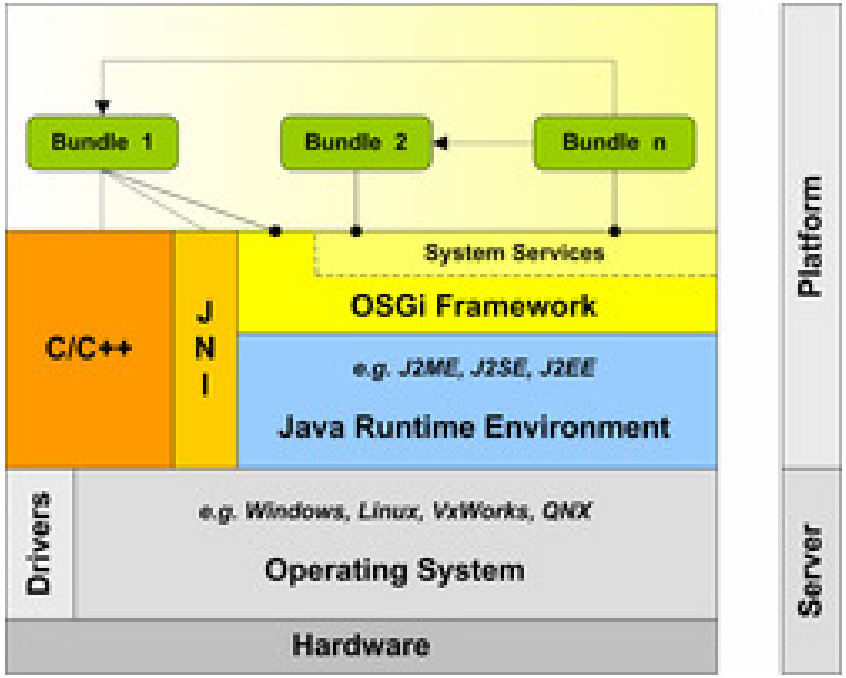
OSGi and Mule

Travis Carlson

What is OSGi?

- Dynamic module system for Java
- A sort of Java OS
- SOA within your app!



mule

Benefits of OSGi

- ✓ Plug & Play!
- ✓ Change parts of your app. dynamically without requiring restarts
- ✓ Components dynamically discover each other for collaboration
- ✓ Increased security: more isolation between modules of your app.

Industry Support

Plug-in model for Eclipse IDE (Equinox)

Embraced by Spring

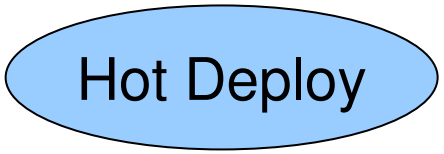
IBM Websphere 6.1

JSR 291: Dynamic Component Support for Java

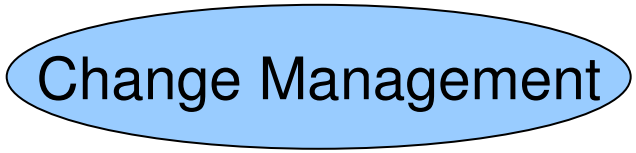
- Supporters: Apache, BEA, IBM, Intel, Nokia, Nortel Networks

Benefits of OSGi for Mule

Hot deploy your Mule app just like you would a WAR file, EAR file, etc. in J2EE



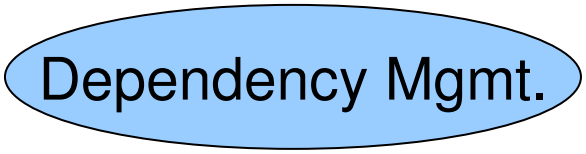
A Mule config can use different versions of the same class



Update and restart a part of your running Mule instance without having to restart all of it

Benefits of OSGi for Mule

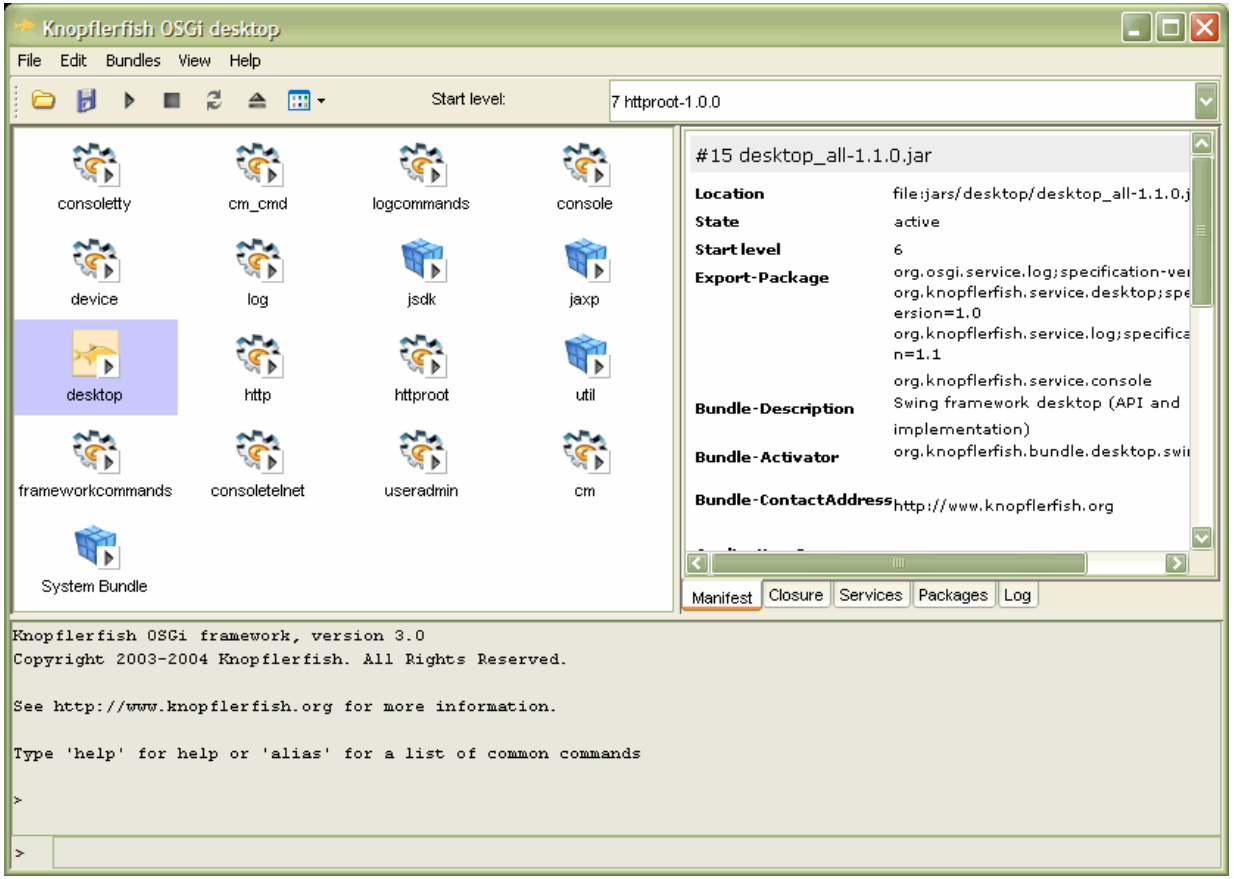
Separate classloader per bundle – not all of Mule has to depend on the same version of a 3rd party library



Enabling technology for Patch Management in MuleHQ

How it works

Mule runs within an “OSGi Framework”

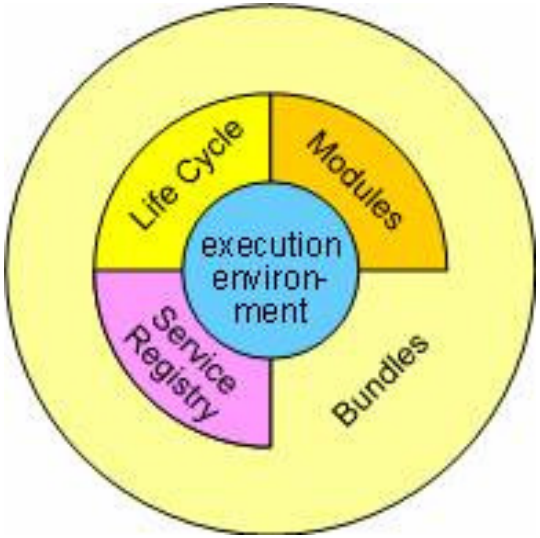


How it works

The Framework manages Bundles and Services:

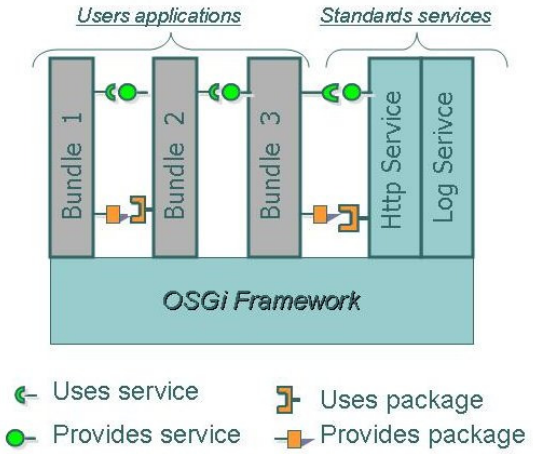
Bundles – static libraries, may or may not be startable/stoppable (via BundleActivator interface)

Services – dynamic components managed by a registry similar to JNDI



Bundles

Each Mule module/transport is an OSGi “bundle” containing a special manifest file



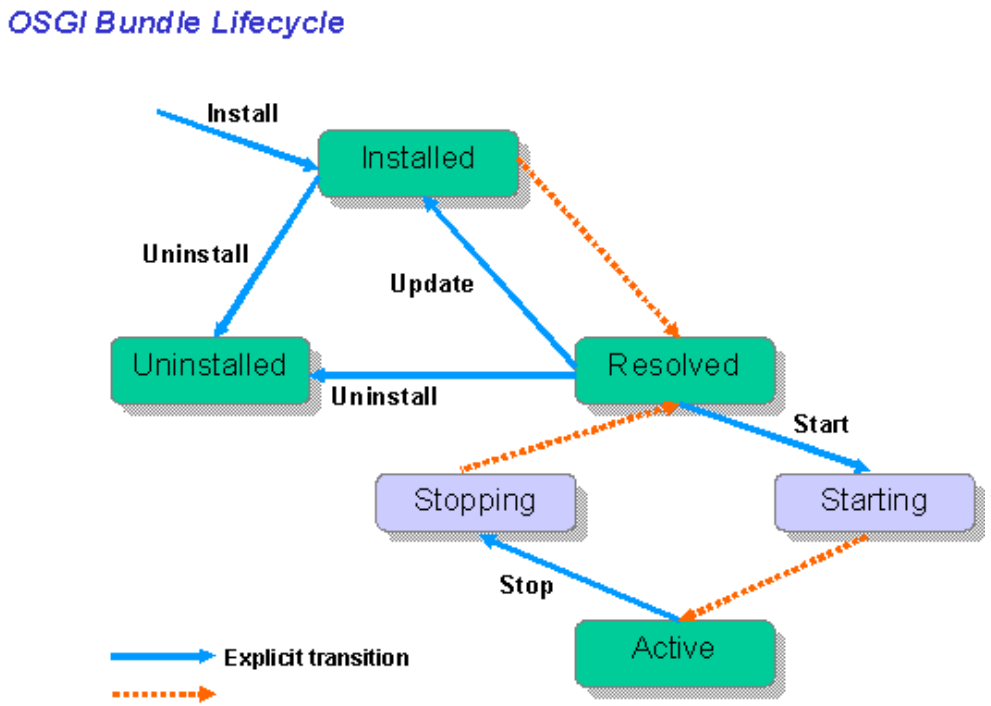
The bundle manifest declares:

- Packages provided by this bundle (static)
- Services provided by this bundle (dynamic)
- Required/optional packages (static dependencies)
- Required/optional services (dynamic dependencies)

Both packages and services have a version associated which is useful for change management

Lifecycle

The framework manages bundles according to a well-defined lifecycle.



Lifecycle

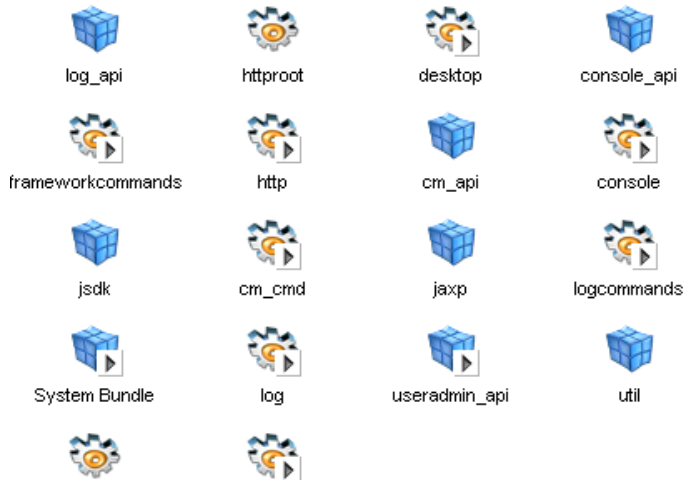
The lifecycle state for each bundle is persistent in case of a system crash or restart

Failover

The framework keeps a timeline of each bundle's lifecycle events for sysadmins

Audit Trail

Bundles can be grouped into “run levels” (like Unix services) and brought up/down in a layered approach



Bundle Id	Name	State	Description	Location	Vendor
2	log_api	resolved	The Knopfle...	log_api-1.0....	Knopflerfish
3	cm_api	resolved	Configuratio...	cm_api-1.0....	Knopflerfish
4	console_api	resolved	Service sys...	console_api...	knopflerfish
5	log	active	The Knopfle...	log-1.0.0.jar	Knopflerfish
6	cm	active	Configuratio...	cm-1.0.0.jar	Knopflerfish
7	console	active	Service sys...	console-1.0....	knopflerfish
8	jsdk	resolved	The servlet j...	jsdk-2.2.jar	knopflerfish
9	consoletty	active	Command li...	consoletty-1...	knopflerfish
10	frameworkc...	active	Framework ...	frameworkc...	Knopflerfish
11	logcommands	active	Log comma...	logcommand...	Knopflerfish
12	useradmin_...	active	User admini...	useradmin_...	Knopflerfish
13	http	active	HTTP Serve...	http_all-1.0....	knopflerfish
14	jaxp	resolved	The Sun JA...	jaxp-1.0.jar	knopflerfish
15	cm_util	resolved	Utilities for ...	cm_util-1.0....	Knopflerfish
16	cm_cmd	active	Command b...	cm_cmd-1.0...	Knopflerfish
17	desktop	active	Swing fram...	desktop_all-...	knopflerfish
18	httproot	active	HTTP root s...	httproot-1.0....	knopflerfish...

Service Registry

- Mule bundles register their available services (connectors, transformers, components, etc.) in the OSGi Service Registry.
- A user's Mule app. then looks up the needed services from the Registry and binds to them.
- Mule services may appear/disappear and consumers must react accordingly.
- This decoupling is what makes hot updates possible.

Spring OSGi

Spring OSGi hides the details of registering and consuming OSGi services, using sensible defaults where possible.

Details may be given in a simple, declarative form within the Spring Beans config file:

- cardinality of dependent services
- handling of ServiceNotAvailable exceptions
- timeouts

Choosing an OSGi Framework

The big 3 Open Source Frameworks are:

- Equinox (a subproject of Eclipse)
- Knopflerfish (Gatespace Telematics)
- Felix (Apache)

Knopflerfish seems to be the best “fit” for Mule in a production environment, but since OSGi is a standard, it should be able to run in any framework.

Looking Forward

- OSGi will be the foundation for future versions of Mule

- It will enable innovative new features for the Mule platform