//Conveyor pseudo-code

Data:

List<Glass> glasses //glasses on the conveyor

List<Glass> waitingGlasses //glass that waiting for the sensor

ConveyorState state;

ConveyorFamily cf //just a existing form of

Enum ConveyorState {STOPED, GLASS\_ARRIVED, WAITING\_FOR\_SENSOR, SENDING\_GLASS\_TO\_SENSOR, FRONT\_SENSOR\_CAN\_SEND\_GLASS, NULL, }

Boolean sensorOccupied;

Boolean stopConveyor;

Messages:

msgStop(){

conveyorState = ConveyorState.Stoped;

stopConveyor = true;

}

msgCanISendGlass(Sensor){//sent from front sensor

if(!stopConveyor){

state = ConveyorState .FRONT\_SENSOR\_CAN\_SEND\_GLASS

}

}

msgHereIsGlass(glass){ //sent from front sensor

glasses.add(glass)

state = ConveyorState.GLASS\_ARRIVED;

}

msgIAmOccupied(){

stopConveyor = true; //if back end sensor is occupied, stop the conveyor unless it will crash

backSensorOccupied = true;

}

msgIAmEmpty(){

if (glasses.size() > 0 && !backSensorOccupied)// if there is glass exist

// in the conveyor, send

// it to the back end sensor

state = ConveyorState.SENDING\_GLASS\_TO\_SENSOR;

else if (backSensorOccupied) {//if backSensorOccupied is true, which means conveyor asked to send glass, but was refused by back end sensor, now, flip the Boolean and change the state

backSensorOccupied = false;

stopConveyor = false;

state = ConveyorState.GLASS\_ARRIVED;

}

}

Scheduler

If(!stopConveyor){

if (state == ConveyorState.GLASS\_ARRIVED && !backSensorOccupied

|| name == "Conveyor2") {

// if there is glass coming in, send to back end sensor (because sensor is empty)

glassArrived(glasses.get(0));

return true;

}

if (state == ConveyorState.GLASS\_ARRIVED && backSensorOccupied) {

// wait

state = ConveyorState.WAITING\_FOR\_SENSOR;

return true;

}

if (state == ConveyorState.WAITING\_FOR\_SENSOR

&& waitingGlasses.size() == 0) {

//

notifySensorGlassIsWaiting();

return true;

}

if (state == ConveyorState.WAITING\_FOR\_SENSOR

&& waitingGlasses.size() > 0 && !backSensorOccupied) {

//push glass to the sensor

pushGlassToSensor(waitingGlasses.remove(0));

notifySensorToSendGlass();

return true;

}

if (state == ConveyorState.FRONT\_SENSOR\_CAN\_SEND\_GLASS) {

//front end sensor can send glass

notifySensorToSendGlass();

return true;

}

if (state == ConveyorState.SENDING\_GLASS\_TO\_SENSOR) {

//send glass to back end sensor

sendGlassToSensor();

return true;

}

}else{

stopConveyor();

}

Methods:

sendGlassToSensor() {

// send glass to sensor

cf.sensor2.msgHereIsGlass(this, glasses.remove(0));

if (waitingGlasses.size() > 0)

state = ConveyorState.WAITING\_FOR\_SENSOR;

else

state = ConveyorState.NULL;

stateChanged();

}

notifySensorToSendGlass() {

state = ConveyorState.NULL;

cf.sensor1.msgIAmEmpty(this);

stateChanged();

}

pushGlassToSensor(Glass glass) {

cf.sensor2.msgHereIsGlass(this, glass);

stateChanged();

}

notifySensorGlassIsWaiting() {

cf.sensor2.msgGlassIsWaiting(this);

waitingGlasses.add(glasses.get(0));

stateChanged();

}

stopConveyor() {

stopConveyor = true;

cf.sensor1.msgIAmOccupied(this);

stateChanged();

}

glassArrived(Glass glass) {

if (name == "Conveyor1") {

cf.sensor2.msgCanISendGlass(this);

} else if (name == "Conveyor2") {// do nothing now.

cf.sensor2.msgCanISendGlass(this);

}

}

getName() {

return name;

}