University of Applied Sciences

Department of Informatics and Media

Apache Camel
Example Application - Earthquake Mashup

Systemintegration 1. Term Master Informatik

Prof. Dr. Preuss

Provided by: Robert Stümer, Nils Petersohn, Steffen Reinkensmeier

27.01.2011.

Contents

1 Introduction / Motivation / Project

System integration is the part of a software architecture. It helps to connect components or subsystems together. Certain patterns are used in the industry today. Using and learning EIP ("Enterprise Integration Patterns") with Apache Camel is the goal of this Project.

The Example for this project is all about earthquake data from around the world. The Application is able to read earthquake data from various RSS Feeds and processes it. During the processing the data will be in form of XML and Java Objects. The data will be enriched, spitted, sorted, aggregated, normalized, marshaled unmarshaled and finally provided again in form of a restful service.

The specified task is as follows:

1. Read Earthquake Data continuously from those two RSS Streams
   - `http://geofon.gfz-potsdam.de/db/eqinfo.php?fmt=rss`
   - `http://earthquake.usgs.gov/eqcenter/catalogs/eqs1day-M2.5.xml`
2. enrich this data with other related information like the weather in this area at this time. Data can be from here: `http://www.programmableweb.com`.
3. sort the earthquakes by the earth parts where they appear
4. if the earthquake has a strength of more than "M 5.5" than send an formatted warning email to an email address.
5. provide this data via a Restful interface in form of a list of the earth parts with an xlink to detailed information of the earthquakes.

2 Information Collection and Normalization

The Normalizer [HW03] (Pic. 2.1) routes each message type through a custom Message Translator so that the resulting messages match a common format. The different message formats are collected in one channel ("*direct* : *collectorChannel*") and than routed according to their xml format. The Routing is accomplished with a xPath condition which routes the messages to the specified Message Translators [HW03] (Pic. 2.2). Here The Translator is implemented with a XSLT formatter `http://www.w3.org/TR/xslt`. The relative source code is listed in 2.1

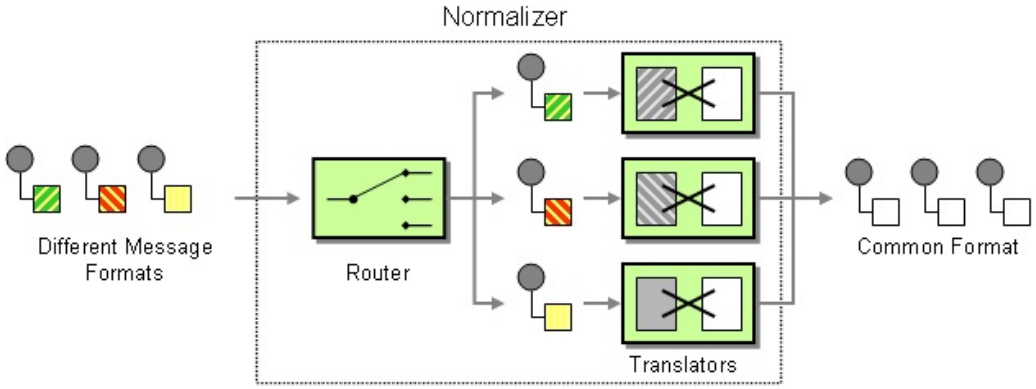

Figure 2.1: Normalizer Pattern [HW03]

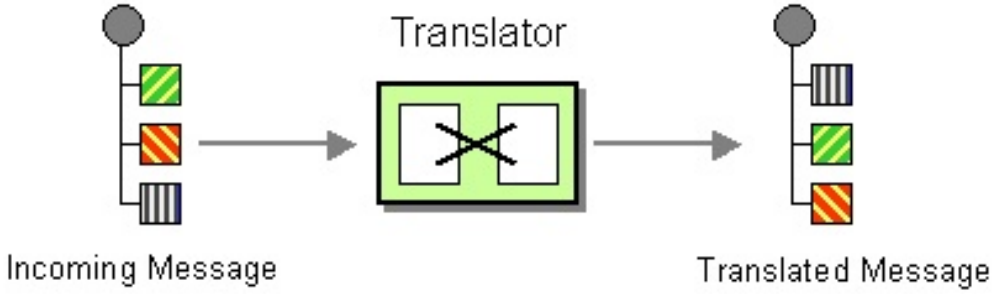

Figure 2.2: Translator Pattern [HW03]

Listing 2.1: Normalizer and Translator Source Code

```
from(
        "http://geofon.gfz-potsdam.de/db/eqinfo.php?fmt=rss&
           splitEntries=false")
        .log("retrieve")
        .to("direct:collectorChannel");

from(
        "http://earthquake.usgs.gov/eqcenter/catalogs/eqs1day-M2
           .5.xml?fmt=rss&splitEntries=false")
        .log("retrieve")
        .to("direct:collectorChannel");

from("direct:collectorChannel")
        .choice()
        .when().xpath("/rss/channel/item/pubDate")
                .to("xslt:data/xsl/transformation2.xsl")
                .setHeader("visited", constant(true))
                .log("true: has /rss/channel/item/pubDate")
                .to("direct:normalizedMsg")
        .otherwise()
                .to("xslt:data/xsl/transformation.xsl")
                .setHeader("visited", constant(true))
                .log("false: has not /rss/channel/item/pubDate")
                .to("direct:normalizedMsg");
```

3 Aggregating Information Sources

> "The Aggregator is a special filter that receives a stream of messages an identifies messages that are correlated. Once a complete set of messages has been received, the Aggregator collects information from each correlated message and publishes a single, aggregated message to the output channel for further processing." [HW03]

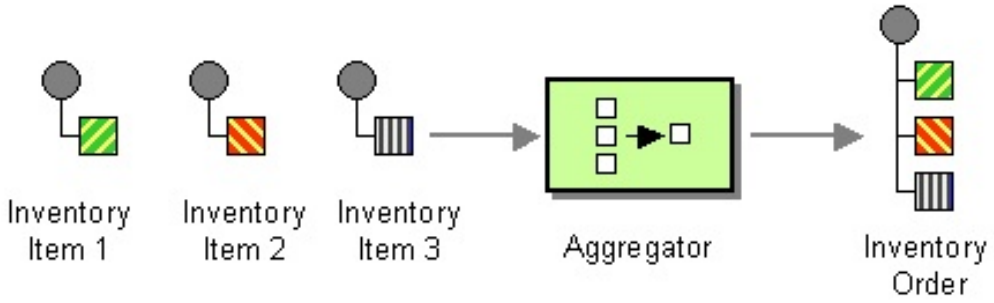

Figure 3.1: Aggregator Pattern [HW03]

Camel provides a method named *aggregate* with two parameters, first the identifier (the message header which was defined before in chapter 2) of the messages to aggregate and second the strategy (as a Object: *SpecialAggregationStrategy* which implements *org.apache.camel.processor.aggregate.AggregationStrategy*). Code listing 3.1 shows the implementation in this project for the normalized messages from Chapter 2. Code listing 3.2 shows the strategy which is fairly simple.

Listing 3.1: Aggregator

```
1 from("direct:normalizedMsg")
2   .aggregate(header("visited"), new XMLAggregationStrategy())
3   .completionSize(2).delay(3000)
4   .to("direct:filterBiggestEarthquakes")
5   .to("direct:UnmarshallMergedSources");
```

Listing 3.2: Aggregation Strategy

```
1 import org.apache.camel.Exchange;
```

```
2  import org.apache.camel.processor.aggregate.AggregationStrategy;
3
4  public class XMLAggregationStrategy implements
       AggregationStrategy {
5
6    public Exchange aggregate(Exchange oldExchange, Exchange
         newExchange) {
7      if (oldExchange == null) {
8        return newExchange;
9      }
10     String oldBody = oldExchange.getIn().getBody(String.class);
11     String newBody = newExchange.getIn().getBody(String.class);
12     String body = oldBody + newBody;
13
14     body = body
15         .replaceAll("<\\?xml version=\"1\\.0\" encoding=\"UTF
               -8\"\\?>",
16             "")
17         .replaceAll("</earthquakes>(.*)<earthquakes>", "")
18         .replaceAll(
19             "</earthquakes><earthquakes xmlns:geo=\"http://www\\.
                   w3\\.org/2003/01/geo/wgs84_pos#\">",
20             "").replaceAll("</earthquakes><earthquakes>", "");
21
22     oldExchange.getIn().setBody(body);
23     return oldExchange;
24   }
25 }
```

4 Enrich Data with related Information

Information can be added to the original Message with the Message Transformation Pattern “Content Enricher” .

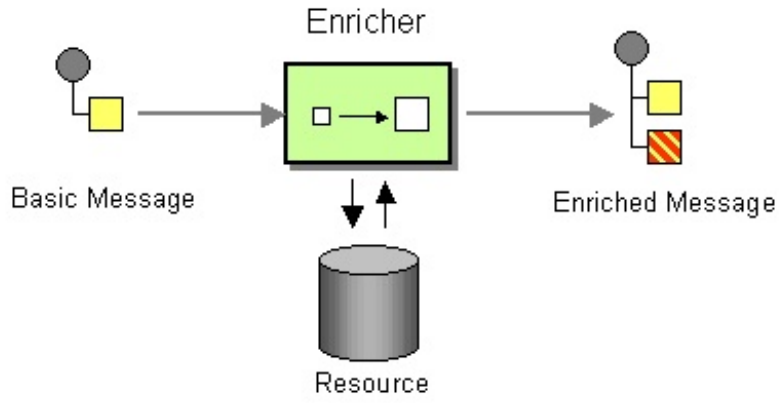

Figure 4.1: Content Enricher Pattern [HW03]

“The Content Enricher (Pic. 4.1) uses information inside the incoming message to retrieve data from an external source.[HW03]” This is accomplished via the coordinates of the earthquake. which are used to retrieve area information
`http://api.wunderground.com/auto/wui/geo/GeoLookupXML`
and weather information from `http://www.worldweatheronline.com/feed/weather.ashx`. See Listing 4.1 for the source code.

Listing 4.1: Enricher

```
1 from("direct:UnmarshallMergedSources")
2   .unmarshal(jaxb)
3   .process(new EnrichmentProcessor())
```

The *EnrichmentProcessor* can now work with the java instances see Listing 4.2

Listing 4.2: EnrichmentProcessor

```
1 final class EnrichmentProcessor implements Processor {
```

```
2     public void process(Exchange exchange) throws Exception {
3       EarthquakeCollection ec = exchange.getIn().getBody(
4           EarthquakeCollection.class);
5       ArrayList<Earthquake> listClone = new ArrayList<Earthquake>()
            ;
6       int i = 1;
7       for (Earthquake e : ec.getEntries()) {
8         String additionalInfo = CommonUtils
9             .findAdditionalInfo(e.getLocation());
10
11        e.setCountry(additionalInfo.contains("not found") ? "
              undefined"
12            : additionalInfo);
13
14        Weather findWeatherInfo = CommonUtils.findWeatherInfo(e.
              getLocation());
15
16        e.setWeather(findWeatherInfo);
17
18        e.setId(i++);
19
20        listClone.add(e);
21      }
22      ec.setEntries(listClone);
23      System.out.println("setting earthquakes now");
24      exchange.getIn()
25          .setBody(ec, EarthquakeCollection.class);
26    }
27  }
```

For the most flexibility the XML Message is automatically unmarshaled to the specified Java objects via the JAXB Unmarshaler.

- Earthpart
- EarthPartCollection
- EarthquakeCollection
- Earthquake
- Weather
- WeatherWrapper

This is easily accomplished with jaxb Annotations in the data models. For more information on JAXB please visit:
`http://www.oracle.com/technetwork/articles/javase/index-140168.html`

In Camel, data formats are pluggable transformers that can transform messages from one form to another and vice versa. Each data format is represented in Camel as an interface in org.apache.camel.spi.DataFormat containing two methods:

- marshal—For marshaling a message into another form, such as marshaling Java objects to XML, CSV, EDI, HL7, or other well-known data models.
- unmarshal—The reverse operation which turns data from well known formats back into a message.

You may already have realized that these two functions are opposites, meaning that one is capable of reversing what the other has done, as illustrated in figure 4.2. [IA10]

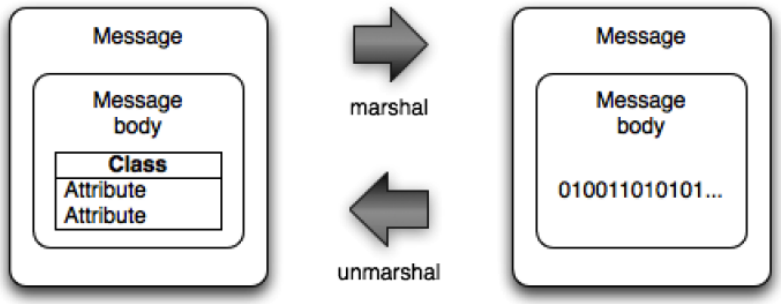

Figure 4.2: Transforming with Data Formats [HW03]

The Marshaled State is shown in the Class Diagram (Pic 4.3) for the package *edu.fhb.sysint.camel.model*
The marshalled state is the XML Form (Pic. 4.4, 4.5)

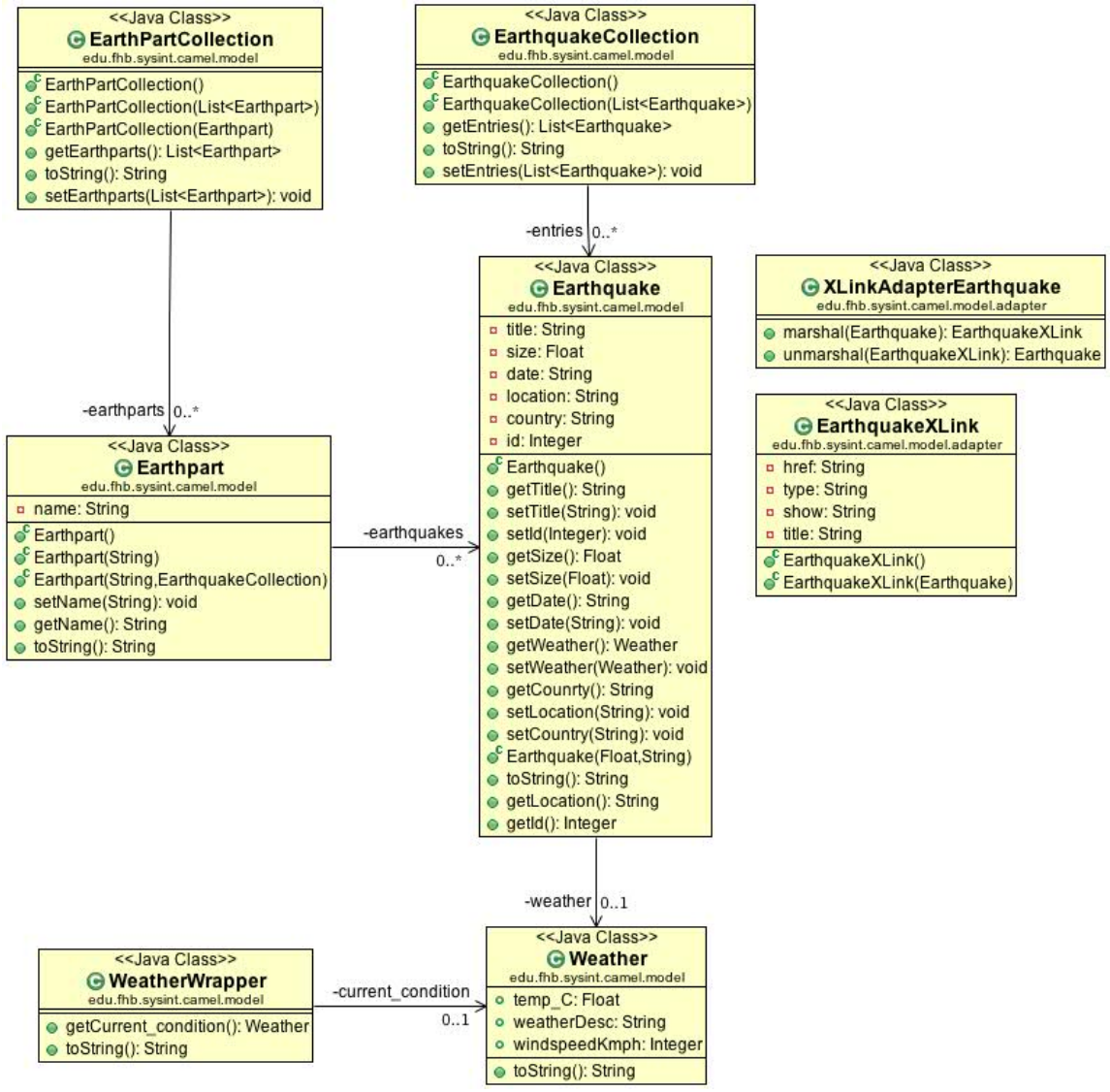

Figure 4.3: edu.fhb.sysint.camel.model Package

Figure 4.4: marshalled Earthparts

Figure 4.5: marshalled Earthquake

5 Email Notification

If the strength of the Earthquakes are more than M 5.5 than an email notification must be delivered. This is implemented with Apache Camel through the splitter Pattern (Pic 5.1)

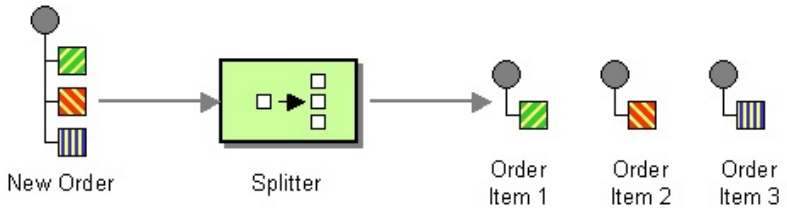

Figure 5.1: Sequencer with Splitter Pattern

The Source is spitted and aggregated. If The xPath evaluates to true the message is reformatted in a human readable format and sent to the provided email address.

Listing 5.1: Splitting Pattern

```
1  from("direct:filterBiggestEarthquakes")
2    .split(xpath("/earthquakes/earthquake[size>5.4]"))
3    .setHeader("splitted", constant(true))
4    .aggregate(header("splitted"), new SimpleAggregationStrategy())
5    .completionInterval(2000)
6    .process(new Processor() {
7      public void process(Exchange exchange) throws Exception {
8        String body = exchange.getIn().getBody(String.class);
9        body = "<earthquakes>" + body + "</earthquakes>";
10       exchange.getIn().setBody(body, String.class);
11     }
12   })
13   .unmarshal(jaxb)
14   .process(new EmailProcessor())
15   .to("smtps://camelfhb@smtp.gmail.com?password=camelfhb31&to=
       camelfhb@googlemail.com")
16   .delay(120000);
```

The Resulting Email is shown in Pic 5.2

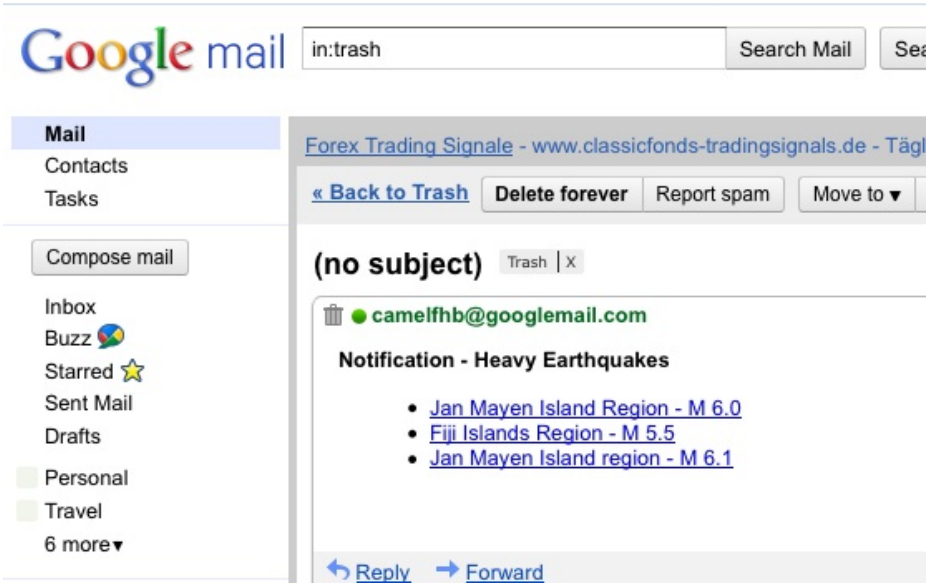

Figure 5.2: Email Result

6 RESTful Service

The Mashup is provided via a REST Web Service (`http://localhost:9000/earthquakeService/`) The provided Data is sorted by Earth parts which were determined via the enrichment Phase. Camel has the availability to establish a REST Service with the Help of the Apache CXF Project which works closely JAXB (marshaling/unmarshaling). The Rest Router (Listing 6.1) determines the requested URL and requests the DAO Layer [Fow02] to send the correct response Data.

Listing 6.1: Rest Router

```
1 String name = RestServiceImpl.class.getName();
2 from("cxfrs://http://localhost:9000?resourceClasses=" + name).
     process(
3                 ...
```

The CXF Endpoint is configured with the Definition class (Listing 6.2)

Listing 6.2: CXF Service Definition

```
1  import javax.ws.rs.GET;
2  import javax.ws.rs.Path;
3  import javax.ws.rs.PathParam;
4  import javax.ws.rs.Produces;
5
6  import edu.fhb.sysint.camel.dao.EarthpartDao;
7  import edu.fhb.sysint.camel.dao.EarthquakeDao;
8  import edu.fhb.sysint.camel.model.EarthPartCollection;
9  import edu.fhb.sysint.camel.model.Earthpart;
10 import edu.fhb.sysint.camel.model.Earthquake;
11
12 @Path("/earthquakeService/")
13 @Produces("application/xml")
14 public class RestServiceImpl {
15
16         public RestServiceImpl() {
17         }
18
19         @GET
20         @Path("/Earthparts")
21         public EarthPartCollection getAllEarthparts() {
22                 return EarthpartDao.all();
```

```
23          }
24
25          @GET
26          @Path("/Earthquake/findById/{id}")
27          public Earthquake getEarthquake(@PathParam("id") Integer
                id) {
28                  return EarthquakeDao.findById(id);
29          }
30
31  }
```

7 XLink with jaxb Adapters

Jaxb allows the use of the Adapter Pattern [GHJV95]. This allows to define special Marshaling for Objects. When the jaxb unmarshaler gets to the Earth part Object than the data field “Earthquakes” is not umarshaled normally. The Annotation @XmlJavaTypeAdapter (Listing 7.1) Redirects that action to special marshalling/umarshalling methods.

Listing 7.1: Earthpart Class

```
1  import java.util.List;
2
3  import javax.xml.bind.annotation.XmlAccessType;
4  import javax.xml.bind.annotation.XmlAccessorType;
5  import javax.xml.bind.annotation.XmlAttribute;
6  import javax.xml.bind.annotation.XmlElement;
7  import javax.xml.bind.annotation.XmlElementWrapper;
8  import javax.xml.bind.annotation.XmlRootElement;
9  import javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter;
10
11 import edu.fhb.sysint.camel.model.Earthquake;
12 import edu.fhb.sysint.camel.model.adapter.XLinkAdapterEarthquake;
13
14 @XmlRootElement(name = "earthpart")
15 @XmlAccessorType(XmlAccessType.FIELD)
16 public class Earthpart {
17
18         @XmlAttribute(name = "name")
19         private String name = "";
20
21         @XmlElementWrapper(name = "erathquakes")
22         @XmlElement(name = "erathquake")
23         @XmlJavaTypeAdapter(XLinkAdapterEarthquake.class)
24         private List<Earthquake> earthquakes;
25
26         ...
```

Since the Classes for marshaling this special requirement are located in an extra package *edu.fhb.sysint.camel.model.adapter* it was possible to use the namespace mechanism at package level.

“Annotations that can be applied to the package element are referred to as package-level annotations. An annotation with ElementType.PACKAGE as one of its targets is a package-

level annotation. Package-level annotations are placed in a package-info.java 7.2 file. This file should contain only the package statement, preceded by annotations. When the compiler encounters package-info.java file, it will create a synthetic interface, package-name.package-info. The interface is called synthetic because it is introduced by the compiler and does not have a corresponding construct in the source code. This synthetic interface makes it possible to access package-level annotations at runtime.[Jay]"

Listing 7.2: package-info.java for the Adapter Package

```
1 @javax.xml.bind.annotation.XmlSchema(namespace = "http://www.w3.
    org/1999/xlink", xmlns = { @javax.xml.bind.annotation.XmlNs(
    prefix = "xlink", namespaceURI = "http://www.w3.org/1999/xlink
    ") }, elementFormDefault = javax.xml.bind.annotation.XmlNsForm
    .QUALIFIED)
2 package edu.fhb.sysint.camel.model.adapter;
```

8 Deployment Steps

The System is deployed via the Apache Servicemix OSGI Runtime Container. “Apache ServiceMix is an open source ESB (Enterprise Service Bus) that combines the functionality of a Service Oriented Architecture (SOA) and an Event Driven Architecture (EDA) to create an agile, enterprise ESB.[sm] ”

The Camel Version used is “2.4.0-fuse-02-00” from the repository “`http://repo.fusesource.com/maven2/`”.

1. download fuse 4-3-0 file and unpack to /downloadedFuse `http://fusesource.com/product_download/fuse-esb-apache-servicemix/4-3-0-fuse-03-00/unix`
2. replace /downloadedFuse/etc/org.apache.karaf.features.cfg key value pair "featuresBoot=..." with

   featuresBoot=config,activemq-broker,camel,camel-http,camel-jaxb,jbi-cluster,war, servicemix-cxf-bc,servicemix-file,servicemix-ftp,servicemix-http,servicemix-jms, servicemix-mail,servicemix-bean,servicemix-camel,servicemix-cxf-se,servicemix-drools, servicemix-eip,servicemix-osworkflow,servicemix-quartz,servicemix-scripting, servicemix-validation,servicemix-saxon,servicemix-wsn2005,camel,camel-spring-osgi, cxf,camel-cxf,camel-jetty,web,cxf-jaxrs,camel-rss,activemq-camel,rome-osgi,camel-mail

3. please modify the Constants in GlobalConstants.java for path settings.
4. in the project root do “mvn install”
5. start fuse with /downloadedFuse/bin/servicemix
6. on the karaf shell type: “install -s mvn:edu.fhb.sysint.camel/ApacheCamelEarthquakeService”
7. browse to http://localhost:9000/earthquakeService/Earthparts

Bibliography

[Fow02] Fowler, M.: *Patterns of enterprise application architecture*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2002. – ISBN 0321127420

[GHJV95] Gamma, E. ; Helm, R. ; Johnson, R. ; Vlissides, J.: *Design patterns: elements of reusable object-oriented software*. Bd. 206. Addison-wesley Reading, MA, 1995

[HW03] Hohpe, G. ; Woolf, B.: *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2003. – ISBN 0321200683

[IA10] Ibsen, C. ; Anstey, J.: *Camel in Action*. Manning Publications, 2010

[Jay] Jayaratchagan, Narayanan: *Declarative Programming in Java*. `http://onjava.com/pub/a/onjava/2004/04/21/declarative.html?page=3`

[sm] *Apache ServiceMix*. `http://servicemix.apache.org/home.html`

List of Figures