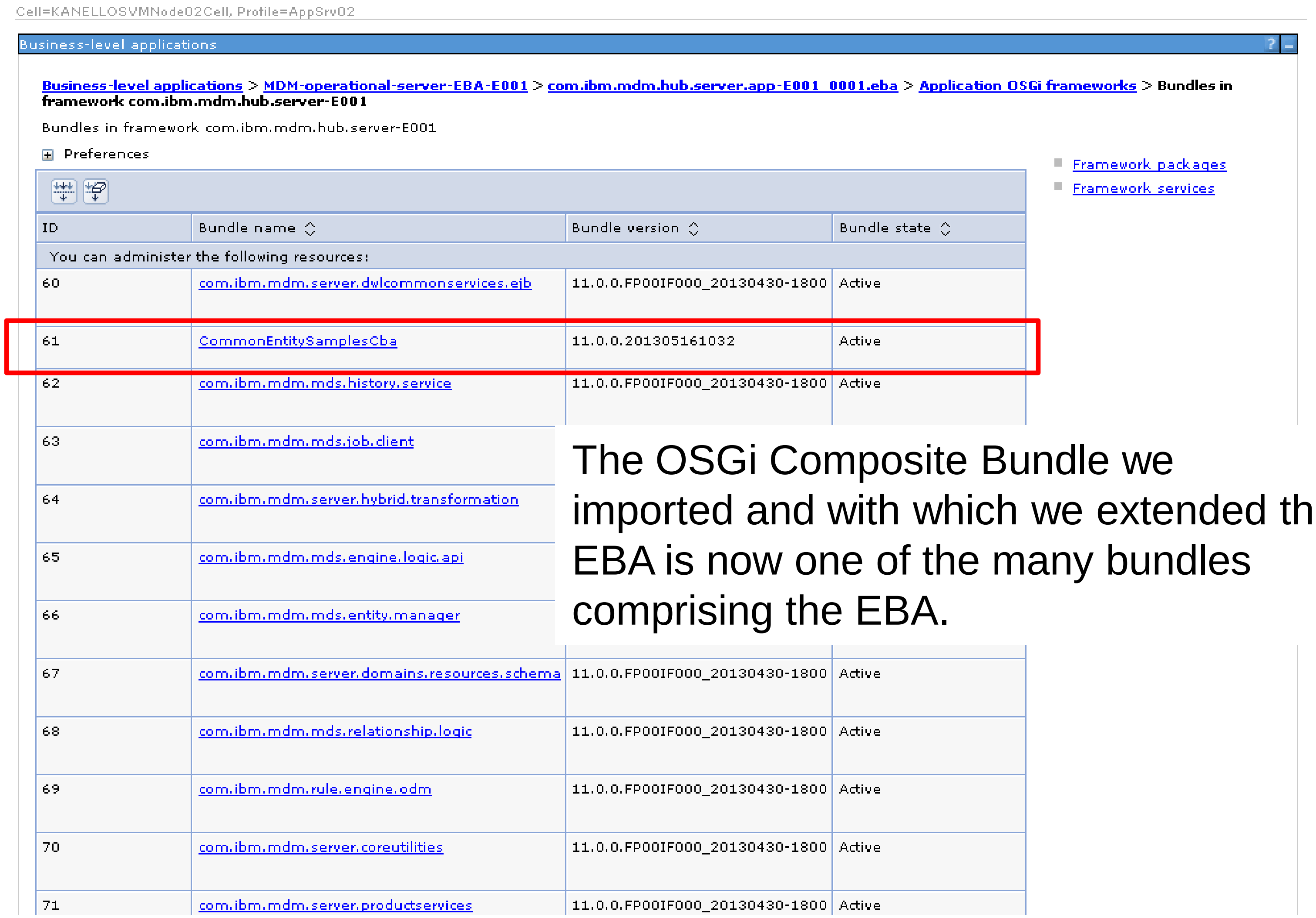
Backing ID: WebSphere:assetname=com.ibm.mdm.hub.server.app-E001.eba

* Starting weight: 1

Additional Properties

- View domain
- Relationship options
- Blueprint resource references
- EJB JNDI names
- EJB message destination references
- EJB references
- EJB resource references
- Listeners for message-driven beans
- Security role to user or group mapping
- RunAs roles for users
- Extensions for this composition unit
- OSGi application console

The Composite Bundle is now part of the Business Level Application



Cell=KANELLOSVMNode02Cell, Profile=AppSrv02

Business-level applications

Business-level applications > MDM-operational-server-EBA-E001 > com.ibm.mdm.hub.server.app-E001_0001.eba > Application OSGi frameworks > **Bundles in framework com.ibm.mdm.hub.server-E001**

Bundles in framework com.ibm.mdm.hub.server-E001

⊞ Preferences

- Framework packages
- Framework services

ID	Bundle name	Bundle version	Bundle state
You can administer the following resources:			
60	com.ibm.mdm.server.dwlcommonservices.ejb	11.0.0.FP00IF000_20130430-1800	Active
61	CommonEntitySamplesCba	11.0.0.201305161032	Active
62	com.ibm.mdm.mds.history.service	11.0.0.FP00IF000_20130430-1800	Active
63	com.ibm.mdm.mds.job.client		
64	com.ibm.mdm.server.hybrid.transformation		
65	com.ibm.mdm.mds.engine.logic.api		
66	com.ibm.mdm.mds.entity.manager		
67	com.ibm.mdm.server.domains.resources.schema	11.0.0.FP00IF000_20130430-1800	Active
68	com.ibm.mdm.mds.relationship.logic	11.0.0.FP00IF000_20130430-1800	Active
69	com.ibm.mdm.rule.engine.odm	11.0.0.FP00IF000_20130430-1800	Active
70	com.ibm.mdm.server.coreutilities	11.0.0.FP00IF000_20130430-1800	Active
71	com.ibm.mdm.server.productservices	11.0.0.FP00IF000_20130430-1800	Active

The OSGi Composite Bundle we imported and with which we extended the EBA is now one of the many bundles comprising the EBA.

Demo

- Ideal MDM Workspace settings
  - Auto-publish disabled
  - “Run on Server” setting
  - Run in debug mode
  - Start server without generated script
- Push sample extension/addition
- Log analysis of deployment
- OSGI application console view

Summary OSGi in general

- OSGi is a technology that isolates the components of an application from other components
- It’s basic unit is the Bundle. A bundle is nothing more than an JAR file having a special manifest that is read and understood by an OSGi container.
- It provides a way for programmers to declare what aspects of their bundles they wish to expose and which they wish to keep internal.
- It provides a way to extend applications with new features without the need to redeploy the application.
- It provides a way to patch applications without the need to redeploy them

Summary MDM using OSGi

- Simplified development
  - Cleaner workspace
  - Ease of customizations deployment
  - Reduces cross-team interdependence

- Simplified update process
  - In-place update of bundles (the ability to apply a bundle update/fix without restarting the server)
  - Separation of core code (IBM code) and extensions (Customer code)
  - Patching simplified; IBM patches against core bundle(s); customer patches against the extension bundle(s)

Thank You - Question?