UNIVERSITÀ DI PISA

Computer engineering

Internet Of Things

# Radaway

**Francesco Berti**

A.Y. 2023/2024

## Contents

## 1 Introduction and use case

The implementation of an Internet of Things (IoT) system to monitor core poisoning in nuclear power plants is crucial for ensuring the safety and efficiency of these facilities.

Core poisoning refers to the accumulation of certain fission products, such as Xenon-135, within the reactor core. Xenon-135 is originated primarily from the decay of Iodine-135, a fission product. The high neutron absorption cross-section of Xenon-135 makes it a powerful neutron poison, which can absorb neutrons that would otherwise sustain the nuclear chain reaction.

Core poisoning was one of the causes for the nuclear power plant incident in Prypiat in 1986, it was ultimately caused by the inadequate and unprepared staff taking part into a test in the power plant[1]: during the test, the plant operators disabled all safety systems and rapidly reduced the reactor core's power by mistake.

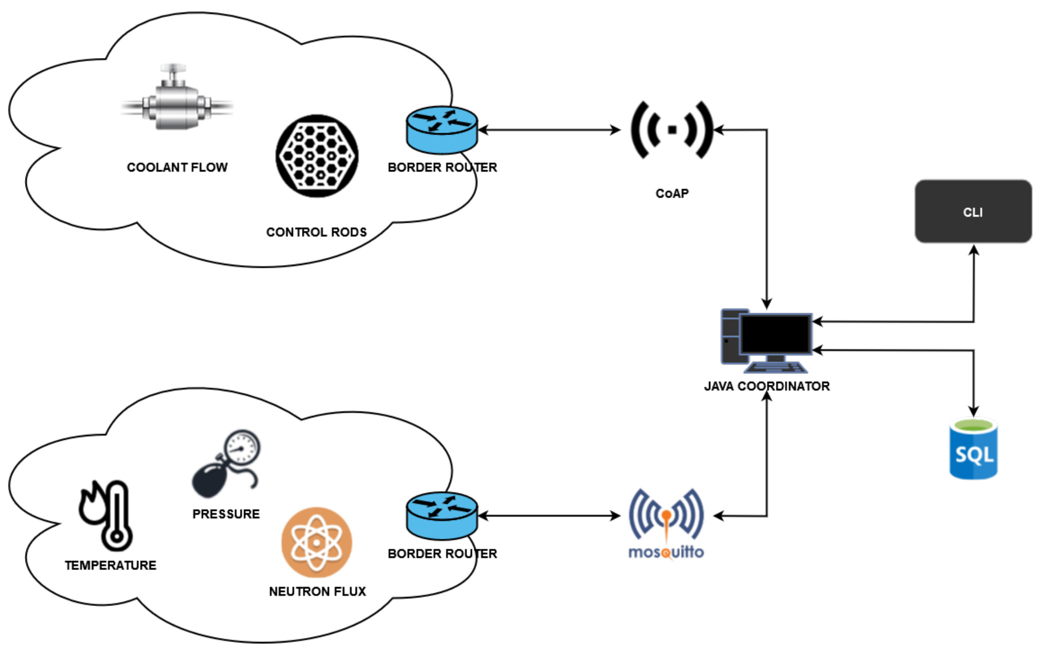
The experiment needed a certain power output from the reactor, thus they removed most of the control rods which regulate the fission reaction. The RMBK-1000 reactor used in Prypiat needed at least 30 control rods to operate safely, the operators reduces the inserted control rods to 6-8.

This lead to an unstable reactor state and the build up of Xenon-135. This instability caused an increase of temperature and coolant pressure, ultimately resulting in a steam explosion that blew off the reactor lid spreading radioactive material into air.

For this reason, an IoT system is crucial for continuous and precise monitoring to prevent catastrophic incidents.

## 2 Architecture

The system architecture is as shown in the following image:



### 2.1 MQTT network

- **Temperature**: senses the temperature in Celsius of the coolant liquid
- **Pressure**: senses the pressure in Bar of the coolant liquid
- **Neutron flux**: senses how many neutrons are present in the reactor's core

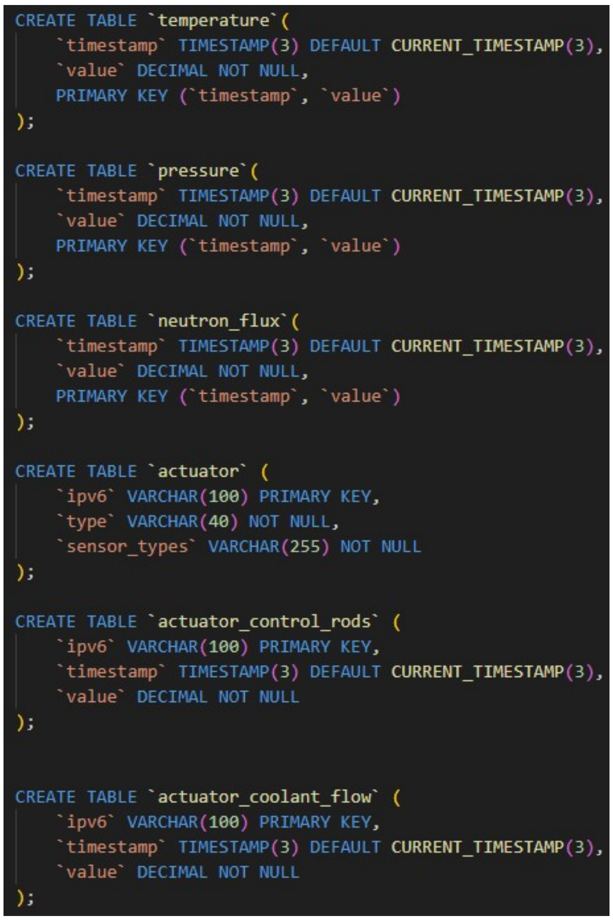
### 2.2 CoAP network

- **Coolant flow**: used to increase/decrease the coolant liquid flow inside the reactor's core
- **Control rods**: used to increase/decrease the amount of fully inserted control rods inside the reactor's core

## 2.3 Database

A MySQL database is used to store registered and active CoAP actuators, as well as their status changing in time which can be useful for later examination in case there was some problem with the actuator themselves. The DB also stores the values published by sensors in the MQTT network, these are then used to adjust the actuator status accordingly.

The DB scheme is the following:



```
CREATE TABLE `temperature`(
    `timestamp` TIMESTAMP(3) DEFAULT CURRENT_TIMESTAMP(3),
    `value` DECIMAL NOT NULL,
    PRIMARY KEY (`timestamp`, `value`)
);

CREATE TABLE `pressure`(
    `timestamp` TIMESTAMP(3) DEFAULT CURRENT_TIMESTAMP(3),
    `value` DECIMAL NOT NULL,
    PRIMARY KEY (`timestamp`, `value`)
);

CREATE TABLE `neutron_flux`(
    `timestamp` TIMESTAMP(3) DEFAULT CURRENT_TIMESTAMP(3),
    `value` DECIMAL NOT NULL,
    PRIMARY KEY (`timestamp`, `value`)
);

CREATE TABLE `actuator` (
    `ipv6` VARCHAR(100) PRIMARY KEY,
    `type` VARCHAR(40) NOT NULL,
    `sensor_types` VARCHAR(255) NOT NULL
);

CREATE TABLE `actuator_control_rods` (
    `ipv6` VARCHAR(100) PRIMARY KEY,
    `timestamp` TIMESTAMP(3) DEFAULT CURRENT_TIMESTAMP(3),
    `value` DECIMAL NOT NULL
);


CREATE TABLE `actuator_coolant_flow` (
    `ipv6` VARCHAR(100) PRIMARY KEY,
    `timestamp` TIMESTAMP(3) DEFAULT CURRENT_TIMESTAMP(3),
    `value` DECIMAL NOT NULL
);
```

the *sensor_types* field contains a JSON array of int values which identifies in which sensors the actuator is interested, this is used in the coordinator to handle autonomous transition of actuator status.

## 2.4 Data encoding

All the data exchanged in this architecture is encoded in JSON. This choice was made since the *nRF52840* dongles used for this project have limited resources and the data exchanged is fairly simple, thus XML would have been too heavy and complex for this purpose.

# 3 Java Coordinator

The coordinator offers a CLI to read and force actuator status but also handles switching actuator status autonomously.

## 3.1 Autonomous system

The coordinator collects all the data published by the MQTT network and interfaces with the MySQL DB to store it. An analysis is performed every 10 seconds in which the coordinator extracts the latest values, in the last 5 and 10 seconds, from the DB for a given sensor and then calculates the averages to compare them in the following way:

```
Math.abs(newAverage - oldAverage) > oldAverage * thresholdPerc
```

*thresholdPerc* is a fixed value set to *0.05* which offers a good balance in updating or not updating the actuator status. The coordinator code offers the possibility of setting this values based on the actuator type for a more in-depth simulation.

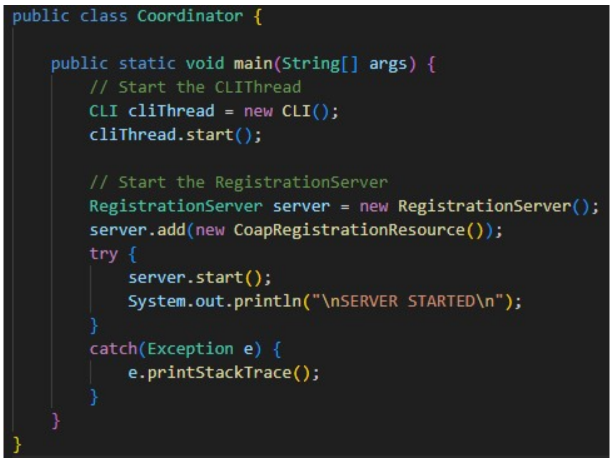
## 3.2 CLI

The commands available in the CLI are the following:

- **!help**: prints all available commands
- **!status**: prints the current actuator status
- **!shutdown**: fully insert all control rods
- **!startup**: 50% of control rods are extracted, starts the fission reaction
- **!regime**: fully inserted control rods are dropped to 10%, used when the power plant is at full capacity
- **!coolant_off**: coolant flow is set to 0%
- **!coolant_on**: coolant flow is set to 70% which is a safe operating value
- **!cooldown**: coolant flow set to 100%

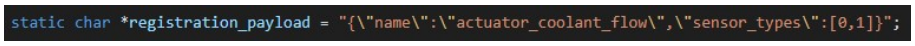
## 4 Deployment

The collector starts two threads, one for the CLI and one for the CoAP registration server:



```java
public class Coordinator {

    public static void main(String[] args) {
        // Start the CLIThread
        CLI cliThread = new CLI();
        cliThread.start();

        // Start the RegistrationServer
        RegistrationServer server = new RegistrationServer();
        server.add(new CoapRegistrationResource());
        try {
            server.start();
            System.out.println("\nSERVER STARTED\n");
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

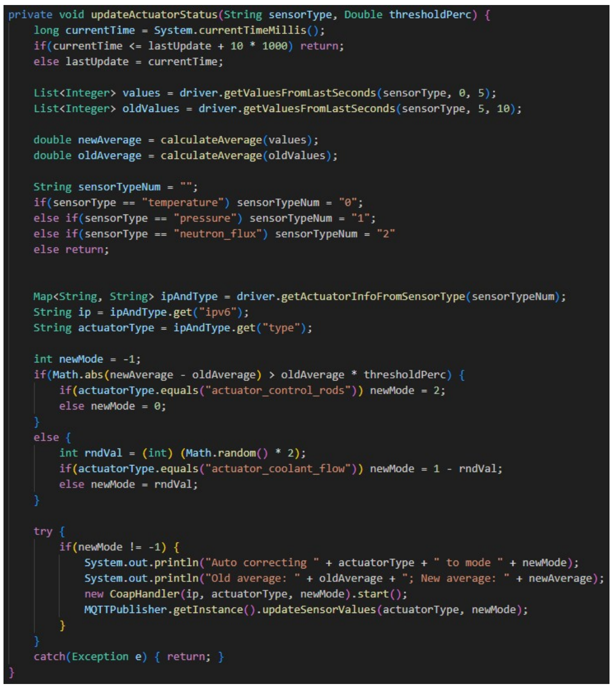
when a CoAP actuator registers it sends both its name and an array of int values to express in which sensor is interested



```c
static char *registration_payload = "{\"name\":\"actuator_coolant_flow\",\"sensor_types\":[0,1]}";
```

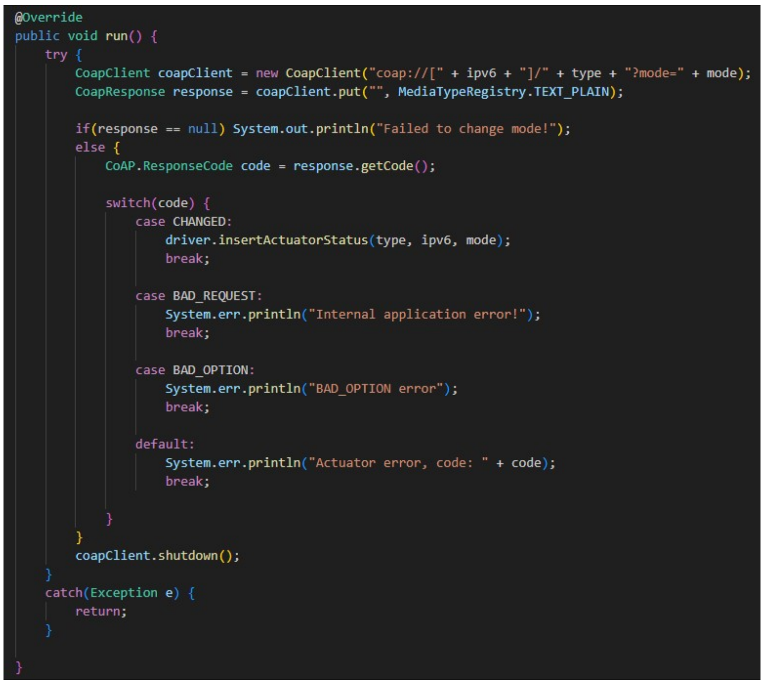
the CoAP registration server extracts the JSON data and stores it in the MySQL DB using a custom driver class called *DBDriver* with appropriate methods.

The CLI thread also registers a MQTT client to subscribe to sensor topics and to set the MQTT callback to a custom class that stores the sensor values in the DB as well as autonomously update the actuators' status:



```java
private void updateActuatorStatus(String sensorType, Double thresholdPerc) {
    long currentTime = System.currentTimeMillis();
    if(currentTime <= lastUpdate + 10 * 1000) return;
    else lastUpdate = currentTime;

    List<Integer> values = driver.getValuesFromLastSeconds(sensorType, 0, 5);
    List<Integer> oldValues = driver.getValuesFromLastSeconds(sensorType, 5, 10);

    double newAverage = calculateAverage(values);
    double oldAverage = calculateAverage(oldValues);

    String sensorTypeNum = "";
    if(sensorType == "temperature") sensorTypeNum = "0";
    else if(sensorType == "pressure") sensorTypeNum = "1";
    else if(sensorType == "neutron_flux") sensorTypeNum = "2"
    else return;


    Map<String, String> ipAndType = driver.getActuatorInfoFromSensorType(sensorTypeNum);
    String ip = ipAndType.get("ipv6");
    String actuatorType = ipAndType.get("type");

    int newMode = -1;
    if(Math.abs(newAverage - oldAverage) > oldAverage * thresholdPerc) {
        if(actuatorType.equals("actuator_control_rods")) newMode = 2;
        else newMode = 0;
    }
    else {
        int rndVal = (int) (Math.random() * 2);
        if(actuatorType.equals("actuator_coolant_flow")) newMode = 1 - rndVal;
        else newMode = rndVal;
    }

    try {
        if(newMode != -1) {
            System.out.println("Auto correcting " + actuatorType + " to mode " + newMode);
            System.out.println("Old average: " + oldAverage + "; New average: " + newAverage);
            new CoapHandler(ip, actuatorType, newMode).start();
            MQTTPublisher.getInstance().updateSensorValues(actuatorType, newMode);
        }
    }
    catch(Exception e) { return; }
}
```

Whether the actuator status switching was made autonomously by the coordinator or via CLI, a new thread is created to update the actuator's status. This uses a custom class called *CoapHandler* which contacts the actuator via PUT method:



```java
@Override
public void run() {
    try {
        CoapClient coapClient = new CoapClient("coap://[" + ipv6 + "]/" + type + "?mode=" + mode);
        CoapResponse response = coapClient.put("", MediaTypeRegistry.TEXT_PLAIN);

        if(response == null) System.out.println("Failed to change mode!");
        else {
            CoAP.ResponseCode code = response.getCode();

            switch(code) {
                case CHANGED:
                    driver.insertActuatorStatus(type, ipv6, mode);
                    break;

                case BAD_REQUEST:
                    System.err.println("Internal application error!");
                    break;

                case BAD_OPTION:
                    System.err.println("BAD_OPTION error");
                    break;

                default:
                    System.err.println("Actuator error, code: " + code);
                    break;

            }
        }
        coapClient.shutdown();
    }
    catch(Exception e) {
        return;
    }

}
```

finally, the change in the actuator's status has to be communicated to the relative sensors, for this purpose the sensors also subscribe to a MQTT topic which is handled by the coordinator to publish actuator's status changes.

## 5 References

[1] NEA, *Chernobyl: Chapter I. The site and accident sequence*, NEA, 2002. Available at: oecd-nea.org