Fuzzy Laboratory 3

Applications implemented using components with FLETPN models

1. Laboratory Objectives

* Acquiring the concepts
  + a description of a first-order system with components,
  + implementing components with FLETPN models,
  + controlling a system with a P, PI algorithm.

developing and testing an application using FLETPN models.

1. Application description

We consider a room heated by a boiler. The hot water in the boiler must ensure a certain desired temperature in the room but this is also influenced by the outside temperature. The architecture of the system is shown in Figure 2.1. The room is modeled by a first-order system with multiple inputs and a single output.



Figure 2.1 Plant component

The Room component models the room temperature and has the following input ports:

* *outsideTemp* - Indicates the outside temperature. This is considered a disturbance;
* *windowOpen* - informs whether the window is open or not. This is considered a disturbance;
* *c(k)* - heat from *HeaterTank*.

The Room component has a single exit port, *roomTemp*, which models the current system temperature.

Model for the room component:

x(k+1)=a\*x(k)+b\*c(k)+d1\*v1(k)+d2\*v1(k) where:

x(k) - room temperature;

c(k) – heat given by *HeaterTank*;

v1 – outside temperature;

d1 – coefficient presenting cooling by opening the window;

d2 – coefficient that models the cooling by walls;

d1\*v1(k) – the disturbance caused by opening the window;

d2\*v1(k) – disturbance caused by cooling by walls.

The HeaterTank component that models the boiler has the following ports:

* *heatingOn/heatingOff* –informs whether the heater is turned on or off. If it is turned on, then the water is recycled by radiators;
* *gasCmd –* is an input port variable that models how hard the gas is released.

The HeaterTank component has a c(k) output port, the HeaterTank heat, and the waterTemp port that models the current boiler temperature. When the outside temperature is very low, the boiler water temperature must be raised to warm the room.

The model for the HeaterTank component:

x2(k+1)=a2\*x2(k)+b2\*u(k) where:

x2(k) is the temperature of the hot water in the *HeaterTank* and u(k) is the gas command.

3. The System Description

The control system consists of the component that models the room temperature (Room temperature Component- RTC) and the Heater Controller (Heather Temperature Controller - HTC) that controls the water temperature.



Figure 3.1 The System’s Components

The RTC component is designed to keep the temperature as close as possible to the reference. The reference is given by a user through the *roomTempRef* input port. The *roomTemp* input port models the room temperature.

The HTC component is designed to heat the water to the desired temperature. Figure 3.1 shows the system’s components with the input and output ports.

The mathematical model implements the logic:

IF (x(k)-ref )<-ɛ THEN *heatingOn* is true

IF (x(k)-ref )<+ɛ THEN *heatingOn* is false.

The *heatingOn* output port corresponds to the boiler startup, controls the water recycling and the room heating. The *heatingOff* port corresponds to stopping the boiler, stops the recycling.

The reference is given through the input port: *waterTempRef*. The input port for reading the current water temperature in the boiler is *waterTemp* and the continuous output port for the gas control is *gasCmd*. The HTC component implements a PI-type controller.

3.1 The RTC component model

The RTC component implements a bipolar controller. Figure 3.2 shows the Petri net model of the RTC component. The net is an FLETPN type, the transition T1 has a FLRS differential type table, and the T3 transition is attached to a FLRS comparator table.

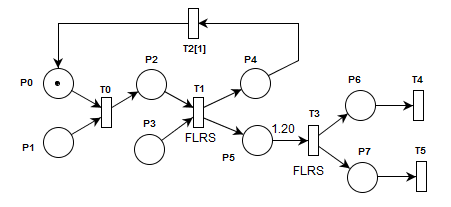


Figure 3.2 The RTC component model

The significance of ports and transitions is as follows:

P1 - is the input port for the *roomTempRef* value;

P3 - is the input port for the current *roomTemp* value;

T0 - reads the reference value and stores it in P2;

T1 - is a differential and calculates the *roomTempRef – roomTemp* error;

T2 - after a delay of 1 unit, the cycle resumes;

T3 - Put a token in the place P6 if the calculated error is NL and for the PL in the place P7, in other cases, the token is consumed (to avoid the accumulation of tokens). In this way, the heating will be started until the temperature reaches *roomTempRef + ɛ* value. It will turn on again when *roomTemp* drops below *roomTempRef - ɛ*. The value of the variable ɛ is the value corresponding to the weight of the weights at the palce P5 at to the transition T3.

3.2 HTC component model

Figure 3.3 shows the Petri net that models the RTC component. The HTC component is built on the P model.

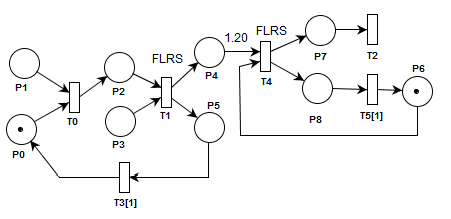


Figure 3.3 HTC component model

The significance of ports and transitions is as follows:

P1 - reads the temperature reference *waterTempRef*;

T0 - stores the reference at place P2;

P3 - reads the water temperature of the boiler *waterTemp*;

T1 - calculates the error *e(k) = waterTempRef(k) - waterTemp(k)* and stores it in P4 and P5;

T3 - after a delay of 1 u.t. (time unit) the cycle resumes;

T4 - computes the current value of the command u(k) = u(k-1) + Δu(k) by summing the two values stored in P7 and P8;

T2 - a continuous output port for the gas control;

T5 –update command with a delay of 1 u.t. the command value towards the boiler, u(k) = u(k-1) and indicates the restart of a new calculation of the command.

4. Implementation in Java

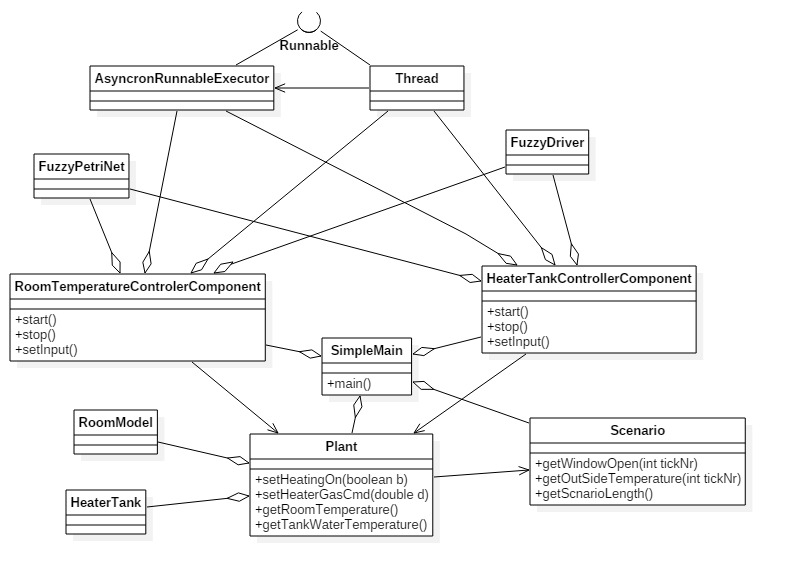
****

Figure 4.1 the class diagram

Figure 4.1 shows the class diagram. The *Plant* class contains the plant model shown above. Methods noted with *set* are input ports and those with *get* are output ports.

The *Scenario* class contains system inputs, user reference, outside temperature, and information about the window if it is closed or open. The class has static methods that randomly generate scenarios such as *extremeEvening*, *winterDay* and *winterMorning*.

The two classes corresponding to the control components contain FLETPN and the executor that is a thread of execution is created with the *AsyncronRunnableExecutor* class. Classes have both input and output ports and are associated with the *Plant* class to be able to execute actions associated with the output ports.

The transitions contain FLRS tables, a fuzzyficator for the input data and a defuzzificator for the output data.

The following is a version of the implementation. It is considered the scenario where the outside temperature varies between -30 and 10 degrees Celsius and the boiler’s water reference varies between 45 and 68 degrees Celsius.

public class HeaterTankControllerComponent {

// HTC component

static String reader = "" +

"{[<NL><NM><ZR><PM><PL>]" +

" [<NL><NM><ZR><PM><PL>]" +

" [<NL><NM><ZR><PM><PL>]" +

" [<NL><NM><ZR><PM><PL>]" +

" [<NL><NM><ZR><PM><PL>]}";

static String doubleChannelDifferentiator = ""//

+ "{[<ZR,ZR><NM,NM><NL,NL><NL,NL><NL,NL>]" //

+ " [<PM,PM><ZR,ZR><NM,NM><NL,NL><NL,NL>]" //

+ " [<PL,PL><PM,PM><ZR,ZR><NM,NM><NL,NL>]"//

+ " [<PL,PL><PL,PL><PM,PM><ZR,ZR><NM,NM>]"//

+ " [<PL,PL><PL,PL><PL,PL><PM,PM><ZR,ZR>]}";

String historyMerger = ""//

+ "{[<NL,NL><NL,NL><NL,NL><NM,NM><ZR,ZR>]" //

+ " [<NL,NL><NL,NL><NM,NM><ZR,ZR><PM,PM>]" //

+ " [<NL,NL><NM,NM><ZR,ZR><PM,PM><PL,PL>]"//

+ " [<NM,NM><ZR,ZR><PM,PM><PL,PL><PL,PL>]"//

+ " [<ZR,ZR><PM,PM><PL,PL><PL,PL><PL,PL>]}";

private AsyncronRunnableExecutor execcutor;

private FullRecorder rec;

private FuzzyDriver tankWaterTemperatureDriver;

private int p1RefInp;

private int p3SysInp;

private FuzzyPetriNet net;

public HeaterTankControllerComponent(Plant plant, long simPeriod) {

// constructing the Petri net for the HTC component

TableParser parser = new TableParser();

net = new FuzzyPetriNet();

int p0 = net.addPlace();

net.setInitialMarkingForPlace(p0, FuzzyToken.zeroToken());

// transition t0 executes Reading

int tr0Reader = net.addTransition(0, parser.parseTwoXOneTable(reader));

p1RefInp = net.addInputPlace();

net.addArcFromPlaceToTransition(p0, tr0Reader, 1.0);

net.addArcFromPlaceToTransition(p1RefInp, tr0Reader, 1.0);

int p2 = net.addPlace();

net.addArcFromTransitionToPlace(tr0Reader, p2);

p3SysInp = net.addInputPlace();

// transition t1 - differentiator

int tr1Diff = net.addTransition(0, parser.parseTwoXTwoTable(doubleChannelDifferentiator));

net.addArcFromPlaceToTransition(p2, tr1Diff, 1.0);

net.addArcFromPlaceToTransition(p3SysInp, tr1Diff, 1.0);

int p4 = net.addPlace();

net.addArcFromTransitionToPlace(tr1Diff, p4);

// transition t2 exit

int tr2Out = net.addOuputTransition(OneXOneTable.defaultTable());

int p5 = net.addPlace();

net.addArcFromTransitionToPlace(tr1Diff, p5);

// transition t3 with delay

int t3 = net.addTransition(1, OneXOneTable.defaultTable());

net.addArcFromPlaceToTransition(p5, t3, 1.0);

net.addArcFromTransitionToPlace(t3, p0);

int p6 = net.addPlace();

net.setInitialMarkingForPlace(p6, FuzzyToken.zeroToken());

// transition t4 adder

int t4Adder = net.addTransition(0, parser.parseTwoXTwoTable(historyMerger));

net.addArcFromPlaceToTransition(p4, t4Adder, 1.2);

net.addArcFromPlaceToTransition(p6, t4Adder, 1.0);

int p7 = net.addPlace();

net.addArcFromTransitionToPlace(t4Adder, p7);

net.addArcFromPlaceToTransition(p7, tr2Out, 1.0);

int p8 = net.addPlace();

net.addArcFromTransitionToPlace(t4Adder, p8);

int t5Delay = net.addTransition(1, OneXOneTable.defaultTable());

net.addArcFromPlaceToTransition(p8, t5Delay, 1.0);

net.addArcFromTransitionToPlace(t5Delay, p6);

// specify the limits for fuzzyfication

FuzzyDriver tankCommandDriver = FuzzyDriver.createDriverFromMinMax(-1.0, 1.0);

// specify the limits for boiler’s water

tankWaterTemperatureDriver = FuzzyDriver.createDriverFromMinMax(-75, 75);

rec = new FullRecorder();

//creating the executor

execcutor = new AsyncronRunnableExecutor(net, simPeriod);

execcutor.setRecorder(rec);

//adding the action for the output transition t2 – gas command

|  |
| --- |
| net.addActionForOuputTransition(tr2Out, new Consumer<FuzzyToken>() { |
| @Override |
| public void accept(FuzzyToken tk) { |
| plant.setHeaterGasCmd(tankCommandDriver.defuzzify(tk)); |
| } |
| }); |

public void start() { (new Thread(execcutor)).start(); }

public void stop() { execcutor.stop(); }

// reading the boiler tempearture

public void setTankWaterTemp(double tankWaterTemperature) {

Map<Integer, FuzzyToken> inps = new HashMap<Integer, FuzzyToken>();

inps.put(p3SysInp, tankWaterTemperatureDriver.fuzzifie(tankWaterTemperature));

execcutor.putTokenInInputPlace(inps); }

// reading the boiler’s temperature reference

public void setWaterRefTemp(double waterRefTemp){

Map<Integer, FuzzyToken> inps = new HashMap<Integer, FuzzyToken>();

inps.put(p1RefInp, tankWaterTemperatureDriver.fuzzifie(waterRefTemp));

execcutor.putTokenInInputPlace(inps); }

//methods for visualizing the Petri net

public FuzzyPetriNet getNet() { return net; }

public FullRecorder getRecorder() { return rec; }

}

public class HeaterTank {

private static final double pipeWaterTemerature = 7;

private static final double maxWaterTemeprature = 75;

private static final double startTemperature = 23;

private static final double theoraticalRoomTemp = 23;

double curentWaterTemepartaure;

public HeaterTank() {

curentWaterTemepartaure = startTemperature; }

public void updateSystem(boolean heaterOn, double gasCmd){

gasCmd = (gasCmd < 0.0) ? 0.0 : ((gasCmd > 1.0) ? 1.0 : gasCmd);

curentWaterTemepartaure += -(curentWaterTemepartaure - pipeWaterTemerature) \* 0.1 \* ((heaterOn) ? 1.0 : 0.0) +

(maxWaterTemeprature - curentWaterTemepartaure) \* 0.4 \* gasCmd

- (curentWaterTemepartaure - theoraticalRoomTemp) \* 0.005; }

public double getHotWaterTemeprature() {

return curentWaterTemepartaure; }

}

public class RoomTemperatureControllerComponent {

static String reader = "" +

"{[<NL><NM><ZR><PM><PL>]" +

" [<NL><NM><ZR><PM><PL>]" +

" [<NL><NM><ZR><PM><PL>]" +

" [<NL><NM><ZR><PM><PL>]" +

" [<NL><NM><ZR><PM><PL>]}";

static String doubleChannelDifferentiator = ""//

+ "{[<ZR,ZR><NM,NM><NL,NL><NL,NL><NL,NL>]" //

+ " [<PM,PM><ZR,ZR><NM,NM><NL,NL><NL,NL>]" //

+ " [<PL,PL><PM,PM><ZR,ZR><NM,NM><NL,NL>]"//

+ " [<PL,PL><PL,PL><PM,PM><ZR,ZR><NM,NM>]"//

+ " [<PL,PL><PL,PL><PL,PL><PM,PM><ZR,ZR>]}";

static String t3Table = "{[<FF,ZR>,<FF,FF>, <FF,FF>, <FF,FF>, <ZR, FF>]}";

private int p1RefInp;

private FuzzyPetriNet net;

private FuzzyDriver temepartureDriver;

private FullRecorder rec;

private AsyncronRunnableExecutor execcutor;

private int p3RealInp;

//receive the reference period for the plant by the constructor

public RoomTemperatureControllerComponent(Plant plant, long simPeriod) {

net = new FuzzyPetriNet();

TableParser parser = new TableParser();

int p0 = net.addPlace();

net.setInitialMarkingForPlace(p0, FuzzyToken.zeroToken());

p1RefInp = net.addInputPlace();

int t0 = net.addTransition(0, parser.parseTwoXOneTable(reader));

net.addArcFromPlaceToTransition(p0, t0, 1.0);

net.addArcFromPlaceToTransition(p1RefInp, t0, 1.0);

int p2 = net.addPlace();

net.addArcFromTransitionToPlace(t0, p2);

p3RealInp = net.addInputPlace();

int t1 = net.addTransition(0, parser.parseTwoXTwoTable(doubleChannelDifferentiator));

net.addArcFromPlaceToTransition(p2, t1, 1.0);

net.addArcFromPlaceToTransition(p3RealInp, t1, 1.0);

int p4 = net.addPlace();

net.addArcFromTransitionToPlace(t1, p4);

int t2 = net.addTransition(1, OneXOneTable.defaultTable());

net.addArcFromPlaceToTransition(p4, t2, 1.0);

net.addArcFromTransitionToPlace(t2, p0);

int p5 = net.addPlace();

net.addArcFromTransitionToPlace(t1, p5);

int t3 = net.addTransition(0, parser.parseOneXTwoTable(t3Table));

int p6 = net.addPlace();

net.addArcFromTransitionToPlace(t3, p6);

int t4 = net.addOuputTransition(OneXOneTable.defaultTable());

net.addArcFromPlaceToTransition(p6, t4, 1.0);

int p7 = net.addPlace();

net.addArcFromTransitionToPlace(t3, p7);

int t5 = net.addOuputTransition(OneXOneTable.defaultTable());

net.addArcFromPlaceToTransition(p7, t5, 1.0);

net.addArcFromPlaceToTransition(p5, t3, 120.0);

net.addActionForOuputTransition(t4, new Consumer<FuzzyToken>() {

@Override

public void accept(FuzzyToken t) {

plant.setHeatingOn(true);

}

});

net.addActionForOuputTransition(t5, new Consumer<FuzzyToken>() {

@Override

public void accept(FuzzyToken t) {

plant.setHeatingOn(false);

}

});

}

temepartureDriver = FuzzyDriver.createDriverFromMinMax(-40, 40);

rec = new FullRecorder();

execcutor = new AsyncronRunnableExecutor(net, simPeriod);

execcutor.setRecorder(rec);

}

public void start() { (new Thread(execcutor)).start(); }

public void stop() { execcutor.stop(); }

public void setInput(double roomTemperatureRef, double roomTemperature) {

Map<Integer, FuzzyToken> inps = new HashMap<Integer, FuzzyToken>();

inps.put(p1RefInp, temepartureDriver.fuzzifie(roomTemperatureRef));

inps.put(p3RealInp, temepartureDriver.fuzzifie(roomTemperature));

execcutor.putTokenInInputPlace(inps);

}

public FuzzyPetriNet getNet() { return net; }

public FullRecorder getRecorder() { return rec; }

}

public class RoomModel {

private static final double StartingTemperature = 24.0;

/\* if the difference between the heated water and the room temperature

Is 1C then the room temperature will increase by <heaterConstant> every

Minute

\*/

private static final double heaterConstant = 0.01;

/\* if the difference between the outside temperature and the room

Temperature is 1C the the room temperature will increase steadily

With <wallConstant> every minute

\*/

private static final double wallConstant = 0.00055;

/\* if the difference between the outside temperature and the room

Temperature is 1C then the window is open and the temperature will

Decrease with <windowConstant> every minuite

\*/

private static final double windowConstant = 0.01;

double currentTemaprature;

public RoomModel() {

currentTemaprature = StartingTemperature; }

public void updateModel(boolean heatingOn, double heaterWaterTemp, boolean windowOpen, double outSideTemp) {

double delatHeater = (heatingOn) ? (heaterWaterTemp - currentTemaprature) : 0.0;

double outsideDelta = currentTemaprature - outSideTemp;

currentTemaprature += delatHeater \* heaterConstant - outsideDelta \* wallConstant -

((windowOpen) ? (outsideDelta \* windowConstant) : 0.0); }

public double getCurrentTemperature() {

return currentTemaprature; }

}

public class Plant {

private volatile boolean heaterOn = false;

private volatile double gasCmd = 0.0;

private int tickCntr = 0;

private long period;

private RoomModel room;

private Scenario scenario;

private HeaterTank tank;

/\* for logs \*/

ArrayList<Double> heaterWaterTempLog = new ArrayList<>();

ArrayList<Double> roomTempLog = new ArrayList<>();

ArrayList<Double> waterHetarCmdLog = new ArrayList<>();

ArrayList<Double> heatOnCmdLog = new ArrayList<>();

int heatOnCntr = 0;

int continousHeatOnMax = 0;

int continousHeatOnCurent = 0;

double tankGasCommandSum = 0.0;

public Plant(long simPeriod, Scenario scen) {

this.period = simPeriod;

room = new RoomModel();

tank = new HeaterTank();

scenario = scen; }

public void setHeatingOn(boolean heaterOn) {

this.heaterOn = heaterOn; }

public void setHeaterGasCmd(double cmd) {

gasCmd = cmd; }

public double getRoomTemperature() {

return room.getCurrentTemperature(); }

public double heatingOnRatio() {

return ((double) heatOnCntr / (double) tickCntr); }

public double gasConsumption() {

return tankGasCommandSum; }

public int maxContiniousHeaterOn() {

return continousHeatOnMax; }

public void start() {

Timer myTimer = new Timer();

TimerTask task = new TimerTask() {

@Override

public void run() {

if (tickCntr < scenario.getScenarioLength()) {

tank.updateSystem(heaterOn, gasCmd);

room.updateModel(heaterOn, tank.getHotWaterTemeprature(), scenario.getWindowOpen(tickCntr),

scenario.getOutSideTemepratue(tickCntr));

makeLogs();

tickCntr++; } else {

myTimer.cancel();

myTimer.purge(); }

}

};

myTimer.scheduleAtFixedRate(task, period, period); }

private void makeLogs() {

heaterWaterTempLog.add(tank.getHotWaterTemeprature());

roomTempLog.add(room.getCurrentTemperature());

waterHetarCmdLog.add(gasCmd);

heatOnCmdLog.add(heaterOn ? 1.0 : 0.0);

heatOnCntr += (heaterOn ? 1.0 : 0.0);

if (heaterOn) {

continousHeatOnCurent++;

} else if (continousHeatOnCurent > 0) {

if (continousHeatOnCurent > continousHeatOnMax) {

continousHeatOnMax = continousHeatOnCurent; }

continousHeatOnCurent = 0; }

tankGasCommandSum += (gasCmd < 0.0) ? 0.0 : gasCmd;

}

public Double getTankWaterTemperature() {

return tank.getHotWaterTemeprature(); }

public Map<String, List<Double>> getTemeartureLogs() {

HashMap<String, List<Double>> logMap = new HashMap<>();

logMap.put("tankTemp", heaterWaterTempLog);

logMap.put("roomTemp", roomTempLog);

return logMap; }

public Map<String, List<Double>> getCommandLogs() {

HashMap<String, List<Double>> logMap = new HashMap<>();

logMap.put("waterCmd", waterHetarCmdLog);

logMap.put("heaterOn", heatOnCmdLog);

return logMap; }

}

public class Scenario {

List<Double> outsideTemperature;

List<Boolean> windowOpen;

public Scenario(List<Double> outsideTempearture, List<Boolean> windowOpen) {

this.outsideTemperature = outsideTempearture;

this.windowOpen = windowOpen; }

public boolean getWindowOpen(int tick) {

return windowOpen.get(tick); }

public double getOutSideTemepratue(int tick) {

return outsideTemperature.get(tick); }

public int getScenarioLength() {

return outsideTemperature.size(); }

private static Scenario scenarioBuilder(double startingTempInHour[], double windowChance[]) {

List<Double> outsideTemperature = new ArrayList<>();

List<Boolean> windowOpen = new ArrayList<>();

Random rnd = new Random();

for (int hour = 0; hour < startingTempInHour.length - 1; hour++) {

double startTemp = startingTempInHour[hour];

double endTemp = startingTempInHour[(hour + 1)];

for (int minute = 0; minute < 60; minute++) {

double temp = startTemp + ((endTemp - startTemp) \* minute) / 60.0 + rnd.nextDouble() \* 0.1;

outsideTemperature.add(temp);

windowOpen.add(rnd.nextDouble() < windowChance[hour]); }

}

return new Scenario(outsideTemperature, windowOpen); }

public static Scenario winterDay() {

double startingTempInHour[] = new double[] { -12.5, -15.0, -17.0, -20.0, -21.0, -19.0, -17.0, -15.0,

-12.0, -8.0, -7.0, -5.0, -4.0, -3.5, -5.0, -4.0, -5.0,

-6.0, -7.5, -8.5, -9.0, -11.0, -11.5, -12.0, -12.0 };

double windowChance[] = new double[] { 0.02, 0.01, 0.01, 0.01, 0.01, 0.01, 0.02, 0.02,

0.08, 0.08, 0.1, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05,

0.05, 0.05, 0.02, 0.02, 0.01, 0.01, 0.01 };

return scenarioBuilder(startingTempInHour, windowChance); }

public static Scenario winterMorning() {

double startingTempInHour[] = new double[] { -19.0, -17.0, -15.0, -12.0 };

double windowChance[] = new double[] { 0.08, 0.04, 0.01, };

return scenarioBuilder(startingTempInHour, windowChance); }

public static Scenario extremeEvening() {

double startingTempInHour[] = new double[] { -5.0, -18.0, -22.0, -27.0 };

double windowChance[] = new double[] { 0.06, 0.03, 0.05, };

return scenarioBuilder(startingTempInHour, windowChance); }

}

public class SimpelMain {

private static final int SIM\_PERIOD = 10;

public static void main(String[] args) {

Scenario scenario = Scenario.winterDay();

Plant plant = new Plant(SIM\_PERIOD, scenario);

HeaterTankControllerComponent tankController = new HeaterTankControllerComponent(plant, SIM\_PERIOD);

RoomTemperatureControllerComponent roomController = new RoomTemperatureControllerComponent(plant, SIM\_PERIOD);

roomController.start();

tankController.start();

plant.start();

double waterRefTemp = 48.0;

double roomTemperature = 24.0;

for (int i = 0; i < scenario.getScenarioLength(); i++) {

tankController.setWaterRefTemp(waterRefTemp);

tankController.setTankWaterTemp(plant.getTankWaterTemperature());

roomController.setInput(roomTemperature, plant.getRoomTemperature());

try {Thread.sleep(10);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace(); }

}

tankController.stop();

roomController.stop();

MainView windowTankController = FuzzyPVizualzer.visualize(tankController.getNet(),

tankController.getRecorder());

MainView windowTermostat = FuzzyPVizualzer.visualize(roomController.getNet(), roomController.getRecorder());

Plotter plotterTemperatureLog = new Plotter(plant.getTemeartureLogs());

Plotter plotterCommandLog = new Plotter(plant.getCommandLogs());

windowTankController.addInteractivePanel("TempLogs", plotterTemperatureLog.makeInteractivePlot());

windowTermostat.addInteractivePanel("TempLogs", plotterTemperatureLog.makeInteractivePlot());

windowTankController.addInteractivePanel("ComandLogs", plotterCommandLog.makeInteractivePlot());

windowTermostat.addInteractivePanel("ComandLogs", plotterCommandLog.makeInteractivePlot());

double[] tankTempStats = SimpelMain.calcStatistics(plant.getTemeartureLogs().get("tankTemp"));

double[] rommTempStsats = SimpelMain.calcStatistics(plant.getTemeartureLogs().get("roomTemp"));

System.out.println("max tank temp :" + tankTempStats[0]);

System.out.println("min tank temp :" + tankTempStats[1]);

System.out.println("avg tank temp :" + tankTempStats[2]);

System.out.println("max room temp :" + rommTempStsats[0]);

System.out.println("min room temp :" + rommTempStsats[1]);

System.out.println("avg room temp :" + rommTempStsats[2]);

System.out.println("heater on ratio:" + plant.heatingOnRatio());

System.out.println("max nr of mins continous heating on:" + plant.maxContiniousHeaterOn());

System.out.println("all consunption ::" + plant.gasConsumption());

System.out.println("avg consunption in a min ::" + plant.gasConsumption() / scenario.getScenarioLength()); }

public static double[] calcStatistics(List<Double> list) {

double min = 1000.0;

double max = 0.0;

double sum = 0.0;

for (Double d : list) {

min = (min > d) ? d : min;

max = (max < d) ? d : max;

sum += d; }

return new double[] { max, min, sum / list.size() }; }

}

Exercises:

1. Test the application for different scenarios.

2. The HTC component controller is constructed on the P model. Starting from this, build the HTC controller component according to the PI model.

3. Try modifying the reference: waterRefTemp to find the value that meets the following scenario:

-the maximum time that the continuous heating is turned on: <8 (min);

- Consumption: <0.10.

4. Starting from the above application, build an application that contains another component called the Outside Reference Calculator (ORC), which has an input port to read the outside temperature and based on that tries to estimate the optimum water temperature in the boiler. The diagram of the components in Figure 4.2 is presented, and in Figure 4.3 is presented the Petri net that model the component.



Figure 4.2 Component diagram

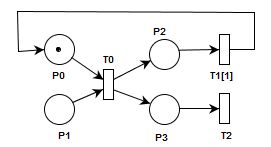
**

Figure 4.3 Petri net of the ORC component

5. Knowledge veirfication

1. What are the system components and what role do they have?
2. When is the executor running and how is it deployed?
3. How does the performance of the system is influenced by the weights of the arcs?
4. What are the instructions that implement the component ports?
5. How to attach FLRS tables to transitions using the FuzzyP utility?