UNIVERSIDADE DE COIMBRA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
Departamento de Engenharia Informática

Project #1
Integração de Sistemas/
Enterprise Application Integration

2014/15 – 1st Semester
MEI

Deadline: 2014-10-17

Nota: A fraude denota uma grave falta de ética e constitui um comportamento não admissível num estudante do ensino superior e futuro profissional. Qualquer tentativa de fraude pode levar à reprovação na disciplina tanto do facilitador como do prevaricador.

XML and XML Manipulation, Java Message Service and Message Oriented Middleware

Objectives

- Learn **XML technologies**. In particular, you will learn XML, XSD, XSL, XSLT and XPATH. This project is mostly about XML processing.
- Understand the technique of "**Screen Scraping**". Screen scraping consists in parsing the information shown in a terminal so that it can be used on a different system. It is the technique used for application integration where the only access point to an application is through its user interface (e.g., a venerable VT100 text terminal). Since, nowadays, web systems are ubiquitous, screen scraping is mostly used to gather and process information from web sites that do not expose APIs to the general public (or their business partners).
- Remember (or learn) how to use **HTML parsers**. These parsers can read HTML code and create data structures representing the web page, such as DOM[1] documents. You may also need to resort to **regular expressions** to clean data available in the DOM document. Regular expressions are an extremely powerful mechanism for cleaning, gathering and processing data embedded in text files.
- Learn how to create simple asynchronous and message-oriented applications.

[1] Document Object Model.

Final Delivery

- This assignment contains two parts: one is for training only, and does not count for the evaluation. You should only deliver the other part.
- You must submit your project in a zip file using Inforestudante. Do not forget to associate your work colleague during the submission process.
- The submission contents are:
  - Source code of the requested applications ready to compile and execute.
  - A small report in pdf format (5 pages max) about the implementation of the project.
- After submitting, you are required to register the (extra-class) effort spent solving the assignment. This step is mandatory. Please fill the effort form at: **https://docs.google.com/spreadsheet/viewform?formkey=dG9KTWpla0dnRW1aQ1JNdzRVTUJJMFE6MA**

Resources

Jsoup

- Jsoup Java HTML Parser, with best of DOM, CSS, and jquery: http://jsoup.org
- Manual at: http://jsoup.org/cookbook/

XML, XSD, XSL and XSLT

- XML: http://www.w3schools.com/xml
- XSD: http://www.w3schools.com/schema
- XPATH: http://www.w3schools.com/xpath
- “Chapter 2: Understanding XML”, in J2EE 1.4 Tutorial http://download.oracle.com/javaee/1.4/tutorial/doc/
- JAXB Tutorial – Java.net: http://jaxb.java.net/tutorial/index.html
- Trang – http://www.thaiopensource.com/relaxng/trang.html

Processing XML/XSLT in Java

- “Chapter 7: Extensible Stylesheet Language Transformations”, in J2EE 1.4 Tutorial
  (Especially, the part “How XPath Works” and “Transforming XML Data with XSLT”)
  http://download.oracle.com/javaee/1.4/tutorial/doc/
- David Jacobs, “Rescuing XSLT from Niche Status – A Gentle Introduction to XSLT through HTML Templates”, http://www.xfront.com/rescuing-xslt.html
- G. Ken Holman, “What is XSLT?”, in XML.COM http://www.xml.com/lpt/a/2000/08/holman/index.html
  (Especially, the part “Getting started with XSLT and XPath”)

- Paul Grosso and Norman Walsh, “XSL Concepts and Practical Use”, in NWalsh.COM
  http://nwalsh.com/docs/tutorials/xsl/xsl/frames.html

Java Message Service

- http://docs.oracle.com/javaee/7/api
- Introducing the Java Message Service: http://www.digilife.be/quickreferences/pt/introducing%20the%20java%20message%20service.pdf
- Mark Richards, Richard Monson-Haefel, and David A. Chappell, “Java Message Service”
- JMS with **JBoss WildFly 8.1.0.Final**: http://eai-course.blogspot.pt, download at: http://wildfly.org

Advice: Skim all the links above before starting to read anything in detail. The recommended IDE to use is *Eclipse IDE for Java EE Developers*, however you are free to use another one.
Note: You have short examples of some of the technologies in the next section.

XML Training Part (doesn’t count for evaluation)

1. Use a tool like *trang* to automatically produce the XSD for the following XML. Change the XSD, to ensure that <direction> can only be one of “dgsg|boinc” or “dgsg|xtremweb”, while <timestamp> must be positive. Note that you should always check if the tool inferred the correct schema, or if it requires some manual adjustment.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<report timestamp="1308046204104" timezone="GMT" version="1.1">
<metric_data>
    <metric_name>cpus_available</metric_name>
    <timestamp>1308046204003</timestamp>
    <value>0.0</value>
    <type>uint32</type>
    <units>cpus</units>
    <spoof>EDGITest|fusion:EDGITest|fusion</spoof>
    <direction>dgsg|boinc</direction>
  </metric_data>
  <metric_data>
    <metric_name>gflops</metric_name>
    <timestamp>1308046204056</timestamp>
    <value>0.0</value>
    <type>float</type>
    <units>gflops</units>
    <spoof>EDGITest|fusion:EDGITest|fusion</spoof>
    <direction>dgsg|boinc</direction>
  </metric_data>
  <metric_data>
    <metric_name>past_workunits</metric_name>
    <timestamp>1308046204058</timestamp>
    <value>0.0</value>
    <type>uint32</type>
    <units>wus</units>
    <spoof>EDGITest|fusion:EDGITest|fusion</spoof>
    <direction>dgsg|boinc</direction>
  </metric_data>
  <metric_data>
    <metric_name>waiting_workunits</metric_name>
    <timestamp>1308046204059</timestamp>
    <value>0.0</value>
    <type>uint32</type>
    <units>wus</units>
    <spoof>EDGITest|dsp:EDGITest|dsp</spoof>
    <direction>dgsg|boinc</direction>
  </metric_data>
  <metric_data>
    <metric_name>success_rate</metric_name>
    <timestamp>1308046204061</timestamp>
    <value>1.0</value>
    <type>float</type>
    <units>percentage</units>
    <spoof>EDGITest|dsp:EDGITest|dsp</spoof>
    <direction>dgsg|boinc</direction>
  </metric_data>
  <metric_data>
    <metric_name>past_workunits_24_hours</metric_name>
    <timestamp>1308046204064</timestamp>
    <value>0.0</value>
    <type>uint32</type>
    <units>wus</units>
    <spoof>EDGITest|fusion:EDGITest|fusion</spoof>
    <direction>dgsg|boinc</direction>
  </metric_data>
  <metric_data>
    <metric_name>cpus_available</metric_name>
    <timestamp>1308046204066</timestamp>
    <value>0.0</value>
    <type>uint32</type>
    <units>cpus</units>
    <spoof>EDGITest|dsp:EDGITest|dsp</spoof>
    <direction>dgsg|boinc</direction>
  </metric_data>
  <metric_data>
    <metric_name>success_rate</metric_name>
    <timestamp>1308046204067</timestamp>
    <value>1.0</value>
    <type>float</type>
    <units>percentage</units>
    <spoof>EDGITest|fusion:EDGITest|fusion</spoof>
    <direction>dgsg|boinc</direction>
  </metric_data>
  <metric_data>
    <metric_name>gflops</metric_name>
    <timestamp>1308046204092</timestamp>
    <value>0.0</value>
    <type>float</type>
    <units>gflops</units>
    <spoof>EDGITest|dsp:EDGITest|dsp</spoof>
    <direction>dgsg|boinc</direction>
  </metric_data>
</report>
```

2. Now use the XML Binding Compiler (*xjc*) command-line tool to generate Java classes that represent the XML Schema that you generated. After this, write a simple program that performs two functions:
    a) Unmarshalls the information contained in the example XML to Java objects (the generated classes will hold the information);
    b) Marshalls the same information, now contained in Java Objects, back to XML.

3. Write an XSL file capable of outputting the XML data into an HTML table. Use a web browser to apply and visualize the transformation (you could also use a Java library, such as Xalan, for this purpose).

4. Let's now try the Java-first approach. In this case you will be writing the Java classes yourself, and using the JAXB notation (e.g., annotations). Check the tutorial first and use JAXB to output the following XML:

a)

```
<?xml version="1.0" encoding="UTF-8"?>
<class>
       <student>
              <name>Alberto</name>
              <age>21</age>
       </student>
       <student>
              <name>Patricia</name>
              <age>22</age>
       </student>
    <student>
              <name>Luis</name>
              <age>21</age>
       </student>
</class>
```

b)

```
<?xml version="1.0" encoding="UTF-8"?>
<class>
```

```
       <student id="201134441110">
             <name>Alberto</name>
             <age>21</age>
       </student>
       <student id="201134441116">
             <name>Patricia</name>
             <age>22</age>
       </student>
    <student id="201134441210">
             <name>Luis</name>
             <age>21</age>
       </student>
</class>
```

c)

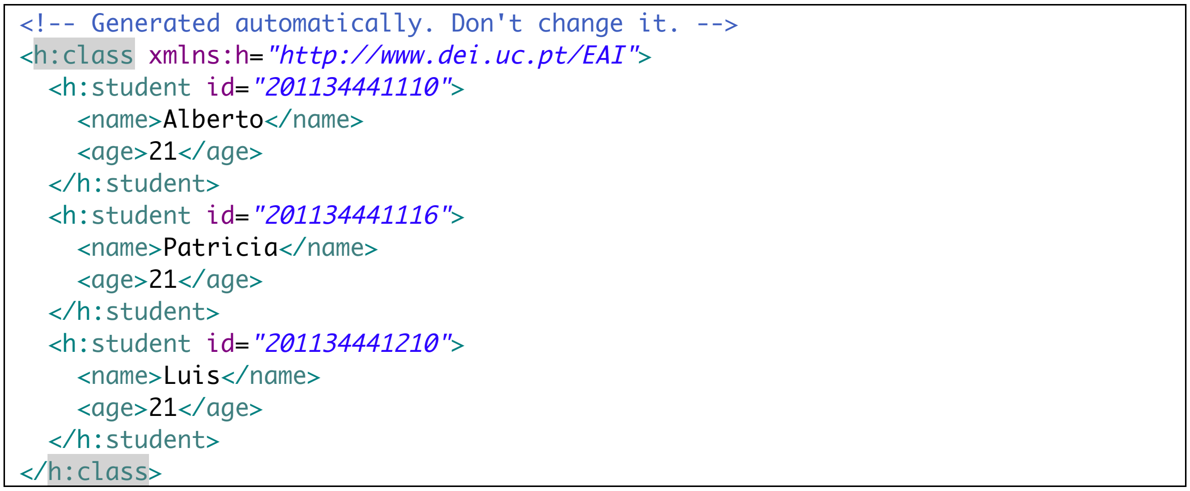
```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated automatically. Don't change it. -->
<class xmlns="http://www.dei.uc.pt/EAI">
  <student xmlns="" id="201134441110">
    <name>Alberto</name>
    <age>21</age>
  </student>
  <student xmlns="" id="201134441116">
    <name>Patricia</name>
    <age>21</age>
  </student>
  <student xmlns="" id="201134441210">
    <name>Luis</name>
    <age>21</age>
  </student>
</class>
```

d)

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated automatically. Don't change it. -->
<h:class xmlns:h="http://www.dei.uc.pt/EAI">
  <student id="201134441110">
    <name>Alberto</name>
    <age>21</age>
  </student>
  <student id="201134441116">
    <name>Patricia</name>
    <age>21</age>
  </student>
  <student id="201134441210">
    <name>Luis</name>
    <age>21</age>
  </student>
</h:class>
```

e)

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="test.xsl"?>
```



```
<!-- Generated automatically. Don't change it. -->
<h:class xmlns:h="http://www.dei.uc.pt/EAI">
  <h:student id="201134441110">
    <name>Alberto</name>
    <age>21</age>
  </h:student>
  <h:student id="201134441116">
    <name>Patricia</name>
    <age>21</age>
  </h:student>
  <h:student id="201134441210">
    <name>Luis</name>
    <age>21</age>
  </h:student>
</h:class>
```

5. Try to manually write the XML Schema Definition for the XML of exercise 4.e).

JMS Training Part (doesn’t count for evaluation)

1. Run the example available at: http://eai-course.blogspot.pt/2012/05/java-message-service-with-jboss-as-7.html.

2. If you remove the `s.close()` from the code of the applications what happens? How do you explain this?

3. Assume now that the sender needs to receive a reply, but you do not want to configure a dedicated queue for that. Which mechanism could you use? Write the necessary code, sending a reply with a set of key-values.

4. Write code that sends text messages to multiple subscribers at once.

5. In the previous code, what happens to the messages that arrive at the topic, before the subscriber actually makes the subscription? Assume now, that a client subscribes a topic, leaves and then subscribes again. We want to know what happens to the messages that enter the topic, while this client is out. Will it receive the messages or not? To ensure that the client receives these messages, which changes do you need to do? Write a new client with these properties and try the code to see if it works. You should check this message: http://eai-course.blogspot.pt/2012/09/a-few-variations-over-jms.html.

6. How do queues behave when there is no receiver? Do they keep the messages or do they drop the messages? Also, what happens if two receivers exist for the same queue? (Relate this question to the 2nd execution question, above).

7. Consider now that you only want to receive messages concerning the Enterprise Application Integration course. How can you avoid the remaining? Will this work, both for queues and for topics? Implement a working example.

8. Assume now that you need to send an XML message for a topic. Which kind of JMS messages should you use? (You do not need to implement this exercise)

9. Explain the parameters of the method `createTopicSession()`. What are the different types of acknowledgment available and what are their differences?

10. Explain the difference between persistent messages and durable subscriptions.

Project Part (for evaluation)

In this assignment you should create three applications. The first one is a *Web Crawler* that collects news from the CNN web site (http://cnn.com), extracts the relevant data to XML, and sends it to a Java Message Service Topic. This Topic serves two other applications that process the data and produce output files. One of the applications, *Stats Producer*, writes statistics information regarding the news site. The other application, *HTML Summary Creator*, summarizes the movies information and creates HTML files for later visualization. Figure 1 illustrates this scenario. The three applications are described in the following paragraphs.

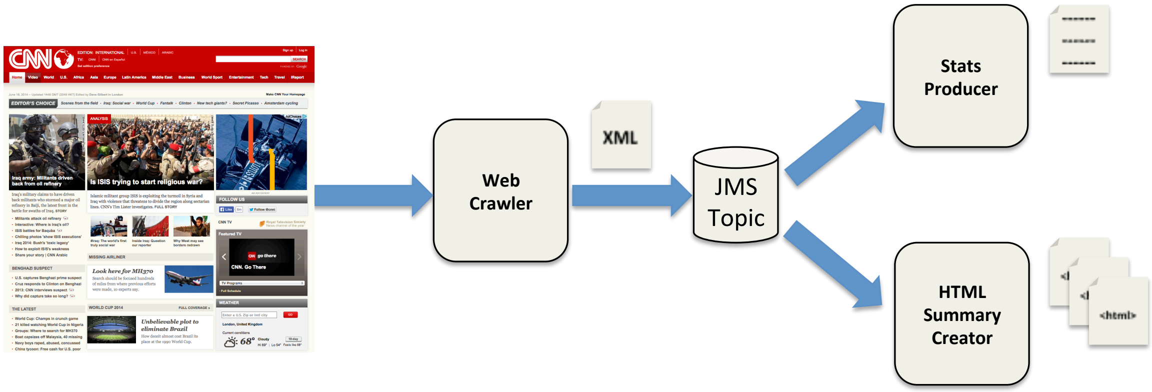


Figure 1 – The information flow

The Web Crawler

The *Web Crawler* is a stand-alone command-line application that reads a web page and sends an XML message to a JMS Topic, carrying details of news from the CNN site. You should use an HTML parser (e.g., Jsoup), to get the data from the web page. **You should not parse the web page directly using regular expressions.** Nevertheless, you are allowed to use regular expressions to extract small pieces of data from the results of the HTML parser. For example, you might find a string that looks like "June 18, 2014" and use regular expressions to extract the different parts of the date.

Once you get the DOM document of the web page, you will need to convert it to XML. If this helps, you can do as follows, although creating the Java classes from scratch is also possible:

- Define the XML schema (this may involve the *trang* tool, to create XSD from an XML sample). **You must include an XML schema file (XSD)** as part of your final submission and be ready to explain it;
- From the XML schema, generate the Java classes using the XML binding compiler, *xjc*);
- Once you have the Java classes that can keep the data, you can instantiate and use them in the normal way, in the *Web Crawler* source code.

Each time the *Crawler* runs, it parses the web page, creates and populates the Java objects that keep the web site's data, outputs an XML document to a JMS message and publishes this message on a JMS Topic. If the topic is down for some reason, you should keep a log with the data that the *Crawler* was unable to publish. In this case, the Crawler should stay in a cycle for a limited amount of time, retrying to publish. The next time it starts, the Crawler automatically checks if there is any unpublished message to send.

You are responsible for defining the exact format of the XML messages. For your reference, you should consider that each message must contain a list of news from the different regions on top of the page (U.S., Africa, Asia, Europe, Latin America, and Middle East). Each news item should carry information like the title, URL, highlights, date, author, news text, photo and video URLs if available. If the website is missing some data you find interesting for the assignment, you can add it to the XML, provided you contact the Professor previously.

Although you only need one site and HTTP access, design your *Crawler* so that:

- Changing web site does not require too much effort;
- Changing to another input data source (e.g., FTP, file access) is simple.

Finally, **keep some test HTML files in your disk**, just in case the website changes.

HTML Summary Creator

This application should be permanently running, waiting for XML messages from the JMS topic. This application must create an HTML file, using the XML files coming from the Topic (keep one file per each reading of the *Crawler*). For this, you should use an XSL template for transforming the resulting XML file into HTML.

You are pretty much free to define the HTML as you want, but you should include all the news, photos and videos collected by the *Crawler*. Organization of the page is important. Use a web browser with a built-in XSLT engine (e.g., Firefox) to apply the transformation and display the resulting HTML page.

Note: Use durable subscriptions to ensure that even if the *HTML Summary Creator* fails, the Topic will keep the messages for later retrieval.

Stats Producer

The purpose of this application is to keep track of the most recent news (based on their creation date). For the sake of simplicity, you are only required to store the information about the number of news having less than 12 hours. You should keep this information on persistent storage (a file would be enough). Finally, the *Stats Producer* should also keep a durable subscription on the Topic, to read all the *Crawler*'s messages, even if the *Stats Producer* fails.

Grading

Grading is performed according to:

- The quality of the data model used for representing data (XML/XSD). Please note that it is better to have the most accurate possible description of the XML nodes (e.g., integer instead of string, if possible).
- The quality of the code (modularity, formatting, comments, code conventions, etc.);
- Simplicity of the solution, including the screen scraping part;
- Final presentation of the work.

The project should be made in groups of **2** students. On your final report you should mention who was mostly involved in what part. Write it down explicitly. Also, we do expect all the members of the group to be fully aware of all the parts of the code that is submitted. Work together!