Middleware for the IoT - TP 2

REST client and XML mapping

Objective

In this session you will develop a REST client with required XML mapping modules for automatic interactions with the OM2M resource tree. Then, the developed modules will enable to perform RESTful communications between M2M monitoring applications and simulated M2M sensors and actuators through the OM2M platform.

Prerequisite / Used libraries

- Eclipse Luna
- Apache HTTP client lib
- Jetty HTTP server lib
- JAXB lib (included in jdk since 1.7)
- oBIX lib
- oneM2M XML files

1. RESTful client

In this part you will develop a Client class to manage the M2M resource tree based on the HTTPClient API from Apache.
First, **download and import** the sample project available on moodle. (File > Import > Maven > import maven project)
Most of the following snippets are already implemented in the sample classes provided in the project.

You can store the response of the platform in a simple Response Class:

```
public class Response {
	String representation;
	int statusCode;
}
```

The Client Class should implement the following "ClientInterface" interface:

```
public interface ClientInterface {
	public Response retrieve(String url, String originator);
	public Response create(String url, String representation, String originator, String
type);
	public Response update(String url, String representation, String originator);
	public Response delete(String url, String originator);
}
```

Therefore you will find some basic snippets you can use to implement the Retrieve, Create, Update, Delete, RESTful methods. Do not forget to replace the parameters with the correct ones. (Refer to what you learned in the first TP to have the information to use in the requests)

```
// Instantiate a new Client
CloseableHttpClient client = HttpClients.createDefault();
// Instantiate the correct Http Method
HttpPost post = new HttpPost(url);
// add headers
post.addHeader(name, value);
// add the body to the request
post.setEntity(new StringEntity(representation));
try {
      // execute the HTTP request and receive the HTTP response
      CloseableHttpResponse reqResp = client.execute(post);
      // get the status code of the response
      response.setStatusCode(reqResp.getStatusLine().getStatusCode());
      // get the body of the response
      response.setRepresentation(IOUtils.toString(reqResp.getEntity().getContent(),
      "UTF-8"));
} catch (Exception e1) {
      e1.printStackTrace();
} finally {
      client.close();
}
```

You can use the following method to read an XML representation from a file (located in the util package):

```
public static String getRequestFromFile(String fileName) {
      Scanner sc = new Scanner(
                  RequestLoader.class
                              .getResourceAsStream("/requests/" + fileName));
      String payload = "";
      while (sc.hasNextLine()) {
            payload += sc.nextLine() + "\n";
      }
      sc.close();
      return payload;
}
```

You can customize the XML representations, create new files, etc., as you need and want. The files must be located in the **main/resources/requests** folder.

1.1. Implementation

Based on the general information above, implement the following steps in a main class to test your client API. You can read resource representation directly from XML files for the body of the requests.

1) Retrieve the attributes of the CSE Base with child resources (query string rcn=5)
2) Register an "appTest" resource (create an AE)
3) Retrieve the "appTest" resource attributes
4) Update the "appTest" resource
5) Retrieve the new "appTest" resource
6) Create a "cntTest" container resource
7) Retrieve the "cntTest" resource
8) Create a "cinTest" content instance resource
9) Retrieve the "cinTest" resource

2. XML Mapping

2.1. JAXB mapping

JAXB is a tool that enables the marshalling and unmarshalling of resources. The marshalling process consists in transforming the in memory representation of an object into a specific data format suitable for exchange. Typically, it is used when your data must be exchanged between different parts of a software or from one software entity to another. The unmarshalling is the reverse operation.

You can find more detailed information in this JAXB COOKBOOK.

In the provided code, you can find the `MapperInterface` which defines the two operations mentioned above. Those two operations have been implemented in the Mapper class. You can use them to convert objects to String (XML) and to convert a String representation (XML) to an object.

TODO: Test the marshall and unmarshall implementations by creating a resource and serializing it in a String (XML) and transform it back to object. You can use the `MapperTest` Class to do so.

2.2. oBIX mapping

In this section you will use the oBIX library to generate oBIX/XML representations.
Learn to use and test the oBIX library:

- create an oBIX object and customize it. A little sample to help you:

`<obj>`
`  <bool name="value" val="true"/>`
`  <str name="location" val="home"/>`
`</obj>`
	`Bool value = new Bool(true);`
	`Str location = new Str("home");`
	`Obj object = new Obj();`
	`object.add("value", value);`
	`object.add("location", location);`

- Encode the oBIX object to an XML string and print it, using this method:
  `String result = ObixEncoder.toString(obj);`
- Decode the oBIX XML string to an oBIX object using this method:
  `Obj obj = ObixDecoder.fromString(content);`

Once you understood how to use the oBIX tool, you can move on to the test section.

2.3. Test

Now, you will use your Client and the Mapper classes with the oBIX API to implement the following test:

1) Create an APPLICATION ENTITY (AE) without specifying any name for it nor AE id. A unique resource id (ri) and a unique Application Entity ID (aei) will be generated by the oneM2M platform.
2) Create a CONTAINER (CNT) without specifying a name, under the previously created AE. To do this, you have to get the resource ID given by the platform once the AE is created and use it as the location for your container creation.
3) Create a CONTENT INSTANCE (CIN), without specifying any name, under the previously created CNT. To do this you will need the resource id of the CNT as before. the content Instance should contain the following oBIX content: (use the oBIX API to create an obj and serialize it)
4) Retrieve the CIN resource, decode the oBIX content and print the sensor data value.

NB:

When creating a resource, the **resource id** is returned in the **Content-Location** header in the response or in the body in the **ri** attribute.

3. Monitoring Application

In this part you will implement a monitoring application that will use an HTTP server to receive oneM2M notifications. The objective of this part is to launch the HTTP server, to open a socket, listen on a specific context and be able to decode the content sent by the platform.

Use the snippet provided in the `Server` class to implement the monitoring application.
You can use the following method to help you get the oBIX content from the received notification:

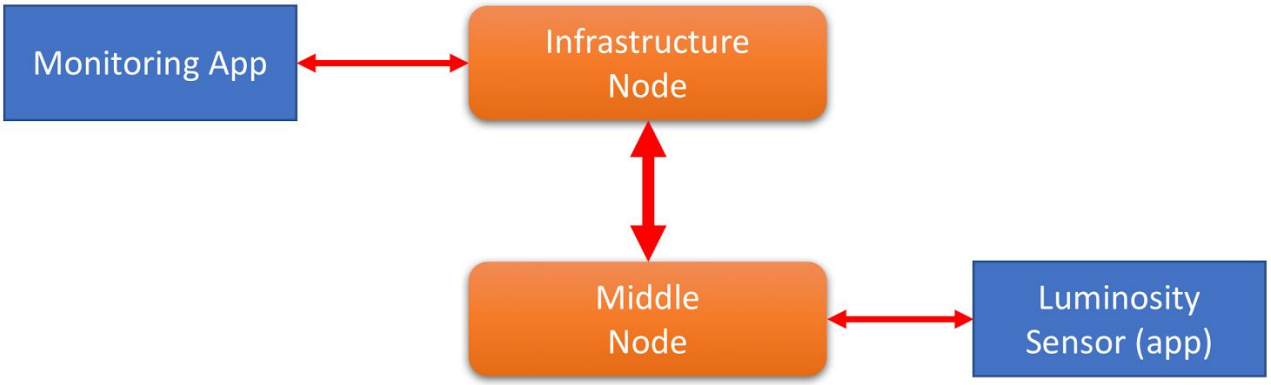
```
public static final Obj getObixFromNotification(Notification notification);
```

NB:

When a subscription is created, a **verification request (vrq)** will be sent to the specified subscription uri to test the validity of the subscription. You have to identify this specific notification and answer to it with a **STATUS OK (200)**.

To test your server, create an application representing your monitoring app and be careful to indicate the **request reachability (rr)** of it to **true** and add a **point of access (poa)** pointing to your listening server. (protocol://ip:port/context). Then, create a subscription resource which will use as POA the URI of the monitoring application, in order to receive notifications when a new instance of data will be pushed in the platform. After that, decode the received notification, the received content instance and the oBIX content. Print the new sensor data value.

4. Scenario



In this section you will implement a oneM2M application simulating a luminosity sensor. This application will send periodically (each 10s for instance) measurements to the platform. The value will be chosen randomly between 0 and 600 lux.
Implement a monitoring server application that will subscribe to the luminosity sensor and follow this behaviour:

- pring **HIGH** when the value of the sensor is **greater than 400**
- print **LOW** when the value of the sensor is **lower than 300**
- print **GOOD** if the value is greater than 300 but **lower than 400**

What you learned during this session

- Implement your own RESTful oneM2M client
- Implement your own monitoring application
- Use the JAXB tool to interact with XML representations and objects
- Use the oBIX library to interact with oBIX representations and objects
- Implement a scenario based on these tools