Conveyor Entry Sensor Agent

Data:

1. List<Part> parts;
2. enum SensorState {Pressed, Released, Nothing};
3. SensorState sensorState = SensorState.Nothing;
4. Conveyor conveyor;
5. ConveyorFamily prevConv;

Messaging:

1. public void msgHereIsParts(Part p) {

parts.add(p);

stateChanged();

}

2. public void msgPressed() {

sensorState = SensorState.*Pressed*;

print("Sensor Pressed");

stateChanged();

}

3. public void msgReleased() {

sensorState = SensorState.*Released*;

print("Sensor Released");

stateChanged();

}

Scheduler:

1. if(sensorState == SensorState.Pressed) {

informPrevConveyor();

return true;

}

1. if(sensorState == SensorState.Released) {

informNextConveyor();

informPrevConveyor();

sensorState = SensorState.Nothing;

return true;

}

Actions:

1. private void informPrevConveyor() {

prevConv.msgisAvailable(sensorState == SensorState.Released);

if (sensorState == SensorState.Pressed) {

print("Stop Conveyor");

}

else {

print("Start Conveyor");

}

}

1. private void informNextConveyor() {

conveyor.msgHereIsParts(parts.remove(0));

}

Conveyor Agent:

Data:

1. List<Part> parts = Collections.synchronizedList(new ArrayList<Part>());
2. enum ConveyorState {Running, Stopped};
3. ConveyorState conveyorState = ConveyorState.Running;
4. boolean changed = false;
5. PostSensor exitSensor;

Messaging:

1. public void msgHereIsParts(Part p) {

parts.add(p);

stateChanged();

}

1. public void msgStopConveyor() {

conveyorState = ConveyorState.Stopped;

changed = true;

stateChanged();

}

1. public void msgStartConveyor() {

conveyorState = ConveyorState.Running;

changed = true;

stateChanged();

}

Scheduler:

1. if (changed) {

doConveyor();

return true;

}

1. if (!parts.isEmpty()) {

sendParts();

return true;

}

Actions:

1. private void doConveyor() {

if (conveyorState == ConveyorState.Running) {

print("Start Conveyor");

transducer.fireEvent(TChannel.CONVEYOR,TEvent.CONVEYOR\_DO\_START, index);

}

else {

print("Stop Conveyor");

transducer.fireEvent(TChannel.CONVEYOR,TEvent.CONVEYOR\_DO\_STOP, index);

}

changed = false;

}

1. private void sendParts() {

print("Give parts to post sensor");

exitSensor.msgHereIsParts(parts.remove(0));

}

Conveyor Exit Sensor Agent

Data:

1. List<Part> parts;
2. Part currentPart = null;
3. enum SensorState {Pressed, Released, Nothing};
4. enum PopupState {Up, Down};
5. enum ConveyorState {Running, Stopped};
6. SensorState sensorState = SensorState.Nothing;
7. PopupState popupState = PopupState.Down;
8. ConveyorState conveyorState = ConveyorState.Running;
9. Conveyor conveyor;
10. Popup popup;
11. Semaphore sem = new Semaphore(0, true);

Messaging:

1. public void msgHereIsParts(Part p) {

print("Receive Part " + p.type);

parts.add(p);

stateChanged();

}

1. public void msgPressed() {

sensorState = SensorState.Pressed;

stateChanged();

}

1. public void msgReleased() {

sensorState = SensorState.Released;

sem.release();

print("Parts released");

stateChanged();

}

1. public void msgPopupLow() {

popupState = PopupState.Down;

print("Pop Lowered");

stateChanged();

}

1. public void msgPopupRaise() {

popupState = PopupState.Up;

print("Pop Raised");

stateChanged();

}

Scheduler:

1. if (sensorState == SensorState.Pressed && popupState == PopupState.Up && conveyorState == ConveyorState.Running) {

stopConveyor();

return true;

}

1. if (popupState == PopupState.Down && conveyorState == ConveyorState.Stopped) {

startConveyor();

return true;

}

1. if (sensorState == SensorState.Released && conveyorState == ConveyorState.Running) {

loadPopup();

sensorState = SensorState.Nothing;

return true;

}

Actions:

1. private void stopConveyor() {

conveyor.msgStopConveyor();

conveyorState = ConveyorState.Stopped;

}

1. private void startConveyor() {

conveyor.msgStartConveyor();

conveyorState = ConveyorState.Running;

}

1. private void loadPopup() {

print("Give parts to popup");

popup.msgHereIsParts(parts.remove(0));

}

Popup Agent

Data:

1. List<MyWorkstation> workstations;
2. MyParts currentPart = null;
3. ConveyorFamily nextConv;
4. static enum ConveyorState {Running, Stopped};
5. static enum PartState {PartsComing, NeedMoving, NeedProcessing, Processed, Leaving, Leaved};
6. static enum WorkstationState {Working, Stopped};
7. ConveyorState conveyorState = ConveyorState.Running;
8. static enum PopupState {Up, Down};
9. PopupState popupState = PopupState.Down;
10. type = "x";
11. PostSensor s;
12. Semaphore sem = new Semaphore(0, true);
13. boolean isStopped = false;
14. boolean flag;

Private Class:

1. private class MyParts {

public Part part;

public PartState partState;

boolean isProcessed;

public MyParts(Part p, PartState s) {

part = p;

partState = s;

isProcessed = false;

}

}

1. private class MyWorkstation {

public Workstation workStation;

public WorkstationState state;

boolean needMove;

public MyWorkstation(Workstation w, WorkstationState s) {

workStation = w;

state = s;

needMove = false;

}

}

Messaging:

1. public void msgConveyorReady() {

conveyorState = ConveyorState.Running;

stateChanged();

}

1. public void msgConveyorStopped() {

conveyorState = ConveyorState.Stopped;

stateChanged();

}

1. public void msgHereIsParts(Part p) {

print("Parts Received");

currentPart = new MyParts(p, PartState.PartsComing);

stateChanged();

}

1. public void msgSendGlassToPopup(Part p) {

msgHereIsParts(p);

currentPart.isProcessed = true;

}

1. public void msgLoadingFinished() {

sem.release();

stateChanged();

}

1. public void msgMovePopup(Workstation ws) {

synchronized (workstations) {

for (MyWorkstation w : workstations) {

if (w.workStation == ws) {

print("found");

w.needMove = true;

}

}

}

stateChanged();

}

1. public void msgMoveFinished() {

sem.release();

stateChanged();

}

1. public void msgIsAvailable(Workstation w, boolean s) {

synchronized (workstations) {

for (MyWorkstation ws : workstations) {

if (ws == w) {

if (s)

ws.state = WorkstationState.Working;

else

ws.state = WorkstationState.Stopped;

}

}

}

stateChanged();

}

1. public void msgPopupReleased() {

sem.release();

stateChanged();

}

Scheduler:

1. if (isStopped && getWorkstation() != null && currentPart == null && popupState == PopupState.Down) {

print("1");

startConveyor();

return true;

}

1. if (flag && popupState == PopupState.Down) {

stopWorkstations();

flag = false;

return true;

}

1. if (currentPart != null && currentPart.partState == PartState.PartsComing) {

loadPopup();

if (popupState == PopupState.Up || (popupState == PopupState.Down && currentPart.part.type == type))

currentPart.partState = PartState.NeedMoving;

else

currentPart.partState = PartState.Leaving;

return true;

}

1. if (there exist w in workstations such that w. needMove == true && popupState == PopupState.Up) {

informRobot(ws);

return true;

}

1. if (there exist w in workstations such that w. needMove == true && currentPart == null) {

stopConveyor();

movePopup(1);

return true;

}

1. if (currentPart != null && currentPart.partState == PartState.NeedMoving) {

if (getWorkstation() == null) {

if (!isStopped)

stopConveyor();

return true;

}

movePopup(popupState == PopupState.Down ? 1 : 0);

if (!currentPart.isProcessed)

currentPart.partState = PartState.NeedProcessing;

else

currentPart.partState = PartState.Leaving;

return true;

}

1. if (currentPart != null && currentPart.partState == PartState.NeedProcessing) {

processPart(currentPart);

currentPart = null;

movePopup(0);

startConveyor();

return true;

}

1. if (currentPart != null && currentPart.partState == PartState.Leaving && conveyorState == ConveyorState.Running) {

releasePopup();

currentPart = null;

return true;

}

Actions:

1. private void loadPopup() {

stopConveyor();

print("Loading popup");

msgLoadingFinished();

while(!sem.tryAcquire());

print("Loading finished");

}

1. private void stopConveyor() {

s.msgPopupRaise();

isStopped = true;

}

1. private void startConveyor() {

s.msgPopupLow();

isStopped = false;

}

1. private void movePopup(int direction) {

if (direction == 0) {

Do("Move Down");

transducer.fireEvent(TChannel.POPUP, TEvent.POPUP\_DO\_MOVE\_DOWN, args);

msgMoveFinished();

popupState = PopupState.Down;

}

else {

Do("Move Up");

transducer.fireEvent(TChannel.POPUP, TEvent.POPUP\_DO\_MOVE\_DOWN, args);

msgMoveFinished();

popupState = PopupState.Up;

}

while(!sem.tryAcquire());

print("Move finished");

}

1. private void releasePopup(){

print("Released");

transducer.fireEvent(TChannel.POPUP, TEvent.POPUP\_RELEASE\_GLASS, args);

nextConv.msgHereIsParts(currentPart.part);

startConveyor();

}

1. private void processPart(MyParts p) {

print(p.part.type + " is processing");

MyWorkstation temp = getWorkstation();

temp.workStation.msgSendGlass(p.part);

temp.state = WorkstationState.Stopped;

}

1. private void informRobot(MyWorkstation ws) {

print("I'm ready");

ws.workStation.msgIsPopupAvailable();

ws.needMove = false;

}

1. private void stopWorkstations() {

for (MyWorkstation ws : workstations) {

ws.workStation.msgIsPopupAvailable(false);

}

}

1. private MyWorkstation getWorkstation() {

for (MyWorkstation ws : workstations) {

if (ws.state == WorkstationState.Working)

return ws;

}

return null;

}

Interfaces

public interface Conveyor {

public void msgHereIsParts(Part p);

public void msgStopConveyor();

public void msgStartConveyor();

public void setPostSensor(PostSensor s);

public String getName();

}

public interface ConveyorFamily{

public void msgisAvailable(boolean state);

public void msgHereIsParts(Part p);

}

public interface Popup {

public void msgConveyorReady();

public void msgConveyorStopped();

public void msgHereIsParts(Part p);

public void msgSendGlassToPopup(Part p);

public void msgLoadingFinished();

public void msgMovePopup(Workstation ws);

public void msgMoveFinished();

public void setPostSensor(PostSensor s);

public void setNextConvFamily(ConveyorFamily conv);

public void addWorkStation(Workstation ws);

public String getName();

}

public interface PostSensor {

public void msgHereIsParts(Part p);

public void msgPressed();

public void msgReleased();

public void msgPopupLow();

public void msgPopupRaise();

public void setConveyor(Conveyor p);

public void setPopupAgent(Popup p);

public String getName();

}

public interface PreSensor {

public void msgHereIsParts(Part p);

public void msgPressed();

public void msgReleased();

public void setConveyor(Conveyor c);

public void setPrevConv(ConveyorFamily conv);

public String getName();

}

public interface Workstation {

public void msgIsPopupAvailable(boolean state);

public void msgSendGlass(Part p);

}