Project #1

Enterprise Application Integration

João Ribeiro
David Cruz
Filipe Norte

12/10/2012

1 Introduction

The current report provides a description of the solution implemented for project 1. It includes static and dynamic perspectives of the architecture and an explanation of some of the major design decisions.

2 Architecture Description

2.1 Dynamic View

The system is composed by two daemons that register themselves into the JMS topic and subscribes to a defined topic that would be feed with data information extracted from the WebProbe. Each of the daemons receives data from the channel and handles the information in different ways. The following dynamic diagram describes the overall system structure of it.

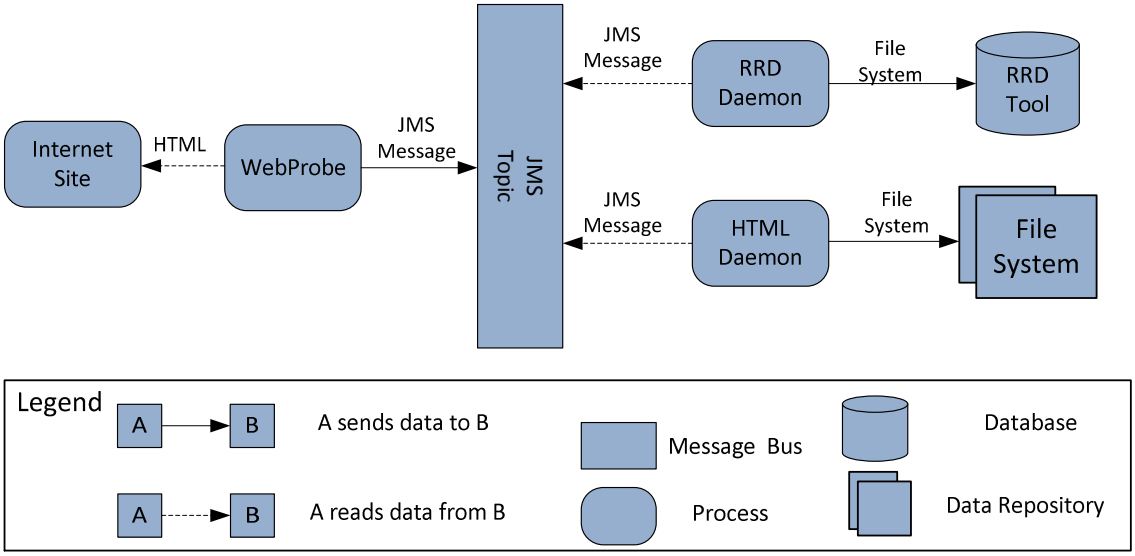

Figure 1 – Dynamic view of the system

2.2 Static View

The project is composed by different packages with different responsibilities on the overall project. The following table gives a brief overview of their responsibilities and relevant information:

Package	Responsabilities
eai.msejdf.apps	Contains the main probe class that extracts the information from external sources into the system
eai.msejdf.config	Handles all the configuration settings required by the system
eai.msejdf.daemons	Contains all the relevant classes related with subscriber clients that will handle all the information supplied by the apps probes
eai.msejdf.data	The data model object that was used internally by the system. Contains the xsd schema definition of the data model.

eai.msejdf.jms	This package is responsible for handling all the Java Messaging System (JMS) interactions (sending and receiving of messages through JMS)
eai.msejdf.rrd	Package responsible for handling the creation and saving of data into Round Robin Databases (RRD)
eai.msejdf.utils	Package responsible for various utilities used by the system. Includes classes for handling the xml/xsl transformations, property loading and string operations
eai.msejdf.web	Package responsible for the probe plugin system implemented in the system. It can handle different providers for different sources.

Table 1 – Package list and responsibilities

Some of these packages are composed of different structures and classes that define object instances used internally by the systems. A brief detail of their structure and design will be described on the following sections for the main packages.

2.2.1 eai.msejdf.daemons package

The daemons static perspective for the project can be seen on the following diagram.

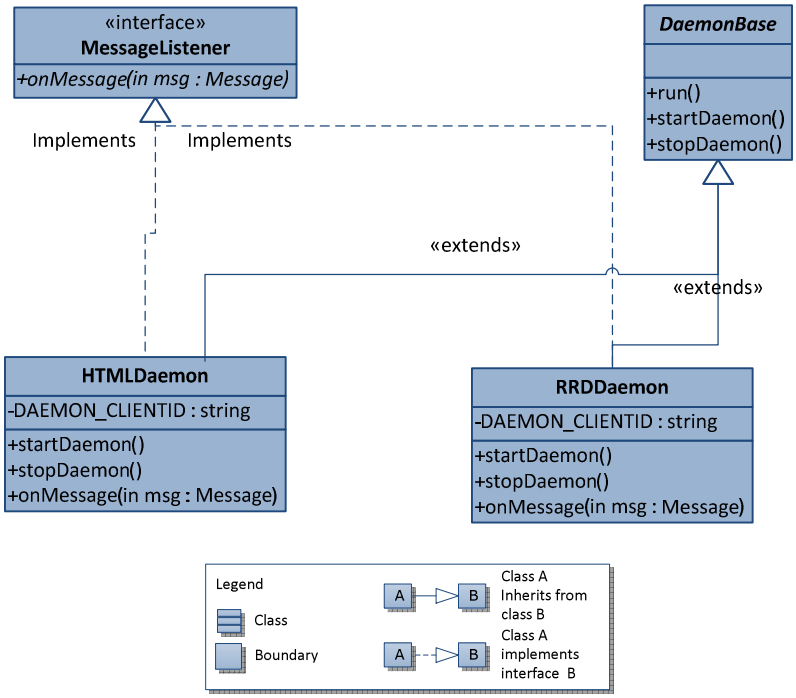

Figure 2 - Daemons static perspective

The daemons are composed of an abstract base class that provides the main execution of programs. All daemons on the system extend this main base class and implement the starting and ending methods of the daemon. These methods normally handle any starting and ending functionalities that are required by the daemons, like registering into the JMS bus and des-

registering from it. The property DAEMON_CLIENTID, identifies the client ID that will be used with the JMS channel in order to persist the messages on the bus.

2.2.2 eai.msejdf.jms package

The JMS package is composed by all the classes responsible for interacting with JMS system. The following diagram describes the main classes involved in the system.

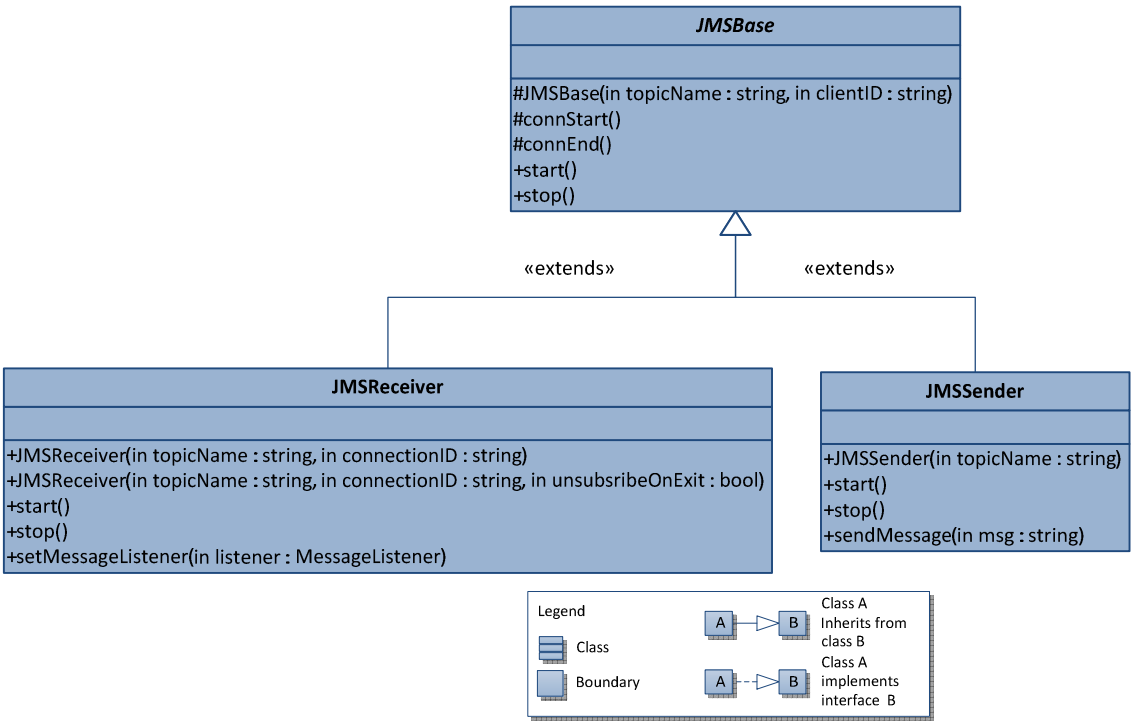

Figure 3 - JMS package static view

The jms package is organized through the use of an abstract class that handles all the common communication starting options with the JMS messaging system. Both the JMS receiver and sender classes rely on this common implementation that set the initial connection and subscription topic name for the system. The JMS sender then defines message producer part of the JMS system and provides a method for sending messages through the topic channel.

The JMS receiver defines the message consumer part of the JMS system and it has two constructors. The default one starts the listening of messages through the channel and maintains the subscription of the topic by the receiver after exiting the system. There's also the possibility to create a receiver that unsubscribes the topic on exiting the system, this can be done through the use of the second constructor indicating the unsubscribeOnExit parameter.

2.2.3 eai.msejdf.apps/eai.msejdf.web package

The static perspective of the Web Probe application is displayed in Figure 4. To better understand the application as a whole, the diagram includes classes from the web plugin and data packages. The parsing of the HTML page is implemented in the ParseStocksPlugin class. The WebProbe application uses this class through a public interface in order to implement a plugin mechanism during run time. It relies on the call to the Parse() method to parse the web

page and return an object representing the page contents. The associated design decisions will be explained in section 2.3.1.

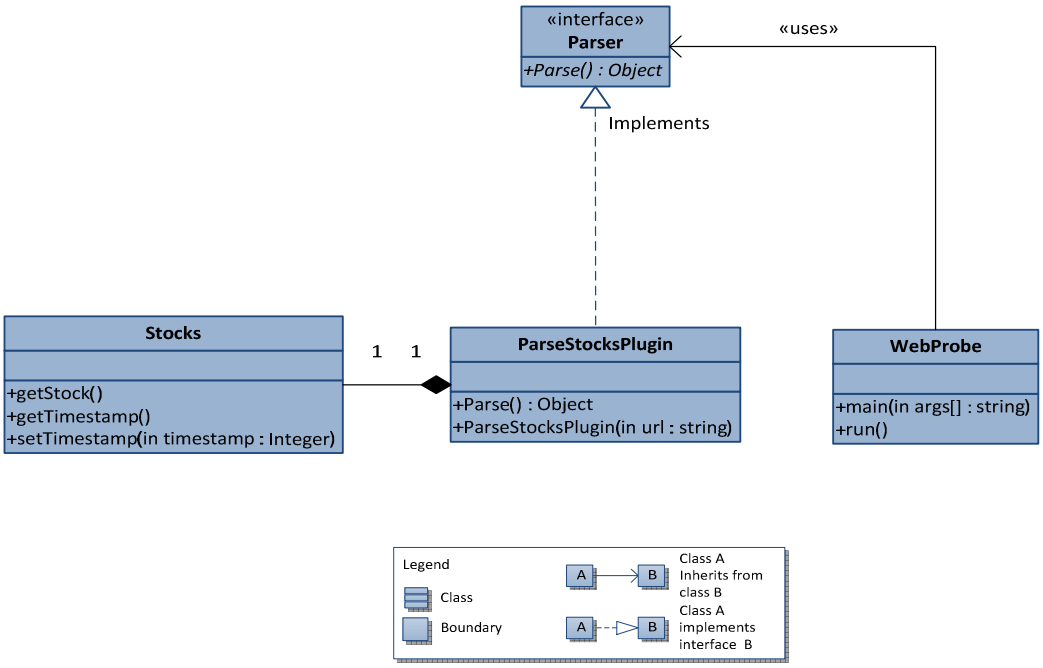

Figure 4 – Web Probe + Plugin static view

2.3 Design Decisions

A few design decisions were made that are worth pointing out. These are related with the Web Probe, XML/XSD file syntax and logging support.

2.3.1 Web Probe

The Web Probe is responsible for parsing web sites and sending a serialized xml file representing the contents to the JMS server. Considering this flow, there is a lot in common with an application that would do exactly the same, but for a different web site, only the parsing implementation would be different.

As this functionality would allow the implementation of a more generic and extensible application, the Web Probe was designed using a plugin mechanism in which different parsers correspond to different plugins. The glue between the probe and the parser is constructed through a parser interface (Parser interface in Figure 4) that each parser must implement. The probe application will receive as arguments when it is invoked the URL of the web site to parse and the name of the parser class (plugin) associated with the site syntax. The probe uses its class loader to load the correct class (parser) on the fly (ParseStocksPlugin.java for the site we selected).

Considering that the objects representing a parsed web site are specific to the site data, the parser will return a generic object. This avoids creating a dependency on the web probe side to the data objects used by the parser, as otherwise each time a new parser would be added with specific data objects, the probe would have to be updated, losing the flexibility of the plugin

mechanism. The probe will use that object and serialize it when sending the containing data to the JMS topic.

In terms of parsing, the site to be parsed contains a table in which the rows contain stock related information for a given company. We had to decide to abort the parsing of the whole page or continuing if a row did not have the expected syntax. The more logical decision would be to abort, as we would most likely be facing a page that has changed and consequently we could not make any assumptions about the correctness of the columns for a given row. However, we verified that the site has recently added a new company to the stock list, but which contains incomplete data that leads the parser to fail (validations). We decided to not abort the whole process and just ignore that row to don't jeopardize the retrieval of good data just because of an incomplete row. If the page syntax changed as a whole, all rows will fail, ensuring that we minimize the potential of sending inconsistent data to the topic. This solution seemed better than assuming default values for unparsed fields in a row, as that would change the accuracy of the data (a receiver would not know whenever the data is real or originally missing).

2.3.2 XML/XSD file syntax

The syntax of the XML file selected by us is slightly different from the suggestion provided in the assignment. Instead of using an attribute specifying the company name in the element containing the stock quotation value, we opted to have dedicated elements for the company data and quotation information, aggregating them under a stock element, as shown in Annex A. The advantage of this approach is that it allows separating/isolating the different types of information, promoting the reuse of data elements. In addition, it eases the extension of independent elements without creating an overly complex single element having everything. For instance, we could easily add an address and web site field to the company in a clean way.

It shall be noted that we did not separate these elements into different files to keep the solution simple considering the purpose of the assignment and value we would get from it, as the code would have to be prepared to load multiple XSD files, considering that they are local files (a main XSD file would import or include the individual XSD files). This also led us to use one single namespace, although to promote the flexibility previously mentioned in a wider way, it would be more beneficial to have the company and quotation elements with dedicated namespaces.

The XML/XSD file also includes an attribute that specifies a time stamp. This time stamp corresponds to the time at which the web site was parsed and the XML file generated. We decided to associate this time with the data as it provides the information about when exactly the data was generated. This is useful when the web probe tries to send the data over to the topic, but the topic is down. Since the data is stored locally to be send a next time the web probe runs, using the time the data was send or received (through topic mechanisms, for example), would not provide an accurate time associated with the data generation.

2.3.3 Logging (log4j)

The team decided from the beginning of the project the use of a logging system that could after help detect any problems or errors in the system. The logging system could also be used

to trace certain calls and message passing between the different components and the JMS system.

Instead of creating a new logging system from scratch, which would be outside the scope of the project context, the team decided to investigate some of the already known java logging platforms and opted in using the highly configurable log4j http://logging.apache.org/log4j/. This system allows an easy integration and configuration within all the system. The platform allows the possibility to direct the logs to different outputs (files, screens, network, ….) based on the different message categories of logging (debug, trace, …) required by each class. For the context of this project we decided to configure our system to output logging information to the standard output and to a log file for all types and classes. This can be easily changed through the setup of the log4j.properties file included in the project and described on section 3.2.2.

To note that all the calls to the logging platform used through the system were optimized according some log4j optimization good practices, through the use of if statements that would enable or not the call to the write method of the logging platform. The following code exemplifies the call

```
if (logger.isDebugEnabled())
{
        logger.debug("Writing HTML file: " + htmlfilename);
}
```

These if statements enable better performance by avoiding some java string concatenation calls before calling a method that might not do anything particular because it was not enabled on the configuration file (log4j.properties).

3 Deploy

3.1 Prerequisites

In order to execute this software it is necessary to have JMS with JBoss AS 7 installed and running with the configurations accordingly. Make sure to configure 1) The JMS system application user 2) the password associated and 3) the topic name; according to the config.propreties file discussed in 3.2.1.

3.2 Settings and Configurations

The following section describes the settings stored in configuration files, their meaning and default values.

3.2.1 config.properties

This file contains the main application settings properties of the system. The following table describes the main keys of this file.

Setting	Default value	Meaning
app.jms.applicationuser	<Appuser>	The application user used for the JMS system
app.jms.applicationpassword	<AppPass>	The password associated with the application user

		for the JMS system
app.jms.topicbasename	EAIProject1	The topic name defined in the JMS system container
app.web.connectiontimeout	20000	The timeout period before error for gathering data from the external websites (in milliseconds)
app.xml.htmlfiledirectory	./html	The default output directory for html files, relative to the base output directory
app.xml.pendingmessagesdirectory	./pending	The default directory for pending messages when the JMS system is down for the WebProbe
app.xml.xsltfile	eai/msejdf/data/Stocks.xsl	The location of the XSL file for parsing the data xml from the probe into html files
app.base.outputdir	./out	The base output directory for all the files generated by the system
app.rrd.outputdir	./rrd	The default output directory for the RRD database files, relative to the base output directory

Table 2 – Configuration settings

3.2.2 log4j.properties

This file contains the main configurations settings for the log4j logging interface. Some of the settings are somehow out of scope of the project. The following table describes the most important ones:

Setting	Default value	Meaning
log4j.rootCategory	DEBUG, R, O	The default log message level for all the classes of the system log message level (e.g. DEBUG, INFO, WARN, ERROR, FATAL)
log4j.appender.*	org.apache.log4j.ConsoleAppender or org.apache.log4j.RollingFileAppender	The type of output associated with the logging message level (Screen, File, ….)
log4j.appender.*.File	log4j.log	The file name of the log file when using an the RollingFileAppender

3.3 How to execute

In the delivered package there are three executable files (consult table). To have the full system working you need to ensure JBOSS is running, start both Daemons and then execute the Webprobe.bat each time you want to analyze the site.

File	Operation
WebProbe.bat	Executes the parser plugin
RRDDaemon.bat	Initiates the RRD Daemon
HTMLDaemon.bat	Initiates the HTML Daemon

4 Participations

The participation by each team member is listed as the packages they mainly worked on. However, each one also worked on other packages, although to less extend in order to update with functionalities or for corrections.

4.1 João Ribeiro

- eai.msejdf.daemons (HTML), eai.msejdf.data (XSD/XLS), eai.msejdf.utils

4.2 David Cruz

- eai.msejdf.daemons (RRD), eai.msejdf.jms, eai.msejdf.utils, eai.msejdf.config, eai.msejdf.rrd

4.3 Filipe Norte

- eai.msejdf.apps, eai.msejdf.web, eai.msejdf.utils

Annex A – XSD Syntax

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="msejdf"
  targetNamespace="msejdf"
  elementFormDefault="qualified">

 <xs:element name="stocks">
  <xs:complexType>
   <xs:sequence>
    <xs:element maxOccurs="unbounded" ref="stock"/>
   </xs:sequence>
   <xs:attribute name="timestamp" use="required" type="xs:unsignedLong"/>
  </xs:complexType>
 </xs:element>

 <xs:element name="stock">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="company"/>
    <xs:element ref="quotation"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>

 <xs:element name="company">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="name"/>
    <xs:element ref="address" minOccurs="0" maxOccurs="1"/>
    <xs:element ref="website" minOccurs="0" maxOccurs="1"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>

 <xs:element name="name" type="xs:string"/>

 <xs:element name="address" type="xs:string"/>

 <xs:element name="website" type="xs:string"/>

 <xs:element name="quotation">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="lastQuotation"/>
    <xs:element ref="time"/>
    <xs:element ref="variation"/>
    <xs:element ref="quantity"/>
    <xs:element ref="maximum"/>
    <xs:element ref="minimum"/>
    <xs:element ref="purchase"/>
```

```
      <xs:element ref="sell"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="lastQuotation">
  <xs:simpleType>
    <xs:restriction base="xs:decimal">
      <xs:minInclusive value="0"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="time">
  <xs:simpleType>
    <xs:restriction base="xs:string" >
      <!-- Support only HH:mm time format from 00:00 (or 0:00) to 23:59 -->
      <xs:pattern value="([0-9]|0[0-9]|1[0-9]|2[0-3]):[0-5][0-9]" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="variation" type="xs:decimal"/>

<xs:element name="quantity" type="xs:unsignedLong"/>

<xs:element name="maximum">
  <xs:simpleType>
    <xs:restriction base="xs:decimal">
      <xs:minInclusive value="0"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="minimum">
  <xs:simpleType>
    <xs:restriction base="xs:decimal">
      <xs:minInclusive value="0"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="purchase">
  <xs:simpleType>
    <xs:restriction base="xs:decimal">
      <xs:minInclusive value="0"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="sell">
  <xs:simpleType>
```

```
        <xs:restriction base="xs:decimal">
          <xs:minInclusive value="0"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>

  </xs:schema>
```