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.

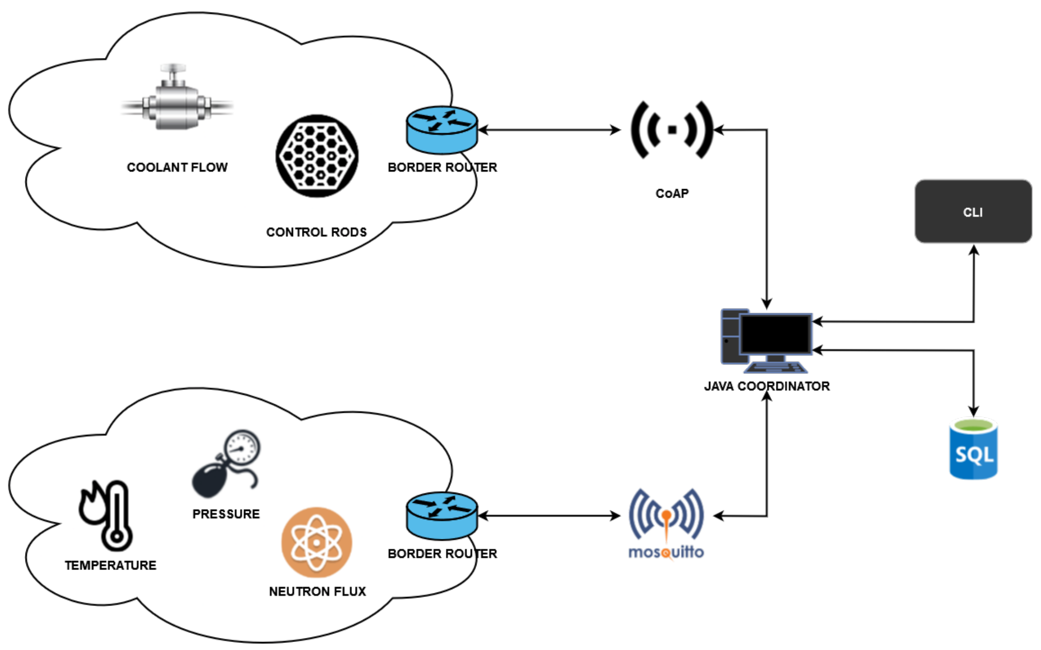
The experiment needed a certain power output from the reactor, thus they removed most of the control rods which regulate the fission reaction. The RMBK-1000 reactor used in Prypiat needed at least 30 control rods to operate safely, the operators reduced the inserted control rods to 6-8.

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 the 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 at the inlet
- **Pressure**: senses the pressure in Bar of the coolant liquid at the inlet
- **Neutron flux**: senses how many neutrons are passing by in a given area, measured in $\frac{\text{neutrons}}{\text{cm}^2 \cdot \text{s}}$

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

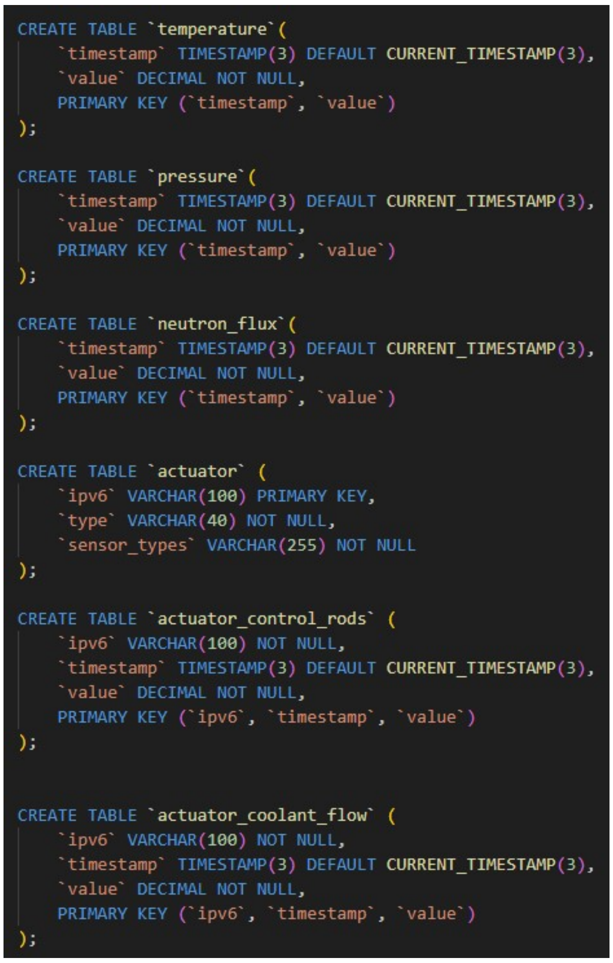
2.3 Database

A MySQL database is used for storing:

- registered CoAP actuators
- the current operating mode of actuators
- values published by sensors in the MQTT network

values stored at point 2 and 3 are then retrieved by the coordinator to take decisions on the next operating mode for actuators, they can also be used as a log for troubleshooting.

The DB scheme is the following:



```sql
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) NOT NULL,
    `timestamp` TIMESTAMP(3) DEFAULT CURRENT_TIMESTAMP(3),
    `value` DECIMAL NOT NULL,
    PRIMARY KEY (`ipv6`, `timestamp`, `value`)
);


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

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 from the DB the values of each sensors in the last 10 seconds, it then calculates the average and compares it to the nominal value for that sensor. In case the average is outside the range, the coordinator issues a change in the actuator operating mode to bring the value back to normal.

Nominal values were extracted by a IAEA - International Atomic Energy Agency document[2] about the RBMK-1000 reactor used at Chernobyl.

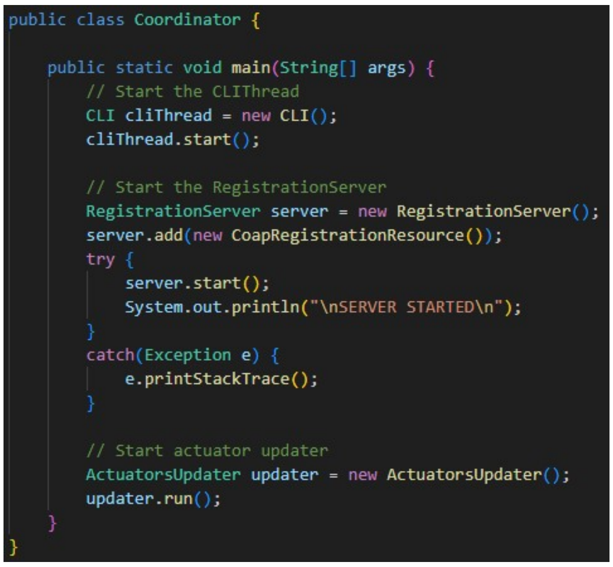
3.2 CLI

The commands available in the CLI are the following:

- **!help**: prints all available commands
- **!actuators**: prints the current actuator status, retrieved via CoAP
- **!sensors**: prints the latest values found in the DB for all the sensors
- **!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% unless the coolant is off, used when the power plant is at full capacity
- **!coolant_off**: coolant flow is set to 0% unless the control rods are at regime
- **!coolant_on**: coolant flow is set to 70% which is a safe operating value
- **!cooldown**: coolant flow set to 100%

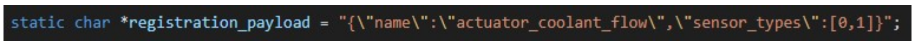
4 Deployment

The collector starts three threads, one for the CLI, one for the CoAP registration server and one to update the actuators statuses:



```
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();
        }

        // Start actuator updater
        ActuatorsUpdater updater = new ActuatorsUpdater();
        updater.run();
    }
}
```

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



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

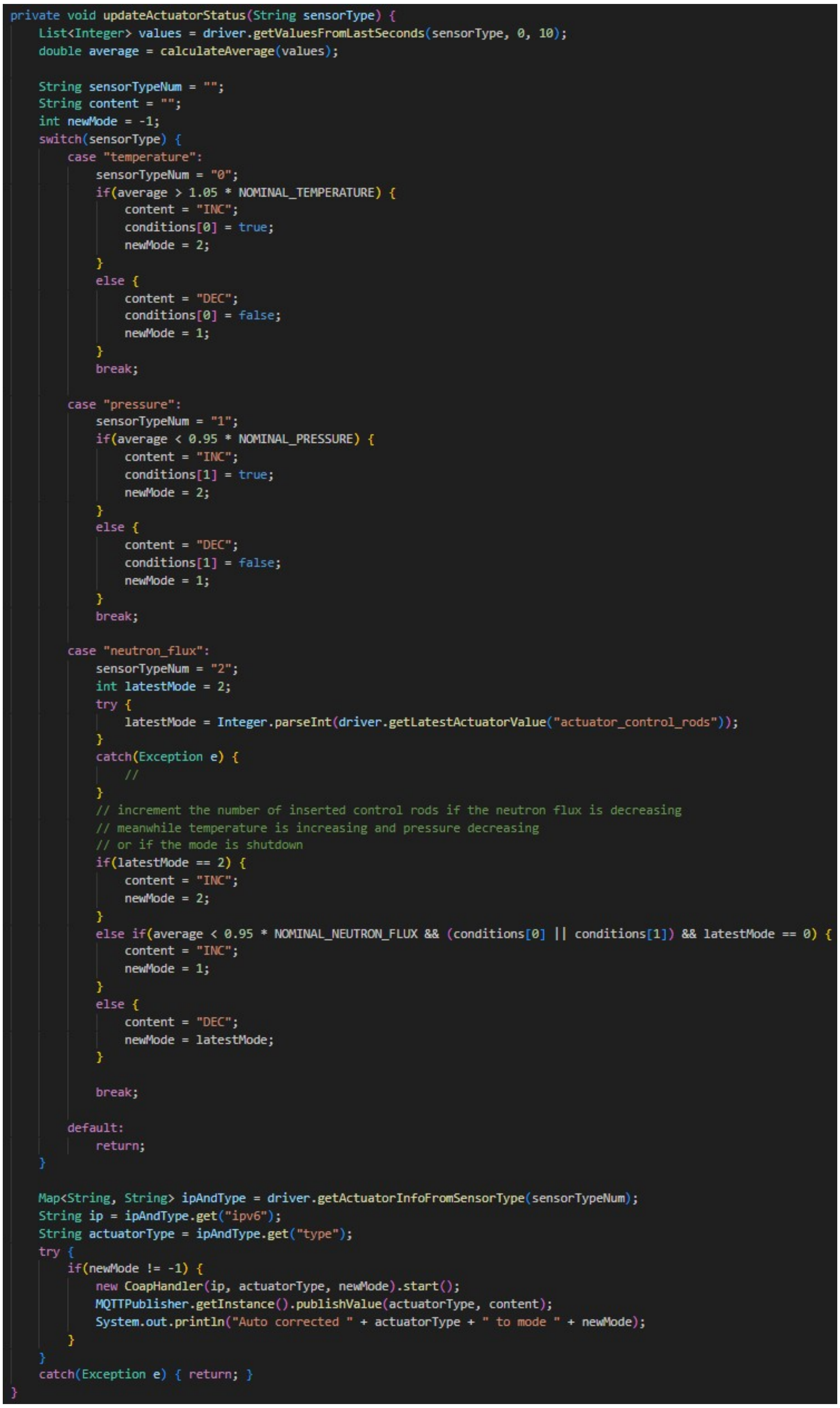
the CoAP registration server extracts the JSON data and stores it in the MySQL DB using a custom driver class called *DBDriver* with appropriate methods.

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.

The thread *ActuatorsUpdater* runs a periodic analysis of the latest values in the DB to which mode an actuator should be set. Every 10 seconds, the values produced by sensors and stored in the DB in the last 10 seconds are extracted and the average is calculated and compared to the nominal value:

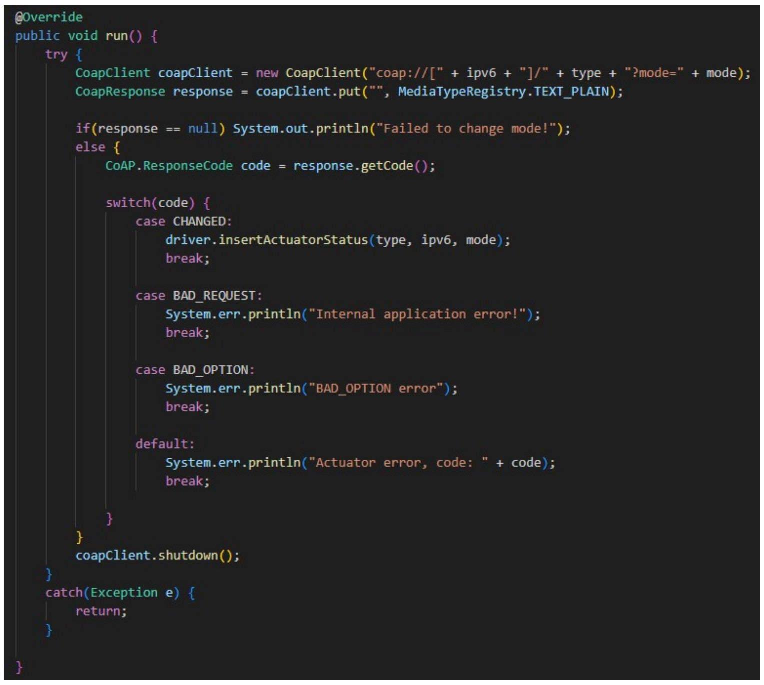
- Coolant flow actuator is straightforward: if the temperature increases or the pressure decreases the coolant flow has to be increased since in both cases the reactor's core is (or will be) getting hotter. A decrease in pressure means a decrease in the boiling point of the coolant (in the RBMK-1000 the coolant is water), steam bubbles might form which can prevent the core from cooling down properly.

- Control rods actuator is slightly more complicated: an indication of core poisoning is when the neutron flux is decreasing with most of the control rods extracted, the temperature is increasing or the pressure is decreasing. This is not an exhaustive indicator, there might be other factors at play, like fuel depletion etc... but this precaution would bring the reactor to a safer level so an operator could verify what is happening with other instruments.



```java
private void updateActuatorStatus(String sensorType) {
    List<Integer> values = driver.getValuesFromLastSeconds(sensorType, 0, 10);
    double average = calculateAverage(values);

    String sensorTypeNum = "";
    String content = "";
    int newMode = -1;
    switch(sensorType) {
        case "temperature":
            sensorTypeNum = "0";
            if(average > 1.05 * NOMINAL_TEMPERATURE) {
                content = "INC";
                conditions[0] = true;
                newMode = 2;
            }
            else {
                content = "DEC";
                conditions[0] = false;
                newMode = 1;
            }
            break;

        case "pressure":
            sensorTypeNum = "1";
            if(average < 0.95 * NOMINAL_PRESSURE) {
                content = "INC";
                conditions[1] = true;
                newMode = 2;
            }
            else {
                content = "DEC";
                conditions[1] = false;
                newMode = 1;
            }
            break;

        case "neutron_flux":
            sensorTypeNum = "2";
            int latestMode = 2;
            try {
                latestMode = Integer.parseInt(driver.getLatestActuatorValue("actuator_control_rods"));
            }
            catch(Exception e) {
                //
            }
            // increment the number of inserted control rods if the neutron flux is decreasing
            // meanwhile temperature is increasing and pressure decreasing
            // or if the mode is shutdown
            if(latestMode == 2) {
                content = "INC";
                newMode = 2;
            }
            else if(average < 0.95 * NOMINAL_NEUTRON_FLUX && (conditions[0] || conditions[1]) && latestMode == 0) {
                content = "INC";
                newMode = 1;
            }
            else {
                content = "DEC";
                newMode = latestMode;
            }

            break;

        default:
            return;
    }

    Map<String, String> ipAndType = driver.getActuatorInfoFromSensorType(sensorTypeNum);
    String ip = ipAndType.get("ipv6");
    String actuatorType = ipAndType.get("type");
    try {
        if(newMode != -1) {
            new CoapHandler(ip, actuatorType, newMode).start();
            MQTTPublisher.getInstance().publishValue(actuatorType, content);
            System.out.println("Auto corrected " + actuatorType + " to mode " + 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. The new mode is provided via the URI instead of using the payload:



```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 at startup to a MQTT topic representing the actuator, these topics are 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

[2] P.Gulshani and A.R. Dastur, *STABILITY ANALYSIS OF SPATIAL POWER DISTRIBUTION IN RBMK-1000 REACTOR*, IAEA, 1987. Available at: inis.iaea.org