Module 02 MDM Architecture

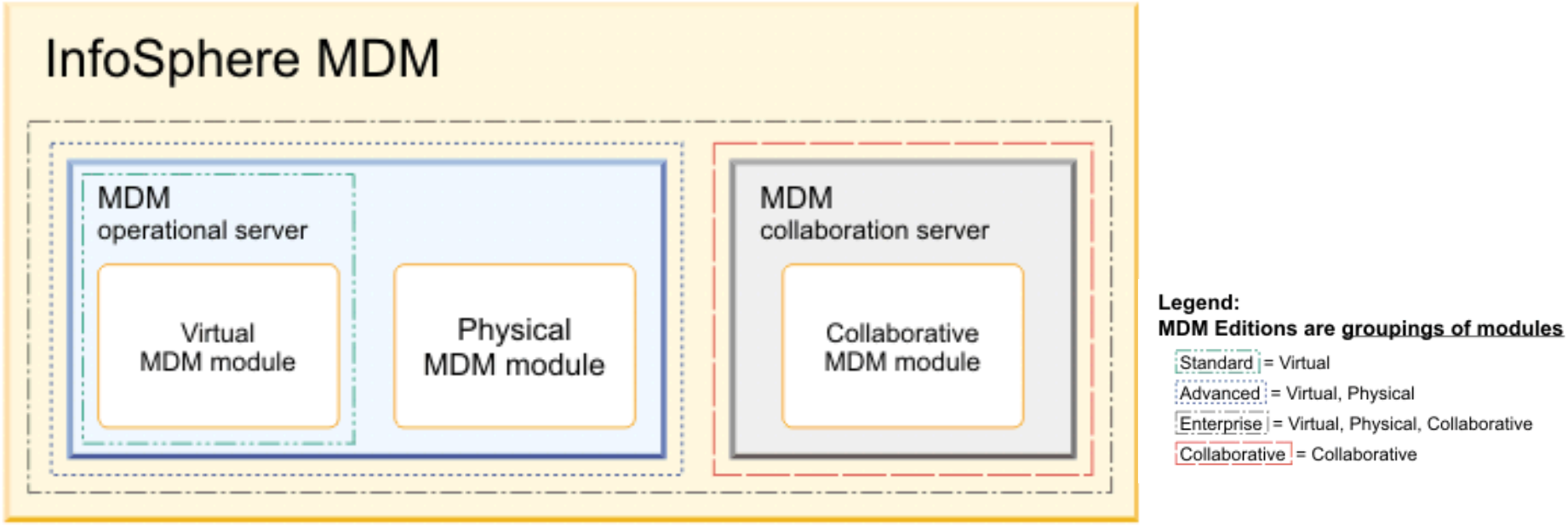
IBM InfoSphere Master Data Management

Module Objectives

After completing this topic, you should be able to explain:

- InfoSphere MDM Implementation Styles
- InfoSphere MDM Architecture
- InfoSphere MDM and OSGi

InfoShere MDM



- The technical capabilities of virtual, physical, and hybrid MDM help you to manage your master data in **any implementation style**, such as ***registry***, ***centralized*** **and** ***coexistence***.

Four Architecture Styles of MDM - Gartner

	Consolidation	Registry	Centralized	Coexistence
				
Authorship vs. Hub	Author is separate from hub	Author is separate from hub	Authorship or harmonization takes place in Hub	Author anywhere
Persistence vs. Hub	Hub stores copy separate from author/source	Hub stores index for master data; master exists at edge	Master persists in hub, though copies may exist at edge	Persist anywhere
Validation	Hub is system of reference	Hub is system of reference	Hub is system of record	Mixed system of record/ reference
Primary Consumer	Downstream analytics and reporting	Both operational and analytical	Upstream operations	Upstream operations
Data Latency	Batch to real time	Batch to event-driven	Real time	Event-driven, pub/sub
Search Complexity	Relatively light	Very complex	Relatively light	Reasonably complex

Explicitly Cleans Up Source Data/Processes

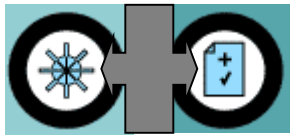
InfoSphere MDM Styles

Advanced Edition

Standard Edition

- Virtual MDM
  - Master data is created in a distributed fashion and remains independent across source systems. MDM provides a central "indexing" service while source systems continue to be the system of record (SOR)
    - ***Registry MDM***: System of reference for master data
    - MDM provides operational inquiry & search functions

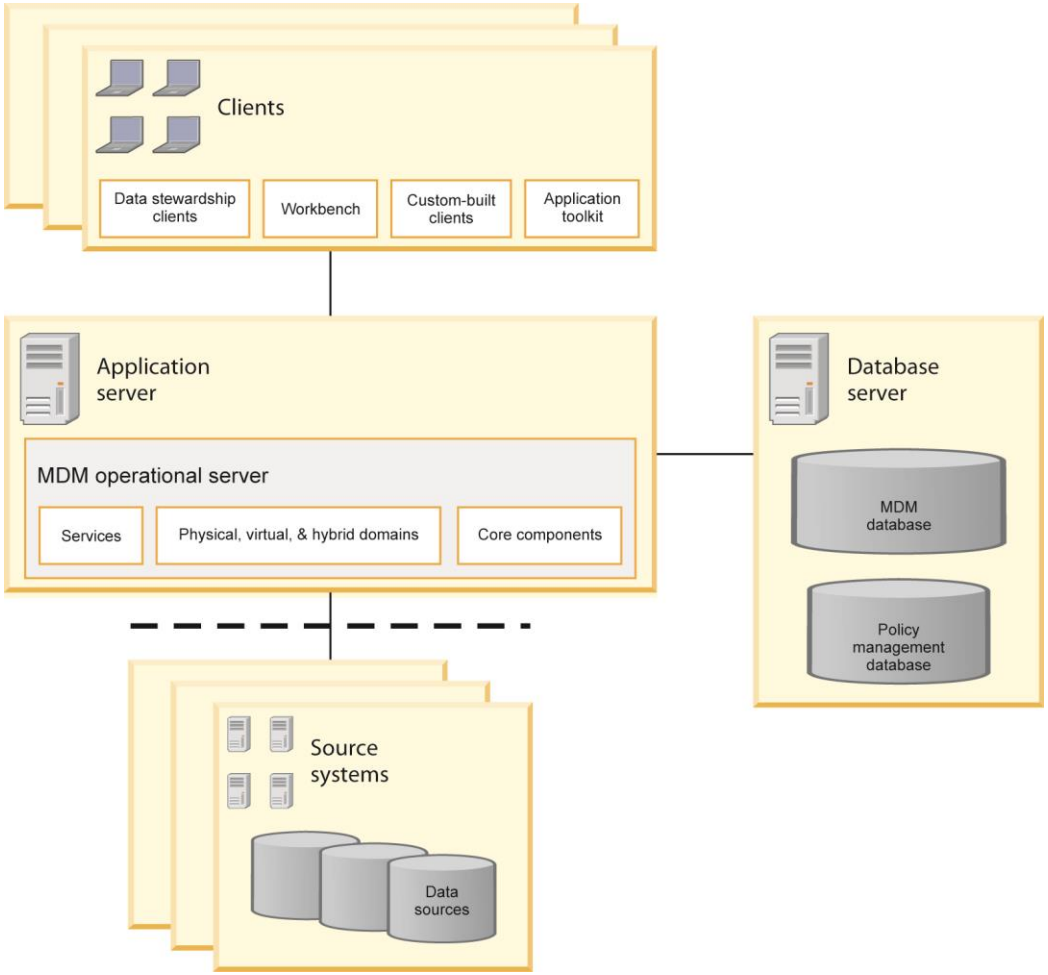
- Physical MDM
  - Master data is created in, stored in, and accessed from a central system
    - ***Centralized MDM***: System of record for master data
    - MDM provides operational data management functions; SOA services



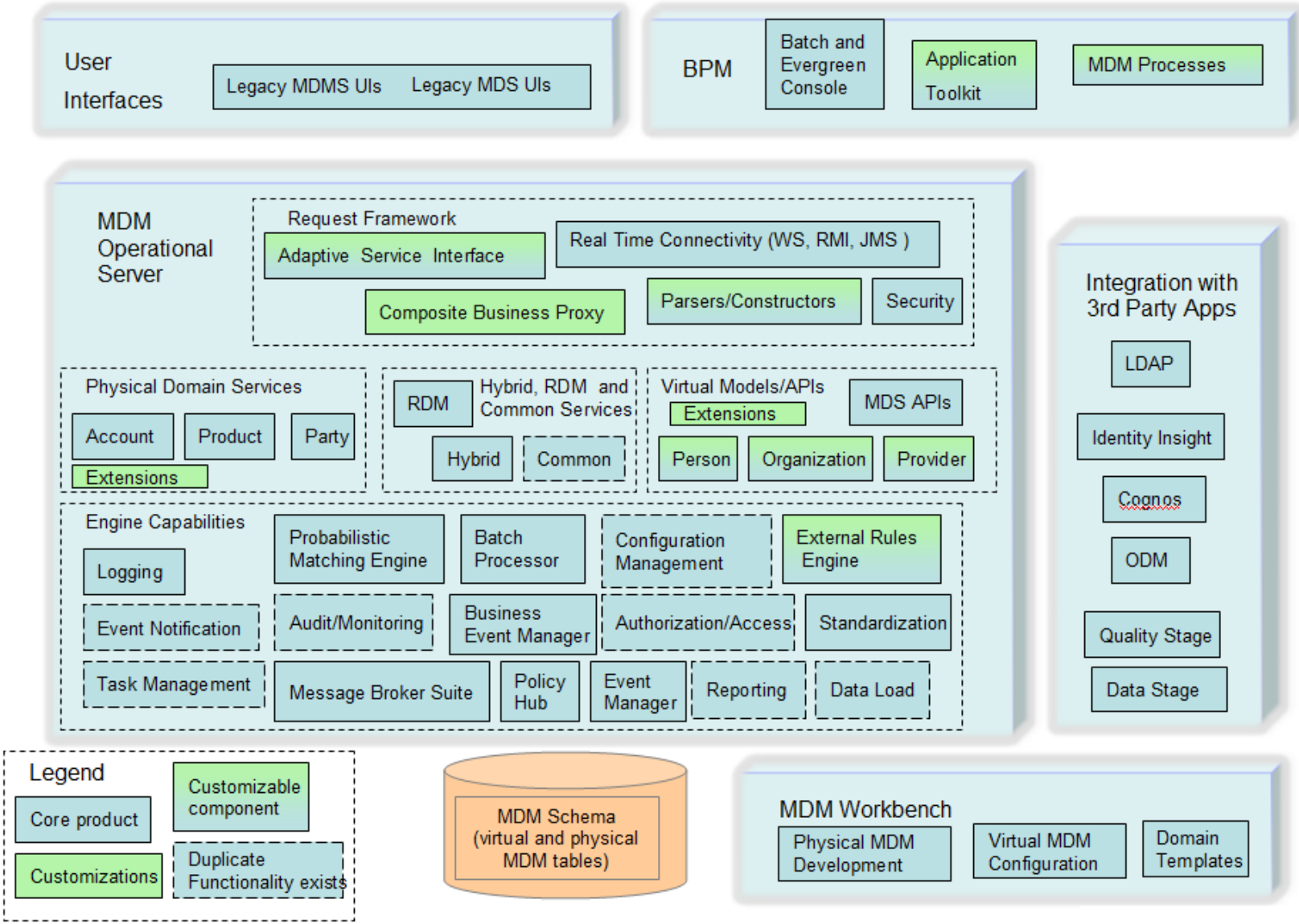
- Hybrid MDM
  - ***Coexistence*** implementation style combines physical and virtual technologies.

InfoSphere MDM Architecture – Unified Engine

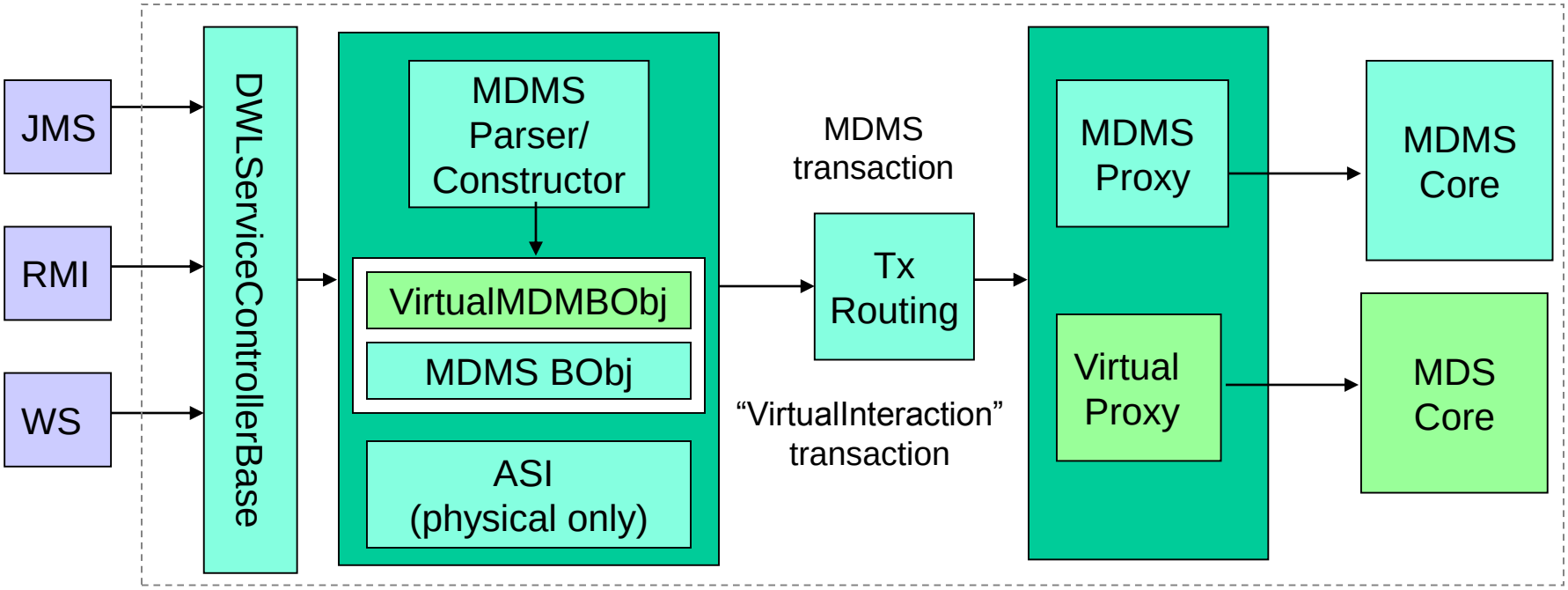
- The virtual and physical MDM modules are combined into **single engine** and share a **single infrastructure**.
- Unification of engine components provides foundation for **Hybrid** use cases
- Virtual and physical MDM tables are combined in **one MDM schema**
- MDM operational server runs within the WebSphere Application Server (WAS) container
- WAS provides infrastructure for authentication and logging that was previously managed by the MDM SE itself, and leverage embedded JMS messaging infrastructure



Unified MDM Architecture – component diagram



Co-resident Operational Server– under the cover



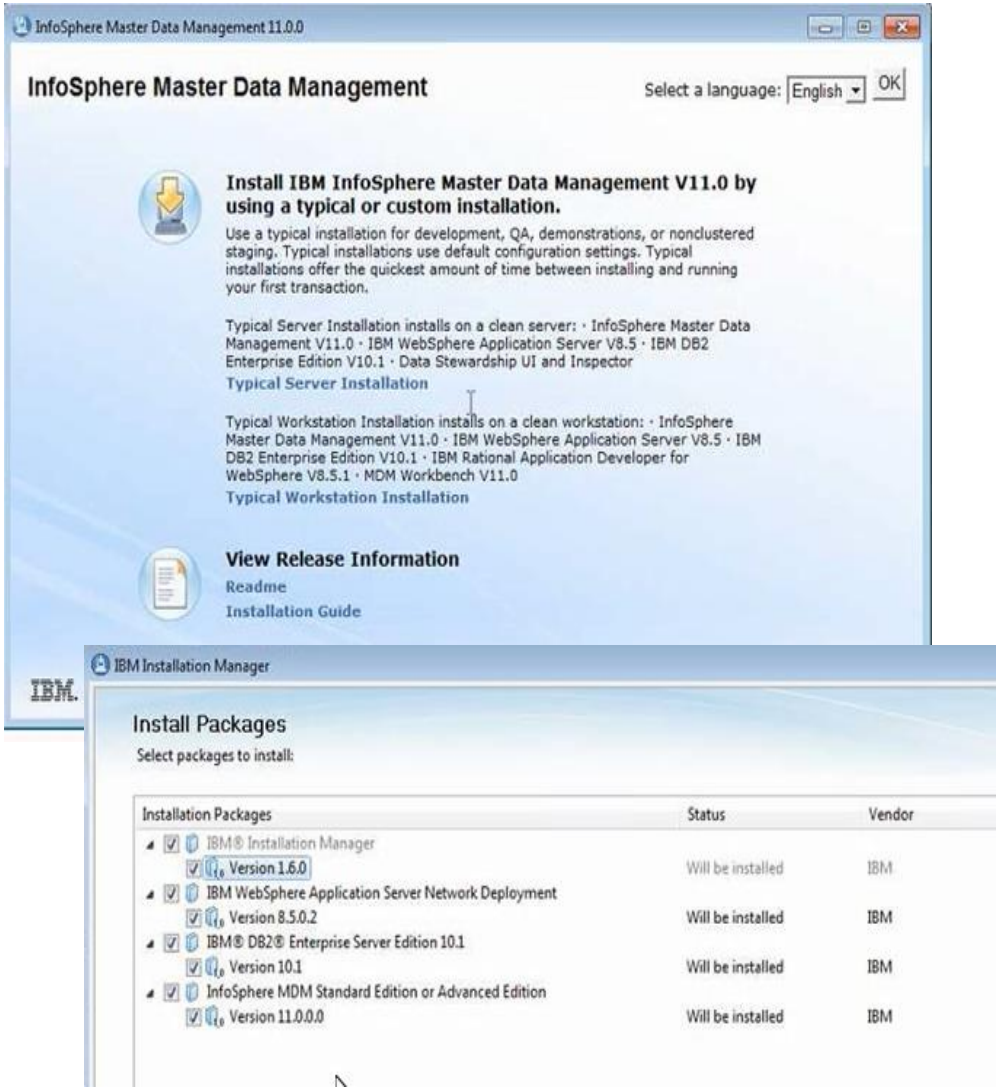
- Existing MDM Server SOA interfaces (RMI, WS and JMS) and Request/Response framework are re-used

Unified MDM Architecture and others

- Unified installation
- Unified workbench
- Unified security
- Unified logging

Unified Installation

- Unified Installer
  - All MDM editions now using IBM Installation Manager based install process
  - MDM AE & SE share the same installer which deploys MDM Operational Server
  - No difference between RDM and CDH Installer other than license
- Typical Installation mode for Server and Workstation
  - Accelerated, simplified installation
  - Deploys all prerequisites (DB2, WAS, RAD) as part of single install session
  - Supported for MDM AE & SE only
- Custom installation mode supports cluster deployment



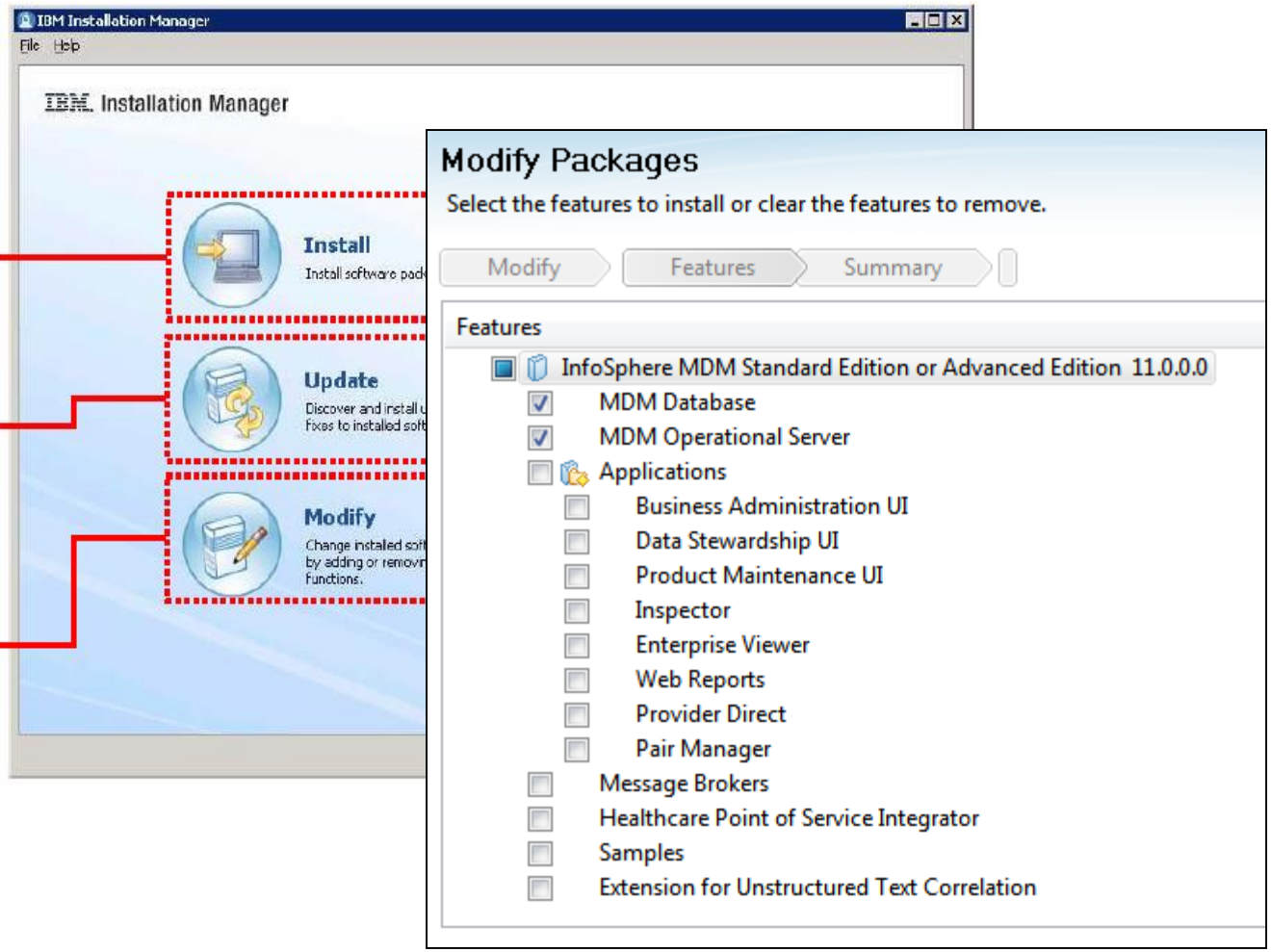
Unified Installation - Installing optional components

- Installation Manager Modify wizard provides ability to install additional optional components

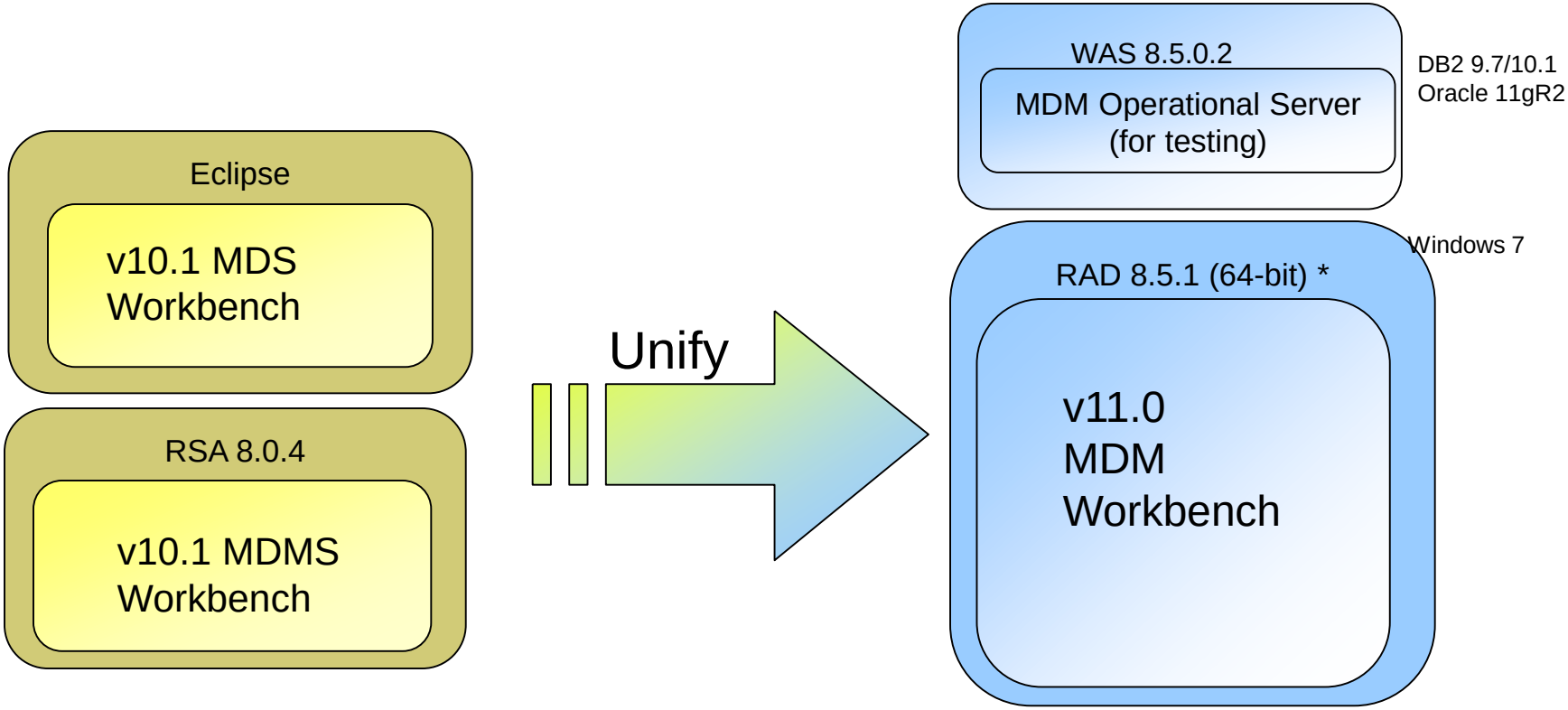
The **Install** wizard walks you through installing new products

The **Update** wizard discovers and installs updates and fixes

The **Modify** wizard can add or remove product features



Unified Workbench



- Evolved from MDMS and MDS tools
- Co-existence in same Eclipse “shell”
- Contains two perspectives:
  - MDM Configuration Perspective for virtual MDM configurations
  - MDM Development Perspective for physical MDM development

Unified Security

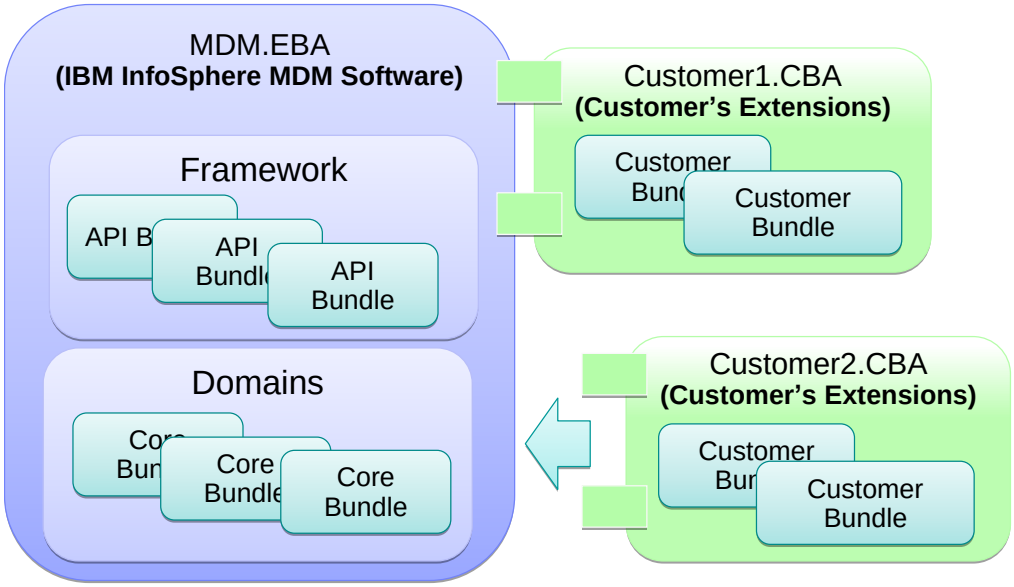
- The current version of InfoSphere MDM **requires** that administrative security is enabled for the application server
- User management for Standard and Advanced Editions
  - To create and populate user groups, use the IBM WebSphere Application Server Administrative Console
  - User management facility in MDM Workbench is removed (but group to access rights mapping still managed in WB)
  - Embedded LDAP is removed
- This release of Virtual MDM makes improvements to the security for the C++ and Java APIs
  - The context constructors now require credentials to create context pools or contexts. Until it is updated, existing code does not compile against the new API JAR file
- When you install InfoSphere MDM SE or AE using WebSphere® Application Server Federated Repository to store authentication users and groups, then IBM Installation Manager creates a set of default user groups required by virtual MDM
  - If your deployment uses external LDAP, you must create the groups manually at the same time that you create the *mdmadmin* user in LDAP, and before you start installing InfoSphere MDM using IBM Installation Manager

Unified Logging

- The MDM operational server utilizes the logging facilities provided by WebSphere.
- Configuration is done within the Websphere Adminstration Console.
  - The Logging options are available under Troubleshooting -> Logs and Tracing
  - A Components and Groups tree will be available to change logging levels
- Log files for the MDM are stored in the default Websphere log directory.
  - The default location for this directory is under the Websphere profiles directory for the server that logging has been configured
    - Example: c:\program files\IBM\Websphere\profiles\AppSrv01\logs\server1
- There are 2 log files that are of primary concern for MDM in this directory.
  - SystemOut.log
  - trace.log
- Virtual MDM performance logging stand-along facility is removed, performance logging is now enabled in WAS and reports are generated via madConfig target
- Physical MDM performance monitor facility remains the same as in MDM 10.1 and covers physical interactions
- Service Activity Monitoring (SAM) remains the same and covers high level virtual interaction summary and detailed physical transaction information

InfoSphere MDM v11 - OSGi

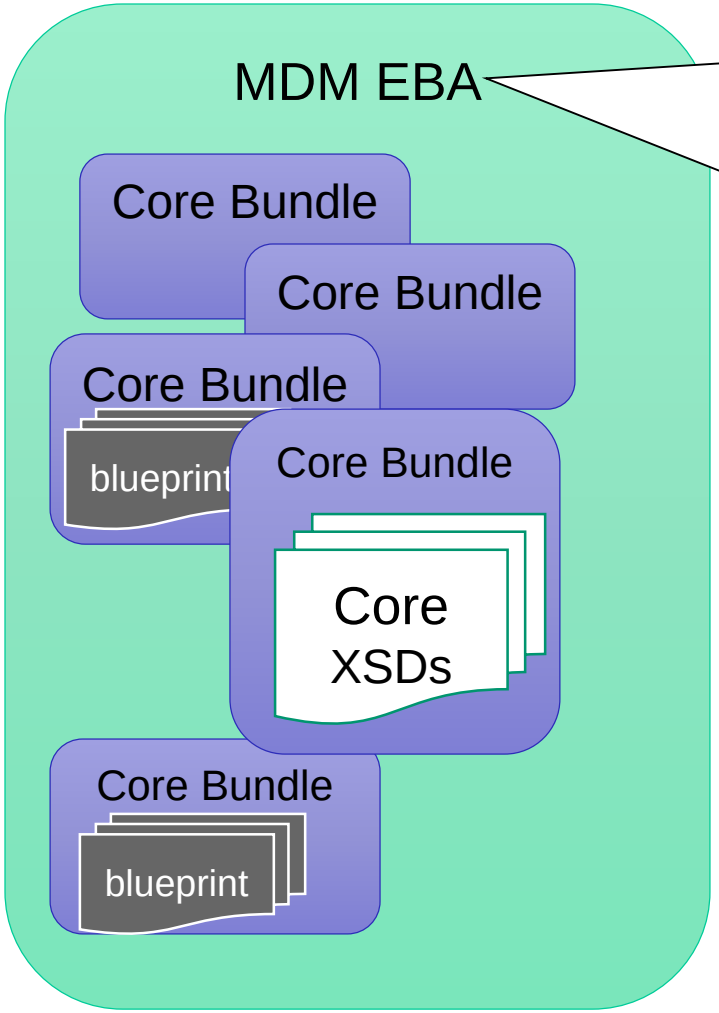
- OSGi has been adopted for Modular/granular InfoSphere MDM deployment
- Key benefits:
  - Separation of core code (IBM code) and extensions (customer's code) as components ("bundles")
  - Ability for customers to create multiple bundles such that separate development teams can work more independently
  - Patching simplified; IBM patches against core code bundle; customer patches against the extension bundle(s)



OSGi

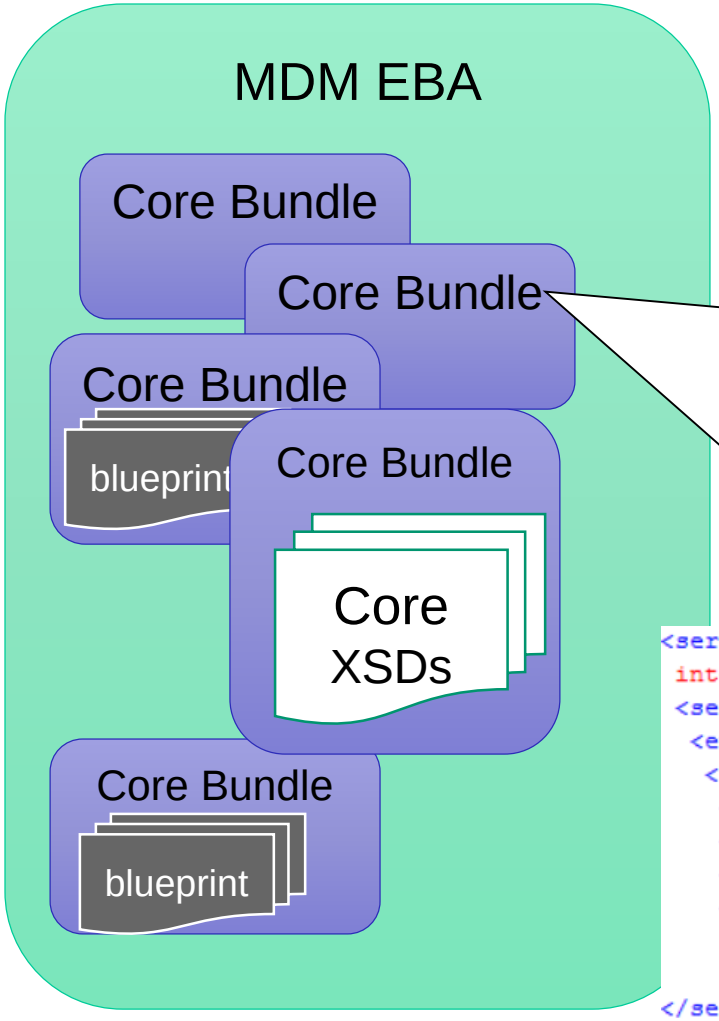
The OSGi framework ("*Open Services Gateway initiative*") is a module system and service platform for the Java programming language that implements a complete and dynamic component model, something that as of 2012 does not exist in standalone Java/VM environments. Applications or components (coming in the form of bundles for deployment) can be remotely installed, started, stopped, updated, and uninstalled without requiring a reboot; management of Java packages/classes is specified in great detail. Application life cycle management (start, stop, install, etc.) is done via APIs that allow for remote downloading of management policies. The service registry allows bundles to detect the addition of new services, or the removal of services, and adapt accordingly.

OSGi EBA



- Enterprise Bundle Archives (EBA)
  - represents a self-contained application.
  - defined by its own manifest, the APPLICATION.MF file

OSGi Bundles

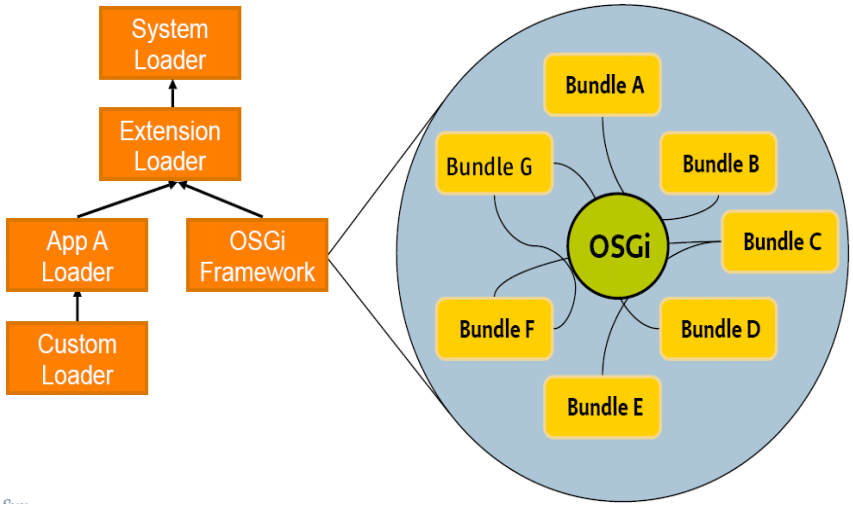


- OSGi bundles
  - the basic building blocks of OSGi
  - just a .jar file with a special manifest file (MANIFEST.MF) that describes its characteristics as an OSGi bundle
  - "blueprint"
    - provides a dependency injection framework for OSGi
    - defines all OSGi services

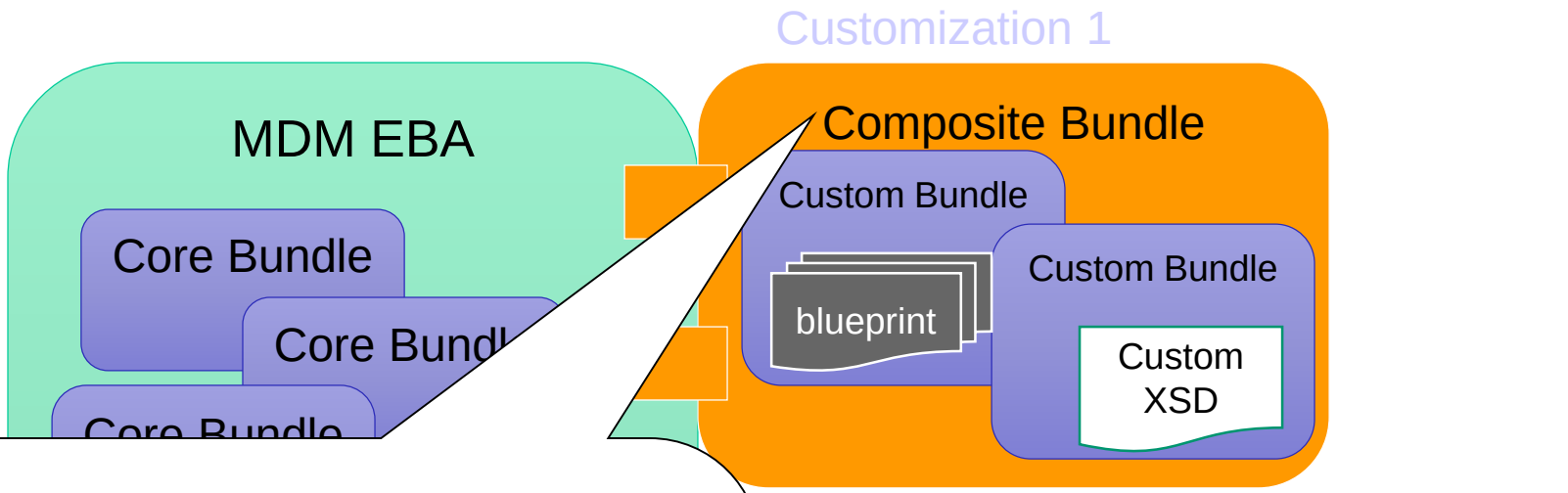
```
<service id="Controller.ITCRMCorePartyTxn"
 interface="com.dwl.tcrm.coreParty.interfaces.ITCRMCorePartyTxn">
 <service-properties>
  <entry key="osgi.jndi.service.name">
   <list>
    <value>collapsePartiesWithRules</value>
    <value>addAddress</value>
    <value>addPartyAddressPrivacyPreference</value>
    <value>collapseParties</value>
    ...

 </service-properties>
  <bean class="com.dwl.tcrm.coreParty.controller.TCRMCorePartyTxnBean"/>
</service>
```

OSGi bundles and class loading

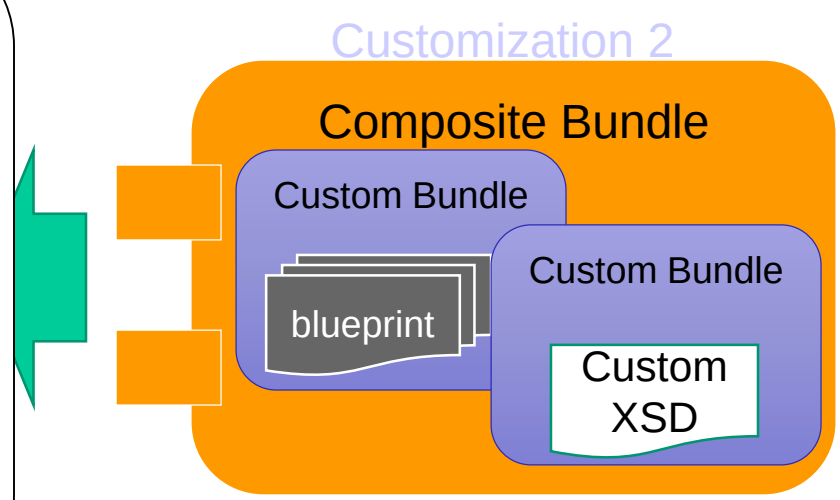
- Class loading
  - Each bundle has its own loader
  - No flat or monolithic class path
  - Class sharing and visibility decided by declarative dependencies, not by class loader hierarchies.
    - A bundle's class loader can only see the classes in the bundle, or
    - The classes in packages imported by the bundle
  - OSGi framework works out the dependencies including versions



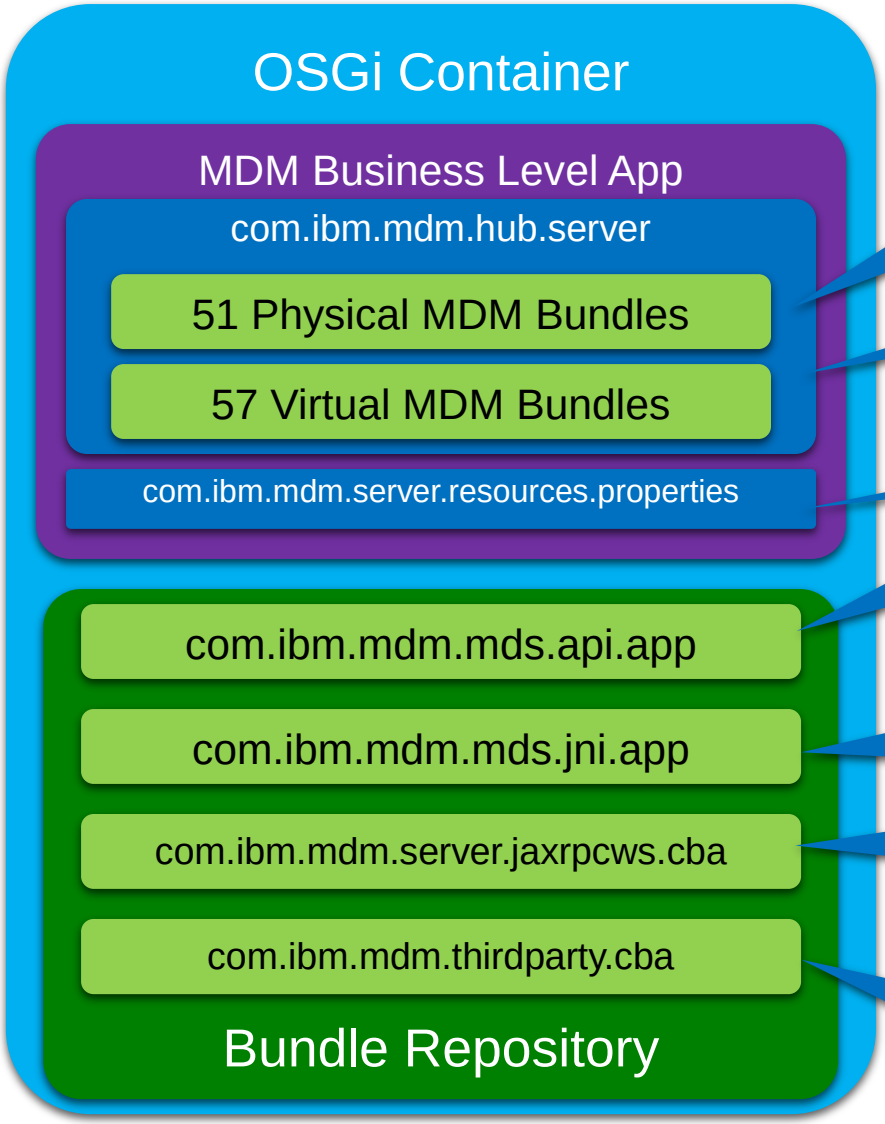
Customizations using OSGi Composite Bundles



- Composite Bundle
  - a bundle that contains multiple bundles that always coexist
  - deployed as a .cba file
  - can be created using the MDM Workbench
  - the manifest file, MANIFEST.MF, contains metadata that lists the bundles contained by the composite bundle



What makes up MDM Operational Server - OSGi



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

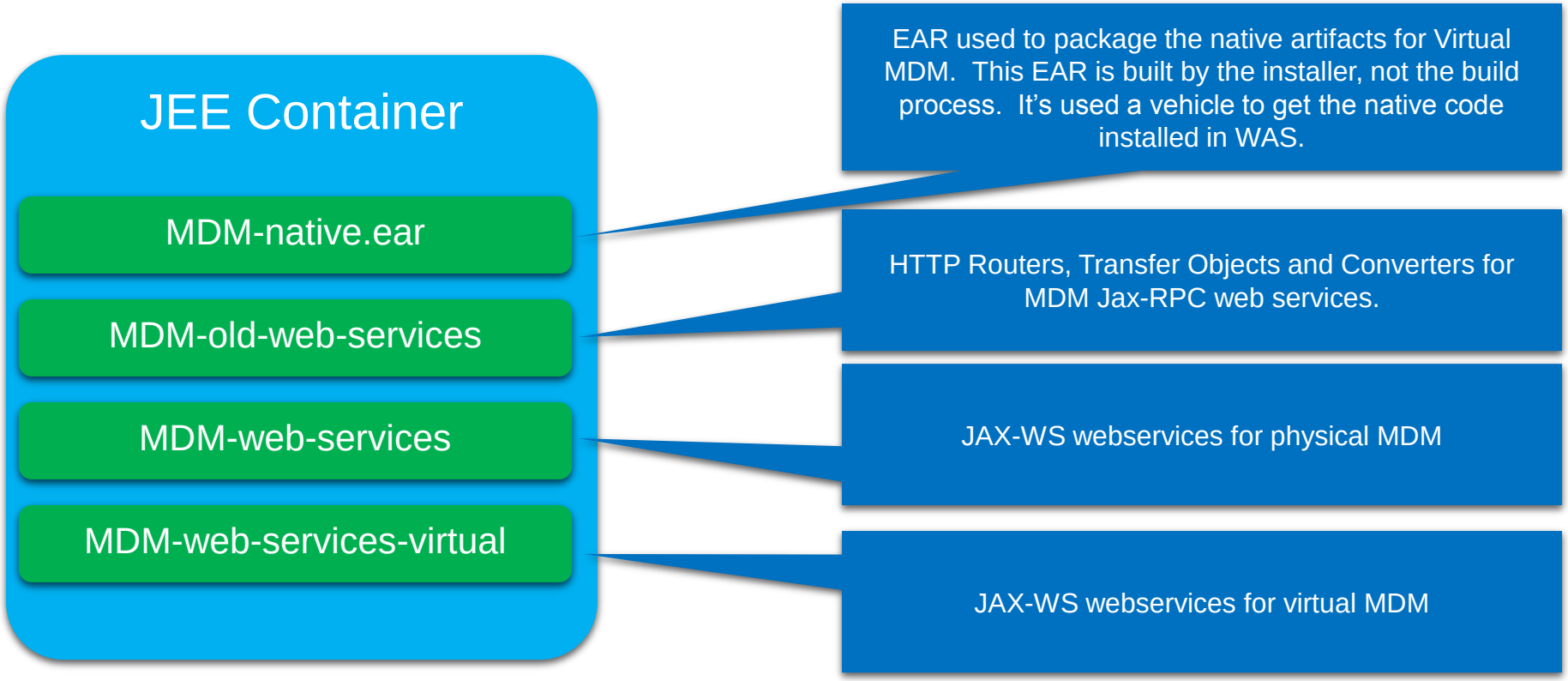
Contains API bundle for virtual engine and API bundle for PME. (Plus some third party bundles)

Contains one bundle: the JNI interface between SE components and native code artifacts

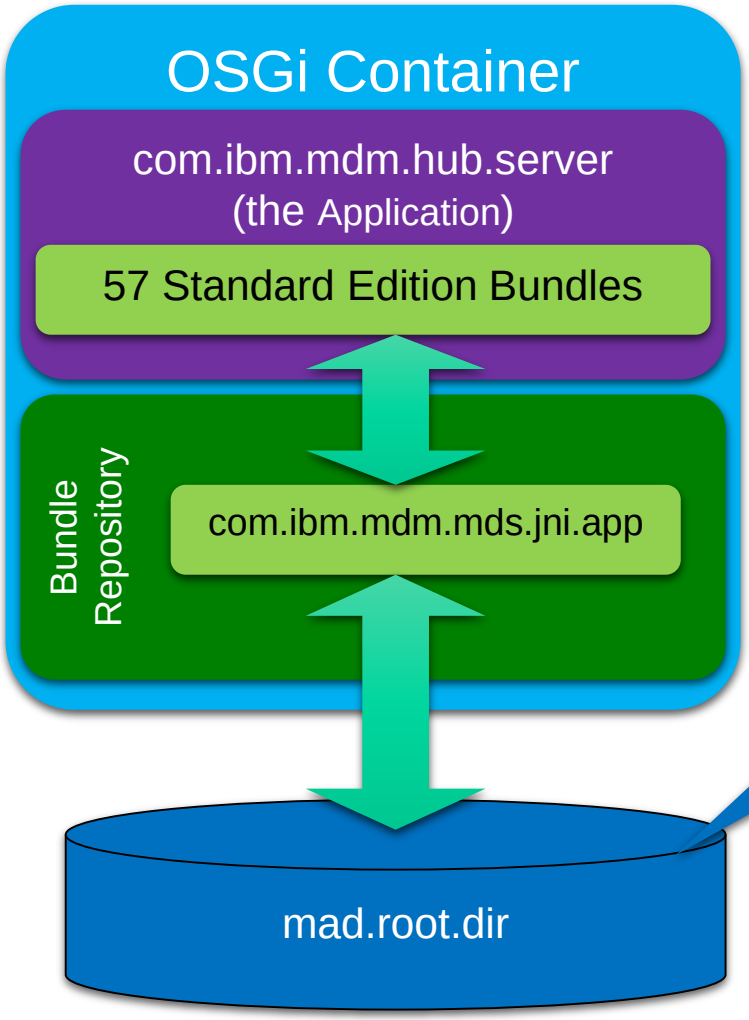
Contains converters and transfer objects for JAX RPC webservices for pMDM

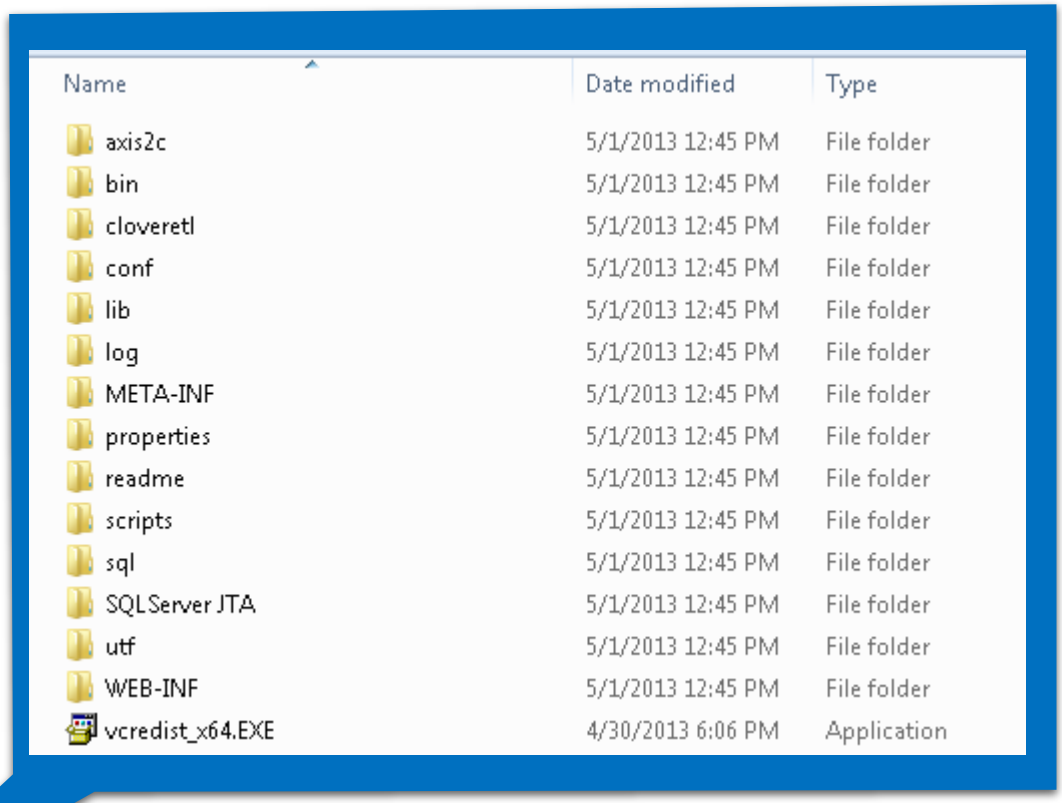
Contains numerous third party bundles used by AE and SE components.

What makes up MDM Operational Server – JEE/OSGi Applications



What makes up MDM Operational Server – Native Code Artifacts



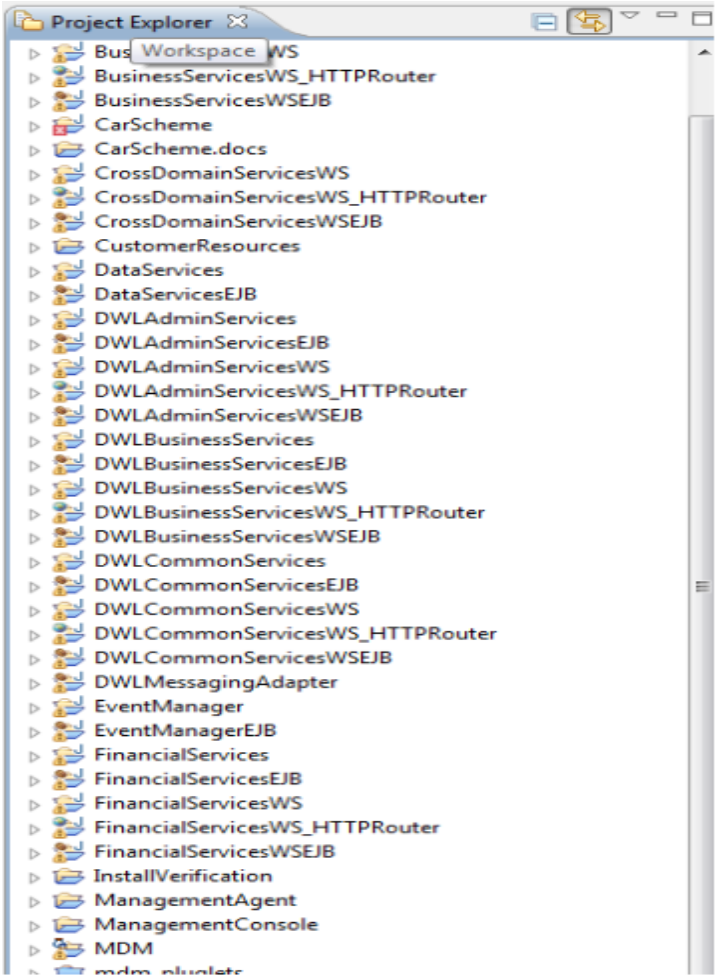


Name	Date modified	Type
axis2c	5/1/2013 12:45 PM	File folder
bin	5/1/2013 12:45 PM	File folder
cloveretl	5/1/2013 12:45 PM	File folder
conf	5/1/2013 12:45 PM	File folder
lib	5/1/2013 12:45 PM	File folder
log	5/1/2013 12:45 PM	File folder
META-INF	5/1/2013 12:45 PM	File folder
properties	5/1/2013 12:45 PM	File folder
readme	5/1/2013 12:45 PM	File folder
scripts	5/1/2013 12:45 PM	File folder
sql	5/1/2013 12:45 PM	File folder
SQLServer JTA	5/1/2013 12:45 PM	File folder
utf	5/1/2013 12:45 PM	File folder
WEB-INF	5/1/2013 12:45 PM	File folder
vcredist_x64.EXE	4/30/2013 6:06 PM	Application

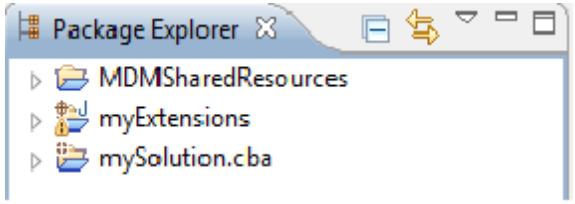
Mad.root.dir is an environment variable in WAS that points to the directory where the Virtual MDM native components reside on each server.

A Visual Indication of Simplicity

Before v11



v11 & Beyond



Project Management Benefits to OSGi

Party Team works on customization independently for the entities and business features of the Party Domain

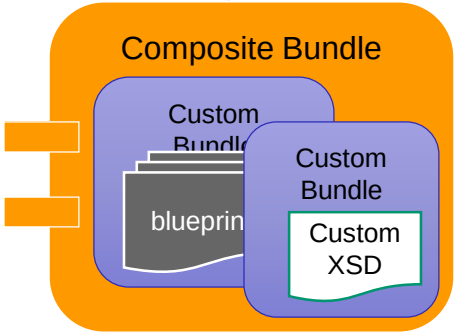
Party Team

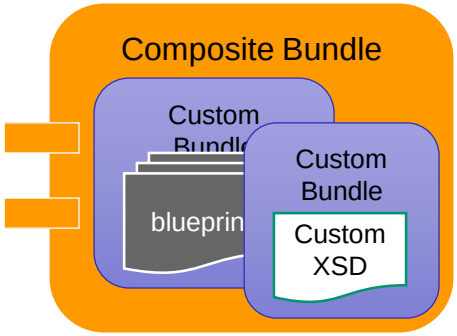
Reduced cross-team interdependence

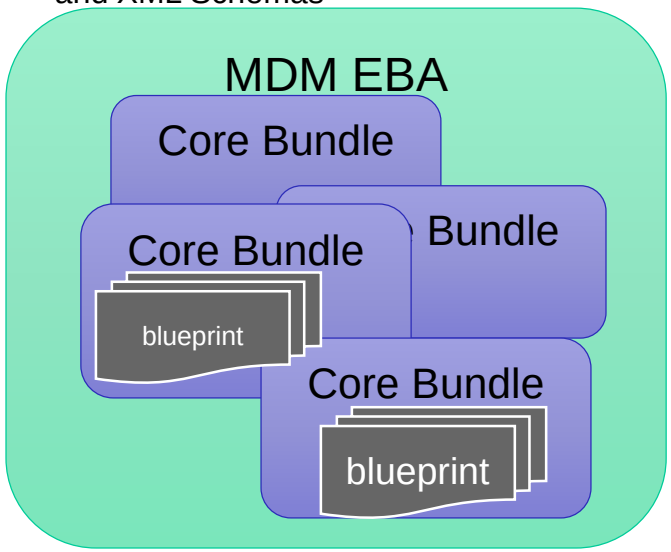
Financials Team

Financials Team works on customization independently

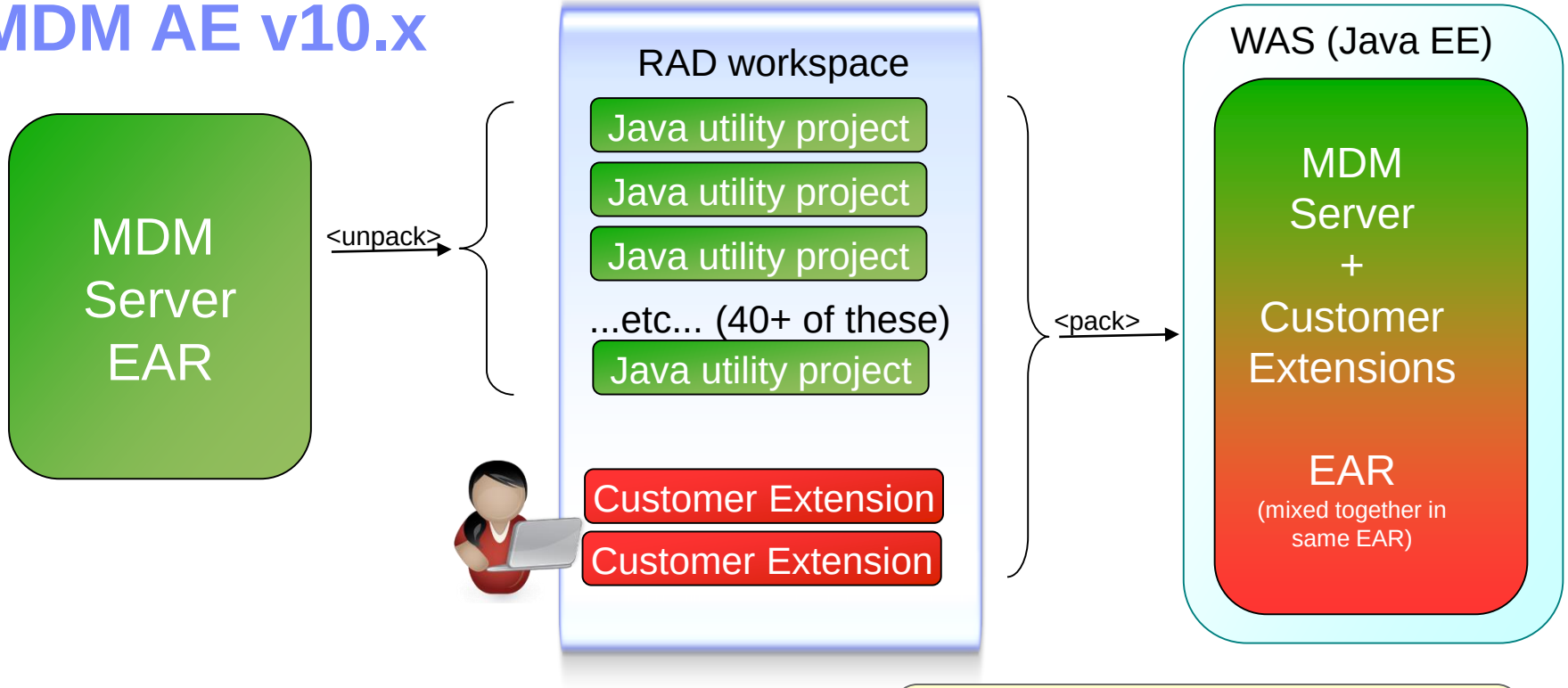
- Teams can work independently on separate features
- Release schedules can be independent
- Minimal effect on other teams when deploying customizations
- Reduced Regression testing across teams' components
- Increased Flexibility of delivery of customized features
- Coordination is required for data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







MDM AE v10.x



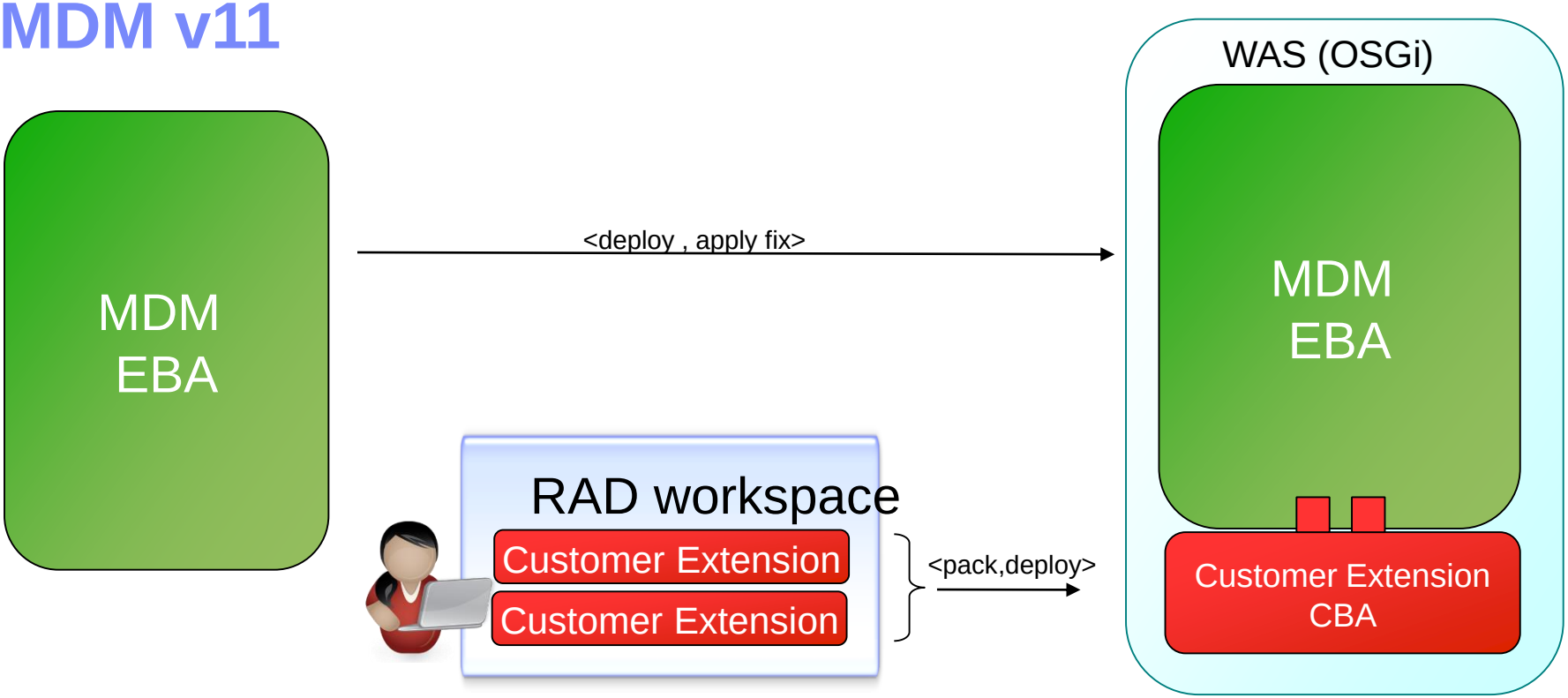
Users' projects "lost"/hard to see amongst the out-of-box imported projects – costs developer time

Applying server fixes to a workspace are hard, can take time

Cleaning/Re-building workspace takes a long time.
Lots of projects/files.
People disable auto-build !

WAS Global Classpath needs to be set
- gets long
- can cause issues
- takes time to maintain

MDM v11



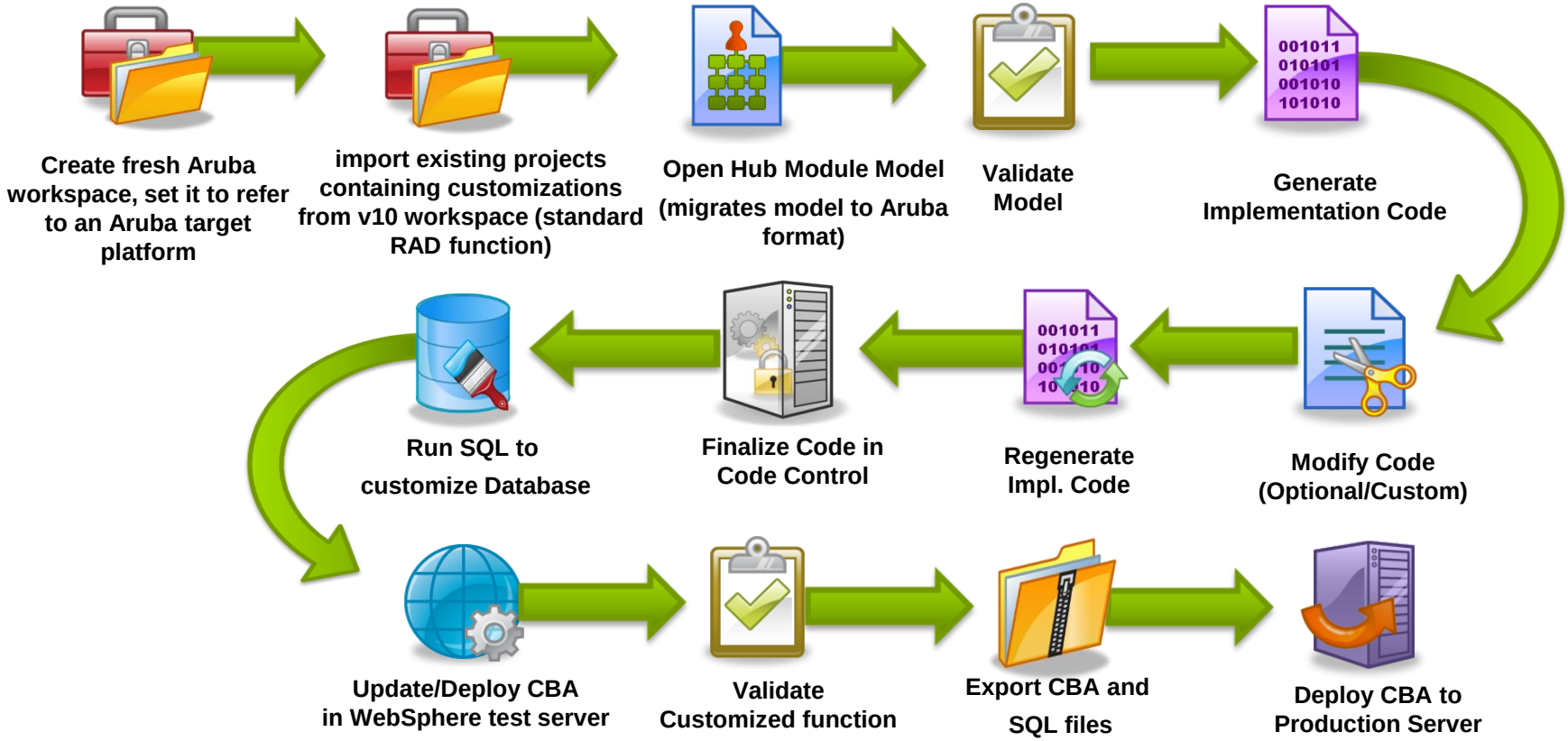
Applying a fix to the operational server is far simpler and quicker

OSGi maintains inter-bundle dependencies. No manual changing of WAS classpath

Less in workspace
- simpler to use
- faster to generate artifacts
- leave auto-build enabled

Server fixes are applied directly to the server
No need to touch the workspace.

Migrating customizations from MDM AE v9/v10 to MDM AE v11



- Workbench wizard helps migrate virtual handlers (callout, composite view, and event notification) and put them into OSGi packaging for deployment