Internet of Things

Domenico Pio Santoro

June 7, 2018

Introduction

This report is a short introduction to the Java application developed for the final project of *Internet of Things*. In this section is described in general, how the system works, the main classes and some other small technical details.
Any other informations on all the other Java classes and its methods are contained in the JavaDoc. The application is divided in three main parts:

- HTTP server written in Golang
- Slot machine class rappresenting a specific slot of the Smart Gambling House
- Remote storage module

System parts

The application is divided into different Java packages, each one rappresent a module of the slot machine or any other important part of the Smart Gambling House.
There are some other packages for the Remote storage module, Coap Gambler Client, and other custom class.

HTTP Server

The simple HTTP server developed, available on GitHub, provides the acceleration of the resource *lever*. The server is developed in Go, thanks to his full HTTP support. The server, hosted on a VPS, provides, on each *GET Request* a pseudo-random value of acceleration that can be a zero value (Lever in QUIET state) or non-zero value (Lever in PULLED state).

Slot Machine

The following section describe the Slot Machine main components.

Lever Controller

This package is made of two important Java classes:

- HttpClient
- LeverCoapServer

HttpClient: this class is in charge of send a *GET Request* to the Go HTTP server, retriving the JSON information and storing it into a Java POJO Object with *GSON* called *LeverActualValue*. *LeverActualValue* contains an *Enum* for the lever status (STARTING, QUIET, PULLED) and an *int* for the actual value of acceleration.

LeverCoapServer: this is a Coap server that handle the *Observable Resource "/lever"*, when this resource is created, autonomously creates an *HttpClient* object and update its status thanks to a Java class call *UpdateStatus*.

Core Module Controller

This package, responsable for the core of the *SlotMachine* class, contains two Java classes:

- ObserveCoreClient
- CoreModuleServer

ObserveCoreClient: has to create an observe relationship with the resource *lever*.

CoreModuleServer: is responsable for the *insertCoin* resource. When created, this resource gets from the *RemoteStorageModule* the minimum coin amount for the instance of the *SlotMachine*. This call has a method in charge of starting the actual game creating the results sequence of numbers and sending all the required requests to the *RemoteStorageModule*.

```
public void startGambling(int soldi, int ID) {

    ...
    CoapClient dummyClient = new CoapClient("coap://localhost:5888/NewGame");
    ...

    //Waiting for the pulled status
    do {
        lAv = obs.getLeverStatus();
    }  while (obs.getLeverStatus().getStatoLeva() != LeverStatus.PULLED);

    ...

    if (((uno == due && due == tre) || lAv.getAcceleration() == 42) && lAv.getAcceleration() > 0 ) {

        System.out.println("WINNER WINNER, CHICKEN DINNER!\n X: "+ uno + " Y:" + due+ " Z" + tre
                );

        CoapClient chickenDinner = new CoapClient("coap://localhost:5888/wonGame");
        ...
    }else {

        System.out.println("insert more coins, the next time you'll be luckier :)\n Resulting sequence-> X:
                "+ uno + " Y: " + due+ " Z:" + tre);

        CoapClient looserDinner = new CoapClient("coap://localhost:5888/lostGame");
        ...

    }
}
```

Remote Storage Module

This package is made of different classes that rappresent all the Coap Resoources described in the project guide line. The class *RemoteStoradgeModule* is in charge of "create and serve" this resources. All this resources in Figure 1, better described in the JavaDoc, implement their own *POST* and *GET* handlers.

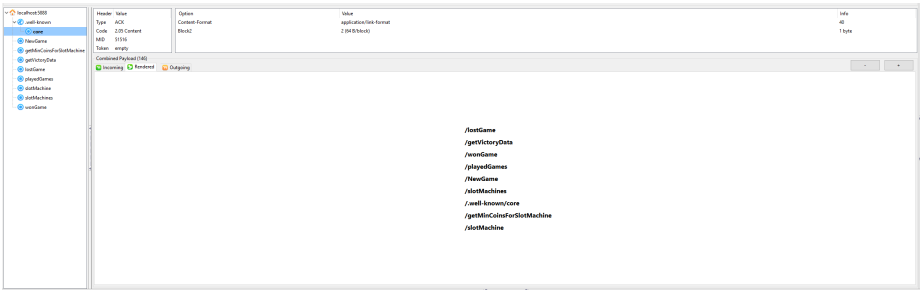
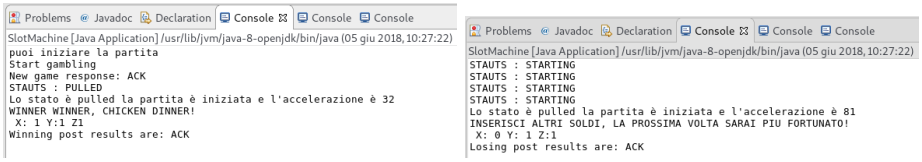


Figure 1: RemoteStorageModule resource

Testing

All the tests are running on local machine, launching the *RemoteStorageModule*, *SlotMachine* and the *GamblerClient*. For the *RemoteStorageModule* there is a Java class rappresents a Coap Client in the *tester* package, but for seek of simplicity it was used the Firefox extension Copper and in this report there are some screenshots of the testing of this resources in Copper, but every resource of the *RemoteStorageModule* is used during the Java execution of the *SlotMachine* class or *GamblerClient* class. Here there are some screenshots of some HTTP Requests to *RemoteStorageModule* resources and some simulations of game.



(a) Wongame (b) Lostgame

Figure 2: Similation of games

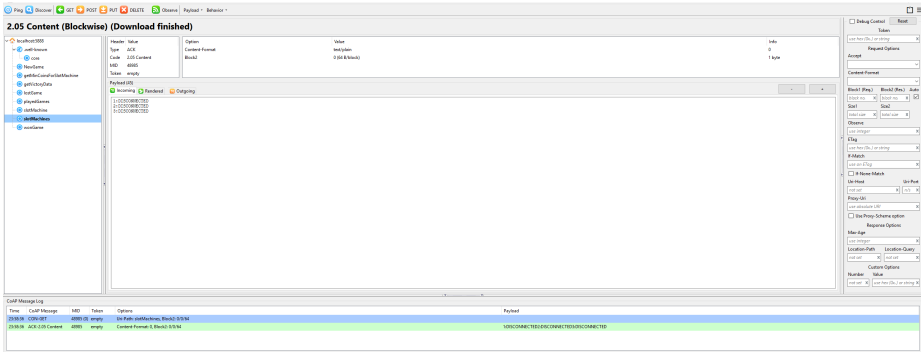


Figure 3: First GET request to CoapResource slotMachines with all slot disconnected

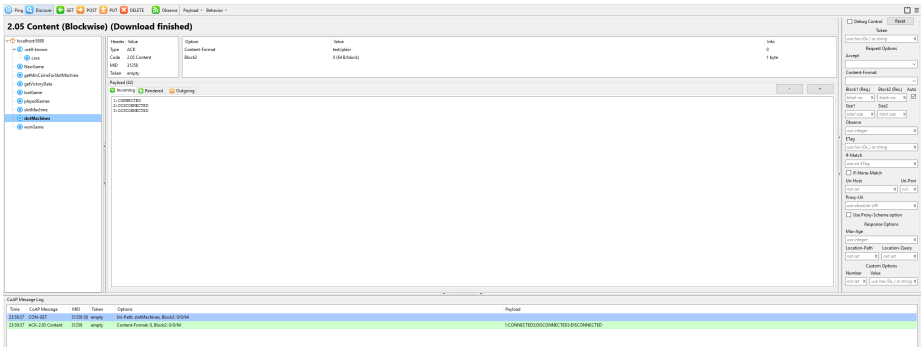


Figure 4: Another GET request to CoapResource slotMachines after POST request to sloatMachine

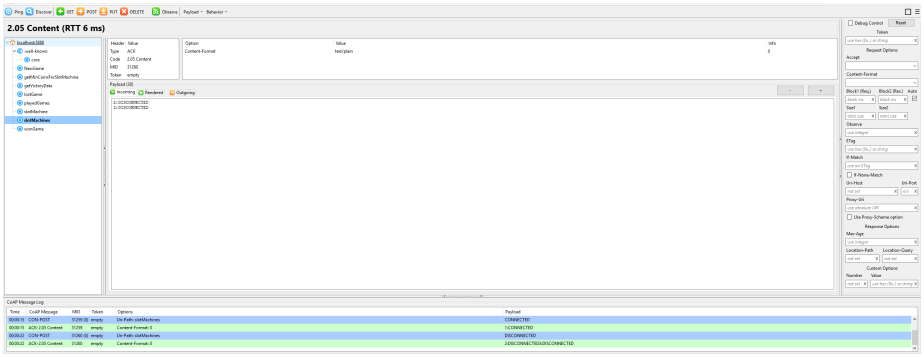


Figure 5: POST to slotMachines to print all the DISCONNECTED machines