



Budapest University of Technology and Economics
Department of Measurement and Information Systems
Fault Tolerant Systems Research Group

Service Integration course

Bonita Connectors

Oszkár Semeráth

Gábor Szárnyas

April 6, 2014

Contents

- 1 Integrating Bonita** **2**
- 1.1 Introduction 2
- 1.2 Creating a Connector 2
- 1.3 Database Integration 3
- 1.4 Web Services 4
- 1.4.1 SOA Web Service 5
- SoapUI 7
- 1.4.2 REST Web Service 7
- 1.5 Creating a SOAP Connector 7
- 1.6 Creating a REST Connector 9
- 1.7 Working with Bonita from Java 10

Chapter 1

Integrating Bonita

Source code snippets and URLs are available in <https://svn.inf.mit.bme.hu/edu/trunk/mdsd/handout/public/2014/>.

A detailed guide in Hungarian is available in <https://njszt.hu/sites/default/files/Informatikai-Navigator-10.pdf>.

1.1 Introduction

1. Start the **Bonita BPM Community** application and choose the **ApplicationStore (1.0)** process.
2. Start the workflow and inspect it. The **Show the application names** step shows the applications available in our application store. Currently this list is generated by the Groovy connector of the **Download the application names** task.
3. Go to the **Connectors** of the **Download the application names** task and delete the Groovy connector.

1.2 Creating a Connector

1. A connector consist of a **definition** (interface) and an implementation.
2. First, define the **definition**. Go to **Development | Connectors | New Definition...** Set the **Definition id** to **ApplicationConnector** and the **Category** to **Script**. Click **Next, Next**. On the the **Output** page, click **Add...*** and set the **Name** to **applications** and the **Type** to **java.util.List**. Click **Finish**.
3. Go to **Development | Connectors | New Implementation**. Pick the **ApplicationConnector** and set the package to **hu.bme.mit.inf.sysint**.
4. An Eclipse JDT-based Java editor will show up with the **ApplicationConnectorImpl** file. Edit the **executeBusinessLogic** method to get the following.

```
@Override
protected void executeBusinessLogic() throws ConnectorException {
    //Get access to the connector input parameters

    //WARNING : Set the output of the connector execution.
    //If outputs are not set, connector fails
    List<String> applications = new ArrayList<String>();
    applications.add("App1");
}
```

```

applications.add("App2");
setApplications(applications);
}

```

5. The connector can be tested in **Development | Connectors | Test connector**. Choose the **ApplicationConnector** and click **Test** and **OK**.
6. Add this connector to the **Download the application names** task. On the **Output operations** page, set that the **applications** variable takes the value of the **applications** output of the connector.
7. Delete the connector from the **Download the application names** task.
8. Test the workflow.

1.3 Database Integration

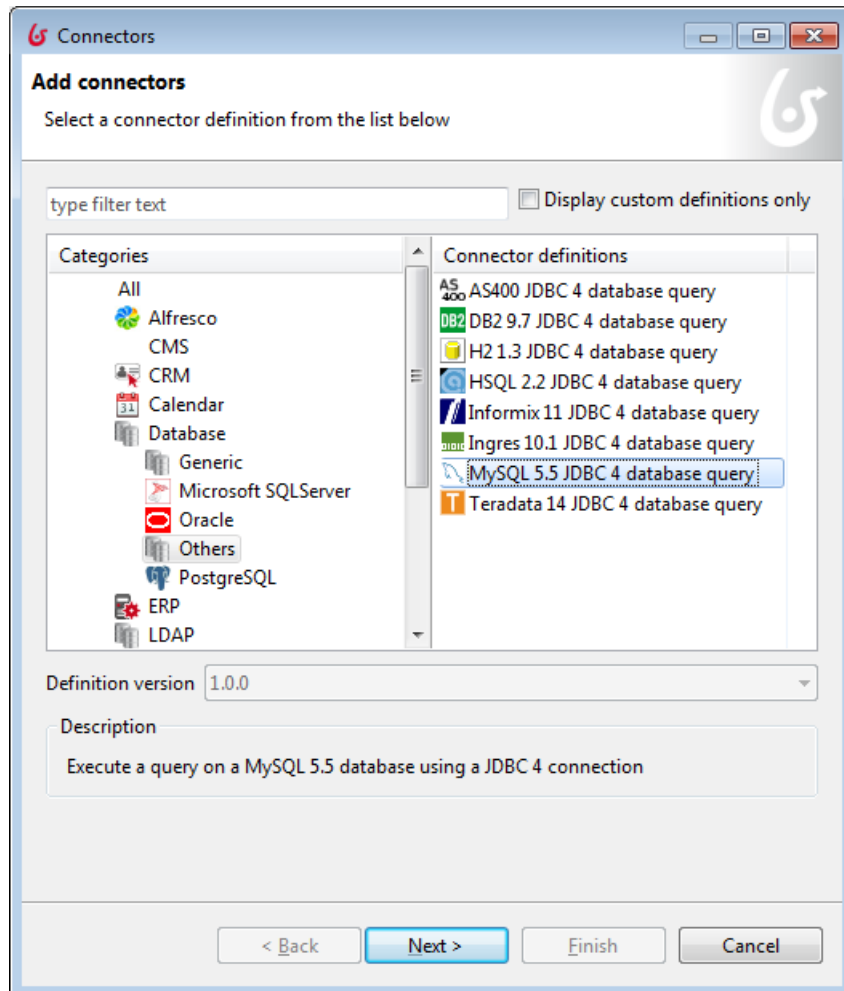


Figure 1.1: Database connector configuration

1. In this section, we integrate our workflow with a MySQL database. Start the **MySQL Workbench 6.0 CE** application.

2. Choose the **SQL development** section and select the **applications** database.
3. The username and password are both root.
4. An SQL development environment will appear. We have created a database table for the workflow: select the **application** table in the **applications** database, **right click** and select **Select Rows - Limit 1000**.
5. The query and the results will appear with the applications Flashlight, News and Weather.
6. Go back to **Bonita**. Select the **Download the application names** task and add a **MySQL 5.5 JDBC4 database query** connector. Name it **MySQLConnector** and click **Next**.
7. On the **Database access information** page, set the following and click **Next**:
 - Driver: `com.mysql.jdbc.Driver`
 - URL: `jdbc:mysql://localhost:3306/applications`
 - Username: root
 - Password: root
8. On the **Enter query** page, use the **Switch editor** button to switch to a single line editor (defining a **Constant** expression). Else you would get an error message (*Unsupported expression type for testing: ...*).
Set the **Query or Batch Script** field in the next page:


```
SELECT * FROM application;
```
9. Click **Test configuration**. It will throw the following error: **java.lang.reflect.InvocationTargetException org.bonitasoft.engine.exception.BonitaRuntimeException: java.lang.ClassCastException: com.mysql.jdbc.JDBC4ResultSet cannot be cast to java.io.Serializable.** The reason for this is that we first have to process the **JDBC4ResultSet** object to a serializable representation.
10. On the **Output operations definition page** set that the **applications** variable takes the value of a Groovy script. The content of the script is the following:


```
def res = []
while (resultset.next()) {
  res.add(resultset.getString("name"))
}
res
```

Set the return type to `java.util.List`
11. Use the **Back** button and press **Test**. The connector should work now and list the applications available in the database.

1.4 Web Services

1. You can start the Tomcat server with the **Start Tomcat** shortcut on the Desktop. The WAR files in the `C:\tomcat\webapps` directory are deployed.
2. You may wish to set the port configuration of the server in the `server.xml`. In the laboratory, we used the following configuration: `Connector port="80" protocol="HTTP/1.1", Connector port="89" protocol="AJP/1.3".)`
3. On the Tomcat server, you have two web applications available: a JAX-WS SOA web service and a JAX-RS REST web service. Both offer the same functionality: they generate an arbitrary number of `Application` objects.

1.4.1 SOA Web Service

To test the SOA web service, open the **Google Chrome** browser and start the **Advanced REST Client** plug-in. Set the **URL** to <http://localhost:80/appstore-ws/services/ApplicationManager> and the **HTTP method** to **POST**. Add a header field SOAPAction (with an empty value) to the **Headers**. Paste the following code to the **Payload** field.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <generateApplications xmlns="http://ws.server.appstore.inf.mit.bme.hu">
      <count>20</count>
    </generateApplications>
  </soapenv:Body>
</soapenv:Envelope>
```

Notes: we can generate the SOA envelope with the **Eclipse WTP platform**. If you generate the client (as seen in the *web service laboratory*), you can observe the SOA envelope in the **TCP/IP monitor**.

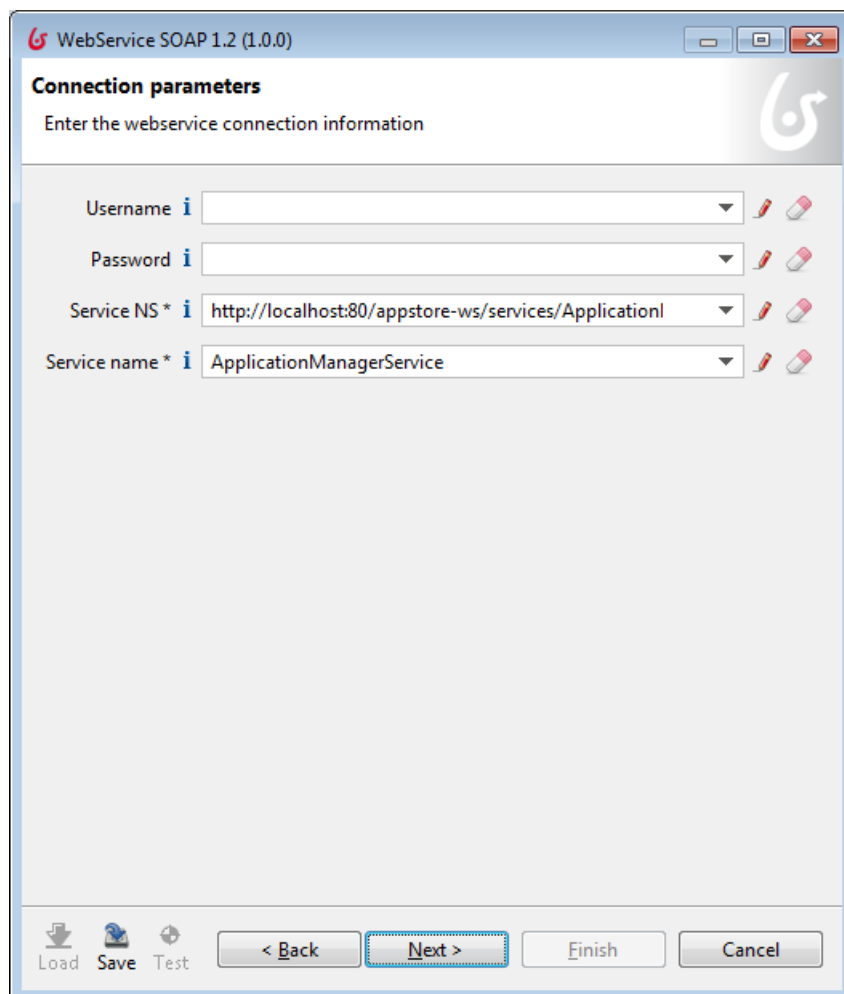


Figure 1.2: SOA web service

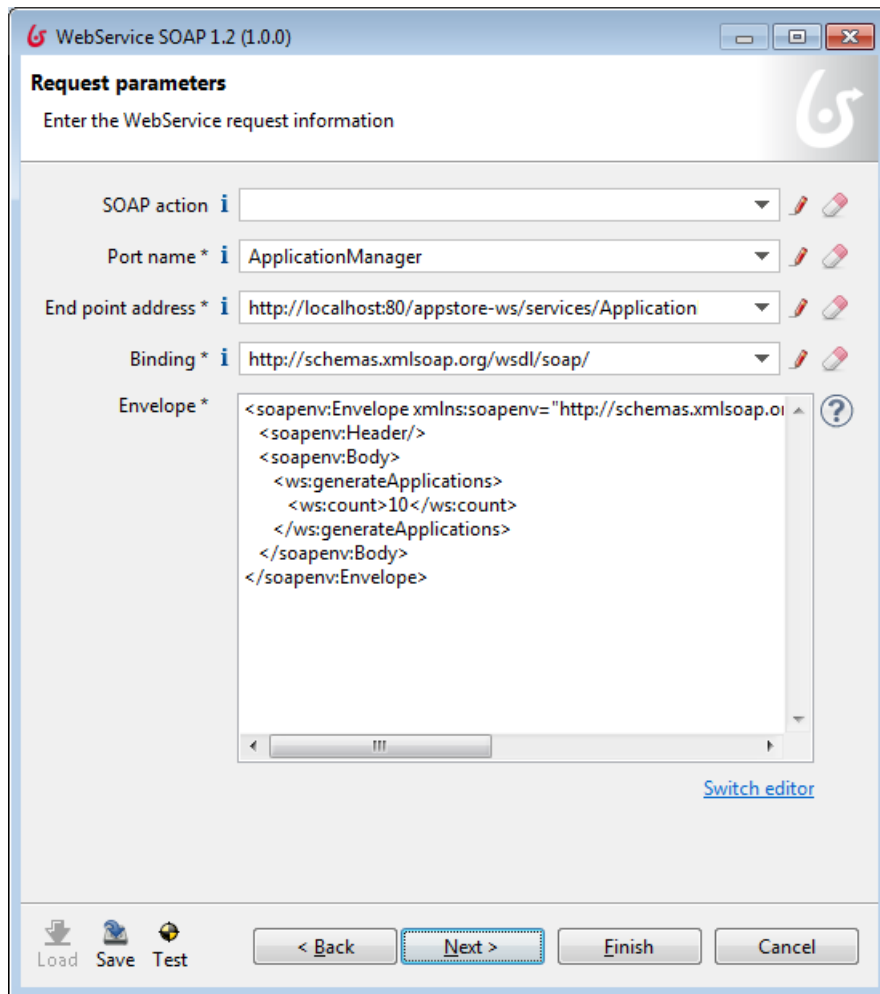


Figure 1.3: SOA web service

SoapUI

SoapUI (<http://www.soapui.org/>) is a tool capable of generating SOA envelopes from the WSDL file.

Just create a **New SOAP Project**, add the `ApplicationManager.wsdl` WSDL file as **Initial WSDL**. Tick **Create Requests** and click **OK**. The SOA envelope will be generated.

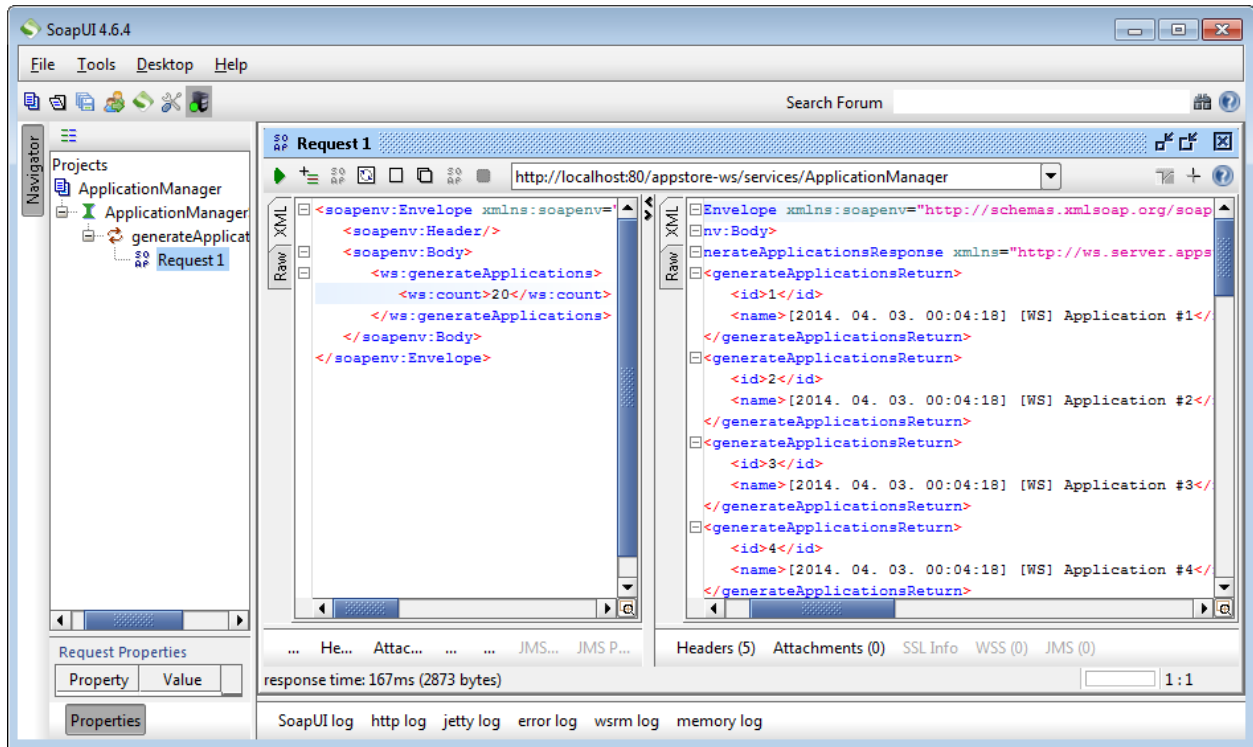


Figure 1.4: SOA web service

1.4.2 REST Web Service

To observe the REST web service, simply visit <http://localhost:80/appstore-rest/rest/applicationmanager/generate/20>.

1.5 Creating a SOAP Connector

1. Go to **Connectors, Add...** and add a **SOAP Web Services | WebService SOAP 1.2** and name it `ApplicationRestConnector`.
 - Service NS: `http://localhost:80/appstore-ws/services/ApplicationManager`
 - Service name: `ApplicationManagerService`
 - Port name: `ApplicationManager`
 - End point address: as above, `http://localhost:80/appstore-ws/services/ApplicationManager`
 - Binding: `http://www.w3.org/2003/05/soap/bindings/HTTP/`
 - Envelope: same XML as above.

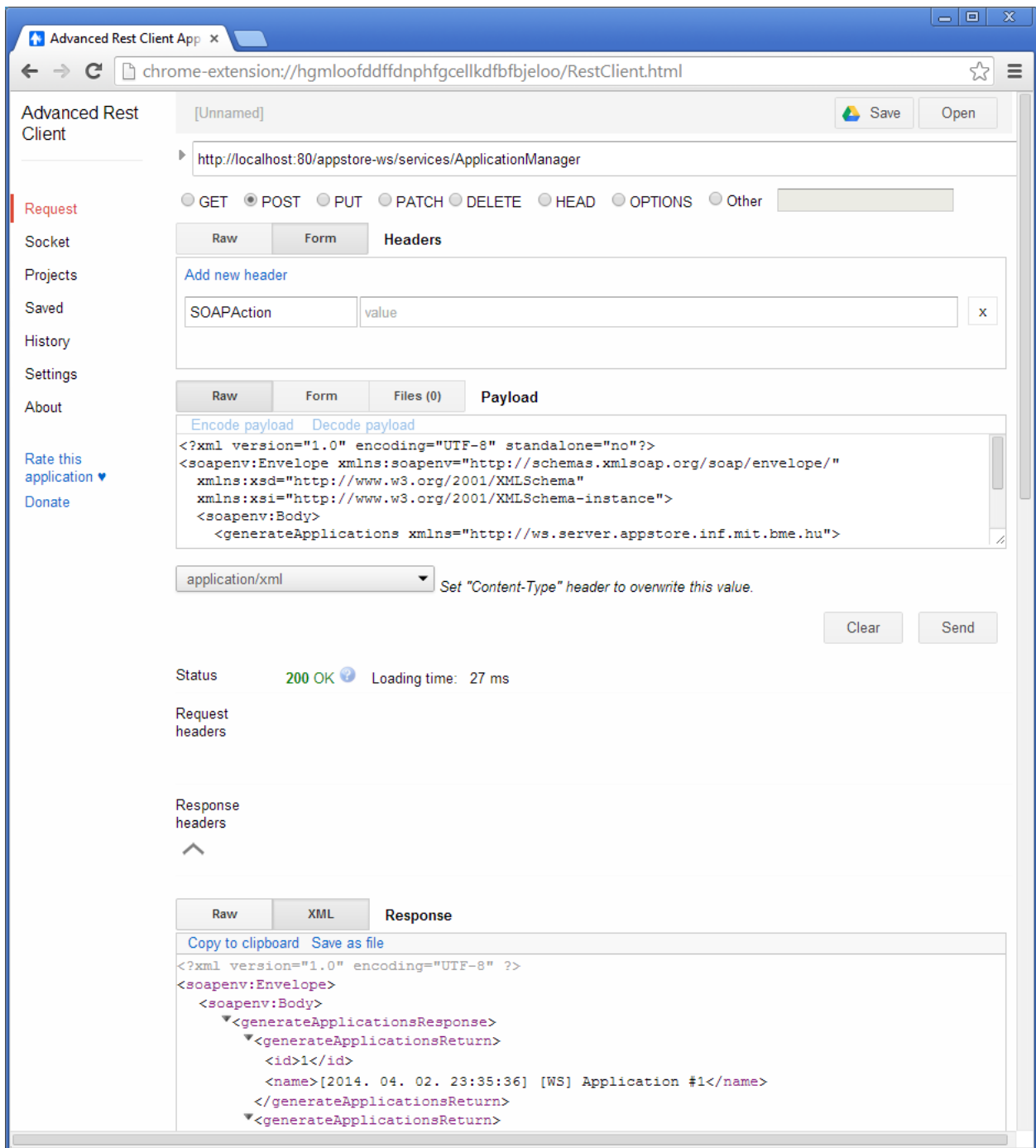


Figure 1.5: Using the **Advanced REST client** for accessing the REST service

2. If you encounter the following exception, change the envelope accordingly: *Caused by: com.sun.xml.internal.ws.protocol.soap.VersionMismatchException.: Couldn't create SOAP message. Expecting Envelope in namespace http://www.w3.org/2003/05/soap-envelope, but got http://schemas.xmlsoap.org/soap/envelope/ ()*.
3. The exception indicates that the error is in the SecureWSCConnector class. A quick web search will guide you to the source code of the class: <https://github.com/bonitasoft/bonita-connector-webservice/blob/master/bonita-connector-webservice-impl/src/main/java/org/bonitasoft/connectors/ws/cxf/SecureWSCConnector.java>.
4. As of now, the error is unresolved. Please drop an email to the author if you have found the solution.

1.6 Creating a REST Connector

1. Create a new implementation for the **ApplicationConnector** definition.
2. Edit the Java code to get the following:

```

@Override
protected void executeBusinessLogic() throws ConnectorException {
    ArrayList<String> results;
    URL url;
    try {
        url = new URL("http://localhost:80/appstore-rest/rest/applicationmanager/generate/3");
    } catch (MalformedURLException e1) {
        throw new ConnectorException(e1);
    }
    URLConnection connection;
    try {
        connection = url.openConnection();
        Document document = parseXmlDom(connection.getInputStream());
        results = new ArrayList<String>();
        NodeList apps = document.getElementsByTagName("applications").item(0)
            .getChildNodes();
        for (int i = 0; i < apps.getLength(); i++) {
            results.add(apps.item(i).getTextContent());
        }
    } catch (IOException e) {
        throw new ConnectorException(e);
    }

    setResults(results);
}

public static Document parseXmlDom(InputStream is) {
    Document document = null;
    try {
        // getting the default implementation of DOM builder
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        dbf.setValidating(false);
        dbf.setIgnoringComments(true);
        dbf.setIgnoringElementContentWhitespace(true);
        dbf.setNamespaceAware(true);
        DocumentBuilder builder = dbf.newDocumentBuilder();
    }
}

```

```
    // parsing the XML file
    document = builder.parse(is);
} catch (Exception e) {
    // catching all exceptions
    System.out.println(e.toString());
}
return document;
}
```

1.7 Working with Bonita from Java

You can work with a Bonita workflow from java using by using the following dependency. The corresponding dependency is:

```
<dependency>
  <groupId>org.bonitasoft.engine</groupId>
  <artifactId>bonita-client</artifactId>
  <version>6.2.6</version>
</dependency>
```

Do not forget to set the version to the actual version number of your Bonita installation.

The BonitaAPI .zip file in https://svn.inf.mit.bme.hu/edu/trunk/mdsd/handout/public/2014/bonita_connector_materials/ contains an example Java code.