CoAP Factory Monitoring

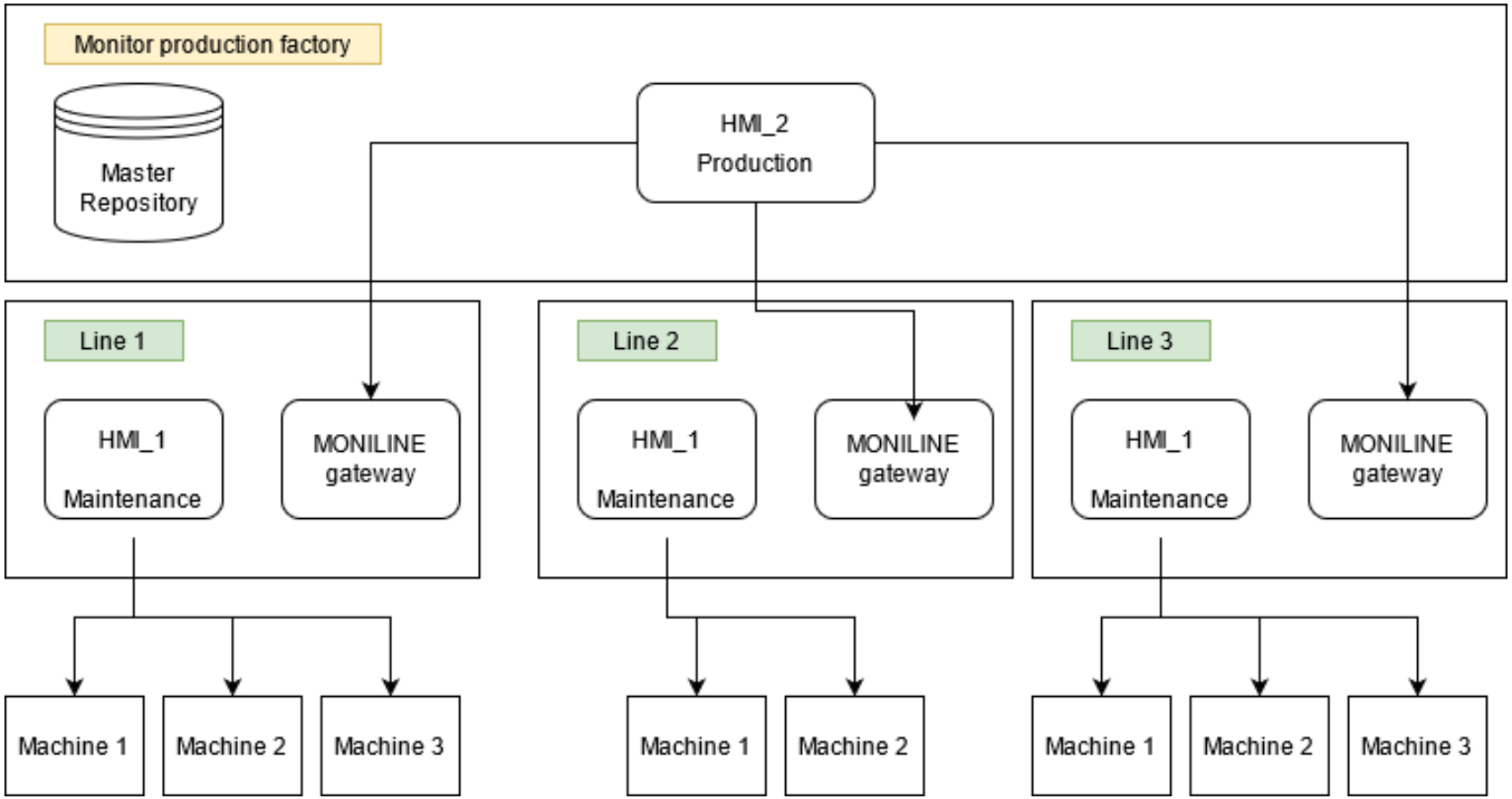
Nicolò Toscani

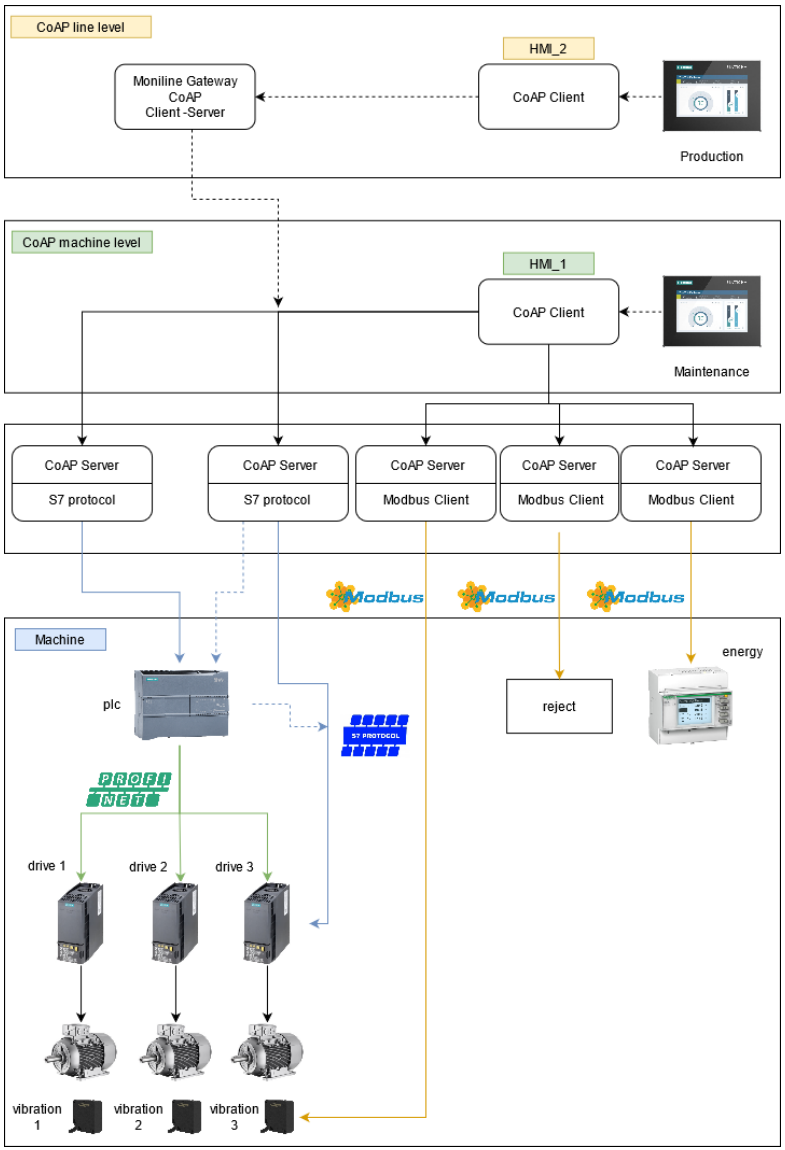
Internet of Things
Final Project

31 Agosto 2021

Goals

- Production monitoring in a manufacturing plant from different company levels
- Data distribution inside the plant using CoAP protocol for future data analysis and predictive maintenance
- Model different data acquisition devices for hiding low level field communication protocol implementation details
- Simplify communication between OT-IT levels


Monitor production factory
Master Repository
HMI_2
Production
Line 1
HMI_1
Maintenance
MONILINE gateway
Line 2
HMI_1
Maintenance
MONILINE gateway
Line 3
HMI_1
Maintenance
MONILINE gateway
Machine 1
Machine 2
Machine 3
Machine 1
Machine 2
Machine 1
Machine 2
Machine 3


CoAP line level
HMI_2
Moniline Gateway
CoAP
Client -Server
CoAP Client
Production
CoAP machine level
HMI_1
CoAP Client
Maintenance
CoAP Server
S7 protocol
CoAP Server
S7 protocol
CoAP Server
Modbus Client
CoAP Server
Modbus Client
CoAP Server
Modbus Client
Modbus
Modbus
Modbus
Machine
plc
reject
energy
drive 1
drive 2
drive 3
vibration
1
vibration
2
vibration
3

Entities

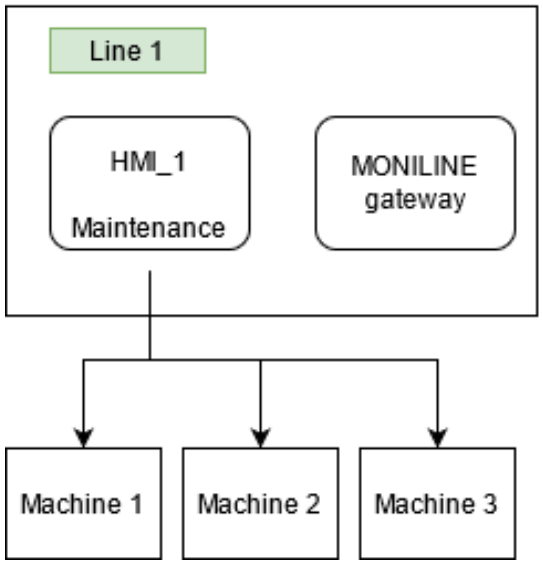
This system can be implemented trying to model the following entities:

- **Lines:** identifies different departments
- **Machines:** identifies different machines installed in a department
- **Maintenance manager:** identifies operator machine *HMI1* managed by maintenance manager who monitors different machines in his department
- **Production manager:** identifies operator interface *HMI2* managed by production manager who wants monitor main KPI of each department to evaluate work metrics

Line

Each line is composed with this entities:

- HM1: used by maintenance manager to interact with all the machines in his department
- Moniline gateway: collect data from different machines and makes it available to higher levels
- $N_{machine}$: set of departmets machines



Machine

Each machine is composed with this entities:

- PLC: used to manage the machine logic and different motors
- Network analyzer: used to get main electrical measurements of the machine
- Reject system: check pieces conformity and keeps production count
- N_{drive}: manages electric motor
- N_{sensor}: vibration sensors installed on motor for reading mechanical measurements

Master repository

Represents the system database used to keep track of configuration parameters used by different entities during application execution.
This allows to store within a queue the configuration parameters of each entity in execution such as:

- Server CoAP port: service port number
- Server CoAP name: service type name
- Server CoAP IPv4 address: device service IP address
- Line ID: line identifier
- Machine ID: machine identifier
- Device ID: device identifier for multiple devices (e.g. drives and vibration sensors)

```
MasterRepository [Java Application] C:\Program Files\Java\jdk-16.0.1\bin\javaw.exe  (22 giu 2021, 09:48:04)
--- ENTITY DATABASE ---
Registered entity: 14
--- ENTITY ---
Device type: plc
Line ID: 1
Machine ID: 1
Device ID: 0
--- ENTITY ---
Device type: plc
Line ID: 1
Machine ID: 2
Device ID: 0
--- ENTITY ---
Device type: plc
Line ID: 2
Machine ID: 1
Device ID: 0
--- ENTITY ---
Device type: plc
Line ID: 2
Machine ID: 2
Device ID: 0
--- ENTITY ---
Device type: energy
Line ID: 1
Machine ID: 1
Device ID: 0
--- ENTITY ---
Device type: energy
Line ID: 1
Machine ID: 2
Device ID: 0
--- ENTITY ---
Device type: energy
Line ID: 2
Machine ID: 1
Device ID: 0
```

config.properties

```
# Line 1
# Machine 1
plc_1_1_name = plc
plc_1_1_port = 5560
plc_1_1_address = 192.168.100.1

energy_1_1_name = energy
energy_1_1_port = 5561
energy_1_1_address = 192.168.100.3

reject_1_1_name = reject
reject_1_1_port = 5562
reject_1_1_address = 127.0.0.1

drive_1_1_1_name = drive
drive_1_1_1_port = 5563
drive_1_1_1_address = 192.168.100.101

drive_1_1_2_name = drive
drive_1_1_2_port = 5564
drive_1_1_2_address = 192.168.100.101

vibration_1_1_1_name = vibration
vibration_1_1_1_port = 5565
vibration_1_1_1_address = 127.0.0.1

vibration_1_1_2_name = vibration
vibration_1_1_2_port = 5566
vibration_1_1_2_address = 127.0.0.1
```

Each system entity is composed by two different threads:

- **entityFieldbusThread:** manages communication with physical device using its communication protocol (e.g. PlcFieldbusThread).
- **entityCoAPServerThread:** used to publish resources for CoAP client devices (e.g. PlcCoAPServerThread).

The classes that model physical devices are:

- PLCGateway: PLC implementation
- DriveGateway: single drive implementation for electric motor management
- RejectGateway: pieces rejection system
- Pm3200Gateway: energy consumption monitoring device implementation
- Qm42vt2Gateway: single vibration motor monitoring sensor implementation

entityFieldbusThread

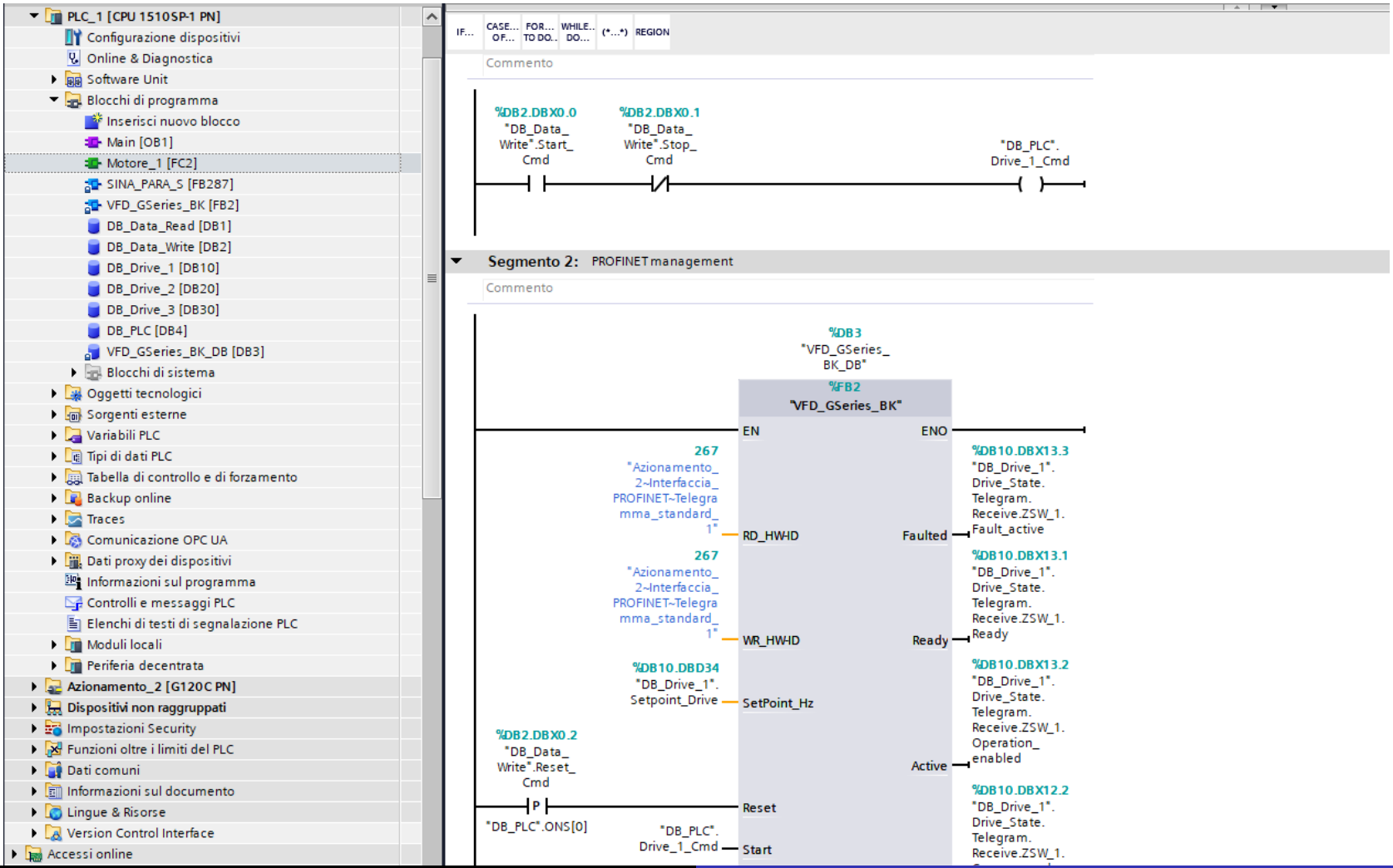
Thread that manages device physical level communication.
It implements fieldbus communication for reading and writing parameters.
Once data has been taken, it is passed to the parent object and then published by the relative CoAP server.
At lower level it uses object responsible for implementing the communication (e.g. PlcS7Service).

Siemens PLC memory

DB_Data_Read

	Nome	Tipo di dati	Offset	Valore di avvio	Ritenzione	Accessibile ...	Scrivi...	Visibile in ..	Valore di i..	Controllo	Commento
1	Static				☐	☐	☐	☐	☐		
2	MachineState	Int	0.0	0	☐	☑	☑	☑	☐		0: stopped; 1: suspended; 2: execute
3	AlarmPresence	Bool	2.0	false	☐	☑	☑	☑	☐		1: used to show alarm list in HMI
4	Alarms	Array[0..9] of Bool	4.0		☐	☑	☑	☑	☐		
5	Alarms[0]	Bool	4.0	false	☐	☑	☑	☑	☐		Alarm 0: Safeties not restored
6	Alarms[1]	Bool	4.1	false	☐	☑	☑	☑	☐		Alarm 1: Compressed air failure
7	Alarms[2]	Bool	4.2	false	☐	☑	☑	☑	☐		Alarm 2: Inappropriate temperature
8	Alarms[3]	Bool	4.3	false	☐	☑	☑	☑	☐		Alarm 3: Lack of external consent
9	Alarms[4]	Bool	4.4	false	☐	☑	☑	☑	☐		Alarm 4: Timeout feedback encoder encoder
10	Alarms[5]	Bool	4.5	false	☐	☑	☑	☑	☐		Alarm 5: Timeout feedback sensor
11	Alarms[6]	Bool	4.6	false	☐	☑	☑	☑	☐		Alarm 6: Inappropriate weight
12	Alarms[7]	Bool	4.7	false	☐	☑	☑	☑	☐		Alarm 7: Labelling machine warning
13	Alarms[8]	Bool	5.0	false	☐	☑	☑	☑	☐		Alarm 8: Pallet missing
14	Alarms[9]	Bool	5.1	false	☐	☑	☑	☑	☐		Alarm 9: Maintenance request

PLC drive command



entityCoAPServerThread

Thread responsible for modelling CoAP server entity. It receives resource type and various parameter for system identification (port and name) inserted during system startup.

```
@Override
public void run() {

    System.out.println("plcCoAPServerThread start at " + new Date());

    this.coapServer.add(this.coapResource);

    this.coapServer.start();

    while(true) {

        System.out.println("Machine state: " + this.plcGateway.getPlcS7Service().getSiemensPLC().state);

        try {
            Thread.sleep(3000);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

}
```

Gateway device resource

Defines device resource data format published by relative CoAP server. Clients that want dialogue with this device can receive reading data in JSON format so they can build an object directly from received format. The same solution is adopted for sending data from client via POST request.

```
public void handlePOST(CoapExchange exchange) {

    this.postResource = exchange.getRequestText();

    System.out.println("PLC POST: " + this.postResource);

    Gson gson = new Gson();

    PLC plcRcv = gson.fromJson(this.postResource, PLC.class);

    this.plcGateway.getPlcS7Service().getSiemensPLC().reset = plcRcv.reset;
    this.plcGateway.getPlcS7Service().getSiemensPLC().startCommand = plcRcv.startCommand;
    this.plcGateway.getPlcS7Service().getSiemensPLC().stopCommand = plcRcv.stopCommand;

}

@Override
public void handlePUT(CoapExchange exchange) {

    this.putResource = exchange.getRequestText();

    System.out.println("PLC PUT: " + this.putResource);

    Gson gson = new Gson();

    PLC plcRcv = gson.fromJson(this.putResource, PLC.class);

    this.plcGateway.getPlcS7Service().getSiemensPLC().reset = plcRcv.reset;
    this.plcGateway.getPlcS7Service().getSiemensPLC().startCommand = plcRcv.startCommand;
    this.plcGateway.getPlcS7Service().getSiemensPLC().stopCommand = plcRcv.stopCommand;

}


@Override
public void handleGET(CoapExchange exchange) {

    jsonDatataFormatting();

    exchange.respond(ResponseCode.CONTENT, this.measures, MediaTypeRegistry.APPLICATION_JSON);
}
```

Each line is equipped with an additional entity called MonilineGateway.
The main task of MonilineGateway is to collect data from all machines line devices, processing and publishing data to higher level which is interested in a subset of the data produced. It is composed by two different threads:

- **monilineGatewayReadThread:** manages the communication to get published data from CoAP servers of the line
- **monilineCoAPServerThread:** exposes processed data using various server resources to higher level clients

Exposed resource are:

- **EnergyAverageResource:** energy consuption average of all machines in the line
- **LineVelocityAverageResource:** average speed of all machine of the line
- **MachinesStateAverageResource:** state of all machines in the line

Once created, this object receives a list of all entities registered in master repository so it can listen related resources published by different servers of the machines. Received data are:

- CoAP port number
- Device type
- Line ID
- Machine ID
- Device ID

HMI1 - HMIMaintenance

Entity used by maintenance manager who controls all machines of specific line. When starting, it specifies line ID that it wants to observe.
It is also possibile to send settings and command values using the following syntax: ***machineID_code_velocity_deviceID***
Command list:

- 0: START
- 1: STOP
- 2: RESET
- 10: MOTOR VELOCITY
- 20: MOTOR THRESOLD

For example, to set 50 Hz speed of motor n.1 in machine n.1, send the following command: 1_10_50_1

HMI1 - HMIMaintenance

```
---------- Machine 1 ----------
---------- PLC ----------
Machine state: STOPPED
---------- ENERGY ----------
I1: 0.0 A
I2: 0.0 A
I3: 0.0 A
I_Avg: 9.1746E-41 A
L1_L2: 410.22797 V
L2_L3: 409.65057 V
L3_L1: 410.6 V
LL_Avg: 410.15952 V
L1_N: NaN V
L1_N: NaN V
L2_N: NaN V
L3_N: NaN V
LN_Avg: NaN V
ActivePower_P1: NaN kW
ActivePower_P2: NaN kW
ActivePower_P3: NaN kW
ActivePower_T: 0.0 kW
ReactivePower_P1: NaN kVAR
ReactivePower_P2: NaN kVAR
ReactivePower_P3: NaN kVAR
ReactivePower_T: 0.0 kVAR
ApparentPower_P1: NaN kVA
ApparentPower_P2: NaN kVA
ApparentPower_P3: NaN kVA
ApparentPower_T: 0.0 kVA
PF_1: 0.0
PF_2: 0.0
PF_3: 0.0
PF_T: 0.0
Frequency: 50.00719 Hz
Temperature: 32.209488 °C
Active_power_Import_Total: 0.0 kWh
---------- VIBRATION ANALYSIS ----------
--- Sensor 10 ---
Z-Axis RMS Velocity: 0.7931676 in/sec
Z-Axis RMS Velocity: 0.49641746 mm/sec
Temperature: 0.81881934 °F
Temperature: 0.6630024 °C
X-Axis RMS Velocity : 0.5346352 in/sec
X-Axis RMS Velocity : 0.15319604 mm/sec
```

HMI2 - HMIProduction

Entity used by production manager who supervises all plant lines.
It is possible to send command for setting weight and speed threshold required on each machine using following syntax:
lineID_machineID_code_value
Command list:

- 0: LINE VELOCITY SETPOINT
- 1: THR UNIT WEIGHT SETPOINT

For example, to set 10 units/min production line velocity on machine n.1 in line n.1 1, send the following command:
1_1_0_10

HMI2 - HMIProduction

```
-------------- Line 1 --------------------
---------- PLC's ----------
PLC lineID: 1
PLC machineID: 1
PLC state: STOPPED
---------- PLC's ----------
PLC lineID: 1
PLC machineID: 2
PLC state: STOPPED
---------- ENERGY ----------
LL_Avg: 409.77582 V
I_Avg: 9.1746E-41 A
LN_Avg: NaN V
ActivePower_T: 0.0 kW
ReactivePower_T: 0.0 kVAR
ApparentPower_T: 0.0 kVA
PF_T: 0.0
Frequency: 49.98537 Hz
Active_power_Import_Total: 0.0 kWh
---------- LINE VELOCITY ----------
Line velocity average: 3 unit/min
-------------- Line 2 --------------------
---------- PLC's ----------
PLC lineID: 2
PLC machineID: 1
PLC state: STOPPED
---------- PLC's ----------
PLC lineID: 2
```

Device resources

Each entity modeled as a CoAP server is also identified as a resource. In this way, every client that wants to receive through a GET the produced values can receive serialized resource in JSON format and rebuild an object of the same format received through deserialization.
Resources are also prepared for receiving data by POST.
System resources are:

- DriveResource: motor drive resource
- PlcResource: PLC resource
- Pm3200Resource: energy meter resource
- Qm42vt2Resource: vibration sensor resource
- RejectResource: reject resource
- RejectObsResorce: reject resource for observable data

Moniline resources

Moniline part modelled as CoAP server is also identified as resource. In this way, each client HMI2 that wants to receive through a GET request produced values can receive serialized resource in JSON format and rerieve an object of the same format received through deserialization.
Resources are also prepared for receiving data by POST.
System resources are:

- EnergyAverageResource: energy data average from different machines from the same line
- LineVelocityAverageResource: line velocity average from different machines from the same line
- MachineStateAverageResource: states from different machines from the same line

Future work

- Using all real devices on a different IP address
- Load network configuration from config.properties file in order to get network device architecture during application boot
- Storing data in a real database for building predictive maintenance algorithms

Thanks for attention.