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 29, 2013

Contents

Chapter 1

Bonita

Source code snippets and URLs are available at https://svn.inf.mit.bme.hu/edu/trunk/mdsd/handout/public/bonita_connector_materials/.

1.1 Introduction

1. Start the Bonita Studio and choose "MyProcessDiagram (1.0)" process.
2. We have changed the browse workflow a bit: it serve as an application collection that can be browsed or extended from various sources.
3. Start the workflow and inspect it: at this page the application available in our database should be displayed. Currently this list is empty.
4. There are buttons in the lower part of the page that shows the available options of extension sources. Those sources are services. Select the WebService option and go to the next page.
5. The **New apps** page displays the newly downloaded pages. Those applications will be inserted to the database. If you click next the first page will be displayed. At the end of the lecture this list will show the newly inserted applications too.
6. Choose **None** and end the workflow.

1.2 Scripting Reminder

1. At first substitute the web service with a script on "Update by web service": `["New"]`.
2. This list should be directed to the newApplications variable, and replacing it.

1.3 Database Integration

1. Lets integrate our workflow with a MySQL database. Start the MySQL Workbench 5.2 CE.
2. Choose the SQL development section and select the Local MySQL database.
3. A dolphin in an aquarium will authenticate you: the password is `root`.

4. An SQL development environment will appear. We have created a database for the workflow: select the application table in the applications database, **right click | Select Rows - Limit 1000**.

5. The query and the result will appear with the single application called `EvilStore`.

6. Our applications will be saved to this table. This query will be copied to the workflow, so do not close it.

7. Select the Load the application names task and add a connector: **Database | MySQL**. Name it to `ApplicationQuery` and fill out the form as described in the picture below.

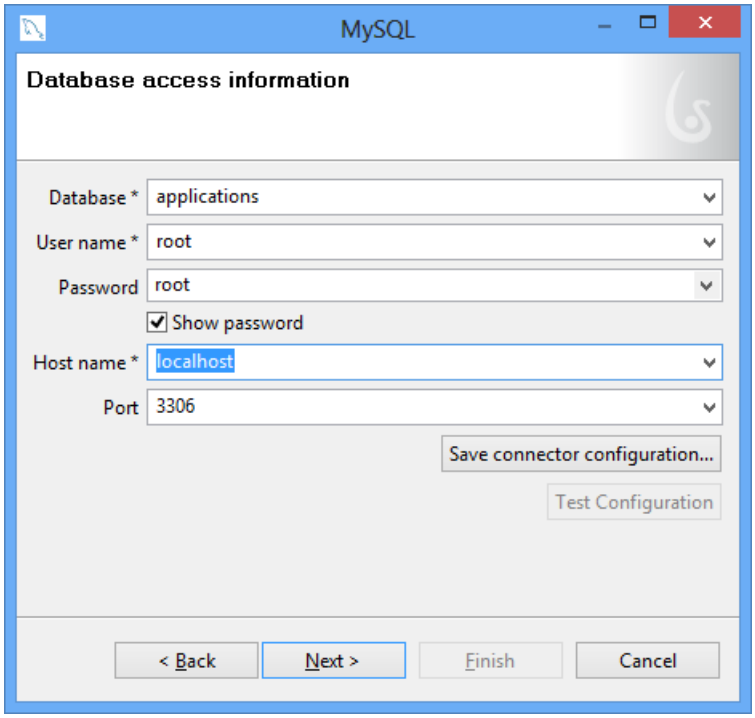

Figure 1.1: Bonita db connector configuration

8. Fill the **Query** field in the next page:

```
SELECT * FROM applications.application
```

9. Click **Test configuration**:

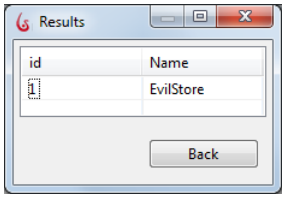

Figure 1.2: Bonita db test

10. Direct the values of the `rowset` to the Applications variable (`java rowSet.getValues()`). Select the replace strategy.

11. Now write the insert script in the "Save to DB" task. Add a new MySQL connector to the task, fill it in the same way.

12. Write the script as a return value of a method:

```
String ret =
  "INSERT INTO `applications`.`application` "+
  "(`Name`) VALUES "
boolean hasItem = false
for(item in newApplications) {
  if(hasItem) ret+=","
  ret+="('"+item.replaceAll("[\\W]", "")+"')"
  hasItem = true
}
return ret
```

13. Test it with the evaluate button. The result will be (explain why):

```
INSERT INTO `applications`.`application` (`Name`) VALUES ('O'),('b'),('j'),('e'),('c'),('t')
```

14. Run the workflow, the result should be inserted to the table.

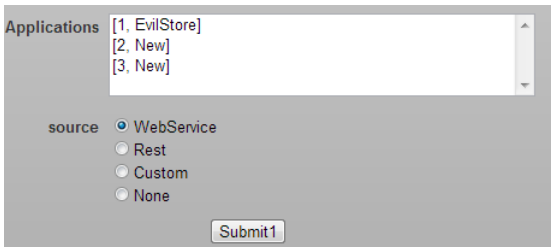

Figure 1.3: Bonita db test

1.4 Web services

You can start the Tomcat server with the `start-tomcat.bat` batch file on the Desktop. The `C:\tomcat\webapps` files are deployed

On the Tomcat server, you have web applications available: a JAX-WS SOA web service and a JAX-RS REST web service.

You have to web services available. 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, start Google Chrome's Advanced REST Client plug-in. Set the URL to http://localhost:8080/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.

SoapUI

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

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

http://www-inf.int-evry.fr/cours/WebServices/TP_Workflow/publish_news.html

1.4.2 REST web service

To observe the REST web service, simply visit http://localhost:8080/appstore-rest/rest/applicationmanager/generate/20.

- Target NS: `http://localhost:8080/appstore-ws/services/ApplicationManager`
- Service name: `ApplicationManagerService`
- Port name: `ApplicationManager`
- Request: same XML as above.
- End point address: as above, `http://localhost:8080/appstore-ws/services/ApplicationManager`
- Binding: `http://schemas.xmlsoap.org/wsdl/soap/`

Write the following code:

```
import javax.xml.transform.dom.DOMSource;

def output = (DOMSource) response
def ret = []
def apps =
   output.getNode().
   childNodes.item(0).
   childNodes.item(1).
   childNodes.item(0).
   childNodes

for(int i=0; i<apps.length; i++){
   ret.add(apps.item(i).textContent)
}
return ret
```

1.5 Creating a new connector

1. Go to **Connectors | New Connector**, and name it to `MyRestConnectorForApplications`.

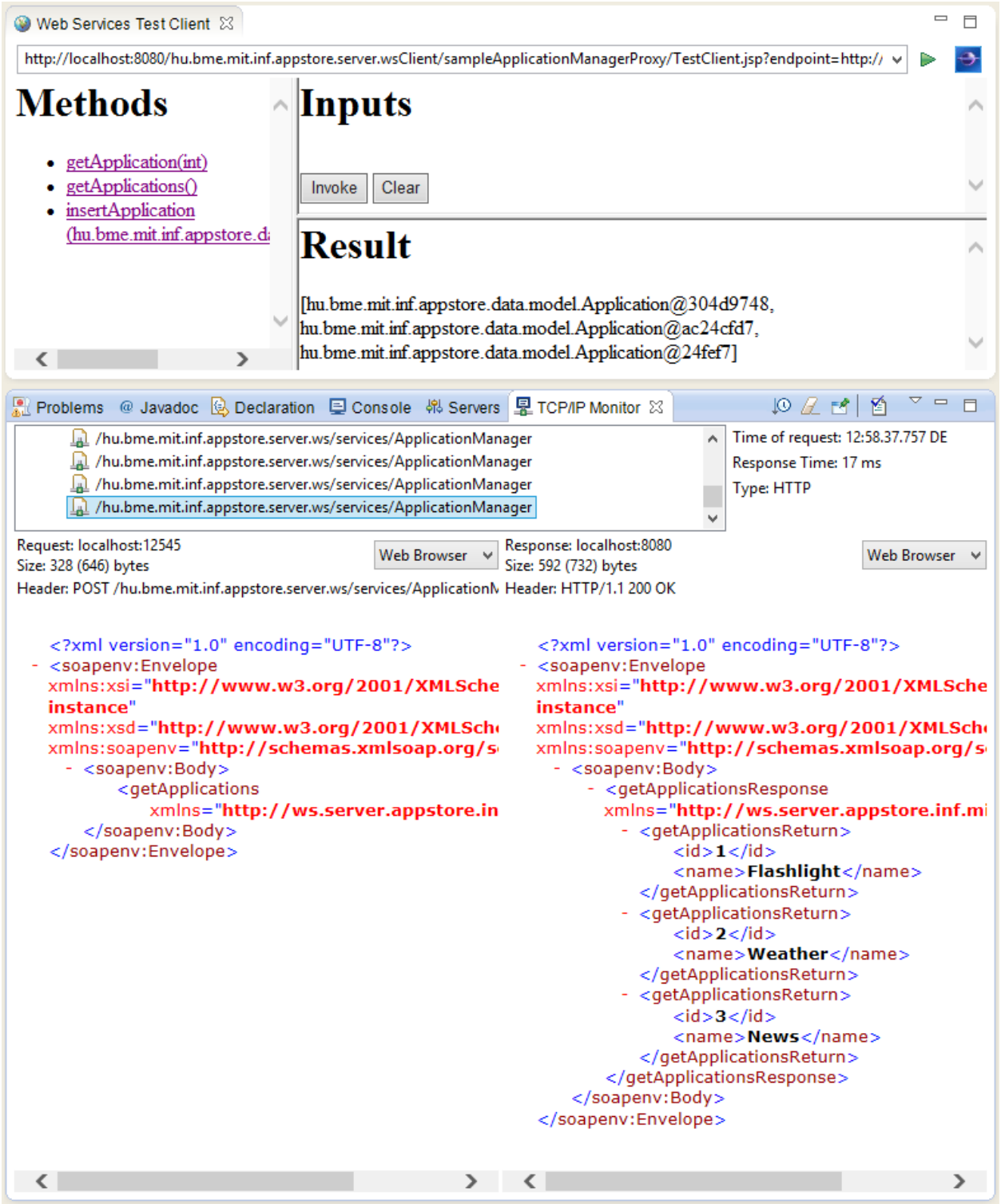

Figure 1.4: The SOA envelope can be observed in the TCP/IP monitor

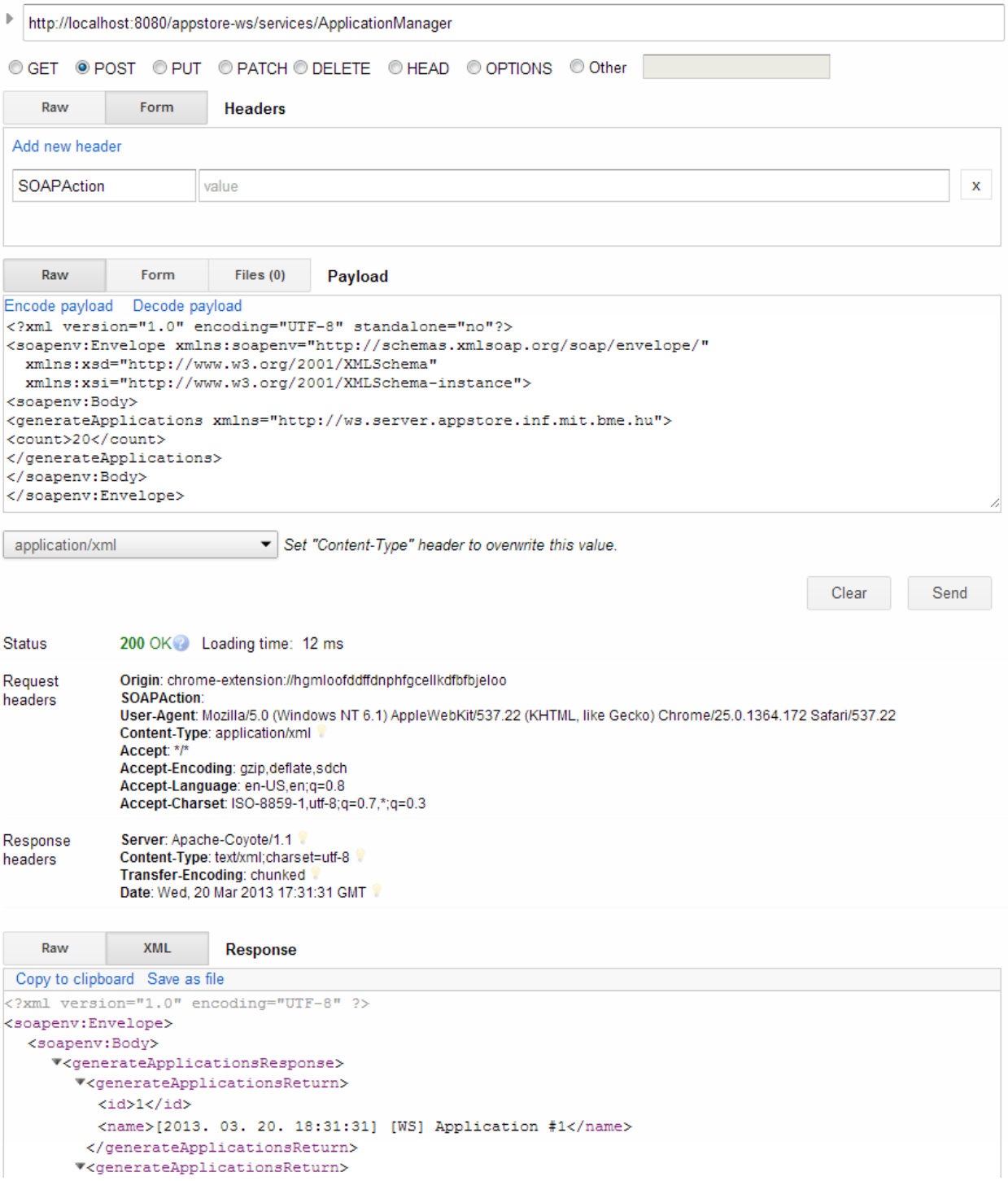

Figure 1.5: SOA web service

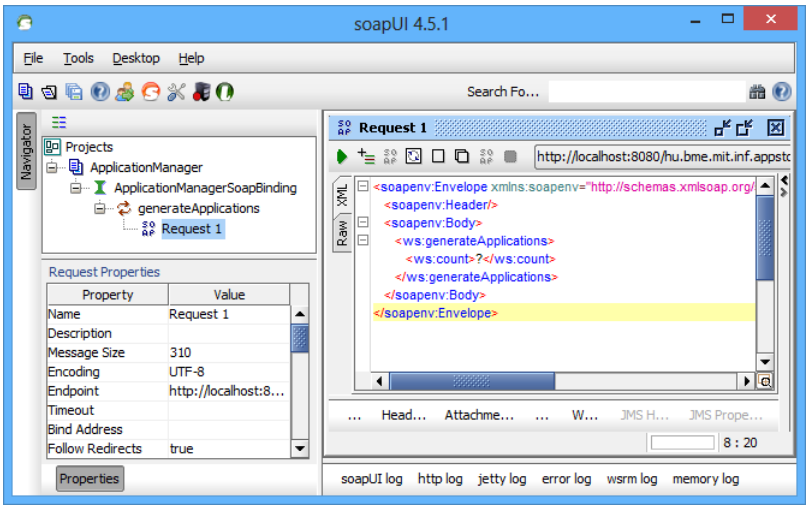

Figure 1.6: SOA web service

2. Add an output parameter named `result` with `List` type.

3. Write the following Java code:

```
import java.util.ArrayList;
import java.util.List;

import org.ow2.bonita.connector.core.ConnectorError;
import org.ow2.bonita.connector.core.ProcessConnector;

public class MyRestConnectorForApplications extends ProcessConnector {

  private ArrayList<String> results;

  @Override
  protected void executeConnector() throws Exception {
    results = new ArrayList<String>();
    results.add("A");
    results.add("B");
  }

  @Override
  protected List<ConnectorError> validateValues() {
    // TODO Auto-generated method stub
    return null;
  }

  /**
   * Getter for output argument 'result'
   * DO NOT REMOVE NOR RENAME THIS GETTER,
   * unless you also change the related entry in the XML descriptor file
   */
  public java.util.ArrayList getResult() {
    return results;
  }
```

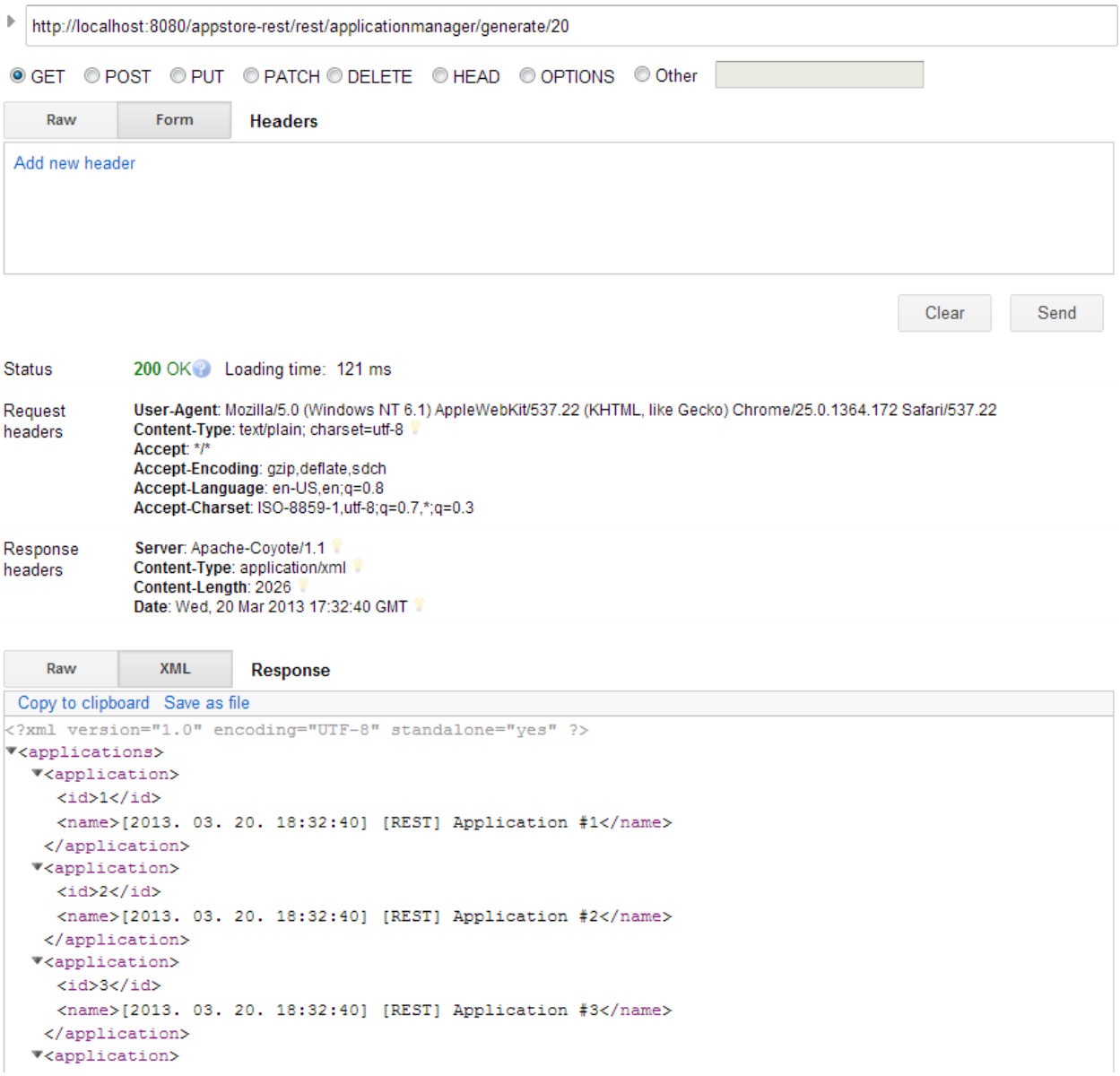

Figure 1.7: REST web service

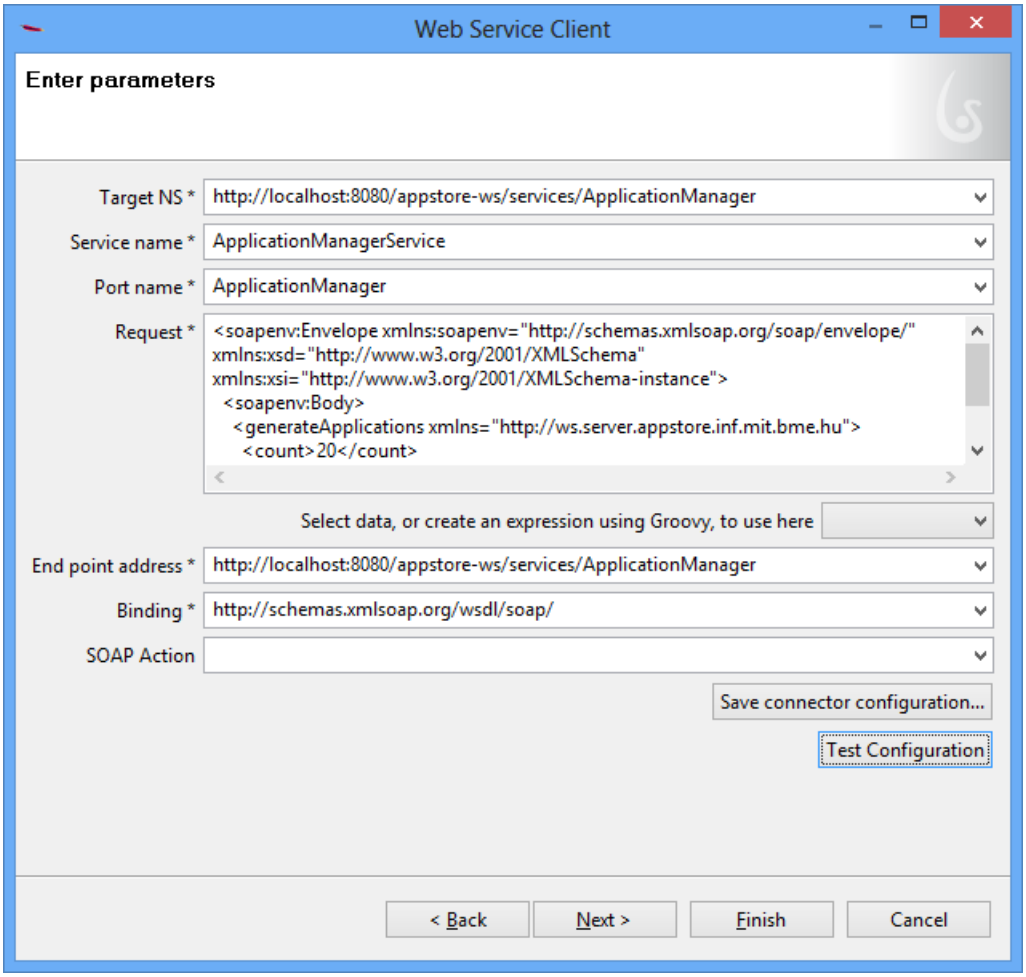

Figure 1.8: **Web Service Client** dialog

```
}
```

4. Put it into the workflow, use **replace** strategy.
5. Edit the code to get the following:

```
import java.io.InputStream;
import java.net.URL;
import java.net.URLConnection;
import java.util.ArrayList;
import java.util.List;

import org.ow2.bonita.connector.core.ConnectorError;
import org.ow2.bonita.connector.core.ProcessConnector;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

public class MyRestConnectorForApplications extends ProcessConnector {

  private ArrayList<String> results;

  @Override
  protected void executeConnector() throws Exception {

    URL url = new URL("http://localhost:8080/appstore-rest/rest/applicationmanager/generate/3");
    URLConnection 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());
    }
  }
  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;
  }

  @Override
  protected List<ConnectorError> validateValues() {
    // TODO Auto-generated method stub
    return null;
  }

  /**
   * Getter for output argument 'result'
   * DO NOT REMOVE NOR RENAME THIS GETTER,
   * unless you also change the related entry in the XML descriptor file
   */
  public java.util.ArrayList getResult() {
    return results;
  }
}
```