@Override

**public** String toString() {

**return** "Robot Agent for " + **this**.robot.toString();

}

//================================================================================

// HLC Information Request

//================================================================================

@Override

**public** DirectedSparseGraph<CapabilitiesNode, CapabilitiesEdge> getCapabilities() {

**return** **this**.robotOutputGraph;

}

@Override

**public** **boolean** scheduleGraph() {

**return** **true**;

}

//================================================================================

// HLC Information Output

//================================================================================

@Override

**public** **void** notify(Object object, CapabilitiesEdge edge) {

**if**(object==**null**){

System.out.println("No object to notify for " + **this**);

**return**;

}

//Notify a listening partController that the part is there

**if** (object.getClass().toString().contains("Part")){

**for** (PartController partController : **this**.listeningParts){

**if** (partController.getTagName() == ((Part) object).getRFIDTag().getName()){

partController.observedEdgeNotification(edge);

}

}

}

}

//================================================================================

// HLC Command Request

//================================================================================

**public** **void** addListeningPart(PartController partController){

**if** (!listeningParts.contains(partController)){

**this**.listeningParts.add(partController);

}

}

**public** **void** removeListeningPart(PartController partController){

**this**.listeningParts.remove(partController);

}

@ScheduledMethod (start = 1 , interval = 1, priority = 5000)

**public** **void** setTransient(){

**if** (working){

**this**.allowAfterTransient = **false**;

**this**.allowAfterTransientTimer = 4;

}

**if** (**this**.allowAfterTransient = **false**){

**this**.allowAfterTransientTimer--;

**return**;

}

**this**.allowAfterTransientTimer = 0;

**this**.allowAfterTransient = **true**;

}

/\*\* NEED TO EDIT THIS LATE WITH PROPER COMMUNICATION

\* **@param** program

\*/

**public** **void** runProgram(String program){

setTransient();

**if** (!**this**.allowAfterTransient){

**return**;

}

CapabilitiesEdge tryEdge = **null**;

//Check the edge you want to try

**for** (CapabilitiesEdge edge: robotOutputGraph.getEdges()){

**if** ((String) edge.getActiveParams()[0] == program){

tryEdge = edge;

//Make sure there is no other part on this edge

**if**(!(tryEdge.getChild().getObject() **instanceof** CollectionEquipment)){

//Check every point on where you want to place it

**for**(Object object: grid.getObjectsAt(tryEdge.getChild().getLocation().x,tryEdge.getChild().getLocation().y)){

//If there is a part, return

**if** (object.getClass().getSimpleName().contains("Part")){

**this**.stopperStation.noMove(**false**);

**return**;

}

}

}

//Make sure there is an empty pallet. If not, return

**if**(tryEdge.getChild().getObject().getClass().getSimpleName().contains("Conveyor")){

ConveyorSystemStationStatus stopperStatus = (ConveyorSystemStationStatus) **this**.stopperStation.query();

**if**(!**this**.stopperStation.getPresenceStatus() || (**this**.stopperStation.getRFIDStatus() != **null**)){

**this**.stopperStation.noMove(**false**);

**return**;

}

}

}

}

//If everything is fine, try setting the program on the robot controller

**if** (!working && setProgram(program)){

//If the program can run, run via statechart

**this**.stopperStation.noMove(**true**);

**this**.robot.setWorking(**true**);

robotStatechart.receiveMessage("Running");

**this**.runningEdge = tryEdge;

}

}

//================================================================================

// Helper methods

//================================================================================

/\*\*

\* Takes in the system graph and modifies the system output graph to hide the subagents

\* For each edge, the object that needs to be called will be this agent instead of the subagents

\*//\*

private void createOutputGraph() {

for (CapabilitiesNode v : robotGraph.getVertices()){

robotOutputGraph.addVertex(v);

}

//Map the index to a program in the conveyor system controller

RobotEdge newEdge = null;

//Make the robot graph into the robot system graph

for (CapabilitiesEdge edge : robotGraph.getEdges()){

try {

newEdge = new RobotEdge(this, edge.getParent(), edge.getChild(),

this.getClass().getMethod("runProgram",new Class[]{String.class}),(String) edge.getActiveParams()[0]);

newEdge.setControllability(edge.getControllability());

newEdge.setObservability(edge.getObservability());

} catch (NoSuchMethodException e) {

System.err.println("[ConveyorSystemController.java] Robot system doesn't have this method");

} catch (SecurityException e) {

System.err.println("[ConveyorSystemController.java] Robot system has security problems...?");

}

robotOutputGraph.addEdge(newEdge, newEdge.getParent(), newEdge.getChild());

}

}

//================================================================================

// Statechart helper methods

//================================================================================

//The following are used by the robot to start/stop the robot

\*//\*\* For the run program command

\* **@param** programInput

\* **@return**

\*//\*

private boolean setProgram(String programInput){

//Set the program if there

if (robot.runMoveObjectProgram(programInput)){

this.program = programInput;

return true;

}

return false;

}

\*//\*\*Change the state of the cnc system controller

\*

\*//\*

@ScheduledMethod (start = 1 , interval = 1, priority = 6000)

public void setIdle(){

if(this.robot.getFree()){

robotStatechart.receiveMessage("Idle");

}

}

\*//\*\* (Call from statechart)

\* **@param** working

\*//\*

public void setWorking(boolean working){

this.working = working;

}

\*//\*\* (Call from statechart)

\* **@return**

\*//\*

public String getProgram(){

return this.program;

}

public CapabilitiesEdge getRunningEdge(){

return this.runningEdge;

}

public void setRunningHoldingObject(Object obj){

if (obj != null)

this.holdingObject = obj;

}

public void setIdleHoldingObject(Object obj){

this.holdingObject = obj;

}

public Object getHoldingObject(){

return this.holdingObject;

}

public Object getRobotHoldingObject(){

return this.robot.getHoldingObject();

}