How Integration works in Motech :

Motech has modular architecture and different module integrate with each using the support from the following software pieces:
-OSGi framework: gives platform to build modular architecture
-Apache Felix: provide implementation of OSGi framework and service platform
-Spring Dynamic Module: services like osgi namespace and Blueprint Container for the easier deployment

Motech takes an advantage of OSGi dynamic execution environment to load and integrate the different modules together. Every module of Motech is the OSGi Bundle. OSGi Framework manages the lifecycle of these bundles, their interaction with each other. The OSGi service platform helps Motech to add, remove these packages on the fly.
Following describes the integration process in the Motech:
1**. Creation of OSGi bundle:**
The packages in the core platforms encapsulate various Open source systems to provide an interface for JMS, Database services, OSGi service interface and other required frequent services. Motech uses the apache.felix's maven-bundle-plugin to automatically creates the OSGi bundle from the compiled jar. A bundle constitutes the .jar file for the given component and Metadata, resources, etc. The maven-resource-plugin copies these bundles into the /user.home/.motech/bundles directory. The META-INF/MANIFEST.MF helps OSGi framework to identify these bundles and dynamically load them. OSGi framework knows how to handle the bundle, export services or import the required dependencies using the metadata from the Manifest.MF.
For instance example the metadata for the HelloWorld motech:
Manifest-Version:
Blueprint-Enabled: true
Bundle-Name: Hello World
Bundle-SymbolicName: org.motechproject.motech-hello-world-tutorial // to get the bundle using the BundleContext interface
Bundle-Version: 0.0.1.SNAPSHOT
Context-File: META-INF/spring/blueprint.xml
Context-Path: helloworld
Export-Package: //packages visible to the other bundles
Import-Package: //specify dependencies on the external packages

2. **Lifecycle of the bundles:**
The following are the modules that take care creating OSGi execution environment, implementing the BundleActivator, BundleContext Interface to manage the lifecycle of different bundle instances:
motech-osgi-platform: It takes care of managing the lifecycle of the Motech core platform bundles.

PlatformActivator: In order to hook the Motech Platforms bundles into the OSGi framework, this class implements the BundleActivator Interface of OSGi framework and overrides the start(), stop() methods.
start() method :
-The method first calls to categorizeBundle() method to divide the bundles in the .motech/bundle library depending on custom BundleType such as Motech_Module, Felix_Framework_bundle , Third_party bundle, etc.
-Next, it registers the listener for the bundles using BundleContext.registerService(). For instance, it records the OsgiBundleApplicationContextListener which listens to the OSGi bundle lifecycle notifications.
- After Registering the required Bundle Listener, it uses the Bundle.start() method to start the bundle instance.
- BundleContext interface uses the symbolic name mentioned in Manifest.MF to dynamically load the module in the OSGi framework
OSGi-web-Utils: It takes care of managing the lifecycle of bundles whose Manifest.MF contain the header Blueprint-Enabled: true. For instance, the lifecycle of HelloWorld bundle is controlled by this package.
BluePrintActivator: This class implement the BundleActivator interface of OSGi framework and override the start(), stop() methods.
start(BundleContext) method tracks the blueprint context (blueprint.xml) using the osgi.web.BlueprintApplicationContextracker. This class takes necessary action to track the application context, verifying bundle's MANIFEST.MF , registering the listener, OSGi services.
stop(BundleContext): stops the bundle instance.
This module exposes the ServerlogService interface as the OSGI service.
MotechRegistrationData: This module enables any module to register within Motech Web UI. For instance, to register HelloWorld module within the Motech Web UI, we configured the bean motechRegistartionData in the Spring context XML(blueprint.xml). It takes two constructor arguments, i.e., module name and URL to instantiate the MotechRegistartionData. The UIFrameworkService.registerModule(MotechRegistartionData) method is used to register the module under Motech web UI. HelloWorld/ any module can use this service.

How OSGi Service Layer works:
OSGi framework provides centralized service registry that follows the publish-find-bind model. Motech uses the BundleContext Interface to access the OSGi services registry. BundleContext interface provides the various method to register, add, or remove services. Any Motech OSGi bundle who want to publish their services, express their services using BundleContext Interface. For instance, the Motech-platform-event bundle publish services for the other bundle to register a listener for the Motech Events, create eventListener using EventListener interface and interface to use the ActiveMQ topic or queue using the EventRelay interface. It provides the list of EXPORT-Packages in the Meta-INF/MANIFEST.IF. It enables the OSGi framework to identify the export packages that can be used by other bundles. The tag <osgi:service> register the mentioned interfaces as OSgi services. Then, we use osgi.framework BundleContext.regiserService() method to register these services in the service registry.

Other bundles such as HelloWord can import these packages to use the published services. The Import-packages list in the Manifest.IF tells OSGi framework to bind these packages at runtime. Also in spring resource configuration file we add the tag <osgi:referrence> to access the OSGi service registry.
For instance , we added following tag in the context XML.
<osgi:reference id="eventRelayOsgi" interface="org.motechproject.event.listener.EventRelay"/>
Thus, the Motech bundles can integrate, communicate with each other using the service-oriented interface.