Team 01

CS201

Factory v.1 Design Doc

4/13/13

**Content:**

I. Interaction Diagrams

II. Pseudo Codes

**Yinong Dai**

**Design of Agents**

*Design: Conveyor Family*

**Data**

int index;

String function;

ConveyorAgent conveyor;

PopupAgent popup;

InlineProcessingAgent inline;

**Messages**

void msgIAmFree() {

popup.msgIAmFree();

}

void msgHereIsGlass(glass) {

conveyor.msgHereIsGlass(glass);

}

void msgHereIsFinishedGlass(glass) {

popup.msgHereIsFinishedGlass(glass);

}

**Scheduler**

N/A

**Actions**

N/A

*Design: Conveyor Agent*

**Data**

int conveyorIndex;

String function;

Mode mode;

enum Mode {Offline, Inline};

Transducer transducr;

List<Glass> glasses;

PopupAgent popup;

PopupState popupState;

enum PopupState {BUSY, FREE};

InlineAgent inline;

boolean inlineFree;

ConveyorFamily previouscf;

boolean conveyorRunning;

SensorState sensor1State;

SensorState sensor2State;

enum SensorState {PRESSED, RELEASED, NOTHING};

**Messages / Eventfires**

void msgHereIsGlass(glass) {

glasses.add(new MyGlass(glass, INITIAL);

stateChanged();

}

void msgPopupBusy() {popupState = BUSY;}

void msgPopupFree() {popupState = FREE;}

void eventFired() { //Registered to SENSOR channel

if( (event == SENSOR\_GUI\_PRESSED) &&

(args[0]/2 == conveyorIndex) ) {

if(args[0]%2 == 0) sensor1State = PRESSED;

if(args[0]%2 == 1) sensor2State = PRESSED;

}

if( (event == SENSOR\_GUI\_RELEASED) &&

(args[0]/2 == conveyorIndex) ) {

if(args[0]%2 == 0) sensor1State = RELEASED;

if(args[0]%2 == 1) sensor2State = RELEASED;

}

}

**Scheduler**

if(mode == OFFLINE) {

if( (popupState == BUSY) && (sensor2State == PRESSED) && (conveyorRunning) ) {stopConveyorAndNotify(); return true;}

if( (popupState == FREE) && (!conveyorRunning) ) {startConveyor(); return true;}

if( sensor2State == RELEASED ) {giveGlassToPopup(); return true;}

}

if(mode == INLINE) {

if( (!inlineFree ) && (sensor2State == PRESSED) && (conveyorRunning ) {stopConveyor(); return true;}

if( (inlineFree) && (!conveyorRunning) ) {startConveyor(); return true;}

if( sensor2State == RELEASED ) {giveGlassToInline(); return true;}

}

if( sensor1State == RELEASED ) {notifyPrevious(); return true;}

return false;

**Actions**

void stopConveyor() {

transducer.fireEvent(CONVEYOR, CONVEYOR\_DO\_STOP);

conveyorRunning = false;

}

void stopConveyorAndNotify() {

stopConveyor();

popup.msgIHaveGlass(glasses.get(0));

}

void startConveyor() {

transducer.fireEvent(CONVEYOR, CONVEYOR\_DO\_START);

conveyorRunning = true;

}

void giveGlassToPopup() {

popup.msgHereIsGlass(glasses.remove(0));

sensor2State == NOTHING;

}

void giveGlassToInline() {

inline.msgHereIsGlass(glasses.remove(0));

sensor2State = NOTHING;

}

void notifyPrevious() {

previous.msgIAmFree();

sensor1State == NOTHING;

}

*Design: Pop-up Agent*

**Data**

int popupIndex;

String function;

Transducer transducr;

Operator[2] operators;

boolean[2] operatorFree;

ConveyorFamily next;

boolean nextFree;

ConveyorAgent conveyor;

List<MyGlass> glasses;

class MyGlass {Glass glass; GlassState state; int conveyorIndex;}

enum GlassState {ON\_ENTRY, PENDING, NEED\_PROCESSING, DONE, PASS, NOTHING};

PopupState popupState;

PopupLevel popupLevel;

enum PopupState {LOADED,EMPTY};

enum LevelState {UP, DOWN};

Semaphore loadSemaphore;

Semaphore elevSemaphore;

**Messages / Eventfires**

void msgIHaveGlass(glass) {

for(MyGlass g: glasses) {

if(g.glass == glass) {

g.state = ONLINE;

return;

}

}

glasses.add(new MyGlass(glass, ONLINE);

stateChanged();

}

void msgHereIsGlass(glass) {

for(MyGlass g: glasses) {

if(g.glass == glass) {

loadSemaphore.acquire();

popupState = LOADED;

g.state = PENDING;

}

stateChanged();

}

void msgIAmFree() {

nextFree = true;

}

void msgIHaveGlassFinished() {

for(MyGlass g: glasses) {

if(g.glass == glass)

g.state = DONE;

}

stateChanged();

}

void msgHereIsFinishedGlass(glass) {

for(MyGlass g: glasses) {

if(g.glass == glass) {

loadSemaphore.acquire();

popupState = LOADED;

g.state = PASS;

}

}

stateChanged();

}

void eventFired(…) {

if( (event == POP\_UP\_LOAD\_FINISHED) && (args[0] == popupIndex) ) {

loadSemaphore.release();

}

if( (event == POP\_UP\_RELEASED\_FINISHED) && (args[0] == popupIndex) ) {

loadSemaphore.release();

}

if( (event == POP\_UP\_MOVED\_UP) && (args[0] == popupIndex) ) {

elevatorSemaphore.release();

}

if( (event == POP\_UP\_MOVED\_DOWN) && (args[0] == popupIndex) ) {

elevatorSemaphore.release();

}

**Scheduler**

if there exists MyGlass g such that g.state == DONE then {acceptFinishedGlass(g);return true;}

if there exists MyGlass g such that g.state == PENDING then identifyGlass();return true;}

if there exists MyGlass g such that g.state == PASS && nextFree then {pushGlass();return true;}

if there exists MyGlass g such that g.state == NEED\_PROCESSING && (operatorFree[0] || operatorFree[1]) then {giveOperatorGlass();return true;}

if there exists MyGlass g such that g.state == ONLINE then {acceptGlass();return true;}

return false;

**Actions**

void acceptFinishedGlass(g) {

raisePopup();

operator[g.operatorIndex].msgIAmFree();

loadSemaphore.acquire();

g.state = PASS;

operatorFree[g.operatorIndex] = true;

}

void giveOperatorGlass(g) {

if(operatorFree[0]) {

raisePopup();

operator[0].msgHereIsGlass(g.glass);

//transducer.fireEvent(WORK\_STATION, WORKSTATION\_DO\_LOAD\_GLASS,popupIndex\*2);

loadSemaphore.acquire();

operatorFree[0] = false;

//Here, Operator Agent will make the animation load the glass. When loading is finished, semaphore should be released

g.state = NOTHING;

} else if(operatorFree[1]) {

raisePopup();

operator[1].msgHereIsGlass(g.glass);

//transducer.fireEvent(WORK\_STATION, WORKSTATION\_DO\_LOAD\_GLASS,popupIndex\*2+1);

loadSemaphore.acquire();

operatorFree[1] = false;

//Here, Operator Agent will make the animation load the glass. When loading is finished, semaphore should be released

g.state = NOTHING;

}

}

void raisePopup() {

transducer.fireEvent(POP\_UP, POP\_UP\_DO\_MOVE\_UP);

conveyor.msgPopupBusy();

elevatorSempahore.acquire();

popupLevel = UP;

}

void lowerPopup() {

transducer.fireEvent(POP\_UP, POP\_UP\_DO\_MOVE\_DOWN);

elevatorSempahore.acquire();

popupLevel = DOWN;

}

*Design: Inline Agent*

**Data**

String function;

Transducer transducr;

TChannel channel;

Glass glassOnSpot;

ConveyorFamily next;

boolean nextFree;

ConveyorAgent conveyor;

Semaphore machineSemaphore;

**Messages**

void msgHereIsGlass(glass) {

glassOnSpot = glass;

stateChanged();

}

void msgIAmFree() {

nextFree = true;

stateChanged();

}

void eventFired(,,,) {

if(Event == WORKSTATION\_GUI\_ACTION\_FINISHED)

machineSemaphore.release();

if(Event == WORKSTATION\_RELEASE \_FINISHED)

machineSemaphore.release();

}

**Scheduler**

if glassOnSpot != null then {processGlass(glassOnSpot); return true;}

return false;

**Action**

void processGlass(glass) {

transducer.fireEvent(channel, WORKSTATION\_DO\_ACTION);

machineSemaphore.acquire();

while(!nextFree) {}

next.msgHereIsGlass(glassOnSpot);

transducer.fireEvent(channel, WORKSTATION\_RELEASE\_PART);

machineSemaphore.acquire();

glassOnSpot = null;

conveyor.msgIAmFree();

}

**Alex Jones**

Bin Agent Design

Data

\*\*\*\*\*

V1\_GUI gui;

Queue<TransducerEvent> trasnducerEvents;

Queue<GlassRequest> requests;

enum TransducerEvent {glassCreated}

class GlassRequest{

public boolean dealtWith = false;

Glass glass;

}

boolean currentlyCreatingGlass = false;

boolean nextCFFree = false;

Messages:

\*\*\*\*\*\*\*\*

//from transducer

msgGlassCreated(){

transducerEvents.add(TransducerEvent.glassCreated);

}

msgIAmFree(){

nextCFFree = true;

stateChanged();

}

msgCreateGlass(Barcode bc){

GlassReqquest glassReq = new Glass(bc.translateToRecipe);

glassReq.dealtWith = false;

requests.add(glass);

stateChanged();

}

Scheduler:

\*\*\*\*\*\*\*\*\*

pickAndExecuteAnAction(){

if (there exists req in requests such that !req.dealtWith && nextCFFree && !currentlyCreatingGlass){

createGlassGUI();

return t;

}

if (there exists req in requests such that !req.dealtWith && !nextCFFree ){

warnInGUIWaitingForCFFree();

return t;

}

if (there exists req in requests such that !req.dealtWith && currentlyCreatingGlass)){

warnInGUICreatingGlass();

return t;

}

else if (there exists req in requests such that req.dealtWith){

removeRequest(req);

}

if (there exists event in transducerEvents such that event.type == glassCreated)

{

giveGlassToNCCutter();

return t;

}

return f;

}

Actions:

\*\*\*\*\*\*\*\*

removeRequest(req){

requests.remove(req);

}

createGlassGUI(){

currentlyCreatingGlass = true;

transducer.fireEvent(CREATE\_BIN);

}

giveGlassToNCCutter(){

nextCFFree = false;

//get first glass in requests

ncCutter.conveyor.msgHereIsGlass(request.remove(0).glass);

currentlyCreatingGlass = false;

}

warnInGUIWaitingForCFFree() {

V1\_GUI.warnWaitingForCFFree();

}

warnCreatingGlass(){

V1\_GUI.warnCreatingGlass();

}

Conveyor Agent Design

public Semaphore popupPush;

ConveyorFamily parentCF;

PopupAgent popupAgent;

EntryAgent entryAgent;

public enum ConveyorEvent {sensorPressed, sensorReleased};

public ArrayList<ConveyorEvent> events;

public enum ConveyorState {busy, freeNotNotified, freeNotified};

public enum PopupStatus {busy, unknown, free};

public PopupStatus popupStatus;

public ConveyorState conveyorState;

void msgSensorPressed() {

events.add(ConveyorEvent.sensorPressed);

stateChanged();

}

void msgSensorReleased() {

events.add(ConveyorEvent.sensorReleased);

stateChanged();

}

void msgPopupIsReady(){

popupStatus = PopupStatus.free;

popupPush.release();

}

public boolean pickAndExecuteAnAction() {

if (!events.isEmpty())

{

ConveyorEvent event;

event = events.remove(0);

if (event == ConveyorEvent.sensorPressed){

pushGlass(); //multistep action, will use a semaphore to guarantee not to push on full/raised popup

/\*

//told by the gui that a glass is on the sensor

if (popupStatus != PopupStatus.free){

stopConveyor();

}

else if (popupStatus == PopupStatus.free){

//let the glass move on through

}\*/

}

else if (event == ConveyorEvent.sensorReleased){

notifyEntryFree();

}

return true;

}

if (conveyorState == ConveyorState.freeNotNotified){

notifyEntryFree();

}

return false;

}

void stopConveyor() {

parentCF.stopConveyor();

conveyorState = ConveyorState.busy;

}

void notifyEntryFree() {

String msg = new String(name + ": notified entry agent that I'm free");

System.out.println(msg);

log.add(new LoggedEvent(msg));

entryAgent.msgConveyorFree();

conveyorState = ConveyorState.freeNotified;;

}

void startConveyor(){

parentCF.startConveyor();

}

void pushGlass() {

String msg = new String(name + ": letting popup know i have glass, stopping conveyor in mean time...");

System.out.println(msg);

//log.add(new LoggedEvent(msg));

popupAgent.msgHereIsGlass();

stopConveyor();

try {

popupPush.acquire();

} catch (InterruptedException e) {

e.printStackTrace();

}

startConveyor();

System.out.println(name + ": glass pushed to popup.");

//assume popup is busy

popupStatus = PopupStatus.busy;

}

}

Conveyor Family Agent Design

DATA:

\*\*\*\*\*\*\*

ConveyorFamily previousCF, nextCF;

EntryAgent entryAgent;

ConveyorAgent conveyorAgent;

PopupAgent popupAgent;

int conveyorNumber;

class myGlassPiece {

boolean needsProcessing; //determined in constructor

Glass glass;

boolean beenProcessed;

}

enum StageInLine {onEntry, onConveyor, onPopup, onlineProcessing, exiting}

Queue<myGlassPiece> glassPieces;

Operator operatorUp, operatorDown;

MSGS:

\*\*\*\*

msgHereIsGlass(Glass g){ //from previous family

entryAgent.msgHereIsGlass(g);

}

msgHereIsFinishedGlass(Operator o, Glass g){

popupAgent.msgHereIsFinshedGlass(o,g);

}

msgIAmFree(){ //from next family

popupAgent.msgIAmFree();

}

sendImFreeMsg(){

prevCF.msgImFree();

}

msgGiveGlassToNext(){

nextCF.msgHereIsGlass(glassPieces.remove(last).glass);

}

SCHEDULER:

\*\*\*\*\*\*\*\*\*

ACTIONS:

\*\*\*\*\*\*\*\*

void notifyPreviousFamilyFree(){

//simple as sending one message to previous

previousCF.msgIAmFree();

}

void giveGlassToNextCF(Glass g){

nextCFFamily.msgHereIsGlass(g);

glassToPush.remove(g);

}

Entry Agent Design

ConveyorFamilyAgent parentCF;

ConveyorAgent conveyor;

enum LineStatus {busy, freeNotNotified, freeNotified};

LineStatus lineStatus = freeNotNotified;

enum ConveyorStatus {busy, free};

ConveyorStatus conveyorStatus = busy;

MSGS:

\*\*\*\*\*\*\*\*

msgSensorPressed(){

linestatus = busy;

event.add(sensorpressed);

stateChanged();

}

msgSensorReleased(){

linestatus = freeNotNotified;

event.add(sensorreleased);

stateChanged();

}

msgConveyorFree(){

conveyorState = free;

}

Scheduler:

\*\*\*\*\*\*\*\*\*

if (!events.empty()){

if (event == sensorpressed && conveyorstatus = busy){

stopConveyor();

}

if (event == sensorreleased)

notifypreviouscffree();

return true;

}

if (lineStatus == freeNotNotified){

notifyPreviousCFFree();

return true;

}

ACTIONS:

\*\*\*\*\*\*\*\*

void stopConveyor(){

transducer.stopconveyor()

linestatus = busy;

}

notifyPreviousCFFree(){

parentCF.msgSendImFreeMsg();

}

Popup Agent Design

Data

AlexsConveyorFamily parentCF;

ConveyorFamily nextCF;

MyOperator upOperator, downOperator;

enum NextCFState {busy, free};

NextCFState nextCFState;

enum UpDownState {Up, Down};

UpDownState upDownState;

enum CurrentState {emptyNoCurrentProcessing, fullNoCurrentProcessing, emptyCurrentProcessing, fullCurrentProcessing};

enum CurrentState {empty,full};

CurrentState popupState;

enum PopupEvent {loadFinished, popupMovedUp, newGlassFromConveyor, releaseFinished, popupMovedDown, finishedGlassFromOperator, nextCFIsFree};

ArrayListPopupEvent popupEvents;

boolean requestToPushGlass = false;

Semaphore waitingForFinishedGlass;

ConveyorAgent conveyor;

ArrayListAlexsConveyorFamily.MyGlass glassOnCF;

public class MyOperator {

Operator operator;

public boolean occupied = false;

public boolean requestOpen = false;

public Operator getOperator(){return operator;}

}

void msgLoadFinished() {

popupState = CurrentState.full;

popupEvents.add(PopupEvent.loadFinished);

stateChanged();

}

void msgReleaseFinished(){

popupState = CurrentState.empty;

popupEvents.add(PopupEvent.releaseFinished);

stateChanged();

}

void msgPopupMovedUp() {

log.add(new LoggedEvent(msg));

upDownState = UpDownState.Up;

popupEvents.add(PopupEvent.popupMovedUp);

stateChanged();

}

void msgPopupMovedDown() {

upDownState = UpDownState.Down;

popupEvents.add(PopupEvent.popupMovedDown);

stateChanged();

}

void msgImFree() {

// next cf is ready

nextCFState = NextCFState.free;

popupEvents.add(PopupEvent.nextCFIsFree);

stateChanged();

}

void msgHereIsFinishedGlass(Operator o) {

popupEvents.add(PopupEvent.finishedGlassFromOperator);

if(upOperator.operator == o ){

upOperator.requestOpen = false;

waitingForFinishedGlass.release();

}

else if(downOperator.operator == o ){

downOperator.requestOpen = false;

waitingForFinishedGlass.release();

}

stateChanged();

}

void msgHereIsGlass() {

requestToPushGlass = true;

popupEvents.add(PopupEvent.newGlassFromConveyor);

stateChanged();

}

void msgIHaveFinshedGlass(Operator o) {

if (o == upOperator.operator)

upOperator.requestOpen = true;

else if (o == downOperator.operator)

downOperator.requestOpen = true;

popupEvents.add(PopupEvent.finishedGlassFromOperator);

stateChanged();

}

public int numOperatorsCurrentlyOccupied(){

int count = 0;

if (upOperator.occupied) count++;

if (downOperator.occupied) count++;

return count;

}

public boolean pickAndExecuteAnAction() {

if (!popupEvents.isEmpty()){

PopupEvent event = popupEvents.remove(0);

switch(event){

case loadFinished

andleLoadFinished();

break;

case nextCFIsFree

handleNextCFIsFree();

break;

case popupMovedDown

handlePopupDown();

break;

case popupMovedUp

handlePopupUp();

break;

case finishedGlassFromOperator

handleNewFinishedGlassRequest();

break;

case newGlassFromConveyor

handleNewGlassFromConveyor();

break;

case releaseFinished

handleReleaseFinished();

break;

}

return true;

}

return false;

}

void handleNewFinishedGlassRequest() {

if (upDownState == UpDownState.Up){

if (popupState == CurrentState.empty)

acceptFinishedGlass();

else if (popupState == CurrentState.full)

at this point, something is weird, move down just in case

moveDown();

}

else if(upDownState == UpDownState.Down){

if (popupState == CurrentState.empty)

moveUp();

else if (popupState == CurrentState.full)

;

}

}

void moveDown() {

parentCF.movePopupDown();

}

void moveUp() {

parentCF.movePopupUp();

}

void acceptNewGlass(){

requestToPushGlass = false;

//get new glass

conveyor.msgPopupIsReady();

//gui will now push glass onto popup

}

void handleReleaseFinished() {

if (upDownState == UpDownState.Down ){

if (numOperatorsCurrentlyOccupied() 2 && requestToPushGlass){

acceptNewGlass();

}

else if (requestToPushGlass && !parentCF.doesLastGlassOnCFNeedProcessing()){

acceptNewGlass();

}

else if (requestToGiveFinishedGlassExists()){

moveUp();

}

}

else if (upDownState == UpDownState.Up){

if (parentCF.doesLastGlassOnCFNeedProcessing() && (numOperatorsCurrentlyOccupied() 2)){

moveDown();

}

}

}

boolean requestToGiveFinishedGlassExists() {

return (upOperator.requestOpen downOperator.requestOpen);

}

void handleLoadFinished() {

if (upDownState == UpDownState.Up){

moveDown();

}

else if (upDownState == UpDownState.Down){

if (parentCF.lastItemBeenProcessed() !parentCF.doesLastGlassOnCFNeedProcessing()){

if (nextCFState == NextCFState.busy)

parentCF.stopConveyor();

else

pushGlass();

}

else

moveUp();

}

}

void handlePopupUp() {

if (popupState == CurrentState.full)

giveGlassToFreeOperator();

else if (popupState == CurrentState.empty)

{

if (requestToGiveFinishedGlassExists())

acceptFinishedGlass();

else if (numOperatorsCurrentlyOccupied() == 0 parentCF.numGlassOnLine() != 0)

moveDown();

}

}

void giveGlassToFreeOperator() {

popupState = CurrentState.empty;

if (!upOperator.occupied)

{parentCF.giveGlassToOperator(upOperator.operator);

upOperator.occupied = true;

}

else if (!downOperator.occupied)

{parentCF.giveGlassToOperator(downOperator.operator);

downOperator.occupied = true;

}

else

print( error in giving glass to operator, none available);

}

void acceptFinishedGlass(){ // multistep action

if (upOperator.requestOpen){

upOperator.operator.msgIAmFree();

try {

waitingForFinishedGlass.acquire();

} catch (InterruptedException e) {

TODO Auto-generated catch block

e.printStackTrace();

}

upOperator.requestOpen = false;

}

if (downOperator.requestOpen){

downOperator.operator.msgIAmFree();

try {

waitingForFinishedGlass.acquire();

} catch (InterruptedException e) {

TODO Auto-generated catch block

e.printStackTrace();

}

//acquired

downOperator.requestOpen = false;

}

}

void handleNextCFIsFree() {

if (upDownState == UpDownState.Down){

if (popupState == CurrentState.full && (parentCF.lastItemBeenProcessed() !parentCF.doesLastGlassOnCFNeedProcessing()) ){

pushGlass();

}

}

}

void handlePopupDown() {

if (popupState == CurrentState.full){

if (nextCFState == NextCFState.free){

pushGlass();

}

else{

}

}

else if (popupState == CurrentState.empty && parentCF.doesLastGlassOnCFNeedProcessing())empty

{

if (numOperatorsCurrentlyOccupied() == 2){

parentCF.stopConveyor();

}

else if (numOperatorsCurrentlyOccupied() 2){

acceptNewGlass();

}

}

else if (popupState == CurrentState.empty && !parentCF.doesLastGlassOnCFNeedProcessing())empty

{

acceptNewGlass();

}

}

void handleNewGlassFromConveyor() {

if (popupState == CurrentState.empty && upDownState == UpDownState.Down){

if (parentCF.doesLastGlassOnCFNeedProcessing()){

if (numOperatorsCurrentlyOccupied() 2)

acceptNewGlass();

else

;conveyor should be stopped, so no need to stop

}

else if (!parentCF.doesLastGlassOnCFNeedProcessing() ){

acceptNewGlass();

}

}

else if (popupState == CurrentState.empty && upDownState == UpDownState.Up){

if (numOperatorsCurrentlyOccupied() 2 && parentCF.doesLastGlassOnCFNeedProcessing())

moveDown();

else if (!parentCF.doesLastGlassOnCFNeedProcessing())

moveDown();

}

}

void pushGlass(){

popupState = CurrentState.empty;

nextCFState = NextCFState.busy;

parentCF.pushGlassOnPopup();

}

boolean inEmptyState() {

TODO Auto-generated method stub

return (popupState == CurrentState.empty);

}

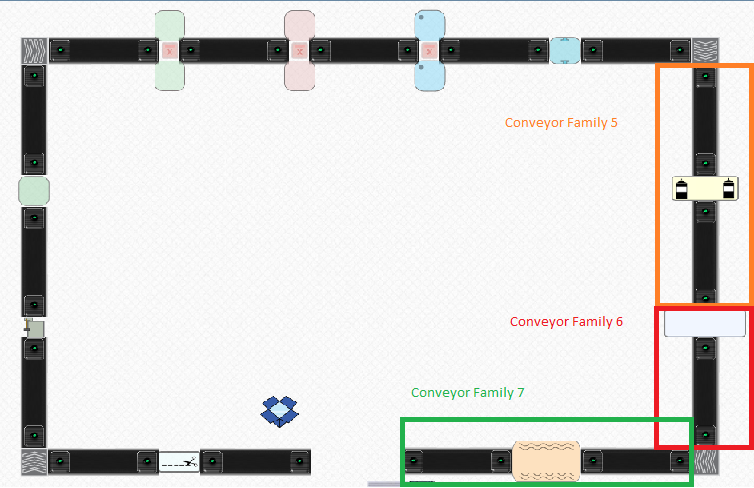
public boolean upNotDown(){

return upDownState == UpDownState.Up;

}

Dongyoung Jung

CSCI201 V1 design document by Dongyoung Jung



Conveyor Family 5 : Conveyor10, Paint Machine, Conveyor11

Conveyor Family 6 : UV Lamp Machine, Conveyor12

Conveyor Family 7 : Conveyor13, Oven Machine, Conveyor14

< Design Hierarchy >

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | Component | |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  | Machine | |  |  | Conveyor | |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  | Paint | |  |  | Conveyor 10 | |  |  |
|  |  | UV Lamp | |  |  | Conveyor 11 | |  |  |
|  |  | Oven | |  |  | Conveyor 12 | |  |  |
|  |  |  |  |  |  | Conveyor 13 | |  |  |
|  |  |  |  |  |  | Conveyor 14 | |  |  |

< Component.java >

// DATA

protected Transducer transducer;

protected String name;

protected Glass tempGlass; // Glass to be received from Previous Conveyor Family

protected boolean nextCompFree = true; // Next Component's status

protected boolean newGlass = false; // New glass on Front Sensor

protected boolean checkPass = false; // New glass on Next Sensor

protected boolean checkDone = true; // Glass passed to next Component safely?

// Constructor

protected Component(String name){

this.name = name;

super.startThread();

}

// MESSAGE

msgIAmFree(){

nextCompFree = true;

if( !checkDone ){ checkPass = true; } // If a glass still stays, it needs to be sent.

stateChanged();

}

msgHereIsGlass(Glass glass){

tempGlass = glass;

}

// SCHEDULER – Nothing for scheduler. This is a super class.

protected boolean pickAndExecuteAnAction() {

return false;

}

// ACTION - Nothing for Action. This is a super class.

< Machine.java >

// DATA

protected Glass glass;

private TChannel channel; // Type of machine

private boolean loadFinished = false, actionFinished = false, releaseFinished = false;

// Constructor

protected Machine(String name, TChannel channel) {

super(name);

this.channel = channel;

}

// SCHEDULER

protected boolean pickAndExecuteAnAction(){

if( loadFinished ){

doWorkAction();

return true;

}

if( actionFinished ){

actionFinishedAction();

return true;

}

if( releaseFinished ){

releaseFinishedAction();

return true;

}

return false;

}

// ACTION

void doWorkAction(){

loadFinished = false;

if( glass.getRecipe( channel ) ){

// Machine work start(Animation)

}

else if( !glass.getRecipe( channel ) ){

actionFinishedAction();

}

}

void actionFinishedAction(){

if( nextCompFree ){

// Release glass from machine(Animation)

passGlassAction();

actionFinished = false;

}

}

void releaseFinishedAction(){

glass = null;

notifyIAmFreeAction();

releaseFinished = false;

}

// These two functions are implemented in each machine class.

protected void notifyIAmFreeAction(){}

protected void passGlassAction(){}

// From Transducer

// Plays a role of message transmission

void eventFired(TChannel channel, TEvent event, Object[] args) {

if( channel == this.channel ){

if( event == TEvent.WORKSTATION\_LOAD\_FINISHED ){

glass = tempGlass;

loadFinished = true;

}

else if( event == TEvent.WORKSTATION\_GUI\_ACTION\_FINISHED ){

actionFinished = true;

}

else if( event == TEvent.WORKSTATION\_RELEASE\_FINISHED ){

releaseFinished = true;

}

stateChanged();

}

}

< Conveyor.java >

// DATA

protected CopyOnWriteArrayList<Glass> glasses = new CopyOnWriteArrayList<Glass>();

private Integer[] conveyorNum = new Integer[1];

private int frontSensorNum, backSensorNum, sensorNum;

protected boolean glassLeaveFront = false;

// Constructor

protected Conveyor(String name, int num, int frontSensorNum, int backSensorNum) {

super(name);

conveyorNum[0] = num;

this.frontSensorNum = frontSensorNum;

this.backSensorNum = backSensorNum;

}

// SCHEDULER

protected boolean pickAndExecuteAnAction(){

// New Glass on Front Sensor

if( newGlass ){

newGlassAction();

return true;

}

// Glass leaves Front Sensor

if( glassLeaveFront ){

glassLeaveFrontAction();

return true;

}

// New Glass on Back Sensor

if( checkPass ){

checkPassAction();

return true;

}

return false;

}

// ACTION

void checkPassAction(){

// Conveyor Stop(Animation)

if( nextCompFree ){

nextCompFree = false;

passGlassAction();

// Conveyor Start(Animation)

checkDone = true;

}

else{

checkDone = false;

}

checkPass = false;

}

void newGlassAction(){

// Conveyor Stop(Animation)

glasses.add( tempGlass );

tempGlass = null;

newGlass = false;

conveyorCheck();

}

void glassLeaveFrontAction(){

glassLeaveFront = false;

notifyIAmFreeAction();

}

// From Transducer

// Plays a role of message transmission

void eventFired(TChannel channel, TEvent event, Object[] args) {

sensorNum = (Integer)args[0];

if( channel == TChannel.SENSOR ){

if( event == TEvent.SENSOR\_GUI\_PRESSED ){

if( sensorNum == frontSensorNum ){

newGlass = true;

}

else if( sensorNum == backSensorNum ){ checkPass = true;

}

stateChanged();

}

else if( event == TEvent.SENSOR\_GUI\_RELEASED ){

if( sensorNum == frontSensorNum ){

glassLeaveFront = true;

stateChanged();

}

}

}

}

void conveyorCheck(){

// Glass on Front Sensor or on Conveyor, but no Glass on Back Sensor

if( ( newGlass || !glasses.isEmpty() ) && !checkPass ){

// Conveyor Stop(Animation)

}

else{

// Conveyor Start(Animation)

}

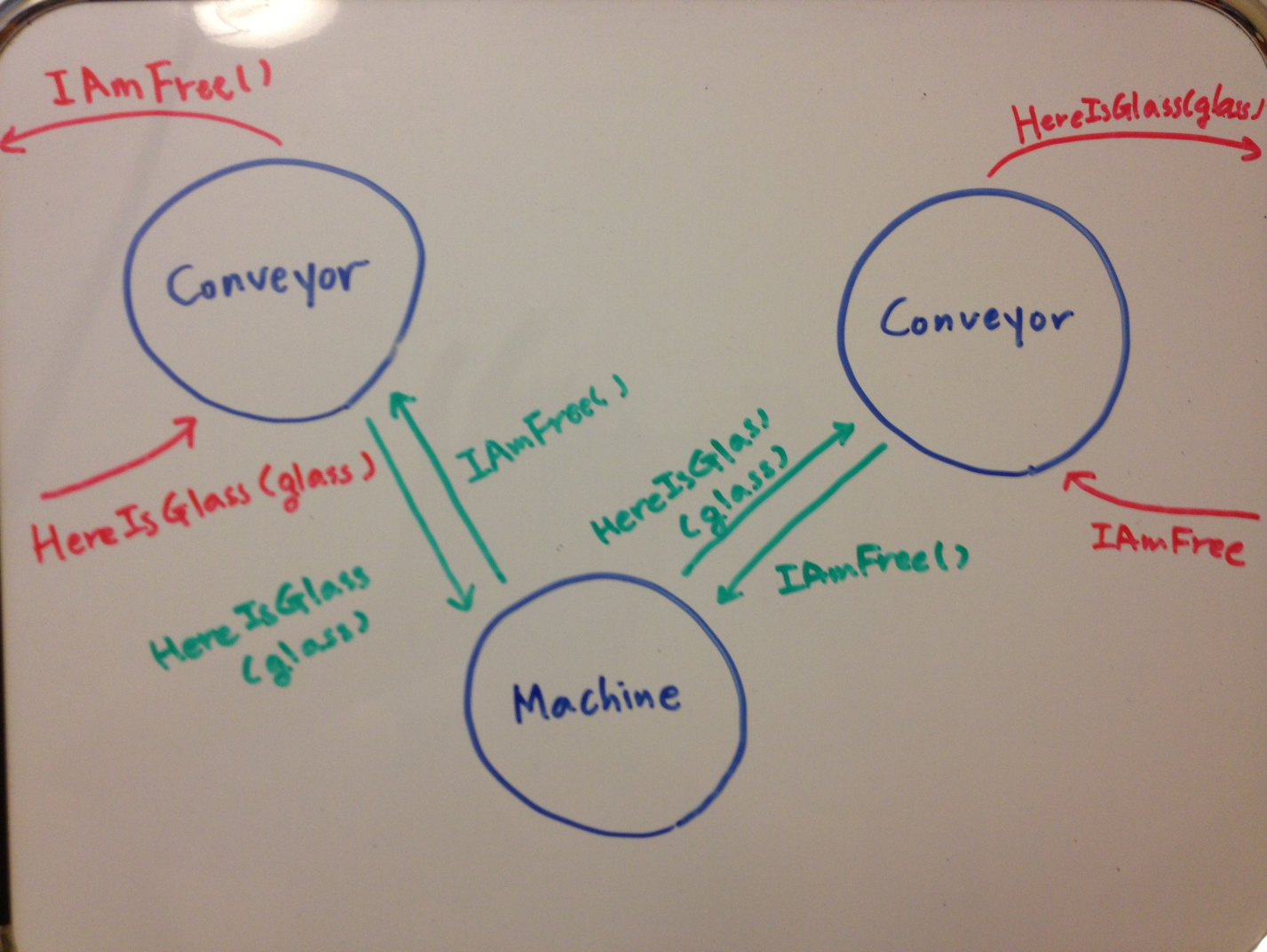
}

// These two functions are implemented in each machine class.

protected void passGlassAction(){}

protected void notifyIAmFreeAction(){}

< Group Diagram >



< Each Conveyor Class >

// DATA

private ConveyorFamily previousFamily;

private PaintAgent paint;

// Constructor

public ConveyorAgentXX (String name, int num, int frontSensorNum, int backSensorNum){

super(name, num, frontSensorNum, backSensorNum);

}

// ACTION

protected void notifyIAmFreeAction(){

previousFamily.msgIAmFree();

}

protected void passGlassAction(){

paint.msgHereIsGlass( glasses.remove(0) );

}

< Each Machine Class >

// DATA

private ConveyorAgent11 conveyor11;

private ConveyorAgent12 conveyor12;

public XXXMachine(String name){

super(name, TChannel.XXXX);

}

// ACTION

protected void notifyIAmFreeAction(){

conveyorXX.msgIAmFree();

}

protected void passGlassAction(){

conveyorXX.msgHereIsGlass( glass );

glass = null;

}

Poojan

public class ConveyorAgent\_PJ extends Agent implements Conveyor\_PJ {

private String name;

private int number;

private Transducer myTransducer;

public ConveyorFamily MyFamily;

public ConveyorFamily NEXTFamily;

public ConveyorFamily PREVIOUSFamily;

private Popup\_PJ mypopup;

private InLineMachine\_PJ myinline;

public enum GlassStatusConveyor{NEW,DONE,ONENTRYSENSOR,CHECKED, ONEXITSENSOR, NEEDSMACHINEPROCESSING, NOMACHINEPROCESSING, CHECKINGPROCESSING, FIRSTDONE, INLINEBUSY};

private Boolean isPopUpBusy;

private Boolean isINLINEBusy;

private Boolean isConveyorRunning;

public boolean isNextConveyorFamilyBusy;

private Boolean secondconveyorfree;

private List<MyCGlass> glassonconveyor = Collections.synchronizedList(new ArrayList<MyCGlass>());

private List<MyOperators> operatorlist = Collections.synchronizedList(new ArrayList<MyOperators>());

public ConveyorAgent\_PJ(String string,int number, ConveyorFamily c1, Transducer transducer,Popup\_PJ p1,InLineMachine\_PJ p2,ConveyorFamily cp) {

// TODO Auto-generated constructor stub

this.name=string;

this.number=number;

this.MyFamily=c1;

this.mypopup=p1;

this.myinline=p2;

this.PREVIOUSFamily=cp;

secondconveyorfree=true;

this.isPopUpBusy=false;

this.isINLINEBusy=false;

myTransducer = transducer;

myTransducer.register(this, TChannel.CUTTER);

myTransducer.register(this, TChannel.SENSOR);

myTransducer.register(this, TChannel.ALL\_AGENTS);

Object[] args={number};

isConveyorRunning=true;

isNextConveyorFamilyBusy=false;

myTransducer.fireEvent(TChannel.CONVEYOR,TEvent.CONVEYOR\_DO\_START,args);

}

public class MyCGlass

{

private Glass pcglass;

private GlassStatusConveyor status;

private Boolean NeedsProcessing;

private MyOperators myoperator;

public MyCGlass(Glass g)

{

this.pcglass=g;

this.status=GlassStatusConveyor.NEW;

}

}

public class MyOperators

{

private Operator\_PJ op;

private boolean occupied;

public MyOperators(Operator\_PJ o)

{

this.op=o;

this.occupied=false;

}

}

public boolean pickAndExecuteAnAction() {

synchronized(glassonconveyor){

for(MyCGlass mg:glassonconveyor){

if(mg.status == GlassStatusConveyor.ONENTRYSENSOR ){

print("CHecking the glass");

checktheglass(mg);

return true;

}

}

};

synchronized(glassonconveyor){

for(MyCGlass mg:glassonconveyor){

if(mg.status == GlassStatusConveyor.ONEXITSENSOR ){

{PassingGlassToInLineMachine(mg);

return true;

}

}

}

};

synchronized(glassonconveyor){

for(MyCGlass mg:glassonconveyor){

if(mg.status == GlassStatusConveyor.INLINEBUSY && secondconveyorfree ){

{starttheconveyor(mg);

return true;

}

}

}

};

synchronized(glassonconveyor){

for(MyCGlass mg:glassonconveyor){

if(mg.status == GlassStatusConveyor.NOMACHINEPROCESSING && isPopUpBusy==false ){

PassingGlassToPopup(mg);

return true;

}

}

};

synchronized(glassonconveyor){

for(MyCGlass mg:glassonconveyor){

if(mg.status == GlassStatusConveyor.NEEDSMACHINEPROCESSING){

// PassingGlassToPopupToProcess(mg);

PassingGlassToInLineMachine(mg);

return true;

}

}

};

synchronized(glassonconveyor){

for(MyCGlass mg:glassonconveyor){

if(mg.status == GlassStatusConveyor.DONE){

// PassingGlassToPopupToProcess(mg);

glassonconveyor.remove(mg);

return true;

}

}

}

// TODO Auto-generated method stub

return false;

}

void starttheconveyor(MyCGlass mg) {

// TODO Auto-generated method stub

print("start conveyor 0");

Object [] no={this.getNumber()};

myTransducer.fireEvent(TChannel.CONVEYOR,TEvent.CONVEYOR\_DO\_START,no);

mg.status=GlassStatusConveyor.ONEXITSENSOR;

stateChanged();

}

void eventFired(TChannel channel, TEvent event, Object[] args) {

// TODO Auto-generated method stub

if(channel == TChannel.SENSOR)

{

if(event == TEvent.SENSOR\_GUI\_PRESSED)

{

if((Integer)args[0]==0)

{

synchronized(glassonconveyor){

for(MyCGlass mg:glassonconveyor){

if(mg.pcglass.getNumber() == (Integer)args[1]){

{

Object [] no={this.getNumber()};

if(isConveyorRunning)

{

myTransducer.fireEvent(TChannel.CONVEYOR,TEvent.CONVEYOR\_DO\_START,no);

isConveyorRunning=true;

}

mg.status=GlassStatusConveyor.ONENTRYSENSOR;

print("SENSOR PRESSED");

print("Conveyor started"+this.getNumber());

stateChanged();

return;

}

}

}

}

};

}

if(event == TEvent.SENSOR\_GUI\_RELEASED)

{

if((Integer)args[0]==0)

{

print("Sensor 0 Released");

this.PREVIOUSFamily.msgIAmFree();

}

}

if(event == TEvent.SENSOR\_GUI\_PRESSED)

{

if((Integer)args[0]==1)

{

synchronized(glassonconveyor){

for(MyCGlass mg:glassonconveyor){

if(mg.pcglass.getNumber() == (Integer)args[1]){

{

print("2nd sensor");

Object [] no={this.getNumber()};

isConveyorRunning=true;

mg.status=GlassStatusConveyor.ONEXITSENSOR;

stateChanged();

return;

}

}

}

}

};

}

if(event == TEvent.SENSOR\_GUI\_PRESSED)

{

if((Integer)args[0]==2)

{

if(isNextConveyorFamilyBusy)

{

}

else

{

Object[] cno ={1};

print("NCCUTTER : 3rd sensor");

myTransducer.fireEvent(TChannel.CONVEYOR,TEvent.CONVEYOR\_DO\_START,cno);

}

secondconveyorfree=false;

}

}

if(event == TEvent.SENSOR\_GUI\_RELEASED)

{

if((Integer)args[0]==2)

{

isINLINEBusy=false;

secondconveyorfree=true;

}

}

if(event == TEvent.SENSOR\_GUI\_PRESSED)

{

if((Integer)args[0]==3)

{

synchronized(glassonconveyor)

{

for(MyCGlass mg:glassonconveyor){

// if(mg.pcglass.getNumber() == (Integer)args[1]){

{

Object[] cno ={1};

myTransducer.fireEvent(TChannel.CONVEYOR,TEvent.CONVEYOR\_DO\_STOP,cno);

if(!(isNextConveyorFamilyBusy))

{

print("4th sensor");

Object[] cno1 ={1};

isNextConveyorFamilyBusy=true;

// myTransducer.fireEvent(TChannel.CONVEYOR,TEvent.CONVEYOR\_DO\_STOP,cno1);

this.NEXTFamily.msgHereIsGlass(mg.pcglass);

print("RELEASE THE GLASS. PROCESSING DONE");

Object[] args1 = {1};

myTransducer.fireEvent(TChannel.CONVEYOR,TEvent.CONVEYOR\_DO\_START,cno);

mg.status=GlassStatusConveyor.DONE;

stateChanged();

return;

}

// }

}

}

}

}

}

if(event == TEvent.SENSOR\_GUI\_RELEASED)

{

if((Integer)args[0]==3)

{

}

}

}

}

// MESSAGES

void msgHereIsGlass(Glass g1) {

// TODO Auto-generated method stub

print("Glass Recieved. Conveyor Start"+this.number);

MyCGlass mcg = new MyCGlass(g1);

mcg.NeedsProcessing = mcg.pcglass.getRecipe(TChannel.CUTTER);

glassonconveyor.add(mcg);

print("glass number "+g1.getNumber());

stateChanged();

}

void msgIamFree() {

// TODO Auto-generated method stub

isPopUpBusy=false;

}

void msgOperatorIsfree(Operator\_PJ operatorAgent) {

// TODO Auto-generated method stub

for(MyOperators mg:operatorlist){

if(mg.op == operatorAgent ){

mg.occupied=false;

}

}

}

// ACTIONS

void checktheglass(MyCGlass mg) {

// TODO Auto-generated method stub

print("Checking the functionalities of glass"+mg.pcglass.getRecipe(TChannel.CUTTER));

mg.NeedsProcessing=mg.pcglass.getRecipe(TChannel.CUTTER);

mg.status=GlassStatusConveyor.CHECKED;

stateChanged();

}

void PassingGlassToPopup(MyCGlass mg) {

// TODO Auto-generated method stub

print("Glass passed to the popup. Conveyor starts again");

this.mypopup.msgGlassDoesNotNeedProcessing(mg.pcglass);

Object[] args1={this.number};

myTransducer.fireEvent(TChannel.CONVEYOR,TEvent.CONVEYOR\_DO\_START,args1);

isConveyorRunning=true;

mg.status=GlassStatusConveyor.DONE;

stateChanged();

}

void PassingGlassToInLineMachine(MyCGlass mg) {

// TODO Auto-generated method stub

if(isINLINEBusy)

{

Object[] cno={0};

myTransducer.fireEvent(TChannel.CONVEYOR,TEvent.CONVEYOR\_DO\_STOP,cno);

print("Conveyor Stop because Inline Is BUSY");

mg.status=GlassStatusConveyor.INLINEBUSY;

}

else

{

this.myinline.msgGlassNeedsProcessing(mg.pcglass,mg.NeedsProcessing);

print("Glass passed to inline machine");

mg.status=GlassStatusConveyor.FIRSTDONE;

isINLINEBusy=true;

}

stateChanged();

}

public String getName(){

return name;

}

public int getNumber(){

return number;

}

public int getglassonconveyorsize(){

return glassonconveyor.size();

}

public int getoperatorlist(){

return operatorlist.size();

}

public Boolean getisPopUpBusy(){

return this.isPopUpBusy;

}

public Boolean getisINLINEBusy()

{

return this.isINLINEBusy;

}

void setOperator(Operator\_PJ o1) {

// TODO Auto-generated method stub

}

public class ConveyorFamily\_PJ implements ConveyorFamily

{

private int ConveryorFamilyNo;

private ConveyorAgent\_PJ conveyor;

private PopupAgent\_PJ popup;

private InLineMachineAgent\_PJ inline;

private ConveyorFamily nextConveyorFamily;

private ConveyorFamily previousConveyorFamily;

public boolean isNextConveyorFamilyBusy;

private Transducer transducer;

private boolean isHalfFamily;

public ConveyorFamily\_PJ(int number, Transducer transducer2,ConveyorFamily cprev)

{

this.previousConveyorFamily=cprev;

this.ConveryorFamilyNo=number;

this.transducer=transducer2;

this.popup = new PopupAgent\_PJ("MyPopup",number,this,transducer);

this.inline = new InLineMachineAgent\_PJ("MyInline",number,this,transducer);

this.inline.setConveyor(conveyor);

this.conveyor = new ConveyorAgent\_PJ("MyConveyor",number,this,transducer, popup, inline,cprev);

// this.popup.setConveyor(conveyor);

isNextConveyorFamilyBusy=false;

}

void msgHereIsGlass(Glass glass) {

// TODO Auto-generated method stub

this.conveyor.msgHereIsGlass(glass);

}

public ConveyorFamily getNextConveyorFamily()

{

return nextConveyorFamily;

}

public String getName() {

// TODO Auto-generated method stub

return null;

}

void msgIAmFree() {

System.out.println("My CFnumber is"+this.ConveryorFamilyNo+"I am Free received from NEXT CONVEYOR FAMILY");

isNextConveyorFamilyBusy=false;

this.conveyor.isNextConveyorFamilyBusy=false;

}

void msgIHaveFinishedGlass(Operator\_PJ o) {

// TODO Auto-generated method stub

System.out.println("operator has finished processing Glass");

this.popup.msgOperatorHasFinishedGlass(o);

}

void msgHereIsFinishedGlass(Glass g, Operator\_PJ operatorAgent) {

// TODO Auto-generated method stub

System.out.println("Sending it back to popup");

this.popup.msgHereIsFinishedGlass(g,operatorAgent);

}

public ConveyorAgent\_PJ getConveyor()

{

return this.conveyor;

}

void msgHereIsFinishedGlass(Operator operator, Glass glass) {

// TODO Auto-generated method stub

}

void msgIHaveGlassFinished(Operator operator) {

// TODO Auto-generated method stub

}

void setNextConveyorFamily(ConveyorFamily c3) {

// TODO Auto-generated method stub

nextConveyorFamily=c3;

this.conveyor.NEXTFamily=nextConveyorFamily;

}

void startThreads() {

// TODO Auto-generated method stub

this.conveyor.startThread();

this.inline.startThread();

//this.popup.startThread();

}

void setPreviousConveyorFamily(ConveyorFamily c2) {

// TODO Auto-generated method stub

this.conveyor.PREVIOUSFamily=c2;

}

}

public class InLineMachineAgent\_PJ extends Agent implements InLineMachine\_PJ {

private int number;

private String name;

private Transducer myTransducer;

public Conveyor\_PJ myconveyor;

public ConveyorFamily MyFamily;

private ConveyorFamily NEXTFamily;

private List<MyPGlass> glassoninline = Collections.synchronizedList(new ArrayList<MyPGlass>());

private List<MyPGlass> finishedglassonpopup = Collections.synchronizedList(new ArrayList<MyPGlass>());

private boolean finisheddone;

public enum GlassStatusInline{NEW,CHECKING,NOPROCESSING,PROCESSING,BEINGPROCESSED, DONE,DONE2, PROCESSINGDONE};

public InLineMachineAgent\_PJ(String string, int i, ConveyorFamily conveyorFamily,

Transducer transducer) {

// TODO Auto-generated constructor stub

this.name=string;

this.number=i;

this.MyFamily=conveyorFamily;

myTransducer = transducer;

myTransducer.register(this, TChannel.ALL\_GUI);

myTransducer.register(this, TChannel.CUTTER);

myTransducer.register(this, TChannel.CONVEYOR);

myTransducer.register(this, TChannel.ALL\_AGENTS);

}

public class MyPGlass

{

private Glass pcglass;

private GlassStatusInline status;

Boolean NeedsProcessing;

public MyPGlass(Glass g, boolean b)

{

this.pcglass=g;

this.status=GlassStatusInline.NEW;

this.NeedsProcessing=b;

print(""+b);

}

}

public boolean pickAndExecuteAnAction() {

synchronized(glassoninline){

for(MyPGlass mg:glassoninline){

if(mg.status == GlassStatusInline.CHECKING ){

checkingforglass(mg);

return true;

}

}

};

synchronized(glassoninline){

for(MyPGlass mg:glassoninline){

if(mg.status == GlassStatusInline.PROCESSING ){

processtheglassaction(mg);

return true;

}

}

};

synchronized(glassoninline){

for(MyPGlass mg:glassoninline){

if(mg.status == GlassStatusInline.NOPROCESSING ){

shiptheglasstonextconveyor(mg);

return true;

}

}

};

synchronized(glassoninline){

for(MyPGlass mg:glassoninline){

if(mg.status == GlassStatusInline.PROCESSINGDONE ){

shiptheglasstonextconveyor(mg);

return true;

}

}

};

synchronized(glassoninline){

for(MyPGlass mg:glassoninline){

if(mg.status == GlassStatusInline.DONE ){

glassoninline.remove(mg);

return true;

}

}

};

// TODO Auto-generated method stub

return false;

}

void shiptheglasstonextconveyor(MyPGlass mg) {

// TODO Auto-generated method stub

mg.status=GlassStatusInline.DONE;

myTransducer.fireEvent(TChannel.CUTTER, TEvent.WORKSTATION\_RELEASE\_GLASS, null);

stateChanged();

}

void glassdone(MyPGlass mg)

{

mg.status=GlassStatusInline.DONE2;

stateChanged();

}

void processtheglassaction(MyPGlass mg) {

// TODO Auto-generated method stub

myTransducer.fireEvent(TChannel.CUTTER, TEvent.WORKSTATION\_DO\_ACTION, null);

print("Workstation do action");

mg.status=GlassStatusInline.DONE2;

stateChanged();

}

void eventFired(TChannel channel, TEvent event, Object[] args) {

// TODO Auto-generated method stub

if(channel == TChannel.CUTTER)

{

if(event == TEvent.WORKSTATION\_LOAD\_FINISHED)

{

synchronized(glassoninline){

for(MyPGlass mg:glassoninline){

if(mg.status == GlassStatusInline.NEW ){

print("cutter called");

mg.status=GlassStatusInline.CHECKING;

stateChanged();

return;

}

}

}

}

if(event == TEvent.WORKSTATION\_LOAD\_FINISHED)

{

synchronized(glassoninline){

for(MyPGlass mg:glassoninline){

if(mg.status == GlassStatusInline.DONE2 ){

mg.status=GlassStatusInline.PROCESSINGDONE;

print("Load fininshed");

stateChanged();

return;

}

}

}

}

if(event == TEvent.WORKSTATION\_GUI\_ACTION\_FINISHED)

{

synchronized(glassoninline){

for(MyPGlass mg:glassoninline){

if(mg.status == GlassStatusInline.DONE2 ){

mg.status=GlassStatusInline.PROCESSINGDONE;

stateChanged();

return;

}

}

}

}

}

}

public String getName(){

return name;

}

public int getNumber(){

return number;

}

public int getglassonpopupsize(){

return glassoninline.size();

}

public int getfinishedglassonpopupsize(){

return finishedglassonpopup.size();

}

void setConveyor(Conveyor\_PJ conveyor) {

// TODO Auto-generated method stub

this.myconveyor=conveyor;

}

// MESSAGES

void msgGlassDoesNotNeedProcessing(Glass glass) {

print("Glass received. Glass does not need processing");

// TODO Auto-generated method stub

glassoninline.add(new MyPGlass(glass,false));

stateChanged();

}

void msgGlassNeedsProcessing(Glass pcglass,Boolean choice) {

// TODO Auto-generated method stub

print("Glass received. Glass needs processing"+choice+pcglass.getNumber());

glassoninline.add(new MyPGlass(pcglass,choice));

stateChanged();

}

void checkingforglass(MyPGlass mg)

{

if(mg.NeedsProcessing==true)

{

mg.status=GlassStatusInline.PROCESSING;

}

else

{

mg.status=GlassStatusInline.NOPROCESSING;

print("NO PRocessing");

}

stateChanged();

}

package engine.conveyorfamily\_Poojan;

public class PopupAgent\_PJ extends Agent implements Popup\_PJ {

private int number;

private String name;

private Transducer myTransducer;

private Conveyor\_PJ myconveyor;

public ConveyorFamily MyFamily;

private ConveyorFamily NEXTFamily;

private Semaphore popupconveyor = new Semaphore(1,true);

private Semaphore popupoperator = new Semaphore(1,true);

private List<ROperator> respondtooperators = Collections.synchronizedList(new ArrayList<ROperator>());

private List<MyPGlass> glassonpopup = Collections.synchronizedList(new ArrayList<MyPGlass>());

private List<MyPGlass> finishedglassonpopup = Collections.synchronizedList(new ArrayList<MyPGlass>());

private boolean finisheddone;

public enum GlassStatusPopup{NOPROCESSING,PROCESSING,BEINGPROCESSED, DONE,DONE2};

public enum opstatus{RESPOND,DONE};

public PopupAgent\_PJ(String string, int i, ConveyorFamily conveyorFamily,

Transducer transducer) {

// TODO Auto-generated constructor stub

this.name=string;

this.number=i;

this.MyFamily=conveyorFamily;

myTransducer = transducer;

myTransducer.register(this, TChannel.ALL\_GUI);

myTransducer.register(this, TChannel.CONVEYOR);

myTransducer.register(this, TChannel.ALL\_AGENTS);

}

public class ROperator

{

private Operator\_PJ myoperator;

private opstatus status;

public ROperator(Operator\_PJ o)

{

this.myoperator=o;

this.status=opstatus.RESPOND;

}

}

public class MyPGlass

{

private Glass pcglass;

private GlassStatusPopup status;

private Operator\_PJ myoperator;

public MyPGlass(Glass g)

{

this.pcglass=g;

this.status=GlassStatusPopup.NOPROCESSING;

}

public MyPGlass(Glass g,Operator\_PJ o)

{

this.pcglass=g;

this.myoperator=o;

this.status=GlassStatusPopup.PROCESSING;

}

}

public boolean pickAndExecuteAnAction() {

synchronized(glassonpopup){

for(MyPGlass mg:glassonpopup){

if(mg.status == GlassStatusPopup.NOPROCESSING ){

glassprocessingfinished(mg);

return true;

}

}

};

synchronized(glassonpopup){

for(MyPGlass mg:glassonpopup){

if(mg.status == GlassStatusPopup.PROCESSING ){

telltheoperator(mg);

return true;

}

}

};

synchronized(glassonpopup){

for(MyPGlass mg:glassonpopup){

if(mg.status == GlassStatusPopup.DONE ){

glassonpopup.remove(mg);

return true;

}

}

};

synchronized(finishedglassonpopup){

for(MyPGlass mg:finishedglassonpopup){

if(mg.status == GlassStatusPopup.PROCESSING ){

shiptheglass(mg);

return true;

}

}

};

synchronized(finishedglassonpopup){

for(MyPGlass mg:finishedglassonpopup){

if(mg.status == GlassStatusPopup.DONE ){

glassonpopup.remove(mg);

return true;

}

}

};

// TODO Auto-generated method stub

return false;

}

void telltheoperator(MyPGlass mg) {

// TODO Auto-generated method stub

mg.status=GlassStatusPopup.DONE2;

mg.myoperator.msgHereIsGlass(mg.pcglass);

stateChanged();

}

void eventFired(TChannel channel, TEvent event, Object[] args) {

// TODO Auto-generated method stub

if(channel == TChannel.POPUP)

{

if(event == TEvent.POPUP\_GUI\_MOVED\_DOWN)

{

if((Integer)args[0]==this.number)

{

if(finisheddone==true)

{

finisheddone=false;

print("finsished");

}

else

{

this.myconveyor.msgIamFree();

try {

popupconveyor.acquire();

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

};

}

if(event == TEvent.POPUP\_GUI\_MOVED\_UP)

{

if((Integer)args[0]==this.number)

{

for(ROperator mg:respondtooperators){

if(mg.status == opstatus.RESPOND ){

mg.myoperator.msgIAmFree();

}

}

try {

popupoperator.acquire();

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

};

}

/\*

if(event == TEvent.POPUP\_GUI\_LOAD\_FINISHED)

{

if((Integer)args[0]==this.number)

{

this.myconveyor.msgIamFree();

try {

popupconveyor.acquire();

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

};

}\*/

if(event == TEvent.POPUP\_GUI\_RELEASE\_FINISHED)

{

if((Integer)args[0]==this.number)

{

for(MyPGlass mg:finishedglassonpopup){

if(mg.status == GlassStatusPopup.BEINGPROCESSED ){

this.myconveyor.msgOperatorIsfree(mg.myoperator);

mg.status=GlassStatusPopup.DONE;

stateChanged();

}

}

}

}

}

}

public String getName(){

return name;

}

public int getNumber(){

return number;

}

public int getrespondtooperatorssize(){

return respondtooperators.size();

}

public int getglassonpopupsize(){

return glassonpopup.size();

}

public int getfinishedglassonpopupsize(){

return finishedglassonpopup.size();

}

void setConveyor(Conveyor\_PJ conveyor) {

// TODO Auto-generated method stub

this.myconveyor=conveyor;

}

// MESSAGES

void msgINeedToPassGlass() {

// TODO Auto-generated method stub

print("Asking for permission");

Object[] args1 = {this.number};

myTransducer.fireEvent(TChannel.POPUP,TEvent.POPUP\_DO\_MOVE\_DOWN,args1);

}

void msgGlassDoesNotNeedProcessing(Glass glass) {

popupconveyor.release();

print("Glass received. Glass does not need processing");

// TODO Auto-generated method stub

glassonpopup.add(new MyPGlass(glass));

stateChanged();

}

void msgGlassNeedsProcessing(Glass pcglass, Operator\_PJ op) {

// TODO Auto-generated method stub

print("Glass received. Glass needs processing");

glassonpopup.add(new MyPGlass(pcglass,op));

stateChanged();

}

void msgHereIsFinishedGlass(Glass g, Operator\_PJ operatorAgent) {

// TODO Auto-generated method stub

print("here is the finished glass");

popupoperator.release();

Object[] args1 = {this.number};

finisheddone=true;

myTransducer.fireEvent(TChannel.POPUP,TEvent.POPUP\_DO\_MOVE\_DOWN,args1);

finishedglassonpopup.add(new MyPGlass(g,operatorAgent));

stateChanged();

}

void msgOperatorHasFinishedGlass(Operator\_PJ o) {

// TODO Auto-generated method stub

print("Operator has finished");

Object[] args1 = {this.number};

myTransducer.fireEvent(TChannel.POPUP,TEvent.POPUP\_DO\_MOVE\_UP,args1);

respondtooperators.add(new ROperator(o));

stateChanged();

}

// ACTIONS

void glassprocessingfinished(MyPGlass mg) {

// TODO Auto-generated method stub

// if(!(this.MyFamily.getStatusOfNextConveyorFamily()))

{

print("RELEASE THE GLASS. PROCESSING DONE");

Object[] args1 = {this.number};

myTransducer.fireEvent(TChannel.POPUP,TEvent.POPUP\_RELEASE\_GLASS,args1);

// this.MyFamily.getNextConveyorFamily().msgHereIsGlass(mg.pcglass);

mg.status=GlassStatusPopup.DONE;

stateChanged();

}

}

void shiptheglass(MyPGlass mg) {

// TODO Auto-generated method stub

Object[] args1 = {this.number};

// if(!this.MyFamily.getStatusOfNextConveyorFamily())

{

myTransducer.fireEvent(TChannel.POPUP,TEvent.POPUP\_RELEASE\_GLASS,args1);

// this.MyFamily.getNextConveyorFamily().msgHereIsGlass(mg.pcglass);

mg.status=GlassStatusPopup.BEINGPROCESSED;

stateChanged();

}

}

Luis

public class ConveyorAgent\_LV extends Agent implements Conveyor\_LV{

int index;

boolean moving;

List<Glass> glassPieces;

enum PopUpState {OPEN, BUSY};

enum SensorState {PRESSED, RELEASED, NULL};

MyPopUp myPopUp;

SensorState sensorOne = SensorState.NULL;

SensorState sensorTwo = SensorState.NULL;

ConveyorFamily previousFamily;

Transducer t;

public class MyPopUp

{

PopUp\_LV popUp;

PopUpState state;

public MyPopUp(PopUp\_LV p)

{

popUp = p;

state = PopUpState.OPEN;

}

}

public ConveyorAgent\_LV(String s, int i)

{

name = s;

index = i;

glassPieces = Collections.synchronizedList(new ArrayList<Glass>());

moving = false;

sensorOne = SensorState.NULL;

sensorOne = SensorState.NULL;

}

/\*

\* MESSAGES

\*/

void msgHereIsGlass(Glass glass)

{

glassPieces.add(glass);

stateChanged();

}

void msgPopUpBusy()

{

myPopUp.state = PopUpState.BUSY;

stateChanged();

}

void msgPopUpFree()

{

myPopUp.state = PopUpState.OPEN;

stateChanged();

}

/\*

\* SCHEDULER

\*/

public boolean pickAndExecuteAnAction() {

if((myPopUp.state == PopUpState.BUSY) && (sensorTwo == SensorState.PRESSED) && (moving))

{

letPopUpKnowGlassIsWaiting();

return true;

}

if((myPopUp.state == PopUpState.OPEN) && (!moving))

{

startUpConveyor();

return true;

}

if(sensorTwo == SensorState.RELEASED && !(myPopUp.state == PopUpState.BUSY))

{

moveToPopUp();

return true;

}

if(sensorOne == SensorState.RELEASED)

{

notifyPreviousFamily();

return true;

}

return false;

}

/\*

\* ACTIONS

\*/

void startUpConveyor()

{

print("starting conveyor");

Integer[] args = new Integer[1];

args[0] = index;

t.fireEvent(TChannel.CONVEYOR, TEvent.CONVEYOR\_DO\_START, args);

moving = true;

}

void stopConveyor()

{

print("stopping conveyor");

Integer[] args = new Integer[1];

args[0] = index;

t.fireEvent(TChannel.CONVEYOR, TEvent.CONVEYOR\_DO\_STOP, args);

moving = false;

}

void letPopUpKnowGlassIsWaiting()

{

stopConveyor();

print("Letting popup know glass is waiting");

myPopUp.popUp.msgIHaveGlassReady(glassPieces.get(0));

}

void moveToPopUp()

{

print("Giving glass to popUp");

myPopUp.popUp.msgHereIsGlass(glassPieces.remove(0));

sensorTwo = SensorState.NULL;

myPopUp.state = PopUpState.BUSY;

}

void notifyPreviousFamily()

{

print("Letting previous Family know I am free");

previousFamily.msgIAmFree();

sensorOne = SensorState.NULL;

}

void eventFired(TChannel channel, TEvent event, Object[] args) {

if(channel == TChannel.SENSOR)

{

if(event == TEvent.SENSOR\_GUI\_PRESSED)

{

if((Integer)(args[0]) == index\*2)

sensorOne = SensorState.PRESSED;

else if((Integer)(args[0]) == index\*2 + 1)

sensorTwo = SensorState.PRESSED;

}

else if(event == TEvent.SENSOR\_GUI\_RELEASED)

{

if((Integer)(args[0]) == index\*2)

sensorOne = SensorState.RELEASED;

else if((Integer)(args[0]) == index\*2+1)

sensorTwo = SensorState.RELEASED;

}

}

stateChanged();

}

void msgHereIsFinishedGlass(Operator operator, Glass glass) {

// TODO Auto-generated method stub

}

void msgIHaveGlassFinished(Operator operator) {

// TODO Auto-generated method stub

}

void setInteractions(ConveyorFamily cf, PopUp\_LV popUp, Transducer trans)

{

previousFamily = cf;

myPopUp = new MyPopUp(popUp);

t = trans;

t.register(this, TChannel.SENSOR);

}

public PopUp\_LV getPopUp()

{

return myPopUp.popUp;

}

void setPopUp(PopUp\_LV p)

{

myPopUp.popUp = p;

}

void setTransducer(Transducer trans)

{

t = trans;

t.register(this, TChannel.SENSOR);

}

public ConveyorFamily getPrevious()

{

return previousFamily;

}

public int getGlassPiecesSize()

{

return glassPieces.size();

}

void setInteractions(ConveyorFamily c2, PopUpAgent\_LV popup) {

// TODO Auto-generated method stub

previousFamily = c2;

setPopUp(popup);

}

}

public class ConveyorFamilyAgent\_LV implements ConveyorFamily{

int index;

ConveyorAgent\_LV conveyor;

PopUpAgent\_LV popup;

Transducer t;

ConveyorFamily previousFamily, nextFamily;

public ConveyorFamilyAgent\_LV(int i, Transducer trans, int popupIndex)

{

index = i;

conveyor = new ConveyorAgent\_LV("Conveyor " + index, index);

popup = new PopUpAgent\_LV("PopUp " + index, popupIndex);

conveyor.setInteractions(previousFamily, popup, trans);

popup.setInteractions(nextFamily, conveyor, trans);

}

void msgHereIsGlass(Glass glass) {

conveyor.msgHereIsGlass(glass);

}

void msgHereIsFinishedGlass(Operator operator, Glass glass) {

popup.msgHereIsFinishedGlass(glass);

}

void msgIHaveGlassFinished(Operator operator) {

popup.msgIHaveGlassFinished(operator);

}

void msgIAmFree() {

popup.msgIAmFree();

}

public String getName() {

return null;

}

void setNextConveyorFamily(ConveyorFamily c3) {

// TODO Auto-generated method stub

popup.setInteractions(c3);

}

void setPreviousConveyorFamily(ConveyorFamily c2) {

// TODO Auto-generated method stub

conveyor.setInteractions(c2, popup);

}

void startThreads() {

conveyor.startThread();

popup.startThread();

}

void setChannel(TChannel tc){

popup.channel = tc;

}

void setOperators(Operator o5up, Operator o5down, TChannel c) {

// TODO Auto-generated method stub

popup.setOperators(o5up, o5down, c);

}

}

public class PopUpAgent\_LV extends Agent implements PopUp\_LV{

int index;

TChannel work;

boolean nextComponentFree;

Status status;

PopUpState state;

List<Machine> operators;

Conveyor\_LV conveyor;

GlassPackage currentGlass;

List<GlassPackage> myGlassPieces = Collections.synchronizedList(new ArrayList<GlassPackage>());

enum GlassState {INCOMING, WAITING, NEEDS\_WORK, FINISHED, MOVE, NONE};

enum PopUpState {FULL, OPEN};

enum Status{RAISED,LOWERED};

Semaphore stateSemaphore = new Semaphore(0,true);

Semaphore statusSemaphore = new Semaphore(0,true);

//Semaphore loadSemaphore = new Semaphore(0,true);

ConveyorFamily next;

Transducer t;

TChannel channel;

public class Machine

{

Operator operator;

int number;

TChannel channel;

boolean occupied;

Semaphore semaphore = new Semaphore(0,true);

public Machine(Operator o, TChannel t, boolean b, int n)

{

operator = o;

number = n;

channel = t;

occupied = b;

}

}

public class GlassPackage

{

Operator operator;

Glass glass;

GlassState state;

int operatorNumber;

public GlassPackage(Glass g, GlassState s)

{

glass = g;

state = s;

}

}

public PopUpAgent\_LV(String n, int i)

{

name = n;

index = i;

nextComponentFree = true;

status = Status.LOWERED;

state = PopUpState.OPEN;

operators = Collections.synchronizedList(new ArrayList<Machine>());

}

/\*

\* MESSAGES

\*/

void msgIAmFree()

{

nextComponentFree = true;

stateChanged();

}

void msgIHaveGlassReady(Glass glass)

{

synchronized(myGlassPieces)

{

for(GlassPackage g : myGlassPieces)

{

if(g.glass == glass)

{

g.state = GlassState.INCOMING;

stateChanged();

return;

}

}

myGlassPieces.add(new GlassPackage(glass, GlassState.INCOMING));

}

stateChanged();

}

void msgHereIsGlass(Glass glass)

{

synchronized(myGlassPieces)

{

for(GlassPackage g : myGlassPieces)

{

if(g.glass == glass)

{

g.state = GlassState.WAITING;

currentGlass = g;

stateChanged();

return;

}

}

GlassPackage g = new GlassPackage(glass, GlassState.INCOMING);

myGlassPieces.add(g);

currentGlass = g;

}

stateChanged();

}

void msgIHaveGlassFinished(Operator operator)

{

synchronized(myGlassPieces)

{

for(GlassPackage g : myGlassPieces)

{

if(g.operator == operator)

{

g.state = GlassState.FINISHED;

}

}

}

stateChanged();

}

void msgHereIsFinishedGlass(Glass glass)

{

synchronized(myGlassPieces)

{

for(GlassPackage g : myGlassPieces)

{

if(g.glass == glass)

{

g.state = GlassState.MOVE;

currentGlass = g;

}

}

}

stateChanged();

}

void msgCannotPass()

{

//popup busy

}

/\*

\* SCHEDULER

\*/

public boolean pickAndExecuteAnAction() {

GlassPackage temp = null;

if(state == PopUpState.OPEN)

{

synchronized(myGlassPieces)

{

for(GlassPackage g : myGlassPieces)

{

if(g.state == GlassState.INCOMING)

temp = g;

}

}

if(temp != null)

{

if((!temp.glass.getRecipe(channel)) || (getOperatorStatus(0) == false || getOperatorStatus(1) == false))

{

takeGlass(temp);

return true;

}

}

}

if(state == PopUpState.OPEN)

{

synchronized(myGlassPieces)

{

for(GlassPackage g : myGlassPieces)

{

if(g.state == GlassState.FINISHED)

temp = g;

}

}

if(temp != null)

{

takeFinishedGlass(temp, temp.operatorNumber);

return true;

}

}

synchronized(myGlassPieces)

{

for(GlassPackage g : myGlassPieces)

{

if(g.state == GlassState.WAITING)

{

temp = g;

}

}

}

if(temp != null)

{

checkGlass(temp);

return true;

}

if(nextComponentFree)

{

synchronized(myGlassPieces)

{

for(GlassPackage g : myGlassPieces)

{

if(g.state == GlassState.MOVE)

{

temp = g;

}

}

}

if(temp != null)

{

moveGlass(temp);

return true;

}

}

if(getOperatorStatus(0) == false || getOperatorStatus(1) == false)

{

synchronized(myGlassPieces)

{

for(GlassPackage g : myGlassPieces)

{

if(g.state == GlassState.NEEDS\_WORK)

{

temp = g;

}

}

}

if(temp != null)

{

if(getOperatorStatus(0) == false)

giveGlassToOperator(temp, 0);

else

giveGlassToOperator(temp, 1);

return true;

}

}

//tweak

if(myGlassPieces.size()==0){

while(stateSemaphore.tryAcquire());

return true;

}

return false;

}

/\*

\* ACTIONS

\*/

void takeGlass(GlassPackage g)

{

print("Taking glass from conveyor");

if(status == Status.RAISED)

lowerPopUp();

conveyor.msgPopUpFree();

//print(""+stateSemaphore.availablePermits());

try{

stateSemaphore.acquire();

} catch(InterruptedException e){

e.printStackTrace();

}

state = PopUpState.FULL;

g.state = GlassState.WAITING;

conveyor.msgPopUpBusy();

}

void giveGlassToOperator(GlassPackage g, int operatorNumber)

{

print("Giving operator glass");

if(status == Status.LOWERED);

raisePopUp();

operators.get(operatorNumber).operator.msgHereIsGlass(g.glass);

g.operator = operators.get(operatorNumber).operator;

try{

operators.get(operatorNumber).semaphore.acquire();

} catch(InterruptedException e){

e.printStackTrace();

}

Do("REACHED HERE !!!");

g.operatorNumber = operatorNumber;

operators.get(operatorNumber).occupied = true;

g.state = GlassState.NONE;

state = PopUpState.OPEN;

currentGlass = null;

}

void moveGlass(GlassPackage g)

{

print("Moving glass to next conveyor");

if(status == Status.RAISED)

lowerPopUp();

next.msgHereIsGlass(g.glass);

myGlassPieces.remove(g);

Integer[] args = new Integer[1];

args[0] = index;

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

try{

stateSemaphore.acquire();

} catch(InterruptedException e){

e.printStackTrace();

}

state = PopUpState.OPEN;

conveyor.msgPopUpFree();

currentGlass = null;

nextComponentFree= false;

}

void checkGlass(GlassPackage g)

{

print("Checking if glass needs work");

if(g.glass.getRecipe(channel))

g.state = GlassState.NEEDS\_WORK;

else

g.state = GlassState.MOVE;

}

void lowerPopUp()

{

print("lowering popup");

Integer[] args = new Integer[1];

args[0] = index; //Note: popup offset

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

try{

statusSemaphore.acquire();

} catch(InterruptedException e){

e.printStackTrace();

}

if(state == PopUpState.OPEN)

conveyor.msgPopUpFree();

status = Status.LOWERED;

}

void raisePopUp()

{

print("raising popup");

Integer[] args = new Integer[1];

args[0] = index;

t.fireEvent(TChannel.POPUP, TEvent.POPUP\_DO\_MOVE\_UP, args);

try{

statusSemaphore.acquire();

} catch(InterruptedException e){

e.printStackTrace();

}

//Do("REACHED HERE");

conveyor.msgPopUpBusy();

status = Status.RAISED;

}

void takeFinishedGlass(GlassPackage g, int operatorNum)

{

print("Taking glass from operator");

if(status == Status.LOWERED)

raisePopUp();

operators.get(operatorNum).operator.msgIAmFree();

try{

operators.get(operatorNum).semaphore.acquire();

} catch(InterruptedException e){

e.printStackTrace();

}

operators.get(operatorNum).occupied = false;

g.state = GlassState.MOVE;

state = PopUpState.FULL;

}

void eventFired(TChannel channel, TEvent event, Object[] args) {

if((channel == TChannel.POPUP) && ((Integer)(args[0]) == index)) //Note: popup offset

{

if(event == TEvent.POPUP\_GUI\_MOVED\_DOWN)

statusSemaphore.release();

if(event == TEvent.POPUP\_GUI\_MOVED\_UP)

statusSemaphore.release();

if(event == TEvent.POPUP\_GUI\_LOAD\_FINISHED)

stateSemaphore.release();

if(event == TEvent.POPUP\_GUI\_RELEASE\_FINISHED)

stateSemaphore.release();

}

else if((channel == operators.get(0).channel) && ((Integer)(args[0]) == 0) )

{

Do("Event: "+event+" Channel: "+channel+" Parameter passed in: "+(Integer)(args[0])+

" Operator No: "+operators.get(0).number);

if(event == TEvent.WORKSTATION\_LOAD\_FINISHED)

operators.get(0).semaphore.release();

if(event == TEvent.WORKSTATION\_RELEASE\_FINISHED)

operators.get(0).semaphore.release();

}

else if((channel == operators.get(1).channel) && ((Integer)(args[0]) == 1) )

{

Do("Event: "+event+" Channel: "+channel+" Parameter passed in: "+(Integer)(args[0])+

" Operator No: "+operators.get(0).number);

if(event == TEvent.WORKSTATION\_LOAD\_FINISHED)

operators.get(1).semaphore.release();

if(event == TEvent.WORKSTATION\_RELEASE\_FINISHED)

operators.get(1).semaphore.release();

}

}

public String getName()

{

return name;

}

void setOperators(Operator operatorOne, Operator operatorTwo, TChannel c)

{

this.operators.add(new Machine(operatorOne,c,false,0));

this.operators.add(new Machine(operatorTwo,c,false,1));

t.register(this, c);

}

void setTransducer(Transducer trans)

{

t = trans;

t.register(this, TChannel.POPUP);

}

void setInteractions(ConveyorFamily cf, Conveyor\_LV c, Transducer trans)

{

conveyor = c;

next = cf;

t = trans;

t.register(this, TChannel.POPUP);

}

void setConveyor(Conveyor\_LV c)

{

conveyor = c;

}

void setNextComponentFree(boolean b)

{

nextComponentFree = b;

}

public boolean getNextComponentFree()

{

return nextComponentFree;

}

public Conveyor\_LV getConveyor()

{

return conveyor;

}

public ConveyorFamily getNextConveyorFamily()

{

return next;

}

public int getOperatorsSize()

{

return operators.size();

}

public int getGlassPiecesSize()

{

return myGlassPieces.size();

}

public Operator getOperator(int i)

{

return operators.get(i).operator;

}

public boolean getOperatorStatus(int i)

{

return operators.get(i).occupied;

}

void setOperatorStatus(int i,boolean b)

{

operators.get(i).occupied = b;

}

void setInteractions(ConveyorFamily c3) {

// TODO Auto-generated method stub

next = c3;

}

}

public class Truck\_Agent\_LV extends Agent implements ConveyorFamily{

String name;

ConveyorFamily previousFamily;

Transducer t;

List<Glass> truckGlass;

enum TruckState{PARKED,COMMUTING,ARRIVED,LOADING};

Semaphore drivingSemaphore = new Semaphore(0,true);

TruckState state;

public Truck\_Agent\_LV(String n)

{

name = n;

truckGlass = new ArrayList<Glass>();

state = TruckState.ARRIVED;

}

//Messages

void msgHereIsGlass(Glass glass)

{

truckGlass.add(glass);

stateChanged();

}

//Scheduler

public boolean pickAndExecuteAnAction() {

if(state == TruckState.ARRIVED)

{

tellConveyorIAmFree();

return true;

}

if(state == TruckState.LOADING)

{

for(Glass g : truckGlass)

{

moveGlass(g);

return true;

}

}

return false;

}

void eventFired(TChannel channel, TEvent event, Object[] args) {

if(channel == TChannel.TRUCK)

{

if(event == TEvent.TRUCK\_GUI\_LOAD\_FINISHED)

{

state = TruckState.LOADING;

}

else if(event == TEvent.TRUCK\_GUI\_EMPTY\_FINISHED)

{

state = TruckState.ARRIVED;

}

}

stateChanged();

}

//Actions

void tellConveyorIAmFree()

{

print("Letting conveyor know I can load glass");

if(truckGlass.size()!=0)

truckGlass.remove(0);

previousFamily.msgIAmFree();

state = TruckState.PARKED;

}

void moveGlass(Glass g)

{

print("Delivering Glass");

Integer[] args = new Integer[1];

args[0] = 0;

t.fireEvent(TChannel.TRUCK, TEvent.TRUCK\_DO\_EMPTY,args);

state = TruckState.COMMUTING;

}

void msgHereIsFinishedGlass(Operator operator, Glass glass) {

// Nothing

}

void msgIHaveGlassFinished(Operator operator) {

// Nothing

}

void msgIAmFree() {

// Nothing

}

void setNextConveyorFamily(ConveyorFamily c3) {

// Nothing

}

void setPreviousConveyorFamily(ConveyorFamily cf) {

previousFamily = cf;

}

void setTransducer(Transducer trans){

t = trans;

t.register(this, TChannel.TRUCK);

}

void startThreads() {

// Nothing

}

public String getName(){

return name;

}

}

End of Design Docs