CoAP Applications

Carlo Vallati

Associate Professor @ University of Pisa

carlo.vallati@iet.unipi.it - http://www.iet.unipi.it/c.vallati/

Java - Californium

Apache Maven - Concept

- Software project management tool
- Based on project object model (POM)
- POM is the fundamental unit (XML file)
  - information
  - configuration
  - dependencies
- de facto standard for large Java projects
- Java library exported in Maven repositories

POM - Minimal

This is an example of a minimal pom.xml file. The file describe the metadata associated with the application and the packaging.

```
<project>
      <modelVersion>4.0.0</modelVersion>
      <groupId>it.unipi.iot</groupId>
      <artifactId>iot-example</artifactId>
      <version>1.0</version>
      <packaging>jar</packaging>
</project>
```

Project inheritance

```
<project>
    <modelVersion>4.0.0<modelVersion>
    <parent>
        <groupId>it.unipi.iot</groupId>
        <artifactId>iot-example</artifactId>
        <version>1.0</version>
    </parent>
    <groupId>it.unipi.iot</groupId>
    <artifactId>server</artifactId>
    <version>1.0</version>
    <packaging>jar</packaging>
</project>
```

|--/iot-example
| --/server
| -- pom.xml
|--pom.xml

A pom.xml can inherit the configuration from a parent project

Project aggregation

```
<project>
    <modelVersion>4.0.0<modelVersion>
    <groupId>it.unipi.iot</groupId>
    <artifactId>iot-example</artifactId>
    <version>1.0</version>
    <packaging>pom</packaging>
    <modules>
        <module>server</module>
    </modules>
</project>
```

```
|--/iot-example
|    --/server
|        -- pom.xml
|--pom.xml
```

The parent project is configured as a collection of subprojects

Dependencies

```
<dependencies>
   ...
   <dependency>
       <groupId>org.eclipse.californium</groupId>
        <artifactId>californium-core</artifactId>
        <version>1.1.0-SNAPSHOT}</version>
   </dependency>
   ...
 </dependencies>
```

- Your project can rely on other project
- Specify each project as a dependency
- See it as “import jar into library path”

Import from where?

Repositories

```
</repositories>
…
    <repository>
        <id>repo.eclipse.org</id>
        <name>Californium Repository</name>

        <url>https://repo.eclipse.org/content/repositories/californium/</url>
    </repository>
 …
</repositories>
```

- Each pom.xml inherits the standard repository
- Some projects, i.e. Californium, have their repository

Plugins

```
<build>
        <plugins>
        …
        </plugins>
</build>
```

- Specify the plugins to be used for compilation and packaging
- Used to construct the final jar

Eclipse plugin & compiler plugin

```
<plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.3.2</version>
</plugin>
<plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-dependency-plugin</artifactId>
        <executions>
                <execution>
                        <id>copy-installed</id>
                        <phase>install</phase>
                        <goals>
                                <goal>copy</goal>
                        </goals>
                        <configuration>
                                <artifactItems>
                                        <artifactItem>

        <groupId>${project.groupId}</groupId>

        <artifactId>${project.artifactId}</artifactId>

        <version>${project.version}</version>

        <type>${project.packaging}</type>
                                        </artifactItem>
                                </artifactItems>

        <outputDirectory>../run/</outputDirectory>
                        </configuration>
                </execution>
        </executions>
</plugin>
```

Assembly plugin

```
<plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-assembly-plugin</artifactId>
            <configuration>
                        <appendAssemblyId>false</appendAssemblyId>
                        <archive>
                                    <manifest>
                                                <addClasspath>true</addClasspath>
                                                <mainClass>iot.server.MyServer</mainClass>
            <addDefaultImplementationEntries>true</addDefaultImplementationEntries>
                                    </manifest>
                        </archive>
                        <descriptorRefs>
                                    <descriptorRef>jar-with-dependencies</descriptorRef>
                        </descriptorRefs>
            </configuration>
```

Assembly plugin (2)

```
        <executions>
          <execution>
            <id>make-assembly</id>
            <phase>package</phase>
            <goals>
              <goal>single</goal>
            </goals>
          </execution>
        </executions>
</plugin>
```

Compile and Install

- In order to compile modules in the root of the project run:

```
mvn clean install
```

- To create the jar:

```
mvn package
```

- Run with:

```
java -jar target/$ARTIFACTID-0.0.1-SNAPSHOT-jar-with-dependencies.jar
```

Californium

- Californium is a Java CoAP library.
- Download with:

```
git clone https://github.com/eclipse/californium.core.git
```

- Compile using Maven:

```
mvn clean install
```

You need to perform this step only if you need to change the internal code of the library, otherwise you can let maven to download the library at the time of deployment of your project

If you just use Californium as a library you can just add it as dependency on your project

Californium overview

- Take a look at the code from github:
  - https://github.com/eclipse/californium
- Take a look at the examples:
  - https://github.com/eclipse/californium/tree/master/demo-apps

Classes

- CoAPServer
  - It is the base class for all custom Servers.
- CoAPClient
  - It is the base class for all custom Clients.
- Requests
  - Provides the functionality of a CoAP request. Clients instantiate such class to issue requests to Servers.
- Response
  - Provides the functionality of a CoAP response. Servers instantiate such class in reply to requests
- Option
  - A message can have several Options with different or same option numbers. Every option is associated with a value of implicit type.

Classes (2)

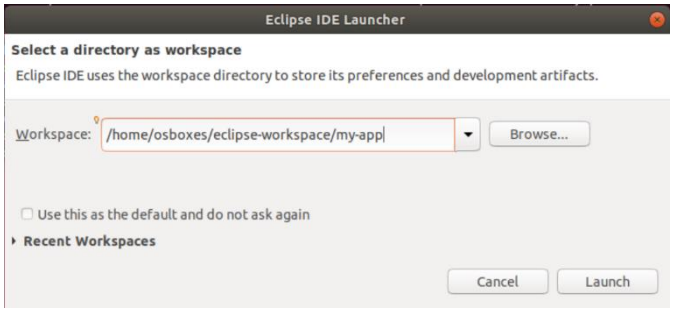
- CoapResource:
  - A server hosts a tree of Resources exposed to clients.
  - Resources can be added and removed dynamically.
  - CoapResource is an element on the resource tree of a server.
  - A resource must have a unique URI.
  - A resource is able to respond to CoAP requests.

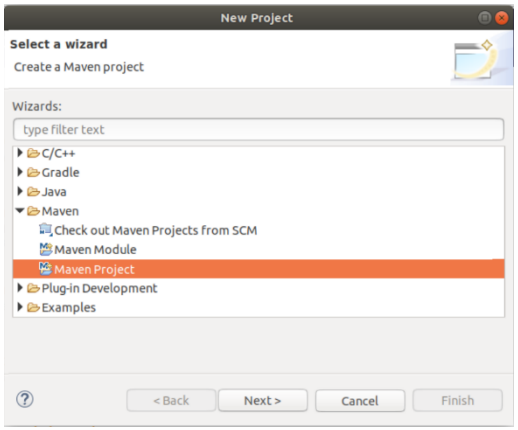
Classes (3)

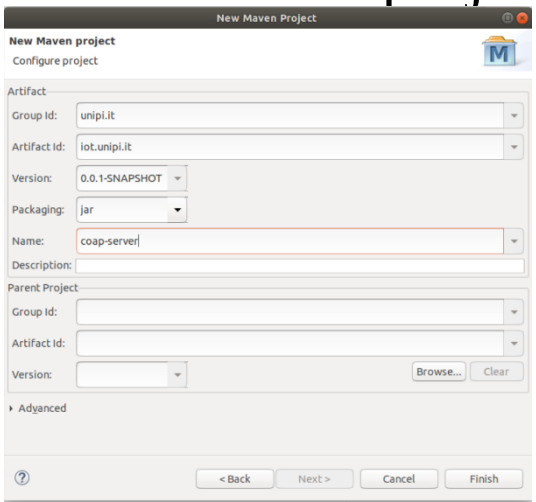
- CoapExchange
  - Used to match each Request to the corresponding Response
  - CoapResources handles CoAPExchange to hide the details of the CoAP protocol
  - Expose useful methods:
    - getRequestOptions: retrieve requests option set
    - getRequestedPayload: retrieve the Payload of the request
    - advanced().getRequest(): retrieve the Request object
    - …

Create a new CoAP project

- Create a new Maven project
  - Open Eclipse
  - Specify a new workspace
  - File->New->Project
  - Create a new Maven->Maven Project
  - Select ‘Create a simple project’
  - Enter the data for the new project
  - Finish







Create a new CoAP project

- Add Californium as dependency

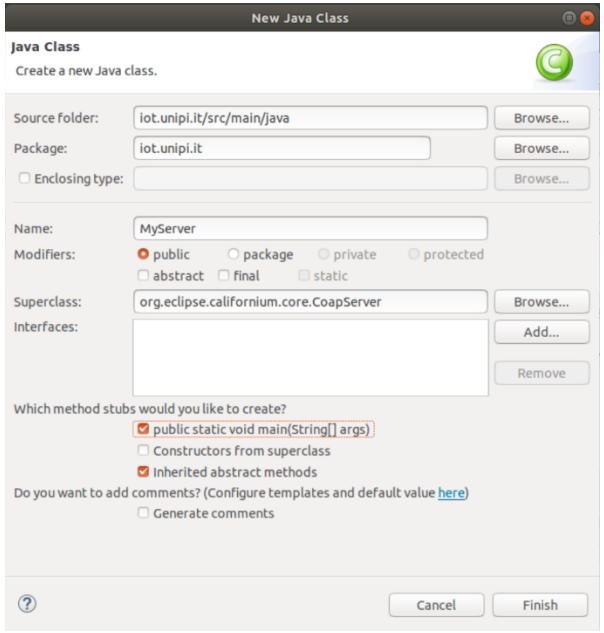
```
 <repositories>
    <repository>
         <id>repo.eclipse.org</id>
         <name>Californium Repository</name>

         <url>https://repo.eclipse.org/content/repositories/californium/
</url>
    </repository>
  </repositories>

  <dependencies>
    <dependency>
        <groupId>org.eclipse.californium</groupId>
         <artifactId>californium-core</artifactId>
         <version>1.1.0-SNAPSHOT</version>
    </dependency>
  </dependencies>
```

CoAP Server main class definition

- Create the main class which extends the base CoapServer with the main:
  - File -> New -> Class
  - Provide class details



- The new file should contain a skeleton of the main function

```
class MyServer extends CoapServer {
        public static void main(String args[]) {...}
        System.out.print("Running it!");
}
```

CoAP Server main class definition

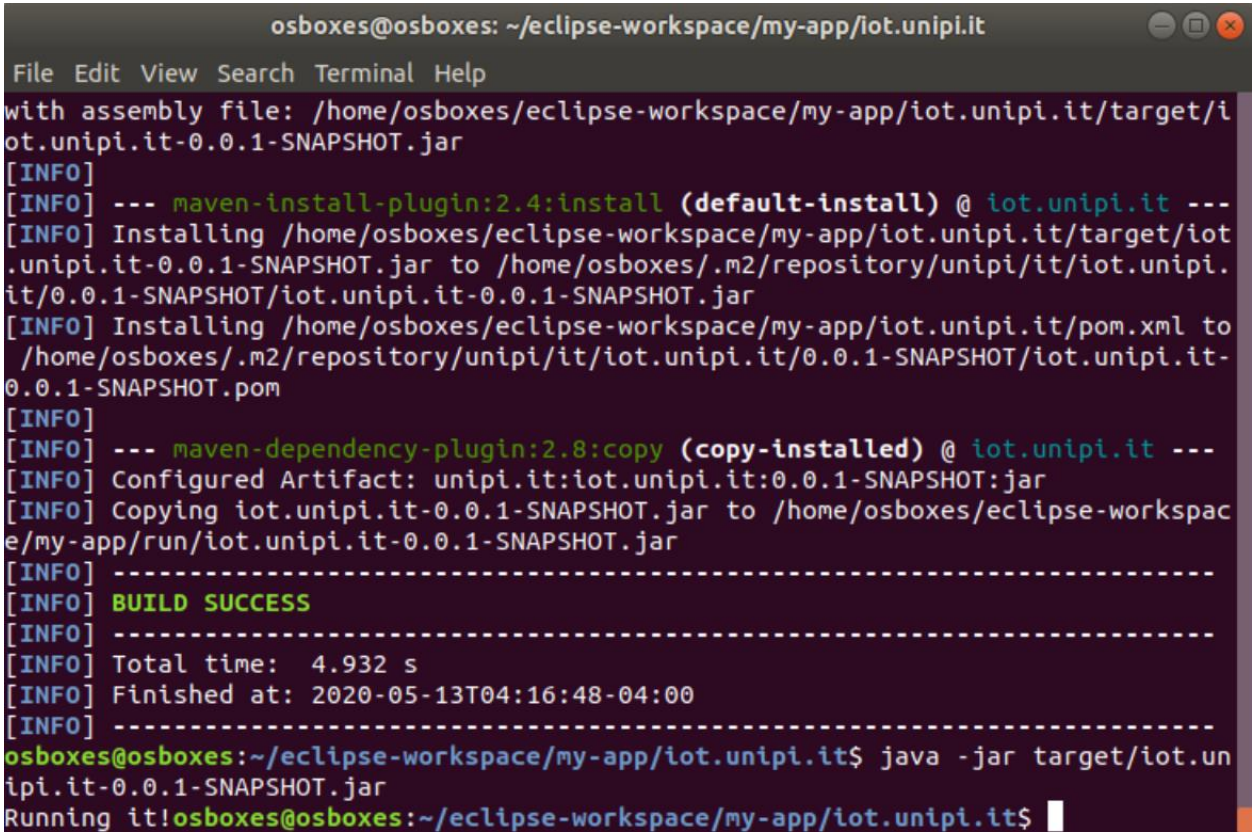
- Detail the build process to specify the main class
- Add the directives at slides #9, #10 and #11 inside a tag build and plugins and change the main class

```
<build>
        <plugins>
                <plugin>
                …

<mainClass>iot.unipi.it.MyServer</mainClass>
                <plugin>
                …
                </plugin>
        </plugins>
</build>
```

Run IT!

- Compile and run this first skeleton application



Define Server Resources

```
class CoAPResourceExample extends CoapResource {
        public CoAPResourceExample(String name) {
                super(name);
                setObservable(true);
        }
        public void handleGET(CoapExchange exchange) {
                exchange.respond(”hello world”);
        }
        public void handlePOST(CoapExchange exchange) {
                /* your stuff */
                exchange.respond(ResponseCode.CREATED);
        }
}
```

Create the server

```
public static void main(String args[]) {
        MyServer server = new MyServer();
        server.add(new CoAPResourceExample("hello"));
        server.start();
}
```

```
osboxes@osboxes:~/eclipse-workspace/my-app/iot.unipi.it$ java -jar target/iot.un
ipi.it-0.0.1-SNAPSHOT.jar
Running it!May 14, 2020 10:27:42 AM org.eclipse.californium.core.network.config.
NetworkConfig load
INFO: loading properties from file /home/osboxes/eclipse-workspace/my-app/iot.un
ipi.it/Californium.properties
May 14, 2020 10:27:42 AM org.eclipse.californium.core.CoapServer start
INFO: Starting server
May 14, 2020 10:27:42 AM org.eclipse.californium.core.CoapServer start
INFO: No endpoints have been defined for server, setting up server endpoint on d
efault port 5,683
May 14, 2020 10:27:42 AM org.eclipse.californium.core.network.CoapEndpoint start
INFO: Starting endpoint at 0.0.0.0/0.0.0.0:5683
^N
```

```
osboxes@osboxes: ~/eclipse-workspace/my-app/iot.unipi.it
File  Edit  View  Search  Terminal  Help
osboxes@osboxes:~/eclipse-workspace/my-app/iot.unipi.it$ coap-client -m get coap
://127.0.0.1:5683/hello
v:1 t:CON c:GET i:4b98 {} [ ]
hello world
```

Create the response

The Response object can be used to create the response.
The CoapExchange object can be used to retrieve information on the request

```
public void handleGET(CoapExchange exchange) {
  Response response = new Response(ResponseCode.CONTENT);
  if(exchange.getRequestOptions().getAccept() ==
       MediaTypeRegistry.APPLICATION_XML){
           response.getOptions().setContentFormat(MediaTypeRegistry.APPLICATION_XML);
           response.setPayload("<value>10</value>")
  }else if(exchange.getRequestOptions().getAccept() ==
       MediaTypeRegistry.APPLICATION_JSON){
           response.getOptions().setContentFormat(MediaTypeRegistry.APPLICATION_JSON);
           ...
  }
  exchange.respond(response);
}
```

Exercise 1

- Create a CoAP Server with a resource which allows GET and POST.
- Implement the POST method in order to read a number from the request payload and reply back with the square of the input number.

Create a Californium Client

- Create a CoapClient object:

```
CoapClient client = new CoapClient(”coap://127.0.0.1/hello”)
```

- Issue a request:

```
CoapResponse response = client.get()
```

Or

```
CoapResponse response = client.post(”10 C°”,
MediaTypeRegistry.TEXT_PLAIN)
```

Exercise 2

- Create a CoAP Client that issue a GET request to the server of the previous exercise.
- Modify the client to add a payload to the request to retrieve the square of the number.

Exercise 3

- Modify the server to reply to GET according to the Accept option in the request. (Possible values: application/xml, application/json)
- Modify the client to retrieve resource representation in json and to parse the results.
- Hints:
  - json-simple: http://www.studytrails.com/java/json/java-json-simple/

```
<!-- https://mvnrepository.com/artifact/com.googlecode.json-simple/json-simple -->
<dependency>
    <groupId>com.googlecode.json-simple</groupId>
    <artifactId>json-simple</artifactId>
    <version>1.1.1</version>
</dependency>
```

Exercise 4

- Write a client which interacts with a server deployed in cooja with contiki.

- Hint: remember to use the rpl-border-router

Advanced Client use

- Create GET Request:

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

- Add options to request:

```
req.getOptions().addOption(new Option(number, value))
```

- Add specific option through utility methods:

```
req.getOptions().addUriQuery("number=3")
```

- Send request:

```
CoapResponse resp = client.advanced(req);
```

Observing

```
CoapClient client = new CoapClient("coap://127.0.0.1/hello");
CoapObserveRelation relation = client.observe(
                                new CoapHandler() {
                                @Override public void onLoad(CoapResponse response) {
                                  String content = response.getResponseText();
                                  System.out.println(content);
                                }
                                @Override public void onError() {
                                  System.err.println("-Failed--------");
                                }
                                });
try { Thread.sleep(6*1000); } catch (InterruptedException e) { }

relation.proactiveCancel(); // to cancel observing
```

Python - CoAPThon

CoAPThon

- CoAPThon is a Python library implementing CoAP
- Two versions are available CoAPThon for Python2 and CoAPThon3 for Python3
- https://github.com/Tanganelli/CoAPthon
- https://github.com/Tanganelli/CoAPthon3

- Its implementation is very easy to use (as it is the Python language) and allows rapid deployment of applications

CoAPThon installation

CoAPThon can be installed using PIP, the package manager for Python

```
sudo apt-get install python-pip/python3-pip
```

```
sudo pip/pip3 install coapthon/coapthon3
```

Create a server

A CoAP server and a resource is created in Python as classes.
Both server and resource must be derived from CoAP and Resoruce, respectively

```
from coapthon.server.coap import CoAP
from coapthon.resources.resource import Resource
class ResExample(Resource):
    def __init__(self, name="ResExample", coap_server=None):
        super(ResExample, self).__init__(name, coap_server,
                      visible=True, observable=True, allow_children=True)
        self.payload = "Basic Resource"
        self.resource_type = "rt1"
        self.content_type = "text/plain"
        self.interface_type = "if1"
    def render_GET(self, request):
        self.payload = "Hello"
        return self

class CoAPServer(CoAP):
    def __init__(self, host, port):
        CoAP.__init__(self, (host, port), False)
        self.add_resource("hello/", ResExample())
```

Create a server

Run the server

```
ip = ”0.0.0.0”
port = 5683



server = CoAPServer(ip, port)


try:
    server.listen(10)
except KeyboardInterrupt:
    print(”Server Shutdown”)
    server.close()
    print(”Exiting...”)
```

Create a client

A simple client can be created via the HelperClient class

```
from coapthon.client.helperclient import HelperClient
from coapthon.utils import parse_uri

host = "127.0.0.1"
port = 5683

path="/hello"

client = HelperClient(server=(host, port))

response = client.get(path)
print(response.pretty_print())
client.stop()
```

Exercise 5

- Create a simple Python coap server and client