INTERNET OF THINGS COURSE
UNIVERSITY OF PARMA

Hands on CoAP: Exercises

Ing. Andrea Gorrieri (andrea.gorrieri@gmail.com)

Ing. Luca Davoli (davoli@ce.unipr.it)

Dott. Ing. Marco Picone (marco.picone@unipr.it)

08-12 May 2015

1 Introduction

Welcome to the Hands On CoAP tutorial! You will learn how to develop RESTful IoT applications based on CoAP. We will connect various devices and services. For this, we will use different CoAP libraries and frameworks, each with its own focus and benefits. You will need the Californium (Cf) CoAP framework and the Contiki OS containing Erbium (Er). We recommend to use the Eclipse IDE for its good Java support. This tutorial assumes that you already finished the “Contiki OS and Cooja Simulator” tutorial and you have a basic understanding of the Java language.

2 Getting Started

Before we dive into the first CoAP exercise, make sure you have a working development environment. If you are working with the virtual machine provided in the Contiki OS and Cooja Simulator tutorial, simply open with Eclipse the project named *IoTCourseProject*:

File -> Import -> General -> Existing Projects into Workspace -> Browse -> Select the project

3 Cf Tutorial Server

First, you will implement your own CoAP server with Californium. You can then test with Copper (Cu) which is a plugin for the Firefox browser which acts as CoAP client.

3.1 Create Main Server Class

The central class that implements the server will also contain the main. Your server class must extend the CoapServer class from the Californium framework. In Eclipse, right-click on /src/main/java and choose:

- New > Class
  - Package: `it.unipr.tlc.iot2015.cf.server`
  - Name: `TutorialServer`
  - Superclass: `org.eclipse.californium.core.CoapServer`
  - Tick public static void main(String[] args)
  - Finish

In the main method, you should run the server. To do so, do the following:

- Create an instance of the `TutorialServer` class
- Start the server

```
public static void main(String[] args) {
        TutorialServer tutorialServer = new TutorialServer();
        tutorialServer.start();
 }
```

Now execute your main class by right-clicking and choosing *Run > Run As > Java Application*.
You should see status logs in the console.

3.2 Add Resources

You can already query your CoAP server with the Copper (Cu) plugin, by typing in Firefox:

coap://127.0.0.1:5683/

in particular the address 127.0.0.1 corresponds to the IP address of your machine (localhost) and 5683 is the default port for CoAP (could be omitted). When you browse your CoAP server with Copper (Cu), it will already reply to pings and provide the */.well-known/core* resource for discovery.
However, there are no resources on your server yet. Let's start adding the obligatory "HelloWorld!"

Create a new resource class in a resources sub-package under `/src/main/java`:

- New > Class
  - Package: `it.unipr.tlc.iot2015.cf.server.resources`
  - Name: `HelloWorldResource`
  - Superclass: `org.eclipse.californium.core.CoapResource`
  - Finish

Once completed the creation of the CoAP resource, implement its constructor, that it has to accept a string parameter. In order to build correctly the CoAP resource, in its constructor the first instruction must be the call to the super constructor, that owns all properties to instantiate a CoAP resource.

- The constructor parameter, passed to the super constructor, represents the name of the resource, with which you can refer during interactions through CoAP clients (e.g., Copper).

```
public class HelloWorldResource extends CoapResource {

    //Constructor
    public HelloWorldResource(String name) {
       super(name);
    }
}
```

Californium manage CoAP requests based on the type of incoming instance. Since that our purpose is to build a HelloWorld resource that reply to simple GET request, what we have to implement is a component that is able to handle GET demands.

- Declare a GET handler, modifying the behavior of the following method:
    - `public void handleGET(CoapExchange exchange)`
- Let the handler respond with “Hello World!”, using a feature of the provided `exchange`:
    - `exchange.respond("Hello World!”);`

Since you have defined only the payload into the GET handler, this means that your resource reply to a request creating a CoAP message with:

- Code: `2.05`
- Payload: `Hello World!`
- Payload MIME-type: `text/plain`

```
public class HelloWorldResource extends CoapResource {

    public HelloWorldResource(String name) {
       super(name);
    }

    @Override
    public void handleGET(CoapExchange exchange) {

      exchange.respond(ResponseCode.CONTENT,
                       "Hello World!",
                       MediaTypeRegistry.TEXT_PLAIN);
    }
```

```
	}
```

Now your HelloWorld resource is ready, but your Californium server doesn't know the existence of this new resource. You have to add your new HelloWorld resource to the Californium server, to be able to query it.

- In your Californium CoAP server implementation, add the following instructions, before starting it:
    - `HelloWorldResource hello = new HelloworldResource("hello-world");`
    - `tutorialServer.add(hello);`

Finally, stop every instances of your Californium server (to avoid that your newly deployed behavior doesn't reflect server work); then run your Californium server and browse it with Copper.
You should be able to receive a "Hello World!" by making a GET call on the "hello-world" resource (also note the Content-Format in the response).

Now it's time to improve your Californium server! Go ahead and add more resources that also handle POST, PUT, or DELETE requests, in an identical fashion of the HelloWorld resource.

3.3 Observing Resources

We now create an observable resource that informs all observers whenever its state changes.

- Create a new Californium CoAP resource called `ObservableResource` , in a similar manner to that described for the `HelloWorld` resource
- Implement the GET handler, so that it responds with the currently stored data.

To make the resource observable, do the following:

- In the TutorialServer Class, use the constructor in order to create a new instance of the `ObservableResource` class:

```
ObservableResource obsRes = new ObservableResource("observable-resource");
```

Then make it observable by writing:

```
	obsRes.setObservable(true);
	obsRes.getAttributes().setObservable();
```

Don't forget to add the resource to the server:

```
	tutorialServer.add(obsRes);
```

Add a private varible to the class `ObservableResource,` which will store the current value of the resource, and initialize it to zero.

```
      private int mValue = 0;
```
As already done for the `hello-world` resource implement the handleGET method in order to reply to GET calls:

```
      public void handleGET(CoapExchange exchange) {
             exchange.respond(ResponseCode.CONTENT,
                              mValue+"",
                              MediaTypeRegistry.TEXT_PLAIN);
      }
```

Now the Observable resource can be accessed via Copper with GET calls and can be observed as well. However, if you try to observe the resource you will note that the observed value is always 0! This is because your variable do not change with time! In order to let the variable change over time you can simply copy-paste this code in the constructor of the class:

```
      Timer timer = new Timer();
      timer.schedule(new UpdateTask(this), 0, 1000);
```

You have also to define the UpdateTask class: copy-paste this code into the `ObservableResource` class:

```
private class UpdateTask extends TimerTask {

       private CoapResource mCoapres;

       public UpdateTask(CoapResource coapres) {
             mCoapres = coapres;
       }

       @Override
       public void run() {
             mValue = new Random().nextInt(20);
             mCoapres.changed();
       }
    }
```

Now the value of your resource change periodically with a period of 1 second and takes a random value beetween 0 and 20. You should see this behaviour by observing the resource with copper.

4 Cf Tutorial Client

Until now, you have used the Copper plugin in order to make query to your CoAP server. Now is time to implement your own java implementation of a CoAP client using the Californium libraries.

4.1 Synchronous CoAP client

We are going to create a simple CoAP client which make a GET call to a predefined resource. In order to do so, as already done for the CoAP server, you have to create a new class with a "main" method. You can denote this class as `SimpleCoapClient` and you will use it in order to run your CoAP client implemetation:
In Eclipse, right-click on /src/main/java and choose:

- New > Class
    - Package: `it.unipr.tlc.iot2015.cf.client`
    - Name: `SimpleCoapClient`
    - Tick public static void main(String[] args)
    - Finish

In you "main" method create a new object of the class `CoapClient` and use its constructor in order to set the address of the resource you want to get (if the server resides on your machine its address will be as usual: 127.0.0.1:5683):

```
CoapClient coapClient=new CoapClient("coap://127.0.0.1:5683/<resource
name>");
```

Then you have to declare a `Request` object. `Request` represents a generic CoAP request (could be GET, POST, PUT and DELETE). In order to create a GET request you can use its constructor:

```
    Request request = new Request(Code.GET);
```

Now you can actually send the GET request and print the CoAP Server response by writing:

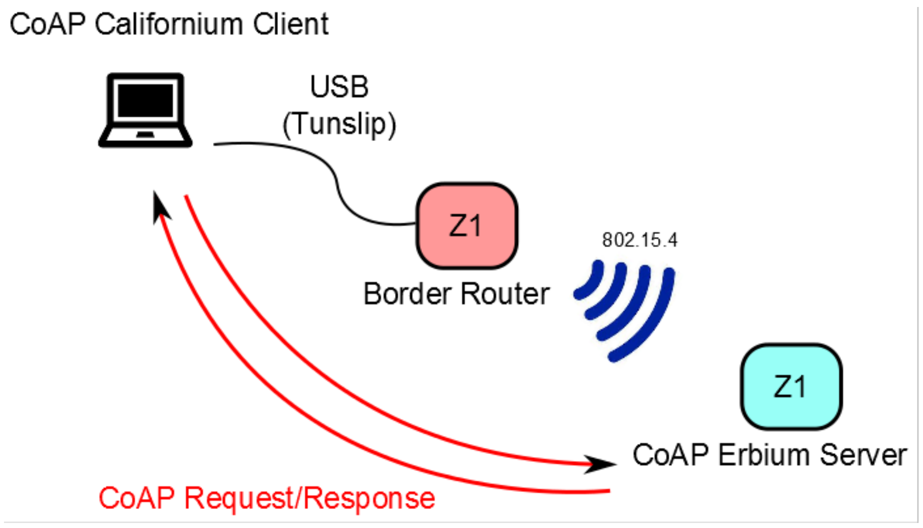
```
    //Synchronously send the GET message (blocking call)
    CoapResponse coapResp = coapClient.advanced(request);

    //The "CoapResponse" message contains the response.
    System.out.println(Utils.prettyPrint(coapResp));
```

Try to query your CoAP server with your CoAP client. Note that this is a **synchronous** implementation of a CoAP client. This means that when the `coapClient.advanced(request);` method is invoked, the program wait for the response (blocking call) before proceeding with the other instructions. The Californium libraries provide also an asynchronous implementation of the CoAP client as well as the observing functionalities.

5 Cf Tutorial Client + Er Tutorial Server

In the “Contiki OS and Cooja Simulator” tutorial you have learned how to create a CoAP server with the Contiki OS by means of the Erbium CoAP libraries. Now you can try to query a contiki Erbium CoAP server (running on a Zolertia Z1) with the Java Californium CoAP client (running on your machine). The configuration is depicted in the following Figure:



Your laptop has to be connected to a border router which is in turn connected to the contiki CoAP Erbium server by means of 802.15.4 link.
In order to configure this 6LoWPAN infrastructure:

- Flash one Zolertia Z1 node with the `border-router` from `contiki/examples/ipv6/rpl-border-router`:
  `make TARGET=z1 border-router.upload`
- Flash a second node with the er-example-server from `contiki/examples/er-rest-examples/er-example-server` :
  `make TARGET=z1 er-example-server.upload`
- Connect tunslip6 to your border router using make:
  `make connect-router`

Now the configuration should be active and you should be able to query the CoAP Erbium Server with your Californium Java client. Note that when you type: `make connect-router` the address of the border router is returned. So you can query the http server of the border router (as already done in the contiki turial) with a simple browser in order to get the IP addresses of all the nodes connected to the border router. In this way you can retrieve the IP address of the Erbium Server.