Middleware for the IoT - TP 1

Interact with the oneM2M RESTful architecture using Eclipse OM2M

Objective

These sessions will explain how to interact with the Eclipse OM2M platform (eclipse.org/om2m)

- configure and launch the platform
- handle the resource tree through a web browser using AJAX interface (JS)
- handle the resource tree though a REST client.

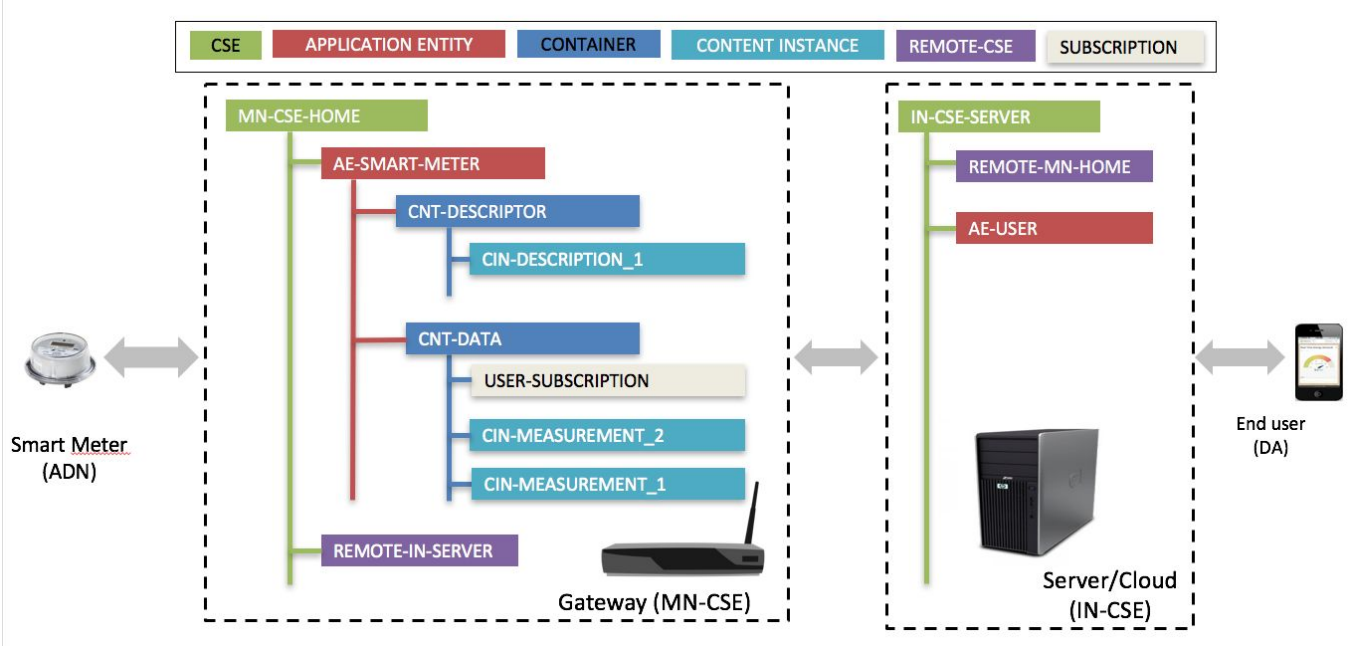

Figure 1: Resource Tree Example

Prerequisite

1) **Java Version 1.7 AT LEAST**
   Configure the "JAVA_HOME" and "PATH" environment as follows if needed:
   - `JAVA_HOME=/usr/local/jdk1...`
   - `PATH=$JAVA_HOME/bin:$PATH`
2) **Install a REST Client**. We propose **Postman** available at https://www.getpostman.com/
3) **Eclipse Luna** (for RCP and RAP developers) or newer (for TP2 and beyond)
   https://eclipse.org/downloads/packages/eclipse-rcp-and-rap-developers/lunasr2

Steps

Retrieve the platform

For the lab session:
Download the binaries available at the Machine to Machine Moodle course.

FACULTATIVE: Clone and build the platform yourself, following this tutorial:
https://wiki.eclipse.org/OM2M/one/Clone
THIS IS NOT REQUIRED FOR THE LAB SESSION. However, it is still recommended to do it at home to have some experience of downloading and compiling such kind of platform.

Configure the platform

Read this tutorial to learn how to configure the OM2M platform:
https://wiki.eclipse.org/OM2M/one/Configuration

Start the platform

To start the IN and a MN, use this tutorial: https://wiki.eclipse.org/OM2M/one/Starting

Handle the resource tree

In this part we will use the "**POSTMAN - REST CLIENT**" an extension of chrome browser, to **CREATE**, **UPDATE**, **RETRIEVE** and **DELETE** resources in the resource tree. These procedures correspond respectively to **POST**, **PUT**, **GET**, **DELETE** methods of the REST client.

Developer interface

To access to the developer interface connect to http://<ip>:<port>/webpage
For example with the default configuration you can use: http://localhost:8080/webpage
Use `admin` as user and password if you have not changed the default configuration. You can use the interface to explore the existing resources.

To have a simple example and browse the oneM2M resources, start the **ipe sample** plugin. Go to the MN shell, type **`ss ipe`** and then **`start <id>`** with the bundle id of the sample ipe plugin.

Interact with the REST API

Authentication: for the originator of a request to be recognized you must add a oneM2M specific header with HTTP. The Header is formatted as follows:

`x-m2m-origin: <originator>`

You can add this header in the header field of the REST client. Use "admin:admin" as originator to begin with and to interact with the REST API and handle the resource tree, refer to this tutorial:
https://wiki.eclipse.org/OM2M/one/REST_API

You can use the Postman application which provides a good interface to build HTTP requests.
A pattern collection is available here for postman:
https://www.getpostman.com/collections/1ebf7b31ec43e97fd12c.

Access Control Management in oneM2M

The ACP Resource

To manage the access of oneM2M resource, a specific resource exists that represents the *rights* given to an entity. This resource is named **AccessControlPolicy (ACP).** Each other oneM2M resource is linked to a set of ACP via the `acpi` attribute.

This resource has 2 specific attributes: *privileges* and *self-privileges*. Both have a list of rules that represent authorized entities with a linked set of operations they are allowed to perform.

- *privileges (`<pv>`)* correspond to the access rights that will be applied to the linked resources. For instance, an `AE_1` will have `ACP_1` has an linked ACP. It means the `ACP_1` identifier will be listed in the attribute acpi of the `AE_1`. Than the *privileges* list of rules will be applied when an entity tries to access or change *AE_1*.
- *self-privileges (`<pvs>`)* provide the rules that will be apply to the current ACP. For instance, only a rule for the admin entity may be present in an ACP, allowing only the admin to genre the Access Rules itself.

The attributes of the rules are:

- Access Control Originator (`<acor>`): the list of originators to whom the rule will be applied (entities are separated by a blank space)
- Access Control Operation (`<acop>`): the list of operations granted to the originators

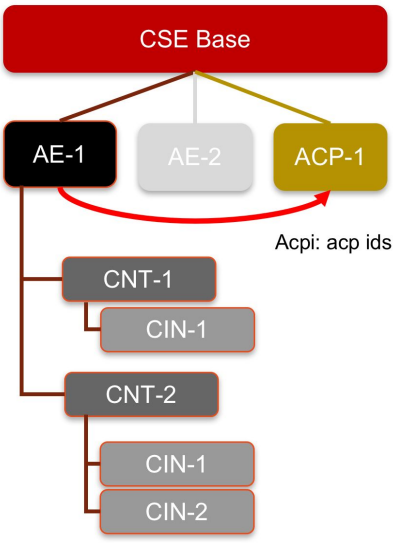

Figure 2: Link Between an AE & an ACP, list of ACP identifiers (uri)

Access Control Operation

The operations allowed by the originator in the Access Control Rule is defined by an integer. This integer is the sum of the Operation value. The following table shows the available operations and their corresponding value.

Access Control Operation	**Value**
CREATE	1
RETRIEVE	2
UPDATE	4
DELETE	8
NOTIFY	16
DISCOVERY	32

For instance, if a rule is aimed to provide rights to perform RETRIEVE + DISCOVERY + NOTIFY operations, the value of the parameter will be 2 + 16 + 32 = 50.

Exercise

The aim is to create an ACP and attach it to an already created data container from the previous exercise.
The aim is to give access to 2 different entities:

- a monitoring application that will only retrieve the data
- a sensor application that will be able to retrieve AND create

To create the ACP resource

- Parent resource: `/<cse-id>`
- ResourceType: 1
- Template payload for ACP creation:

```
<m2m:acp xmlns:m2m="http://www.onem2m.org/xml/protocols" rn="MY_NAME">
    <pv>
        <acr>
            <acor></acor>
            <acop></acop>
        </acr>
    </pv>
    <pvs>
        <acr>
            <acor></acor>
            <acop></acop>
        </acr>
    </pvs>
</m2m:acp>
```

Scenario

Realize this scenario based on the previous sections. The aim of this scenario is to simulate several sensors and to register an application that will monitor the new values pushes by the sensors using the Subscription/Notification mechanism.

1) **Create 3 AE on the MN with the following names:**
    a) SmartMeter
    b) LuminositySensor
    c) TemperatureSensor

Each application representing each device should have 2 containers with the following names: **DESCRIPTOR** and **DATA**

2) **Create the content instances as indicated below:**
    - The DESCRIPTOR container should have 1 content instance containing the description of the sensor. Use the following oBIX model for the content representation

```
<obj>
    <str name="Type" val="Sensor" />
    <str name="Category" val="Light" />
    <str name="Unit" val="Lux" />
    <str name="Model" val="1142_0" />
    <str name="Location" val="Home" />
    <str name="Manufacturer" val="PHIDGETS" />
    <str name="Consumption Max" val="27 mA" />
    <str name="Voltage Min" val="4.8 V DC" />
    <str name="Voltage Max" val="5.3 V DC" />
    <str name="Operating Temperature Min" val="0 C" />
    <str name="Operating Temperature Max" val="70 C" />
</obj>
```

    - The DATA container should store the measurements values of the sensor. Use the following oBIX model for the content representation

```
<obj>
    <str name="Category" val="Light"/>
    <str name="Data" val="300" />
    <str name="Unit" val="Lux" />
    <str name="Location" val="Home" />
</obj>
```

3) **Start the monitoring application**

4) **Create a new AE with a *request reachability attribute* (rr) at *true* and a *point of access* (poa) with the url of the monitor.** Create required subscriptions to enable the AE representing the monitoring application to receive electricity, luminosity and temperature data when new values are stored in the platform.