# History

<http://gnodet.blogspot.ru/2010/12/thoughts-about-servicemix.html>

I’ve been involved in ServiceMix, the Open Source ESB, nearly since its inception. ServiceMix was started by [James Strachan](http://macstrac.blogspot.com/) and [Rob Davies](http://rajdavies.blogspot.com/) in May 2005 and the 1.0 version was released in August 2005. I joined LogicBlaze in October 2005 (leaving Mule behind, as I was a committer on Mule) to work on the 2.x releases (2.0 was released in November 2005) and then on the 3.x after the move from Codehaus to the Apache Software Foundation (3.0 was released in May 2007) and the 4.x versions based on OSGi (4.0 was released in March 2009). Even though I’m now focusing more on the OSGi side now (being the VP of Karaf), I’ve done my share of work on ServiceMix (I’m still the first committer in terms of number of commits according to http://svnsearch.org/svnsearch/repos/ASF/search?path=/servicemix) and I’ve been the VP of ServiceMix at the ASF for a few years.  
  
ServiceMix started as a very lightweight implementation of JBI spec. The 2.x version brought much more JBI compliance and the 3.x has seen migration from the lightweight component to fully fledge JBI components and full JBI compliance. In doing so, ServiceMix became a container, as required per the JBI spec and over-time lost a bit of its lightweight-ness. That’s exactly at the same time that Camel was created as a routing library, based on the experience with ServiceMix. After the 3.0 was released it became apparent that the JBI spec was way too limited with respect to the container (classloaders and even the packaging are a real pain in JBI), so we started experimenting with OSGi at that time and that led the path to ServiceMix 4.x and the Karaf project, which started as ServiceMix Kernel.  
  
In 2007, the SCA spec came out, backed by IBM and Oracle, and during a few months, there were a lot of heated debate around JBI versus SCA. It eventually settled a bit when people started understanding that those specs were not really competing, as JBI was targeted around components interoperability while SCA target was application development and could be built on top of JBI. However, JBI was not backed by the big vendors (only really SUN), and when the spec lead for JBI 2.0 left SUN with no replacement, it became clear that the JBI spec was dead.  
  
Another point is that over time, Camel grew to become a top level project and be the very successful project we know, and for some time, we did not really know what to do with the overlap between Camel components and ServiceMix components.   
  
Since JBI 2.0 doesn't appear to be going anywhere we realised we should focus on Camel for the EIP support and connectivity and that OSGi was a better long term standard to represent the registry of endpoints, so we've refactored ServiceMix NMR to be more lightweight, based on the lightweight OSGi container and based around the OSGi registry; with JBI support an optional legacy connector. So we now have a simple lightweight approach to providing a middleware agnostic registry of endpoints with full hot-deploy and supporting powerful class loader versioning schemes via the OSGi registry.   
  
Long live ServiceMix, Camel and OSGi!

# Java Business Integration

From Wikipedia, the free encyclopedia <http://en.wikipedia.org/wiki/Java_Business_Integration>

## [Is JBI dead? Not of course !!!](http://www.pymma.com/index.php?option=com_content&view=article&id=127:is-jbi-dead-not-of-course&catid=82:blog-paul-perez)

**Java Business Integration** (**JBI**) is a specification developed under the [Java Community Process](http://en.wikipedia.org/wiki/Java_Community_Process) (JCP) for an approach to implementing a [service-oriented architecture](http://en.wikipedia.org/wiki/Service_oriented_architecture) (SOA). The JCP reference is JSR 208 for JBI 1.0 and JSR 312 for JBI 2.0.

JBI is built on a [Web Services](http://en.wikipedia.org/wiki/Web_Service) model and provides a pluggable architecture for a container that hosts service producer and consumer components. Services connect to the container via binding components (BC) or can be hosted inside the container as part of a service engine (SE). The services model used is [Web Services Description Language 2.0](http://en.wikipedia.org/wiki/Web_Services_Description_Language). The central message delivery mechanism, the normalized message router (NMR), delivers normalized messages via one of four [Message Exchange Patterns](http://en.wikipedia.org/wiki/Message_Exchange_Pattern) (MEPs), taken from WSDL 2.0:

## Normalized Message Router

<http://fusesource.com/docs/esb/4.4/jbi/ESBJBINMR.html>

*The normalized message router is a bus that shuttles messages between the endpoints deployed on the ESB.*

## Overview

The normalized message router(NMR) is the part of the JBI environment that is responsible for mediating messages between JBI components. The JBI components never send messages directly to each other; instead, they pass messages to the NMR, which is responsible for delivering the messages to the correct JBI endpoints. This allows the JBI components, and the functionality they expose, to be location independent. It also frees the application developer from concerns about the connection details between the different parts of an application.

## Message exchange patterns

The NMR uses a WSDL-based messaging model to mediate the message exchanges between JBI components. Using a WSDL-based model provides the necessary level of abstraction to ensure that the JBI components are fully decoupled. The WSDL-based model defines operations as a message exchange between a service provider and a service consumer. The message exchanges are defined from the point of view of the service provider and fit into one of four message exchange patterns:

# Что такое OSGi

<http://samolisov.blogspot.ru/2009/03/osgi.html>

Начнем рассмотрение снизу вверх. Начиная с версии 3.0 Eclipse перевели на новую основу. Имя ей - OSGi.  
  
[OSGi (Open Services Gateway Initiative)](http://osgi.org/) - спецификация динамической [плагинной (модульной) шины](http://samolisov.blogspot.com/2008/04/guice.html) для создания Java-приложений, разрабатываемая консорциумом [OSGi Alliance](http://osgi.org/" \t "_blank). Круг применений данной спецификации довольно широк: изначально разрабатывалась для создания встроенных систем (в частности, для автомобилей BMW, также в разработке спецификации активно учавствует Siemens), но сейчас на базе OSGi строят многофункциональные десктоп приложения (например, Eclipse SDK) и Enterprise-системы (например, Naumen DMS). Последняя версия спецификации носит номер 4.1 и доступна [вот здесь](http://www.osgi.org/Specifications/HomePage?section=2#Release4).  
  
  
Основная идея фреймворка OSGi - все в системе есть **плагины** (в терминах OSGi - **бандлы (bundles)**). Основной способ взаимодействия между бандлами - сервисы: объекты, зарегистрированные в ядре системы с заявленными реализованными интерфейсами. Бандлы регистрируют сервисы для предоставления определенной функциональности другим бандлам. Помимо этого OSGi предоставляет механизм создания и обработки событий, управление импортом/экспортом java-пакетов и библиотек, набор[класслоадеров](http://samolisov.blogspot.com/2008/01/java.html), методы адресации ресурсов.