Spring & Mule

Andrea Bozzoni
andrea.bozzoni@gmail.com

Contenuti

Panoramica su un ESB (Enterprise Service Bus)

Cos'è

- ESB è un'architettura che consente l'integrazione di applicazioni di business attraverso l'utilizzo di un bus di messaggistica condiviso

Perchè ne abbiamo bisogno

- La maggior parte dei progetti oggi sono basati sull'integrazione di applicazioni eterogenee

SOA

- ESB supporta le regole della Service Orientated Architecture
  - Utilizzo dei servizi attraverso l'uso di interfacce indipendenti dall'implementazione
  - Uitlizzo di protocolli di comunicazione che consentono l'iteroperatibilità e la trasparenza della locazione fisica dei servizi
  - Definizione di servizi che espongono funzioni di business nuove o già esistenti

La novità?

- Apparentemente non c'è nulla di nuovo! Soluzioni di integrazione senza ESB esistono e sono funzionanti!

Panoramica su un ESB (Enterprise Service Bus)

- **Basso accoppiamento**
  - Le applicazioni dialogano attraverso un bus comune basato su un sistema di messagistica
- **Event-Driven**
  - Le applicazioni generano eventi che avviano dei processi nell'ESB
- **Highly Distributed**
  - Un ESB può essere distribuito su più nodi
- **Security/Authorization**
  - Supportano il concetto di autenticazione e autorizzazione
- **Abstract Endpoints**
  - Rappresentano dei punti d'ingresso attraverso i quali le applicazioni posso spedire e ricevere messaggi nel o dal sistema
- **Routing intelligente**
  - Ogni evento in ingresso o in uscita può essere distribuito secondo regole configurabili
- **Trasformazione dei dati (in ingresso e uscita)**
  - I dati in ingresso e in uscita possono essere trasformati per gestire l'impedenza fra le applicazioni
- **Multi-Protocol Message Bus**
  - Le applicazioni possono accedere attraverso diversi protocolli (HTTP, SOAP,JMS, File, Email)
- **Light Weight**

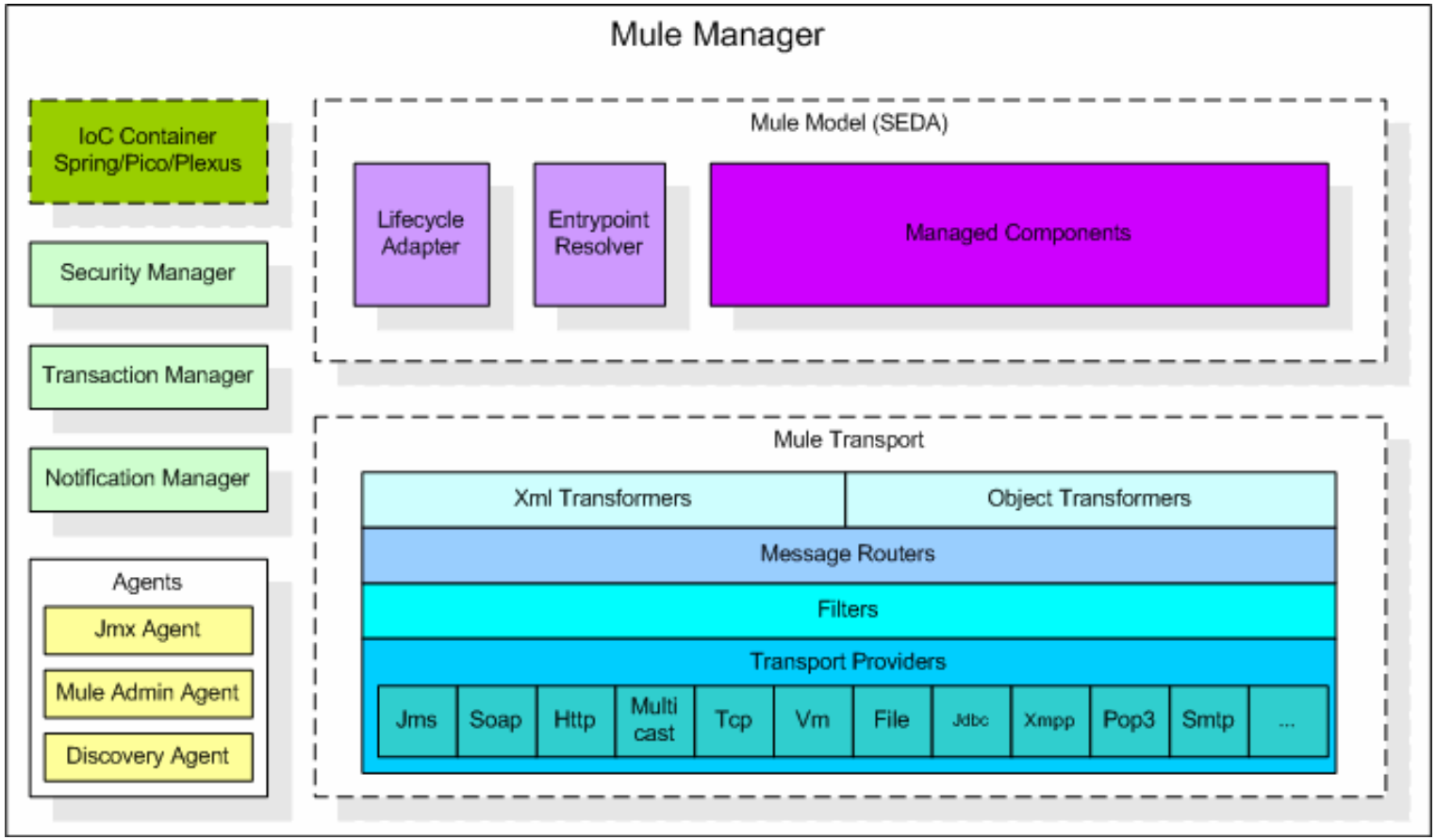
Introduzione a MULE

- Piattaforma di messaggistica basata sul concetto di Enterprise Sevice Bus
- Supporta diverse topologie oltre l'ESB
  - Pipeline
  - Client Server
  - Hub&Spoke
  - Peer2Peer
- Utilizza un modello di tipo SEDA altamente scalabile
- Service container (Object Broker)
- Supporta qualsiasi tipo di componenti (UMO): POJO, EJB, Spring beans
- Integrabile con diversi conteiners
  - Spring
  - Pico Cntainer
  - HiveMind
  - Plexus

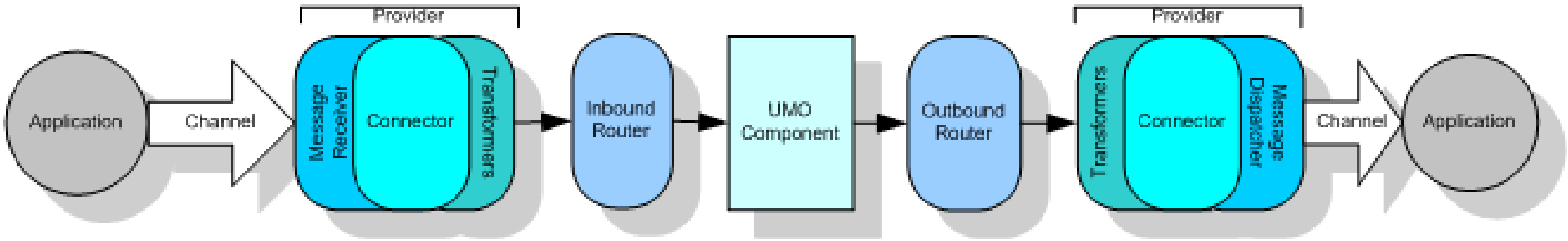
Introduzione a MULE

- Supporta molti protocolli di messagistica e di trasporto
- Gestisce parecchie tipologie di routing degli eventi
- Trasforma I dati sia in ingresso che in uscita
- Supporta un sistema di autenticazione/autorizzazione (attraverso Spring/Acegi)
- E' Leggero ed embeddable
- Utilizzabile in diversi contesti e in diverse modalità:JAR, WAR, EAR, RAR)
- Instrumentabile attraverso JMX (Consolle di gestione e visualizzazione del suo stato)
- E' adattativo rispetto alla tecnologia che lo circonda invece che imporre la sua

Introduzione a MULE



Introduzione a MULE



Integrazione tra Spring e Mule

- **Spring come component Factory**
- **Configurare Mule in un constesto di Spring**
- **Configurare un contesto Spring attraverso la configurazione di Mule**
- **Gestione degli eventi tra Mule e Spring**

Spring come component factory

Questa è la prima modalità d'integrazione tra Mule e Spring. In questo caso il Mule manager viene configurato attraverso il suo formato di configurazione, specificando il riferimento ad un Application Context è possibile utilizzare i beans in esso definiti ed utilizzarli come componenti di Mule.

Per questo è sufficiente indicare a Mule che deve utilizzare un ApplicationContext esterno

```
<mule-configuration id="Default" version="1.0">
      <description>Spring</description>
   <container-context
     className="org.mule.extras.spring.SpringContainerContext">
       <properties>
          <property name="configFile"
              value="conf/springSimpleApplicationContext.xml"/>
          </properties>
   </container-context>
...
```

SpringContainerContext è uno Spring Context che può esporre degli spring managed components per l'utilizzo in Mule

Spring come component factory

La differenza nella dichiarazione di un UMO nel file di configurazione di Mule

- **senza l'utilizzo di un container esterno**

```
<mule-descriptor name="service"
        class="it.bozzoni.spring.services.BusinessServicesImpl">
        <inbound-router>
...
```

- **attraverso l'utilizzo di un container esterno come Spring**

```
<mule-descriptor name="service"  implementation="businessService">
      <inbound-router>
...
```

Spring come component factory

L'attributo id del bean di Spring deve corrispondere all'attributo implementation della definizione del componente in Mule

```
<bean id="auditService"
  class="it.bozzoni.spring.audit.AuditServicesImpl">
</bean>

<bean id="businessService"
    class="it.bozzoni.spring.services.BusinessServicesImpl"
    scope="prototype">
  <property name="auditService">
    <ref local="auditService" />
  </property>
</bean>
...
<mule-descriptor name="service" implementation="businessService">
  <inbound-router>
     <endpoint address="vm://myComponent" synchronous="true"/>
  </inbound-router>
</mule-descriptor>
```

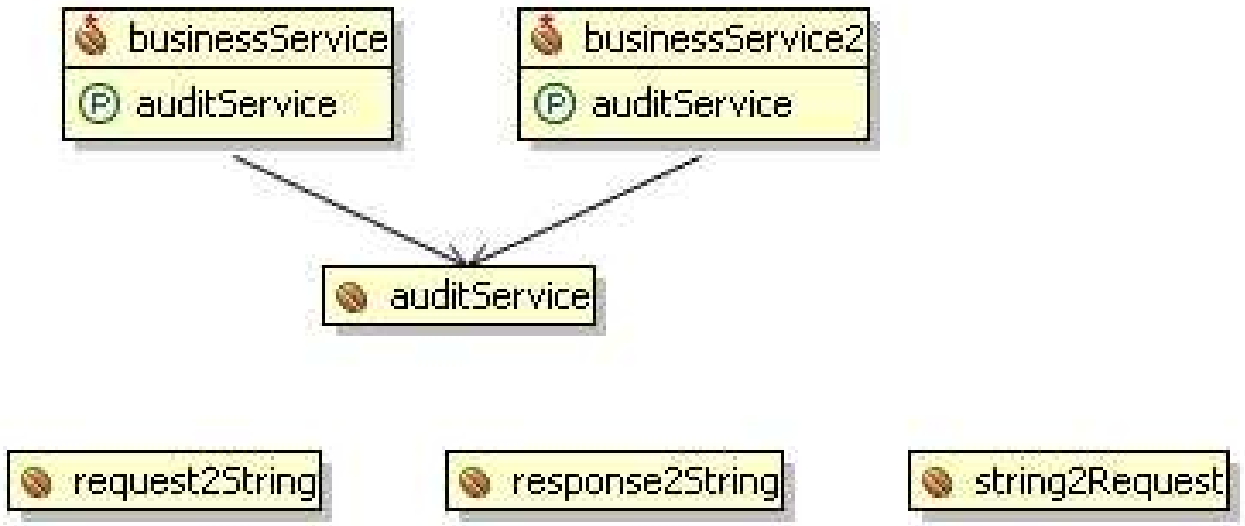
Spring come component factory

E' possibile definire nell'application context un qualsiasi componente di Mule.
In questo caso abbiamo dichiarato un transformer che può essere utilizzato dagli endpoints

```
<bean id="string2Request"
   class="it.bozzoni.mule.transformers.String2Request">
</bean>
...
<transformers>
  <transformer name="String2Request" ref="string2Request"/>
</transformers>
...
<mule-descriptor name="service2" implementation="businessService">
  <inbound-router>
    <endpoint address="vm://myComponent2" remoteSync="true"
        transformers="String2Request"/>
  </inbound-router>
...
```

Spring come component factory

Esempio di application context in cui sono dichiarati due UMO e tre transformers.



Spring come component factory

Il Mule manager viene inizializzato indicando dov'è il file di configurazione con due semplici istruzioni

```
MuleXmlConfigurationBuilder builder = null;
UMOManager manager = null;
try
{
	builder = new MuleXmlConfigurationBuilder();
	manager = builder.configure(getConfigResources());
}
catch (org.mule.config.ConfigurationException e)
{
	// TODO Auto-generated catch block
	e.printStackTrace();
}
```

Integrazione tra Spring e Mule

- Spring come component Factory
- Configurare Mule in un constesto di Spring
- Configurare un contesto Spring attraverso la configurazione di Mule
- Gestione degli eventi tra Mule e Spring

Configurare Mule in un contesto Spring

E' possibile configurare un Mule manager direttamente come managed beans in un application context di Spring

Per questo Mule mette a disposizione due FactoryBean:

- **org.mule.extras.spring.config.UMOManagerFactoryBean**
- **org.mule.extras.spring.config.AutowireUMOManagerFactoryBean**

Il primo è stato deprecato poichè richiedeva il wiring esplicito di tutte le proprietà del Mule manager dando luogo ad application contest eccessivamente verbosi.
Il secondo effettua l'auto-wiring in funzione di quello che è disponibile all'interno dell'application context. E' molto utile in quanto riduce la verbosità dell'application context.

Configurare Mule in un contesto Spring

org.mule.extras.spring.config.AutowireUMOManagerFactoryBean

```
...
<beans>
  <bean id="muleManager"
    class="org.mule.extras.spring.config.AutowireUMOManagerFactoryBean" />

  <bean id="muleNameProcessor"
    class="org.mule.extras.spring.config.MuleObjectNameProcessor" />
...
```

Questa factory viene dichiarata come bean nell'application context e viene utilizzata da Mule come factory per la configurazione del manager in funzione di ciò che è definito all'interno del contesto

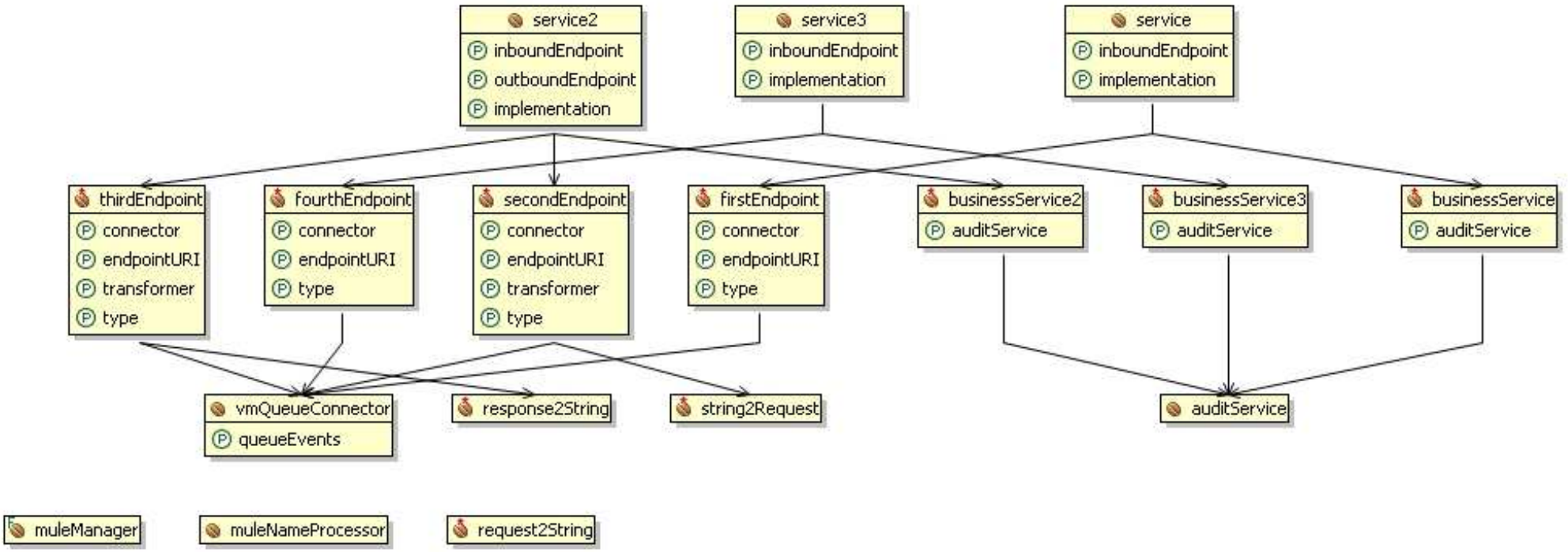
Configurare Mule in un contesto Spring

Ecco un esempio di dichiarazione di un UMO descriptor che viene utilizzato da Mule per inizializzare un UMO

```
<bean id="service" class="org.mule.impl.MuleDescriptor">
  <property name="inboundEndpoint">
      <ref local="firstEndpoint" />
  </property>
  <property name="implementation">
      <ref local="businessService"/>
  </property>
</bean>
...
<bean id="businessService"
  class="it.bozzoni.spring.services.BusinessServicesImpl"
  singleton="false">
  <property name="auditService">
      <ref local="auditService" />
  </property>
</bean>
...
```

Configurare Mule in un contesto Spring

Esempio di application context in cui è configurata una istanza di mule, una serie di UMO e una serie di transformers



Configurare Mule in un contesto Spring

Il Mule manager viene inizializzato indicando dov'è il file di configurazione con due semplici istruzioni. Notare che in questa circostanza è cambiato il builder

```
SpringConfigurationBuilder builder = null;
UMOManager manager = null;
try
{
   builder = new SpringConfigurationBuilder();
   manager = builder.configure(getConfigResources());
}
catch (org.mule.config.ConfigurationException e)
{
  // TODO Auto-generated catch block
  e.printStackTrace();
}
```

Integrazione tra Spring e Mule

- Spring come component Factory
- Configurare Mule in un constesto di Spring
- Configurare un contesto Spring attraverso la configurazione di Mule
- Gestione degli eventi tra Mule e Spring

Configurare un contesto Spring attraverso la configurazione di Mule

In questo caso l'application context è implicitamente definito nella configurazione di Mule.

Occorre dichiarare un differente DOCTYPE nella configurazione di Mule:

```
<!DOCTYPE mule-configuration PUBLIC
        "-//MuleSource //DTD mule-configuration XML V1.0//EN"
       "http://mule.mulesource.org/dtds/mule-spring-configuration.dtd">
```

Configurare un contesto Spring attraverso la configurazione di Mule

E' quindi possibile sfruttare le capacità di wiring di Spring direttamente nella configurazione di Mule

```
<mule-descriptor name="service"
implementation="it.bozzoni.spring.services.BusinessServic
esImpl">
  <inbound-router>
    <endpoint address="vm://myQueue" synchronous="true"/>
  </inbound-router>
  <properties>
     <spring-property name="auditService">
        <ref local="auditService"/>
     </spring-property>
   </properties>
</mule-descriptor>
```

In questo caso dichiarando una spring-property iniettiamo una dipendenza su un UMO dichiarato in Mule.

Configurare un contesto Spring attraverso la configurazione di Mule

In realtà Mule all'avvio trasforma il file della sua configurazione in un application context vero e proprio:

```
<bean class="org.mule.impl.MuleDescriptor" name="service">
  <property name="implementation">
    <value>it.bozzoni.spring.services.BusinessServicesImpl
    </value>
  </property>
  <property name="containerManaged">
    <value>true</value>
  </property>
  <property name="properties">
    <map>
      <entry key="auditService">
        <ref local="auditService" />
      </entry>
    </map>
  </property>
```

...

Integrazione tra Spring e Mule

- Spring come component Factory
- Configurare Mule in un constesto di Spring
- Configurare un contesto Spring attraverso la configurazione di Mule
- Gestione degli eventi tra Mule e Spring

Gestione degli eventi tra Mule e Spring

Spring offre un sistema per la pubblicazione/ricezione di eventi basato sul pattern Observer.
In questo modo è possibile per un bean sia:

- ricevere eventi dal container (applicativi e/o generati dal container)
- pubblicare eventi nel container.

Affinchè un bean riceva eventi dal container occorre che implementi l'interfaccia :

- **org.springframework.context.ApplicationListener**

Mentre per la pubblicazione è sufficiente avere a disposizione una reference all'application context. Sarebbe, comunque, più elegante implementare l'interfaccia:

- **org.springframework.context.ApplicationEventPublisherAware**

Gestione degli eventi tra Mule e Spring

Per la gestione degli eventi all'avvio l'application context effettua le seguenti operazioni:

- verifica se nella configurazione è presente un bean il cui attirbuto id o name vale '**applicationEventMulticaster**'
- Se è presente lo utilizza come suo broadcaster di eventi, altrimenti ne stanzia uno di default: SimpleEventMulticaster
- Memorizza tutti i beans che implementano l'interfaccia ApplicationListener come subscribers nell'applicationEventMulticaster

Quando viene pubblicato un evento nel container l'application context effettua le seguenti operazioni:

- chiede all' event multicaster la lista dei subscribers
- notifica a ciascuno di essi l'evento attraverso l'invocazione del metodo '**onApplicationEvent(ApplicationEvent evento)**'

Gestione degli eventi tra Mule e Spring

E' possibile attivare lo scambio di messaggi tra Mule e Spring, configurando nell'application context il seguente multicaster fornito con Mule:

- **org.mule.extras.spring.events.MuleEventMulticaster**

```
<bean id="applicationEventMulticaster"
    class="org.mule.extras.spring.events.MuleEventMulticaster">
  <property name="subscriptions">
     <list>
        <value>vm://myQueue4</value>
        <value>jms://eventQueue</value>
     </list>
  </property>
</bean>
```

In questa lista saranno indicati gli endpoints definiti in Mule da cui si desidera ricevere eventi in Spring

Gestione degli eventi tra Mule e Spring

Un bean che desidera ricevere eventi solo da alcuni endpoints può implementare l'interfaccia fornita con Mule:

- **org.mule.extras.spring.events.MuleSubscriptionEventListener**

```
 <bean id="service"
class="it.bozzoni.spring.services.BussinessServicesImpl">
    <property name="subscription">
        <list>
            <value>vm://myQueue</value>
        </list>
    </property>
</bean>
```

In questa listà saranno indicati gli endpoints che generano eventi al cui il bean è interessato

Gestione degli eventi tra Mule e Spring

Viceversa se un bean desidera pubblicare messaggi verso qualche endpoint esposto da Mule non deve fare altro che costruire un apposito evento fornito da Mule:

- **org.mule.extras.spring.events.MuleApplicationEvent**

```
String url = "vm://myQueue";
Request req = new Request();
req.setServizio("Servizio richiesto sincrono");
req.setAzione("Azione servizio");
req.setData("Dati servizio");

MuleApplicationEvent muleEvent = new MuleApplicationEvent
(req, url);
applicationContext.publishEvent(muleEvent);
```

Per quanto riguarda la pubblicazione non deve fare altro che richiamare, come di consueto,il metodo publish sull'application context.

Riferimenti

ESB – Enterprise Service Bus
http://www-128.ibm.com/developerworks/xml/library/ws-esbscen/
http://www-128.ibm.com/developerworks/xml/library/ws-esbscen2.html
http://www.ibm.com/developerworks/webservices/library/ws-esbscen3/

EIP – Enterprise Integration Pattern:
http://www.eaipatterns.com/index.html

SEDA – An Architecture for Highly Concurrent Server Applications
http://www.eecs.harvard.edu/~mdw/proj/seda/

Spring:
http://www.springframework.org

Mule:
http://mule.mulesource.org/display/MULE/Home

Grazie per l'attenzione.