@Override

public String toString() {

return "Machine Agent for " + this.cnc;

}

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

// HLC Information Output

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

@Override

public void notify(Object object, SystemEdge edge) {

if(object==null){

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

return;

}

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);

}

public void runProgram(Integer program){

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

cncStatechart.receiveMessage("Running");

for (SystemEdge edge: cncOutputGraph.getEdges()){

if ((edge.getActiveParams() != null) && (Integer) edge.getActiveParams()[0] == program){

this.runningEdge = edge;

}

}

}

}

public void doNothing(){

for (SystemEdge edge: cncOutputGraph.getEdges()){

if (edge.getActiveParams() == null && runningEdge.getChild().equals(edge.getParent())){

this.runningEdge = edge;

break;

}

}

}

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

// HLC Request

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

@Override

public DirectedSparseGraph<SystemVertex, SystemEdge> getGraph(int time) {

if (!working){

return cncOutputGraph;

}

int timeLeft = this.cnc.getTimeLeft();

DirectedSparseGraph<SystemVertex, SystemEdge> newGraph = new DirectedSparseGraph<SystemVertex, SystemEdge>();

for (SystemEdge edge:cncOutputGraph.getEdges()){

SystemEdge newEdge = edge.copy();

newEdge.setWeight(edge.getWeight()+timeLeft);

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

}

return newGraph;

}

@Override

public boolean scheduleGraph(Object object) {

return false;

}

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

// 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 (SystemVertex v : cncGraph.getVertices()){

cncOutputGraph.addVertex(v);

}

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

CncEdge newEdge = null;

for (SystemEdge edge : cncGraph.getEdges()){

try {

if (edge.getActiveParams() == null){

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

this.getClass().getMethod("doNothing"),

null,edge.getWeight());

}

else{

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

this.getClass().getMethod("runProgram",new Class[]{Integer.class}),

(Integer) edge.getActiveParams()[0],edge.getWeight());

}

newEdge.setControllability(edge.getControllability());

newEdge.setObservability(edge.getObservability());

} catch (NoSuchMethodException e) {

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

} catch (SecurityException e) {

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

}

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

}

}

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

// Statechart helper methods

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

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

\* **@param** programInput

\* **@return**

\*//\*

private boolean setProgram(int programInput){

//Set the program if there

if (cnc.runProgram(programInput)){

this.program = programInput;

return true;

}

return false;

}

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

\*

\*//\*

@ScheduledMethod (start = 1 , interval = 1, priority = 200)

public void setIdle(){

if(!this.cnc.queryWorking()){

cncStatechart.receiveMessage("Idle");

cnc.setNotDone();

}

}

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

\* **@param** working

\*//\*

public void setWorking(boolean working){

this.working = working;

}

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

\* **@return**

\*//\*

public int getProgram(){

return this.program;

}

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

\* **@param** program

\*//\*

public void runCnc(int program){

if (program != -1){

this.cnc.runProgram(program);

}

}

public Object getHoldingObject(){

return this.cnc.getHoldingObject();

}

public SystemEdge getRunningEdge(){

return this.runningEdge;

}

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

// Statechart auto methods

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

@ProbedProperty(displayName="CncStatechart")

CncStatechart cncStatechart = CncStatechart.createStateChart(this, 0);

public String getCncStatechartState(){

if (cncStatechart == null) return "";

Object result = cncStatechart.getCurrentSimpleState();

return result == null ? "" : result.toString();

}