Spoilt for Choice – Which Integration Framework to choose?
Main Tasks

- Evaluation of Technologies and Products
- Requirements Engineering
- Enterprise Architecture Management
- Business Process Management
- Architecture and Development of Applications
- Planning and Introduction of SOA
- Integration of Legacy Applications
- Cloud Computing

Contact

- Email: kai.waehner@mwea.de
- Blog: www.kai-waehner.de/blog
- Twitter: @KaiWaehner
- Social Networks: Xing, LinkedIn

Consulting
Developing
Speaking
Coaching
Writing
What is the Key Message?
Do not reinvent the „integration wheel“!
Do not reinvent the „integration wheel“!
There are some good alternatives for Integration!
Key Messages

Do not reinvent the „integration wheel“!
There are some good alternatives for Integration!
Often an ESB is the wrong Choice!
1) Systems Integration

2) Integration Frameworks

3) Spring Integration

4) Mule ESB

5) Apache Camel

6) And the Winner is ...
1) Systems Integration

2) Integration Frameworks

3) Spring Integration

4) Mule ESB

5) Apache Camel

6) And the Winner is ...
What is the Problem?

Growth

- Applications
- Interfaces
- Technologies
- Products
NEW PARADIGM AHEAD
Everybody communicates to everybody
All Roads lead to Rome ...
Alternatives for Systems Integration

- **Low Complexity of Integration**
  - No Tool
  - Integration Framework

- **High Complexity of Integration**
  - Enterprise Service Bus (ESB)
Alternatives for Systems Integration

- No Tool
- Integration Framework (Low Complexity)
- Enterprise Service Bus (ESB) (High Complexity)

Spring Integration vs. Mule ESB vs. Apache Camel
Agenda

1) Systems Integration

2) Integration Frameworks

3) Spring Integration

4) Mule ESB

5) Apache Camel

6) And the Winner is ...
Wishes

• Standardized Modeling
• Efficient Realization
• Automatic Testing
Enterprise Integration Patterns (EIP)
Relation between EIPs and Integrations Frameworks

Integration

spring

vs. Mule ESB

vs.

Apache Camel

implement
Comparison Criteria

- Open source
- Basic concepts / architecture
- Testability
- Commercial support
- Error handling
- Monitoring
- Enterprise readiness
- Developer-centric vs. designer-centric
- Expandability
- Deployment
- Popularity
- Tool support
- Connectivity
- Domain specific language (DSL)
Similarities
Integration of different Technologies
„A domain-specific language (DSL) is a programming language or specification language dedicated to a particular problem domain, a particular problem representation technique, and / or a particular solution technique.“

Wikipedia
(Exemplarily: Apache Camel => Concepts are all the same, only different names)
Deployment

Standalone  Application  Server
Web Container

Spring Container

OSGi

Cloud
Enterprise-Ready

• Maturity
• Transactions
• Concurrency
• Error handling
• Monitoring
• Testability
Integration vs. Mule ESB vs. Camel

- Spring
  - Apache License
- Mule ESB
  - Common Public Attribution License (CPAL)
- Camel
  - Apache License
Integration vs. Mule ESB vs. Apache Camel
Tool Support

Integration Graph for Spring Integration

Mule Studio for Mule ESB

Fuse IDE for Apache Camel

Talend Studio for Apache Camel
Upcoming Procedure

- Concepts of each Framework
- Code Example
- Live Demo
Comparison Criteria

- Open source
- Basic concepts / architecture
- Testability
- Commercial support
- Error handling
- Monitoring
- Enterprise readiness
- Developer-centric vs. designer-centric
- Expandability
- Deployment
- Popularity
- Tool support
- Connectivity
- Domain specific language (DSL)

Focus
1) Systems Integration

2) Integration Frameworks

3) Spring Integration

4) Mule ESB

5) Apache Camel

6) And the Winner is ...
How to create own custom adapter?

Spring Integration Samples / INTSAMPLES-73
There is no sample how to create an own custom adapter

Details
- Type: Task
- Priority: Minor
- Affects Version/s: None
- Component/s: None
- Labels: None
- Status: Open
- Resolution: Unresolved
- Fix Version/s: None
- Security Level: Public

Spring Forum Reference:
XML

(Not production-ready yet)
<file:inbound-channel-adapter
    id="incomingOrders"
    directory="file:incomingOrders"/>

<payload-type-router input-channel="incomingOrders">
    <mapping type="com.kw.DvdOrder"
        channel="dvdOrders" />
    <mapping type="com.kw.VideogameOrder"
        channel="videogameOrders" />
    <mapping type="com.kw.OtherOrder"
        channel="otherOrders" />
</payload-type-router>

<file:outbound-channel-adapter
    id="dvdOrders"
    directory="dvdOrders"/>

<jms:outbound-channel-adapter
    id="videogamesOrders"
    destination="videogameOrdersQueue"
    channel="videogamesOrders"/>

<logging-channel-adapter id="otherOrders" level="INFO"/>
val messageFlow =
  filter.using{payload: String => payload == "World"} -->
  transform.using { payload: String => "Hello " + payload} -->
  handle.using { payload: String => println(payload) }

messageFlow.send("World")

http://blog.springsource.org/2012/03/05/introducing-spring-integration-scala-dsl/
Spring Integration in Action
Pro
• Visual Designer for Spring Integration flows
• Vice versa Editing (code generation vs. coding by hand)
• Open Source (part of Spring IDE at github)
• Simple Eclipse plugin – „Just Spring Integration“
• Export of flow diagrams

Contra
• Non-intuitive usability
• Unclear diagrams
• Immature (e.g. missing documentation, problems with code examples)
1) Systems Integration

2) Integration Frameworks

3) Spring Integration

4) **Mule ESB**

5) Apache Camel

6) And the Winner is ...
Embedding Mule in a Java Application or Webapp

This page describes how to start and stop Mule from a Java application or to embed it in a Webapp (such as a JSP or servlet), and how to interact with Mule from your code in both scenarios.

Starting Mule from a Java Application

To start Mule from any Java application, you can call one of its configuration builders. To use Mule XML configuration:

```java
DefaultMuleContextFactory muleContextFactory = new DefaultMuleContextFactory();
SpringXmlConfigurationBuilderFactory configBuilderFactory = new SpringXmlConfigurationBuilderFactory("mule-context.xml");

muleContext = muleContextFactory.createMuleContext(configBuilderFactory);
```

If you have multiple configuration files, you can provide a comma-separated list or an array of configuration files:

```java
SpringXmlConfigurationBuilderFactory configBuilderFactory
    new SpringXmlConfigurationBuilderFactory(new String[] { "mule-config.xml", "another-config.xml" });
```

You then call the start method to start the server:

```java
muleContext.start();
```

Stopping Mule from a Java Application

To stop Mule, you stop its context like this:

```java
muleContext.stop();
muleContext.dispose();
```

http://www.mulesoft.org/documentation/display/MULE3USER/Embedding+Mule+in+a+Java+Application+or+Webapp
### Connectivity

<table>
<thead>
<tr>
<th>AS400 Data Queue</th>
<th>Abdera</th>
<th>Amazon SQS</th>
<th>Amazon S3</th>
<th>Authorize.net</th>
<th>JCR</th>
</tr>
</thead>
<tbody>
<tr>
<td>jBPM</td>
<td></td>
<td>CICS CTG</td>
<td></td>
<td>Apple Push</td>
<td>JDBC</td>
</tr>
<tr>
<td>CXF</td>
<td></td>
<td>Email</td>
<td></td>
<td>Bit.ly</td>
<td>Jersey</td>
</tr>
<tr>
<td>FTP</td>
<td></td>
<td>FTP</td>
<td></td>
<td>CMIS</td>
<td>Jetty</td>
</tr>
<tr>
<td>Hibernate</td>
<td></td>
<td>HTTP/S</td>
<td></td>
<td>CyberSource</td>
<td>JMS</td>
</tr>
<tr>
<td>Legs4Mule</td>
<td></td>
<td>IMAP/S</td>
<td></td>
<td>Facebook</td>
<td>LDAP</td>
</tr>
<tr>
<td>IMAP/S</td>
<td></td>
<td></td>
<td></td>
<td>Flickr</td>
<td>Multicast</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>HBase</td>
<td>POP3/S</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Magento</td>
<td>Quartz</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RMI</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SAP</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Atom</th>
<th>Base64 encoded</th>
<th>Byte arrays</th>
<th>CSV</th>
<th>Encrypted</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GZIP</td>
<td>Hex Strings</td>
<td>HTML/ XHTML</td>
<td>Java Objects</td>
</tr>
<tr>
<td></td>
<td>JSON</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EDI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>COBOL Copybook</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>XML</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Many more Connectors +

easy to create own Connectors
Several Proprietary Connectors available, for instance:
There have been bubbles of interest about OSGi in the Java community over recent years. I for one was very excited about the advent of a modular Java platform that freed us from the classloader issues in the JDK manifested best by Jakarta commons-logging (logging our app servers).

OSGi was going to change everything, dependencies would be completely isolated (no more tripping over conflicting dependency versions), visibility and would be strictly enforced between each ‘bundle’. I was so bought into the promise of OSGi, like many others, I focused our engineering team on making Mule OSGI-enabled.

„OSGi adds another complexity to building applications. [...] OSGi is a great specification for middleware vendors, but a terrible specification for the end user.“

Ross Mason, MuleSoft, November 2010
XML
<flow name="muleFlow">
  <file:inbound-endpoint path="incomingOrders"/>
  <choice>
    <when expression="payload instanceof com.kw.DvdOrder"
      evaluator="groovy">
      <file:outbound-endpoint path="incoming/dvdOrders"/>
    </when>
    <when expression="payload instanceof com.kw.DvdOrder"
      evaluator="groovy">
      <jms:outbound-endpoint queue="videogameOrdersQueue"/>
    </when>
    <otherwise>
      <logger level="INFO"/>
    </otherwise>
  </choice>
</flow>
Live Demo

Mule in Action
**Pro**

- Visual Designer for Mule Flows
- Visual „Live Monitoring“
- Vice versa Editing (Code generation vs. coding by hand)
- Simple Eclipse plugin – „Just Mule“
- Intuitive GUI

**Contra**

- Proprietary
- Subscription required for many enterprise features (such as monitoring)
1) Systems Integration
2) Integration Frameworks
3) Spring Integration
4) Mule ESB
5) Apache Camel
6) And the Winner is ...
DSLs

XML

Java

Groovy

Scala
Many more Components + easy to create own Components
<route>
  <from uri="file:incomingOrders"/>
  <choice>
    <when>
      <simple>${in.header.type} is 'com.kw.DvdOrder'</simple>
      <to uri="file:incoming/dvdOrders"/>
    </when>
    <when>
      <simple>${in.header.type} is 'com.kw.VideogameOrder'</simple>
      <to uri="jms:videogameOrdersQueue"/>
    </when>
    <otherwise>
      <to uri="log:OtherOrders"/>
    </otherwise>
  </choice>
</route>
from("file:incomingOrders ")
    .choice()
    .when(body().isInstanceOf(com.kw.DvdOrder.class))
        .to("file:incoming/dvdOrders")
    .when(body().isInstanceOf(com.kw.VideoGameOrder.class))
        .to("jms:videoGameOrdersQueue ")
    .otherwise()
        .to("mock:OtherOrders ");
Apache Camel in Action
Fuse IDE

vs.

Talend Open Studio for ESB
Fuse IDE

Diagram showing a flowchart with components such as 'choice', 'when', and 'otherwise', connected with arrows and labeled 'filetarget/messa...'.
Pro
• Visual Designer for Camel routes
• Visual „Live Monitoring“ (for debugging, browsing, tracing)
• Vice versa Editing (code generation vs. coding by-hand)
• Just a simple Eclipse-Plugin – „Just Camel“
• Intuitive GUI
• Export of route diagrams
• JUnit Test Wizard (generates scaffolding for tests)

Contra
• Proprietary
• Subscription required
• Only XML DSL (this is not really a contra, because only Camel offers other DSLs)
Talend Open Studio for ESB
Pro
• Visual Designer for Camel routes
• Visual „Live Monitoring“
• Open Source (at github)
• Community Edition (not all features)
• Zero Coding

Contra
• Zero Coding
• Only Java DSL (plus a lot of boilerplate code)
• No vice versa code editing (only code generation)
• No intuitive user interface
  => no simple Eclipse plugin
  => based on Eclipse, but it is more an own product (1GB)
  => full ESB only
1) Systems Integration
2) Integration Frameworks
3) Spring Integration
4) Mule ESB
5) Apache Camel
6) And the Winner is ...
And the Winner is ...

Integration

Mule ESB

Apache Camel
When to use?

Integration

Mule ESB

Apache Camel
When to use?

- Spring Project
- “Typical“ JVM Technologies
- No additional Framework
When to use?

- One of the available proprietary Connectors is required
When to use?

- In all other Cases
Alternatives for Systems Integration

- **No Tool**
- **Integration Framework**
- **Enterprise Service Bus (ESB)**

The complexity of integration ranges from low to high.
Alternatives for Systems Integration

Integration Framework

- No Tool
- Low Complexity of Integration

Enterprise Service Bus (ESB)

Integration
- Connectivity
- Routing
- Transformation

High Complexity of Integration
Alternatives for Systems Integration

- Integration Framework
- Enterprise Service Bus (ESB)

Low vs. High Complexity of Integration

Integration Connectivity Routing Transformation

Spring Integration vs. Mule ESB vs. Apache Camel
Alternatives for Systems Integration

- No Tool
- Integration Framework
- Enterprise Service Bus (ESB)

Low Complexity of Integration to High Complexity of Integration
Alternatives for Systems Integration

No Tool

Integration Framework

Enterprise Service Bus (ESB)

Integration Connectivity Routing Transformation

Complexity of Integration

Low

High

Middleware Product

Alternatives for Systems Integration

Complexity of Integration

Low

No Tool

Integration Framework

High

Enterprise Service Bus (ESB)

Middleware Product

Integration

Connectivity

Routing

Transformation

Integration

Business Process Management

Business Activity Monitoring

Registry / Repository

Rules Engine

some use
Alternatives for Systems Integration

Enterprise Service Bus (ESB)

Integration Framework

No Tool

Low

High

Complexity of Integration

Middleware Product

Integration

Business Process Management

Business Activity Monitoring

Registry / Repository

Rules Engine

Apache ServiceMix, Mule ESB, WSO2 ESB vs.
IBM Message Broker, IBM ESB, Oracle ESB, webMethods ESB
Did you get the Key Messages?
Do not reinvent the „integration wheel".
There are some good alternatives for Integration.
Often an ESB is the wrong Choice.
Did you get the Key Messages?
Become a Part of the Open Source Community
Thank you for your Attention. Any Questions?

Kai Wähner

MaibornWolff et al:
www.mwea.de
Email: kai.waehner@mwea.de
Twitter: @KaiWaehner

Blog: www.kai-waehner.de/blog

integration or Mule ESB or Camel = Smart EAI