Part 1

Modeling a Road Traffic Network of a City using Multi-Agent Systems

The sample problem which I attempted to solve in my assignment is on how to model a road traffic network in a city using the concepts of multi-agent systems.

The approach here is to think about places on the road in the city border and city junctions (3-way junctions, 4-way junctions etc.) as agents operating "independently" from each other.

The places near the city border are simulated by **RoadAgent**s. Junctions within the city are simulated as **JunctionAgent**s (**ThreeWayJunctionAgent**, **FourWayJunctionAgent**).

A **RoadAgent** will always report the incoming vehicle rate into the city from the city limits to its nearest junction once every given period.

On the other hand a **JunctionAgent** will do the following 3 activities periodically.

1. Reporting the outgoing vehicle rate from its each road to its associated JunctionAgents.
2. Receive the latest incoming vehicle rates for each of its roads from the associated JunctionAgents/RoadAgents.
3. Alternate its traffic schedule based on the updated incoming vehicle rates from each of its roads.

Every RoadAgent/JunctionAgent in the system will operate independently from each other while doing the above activities periodically.

An interesting question is how the system would handle an accident? During an accident the particular road subjected to the accident will be blocked. But as long as the damaged vehicles are removed all the other vehicles those were blocked due to the accident on the road will start to move. Thus the associated junctions will experience higher vehicle incoming/outgoing rates from each of their roads. So the junction agents will adjust their traffic schedule accordingly. This is the power of a Multi-agent System. Any of the agents doesn't have a complete knowledge on the entire system. However their combination as a whole is able to solve any problem related with any complex system.

A Note on Implementation

I have used JADE (Java Agent Development Framework) [2][7] as the toolkit for implementing the traffic simulator. The implementation contains the following classes.

RoadAgent.java – Represents a *RoadAgent*

NWayJunctionAgent.java – An abstract class, which represents a *JunctionAgent*, with any number of roads connecting in the junction.

ThreeWayJunctionAgent.java – Representing the behavior of a *JunctionAgent* in a 3-way junction. This class is written as a subclass of NWayJunctionAgent.java.

FourWayJunctionAgent.java – Representing the behavior of a *JunctionAgent* in a 4-way junction. This class is also written as a subclass of NWayJunctionAgent.java.

Please refer to Appendix A for the source code of these classes.

Running the Implementation

I have compiled the JADE libraries and tested the implementation under java version ***"1.6.0_17"***

Copy the contents inside **JADE_Library** folder in the CD to the hard disk.

From a command prompt set up the relevant classpaths as follows. (In this case I have copied the jade folder found inside JADE_Library folder to the C:\ in my hard disk)

SET JADE_HOME=c:\jade\

**SET CLASSPATH=
%CLASSPATH%;.;%JADE_HOME%\lib\jade.jar;%JADE_HOME%\lib\jadeTools.jar;%JADE_HOME%\lib\http.jar;%JADE_HOME%\lib\iiop.jar;%JADE_HOME%\lib\commons-codec\commons-codec-1.3.jar;%JADE_HOME%\classes**

Copy the contents inside **Assignment_Source** folder in the CD to the hard disk.

From the ***same command prompt*** you used earlier edit the classpath again as follows. (In this case I've copied the assignment folder found within Assignment_Source folder into the Src folder found in C:\ of my hard disk)

SET CLASSPATH=%CLASSPATH%;C:\Src

Go to the C:\Src folder from the ***same command prompt***.

Compile the program using javac

```
javac assignment\*.java
```

Let's consider the case of a simple 4-way junction. So we can have a 4-way junction agent called j1 and 4 road agents reporting statistics to the junction as r1, r2, r3 and r4.

Issue the following command in the ***same command prompt*** to run the program.

```
start java jade.Boot -gui r1:assignment.RoadAgent(j1 6000) r2:assignment.RoadAgent(j1
6000)  r3:assignment.RoadAgent(j1 6000) r4:assignment.RoadAgent(j1 6000)
j1:assignment.FourWayJunctionAgent(r1 r2 r3 r4 6000 6000)
```

As you can see for a RoadAgent we need 2 arguments; the junction agent which it reports and the period it reports (in milliseconds).

For a 4-way junction agent we need 6 arguments; the 4 junction agent(s)/road agents(s) which it is connected, the period which it reschedules its traffic (in milliseconds) and the period which it notifies its outgoing vehicle rates to its surrounding junction agents.

Similarly for a 3-way junction agent we need only 5 arguments (since there are only 3 roads); the 3 junction agents, traffic scheduling period and statistics reporting period.

Please note that you can specify any name as the local name when adding agents to the jade runtime (such as r1, r2, r3, r4 and j1 specified in the example). However when adding a junction agent, make sure that the local name starts with letter j.

Part 2

Multi-Agent Development Toolkits – A Survey

Multi-agent systems (MAS) consist of a set of multiple agents who are interacting with each other [1]. They can be used to solve problems which are hard or almost impossible to be solved only in the capacity of a single individual agent, but can be solved due the interaction between a set of individual agents with each other. Unlike conventional programs agents are small programs those are loaded and executed in memory only when required. Most of the time each of the agents in a system will have a limited knowledge comparing with the overall system. But the interesting part is that although the knowledge of each agent is limited; when multiple agents combine to form a multi-agent system, with interactions between each agent and interactions with the surrounding environment, this "***combination of agents***" serves as a powerful tool to model any complex system. Some examples for such complex systems involving heavy use of multi-agent system are online trading, disaster response and modeling social networks.

Due to the above reasons multi-agent programming is becoming popular as a separate programming paradigm to model real-world complex systems. Thus there have been many toolkits and frameworks, those are developed adhering to common standards defined for multi-agent programming and facilitating with basic services (such as creating, managing and destroying agents), thus saving development time. Following are some of such well known multi-agent development systems.

1. JADE [2][7] or Java Agent Development Framework is a framework designed for multi-agent development using java. JADE works on any platform which is supported by Java. JADE can create multiple containers where several agents can reside within a single container. The agents created by JADE adheres FIPA (Foundation for Intelligent Physical Agents) specification. Further a JADE multi-agent system can have several containers, either placed in a single host, or distributed between multiple hosts in a network. JADE uses RMI (Remote Method Invocation) to implement the communication of agents within a distributed environment. Agent programs can be written in Java and added into the the JADE runtime. Further communication among agents is implemented by adhering to standards specified under ACL (Agent Communication Language).
2. ZEUS [3][7] is a toolkit consisting of a set of components, written in Java, that can be categorized into three functional groups (or libraries). An agent component library, an agent building tool and a suite of utility agents comprising name server, facilitator and visualizer agents. Communication between agents is implemented by adhering to both standards ACL and KQML (Knowledge Query Manipulation Language). In addition it provides a set of utilities consists of code generators and visual editors which can be used to generate code for agent programs and design ontologies for adding common domain knowledge and communication between agents.
3. AgentBuilder [4][7] is another multi-agent development software toolkit which is implemented using both Java & C++. This toolkit uses KQML as the underlying standard to establish communication between agents. Further CORBA and TCP/IP communication

is used as the underlying technology behind extending the AgentBuilder toolkit for distributed environments. In terms of license, unlike JADE & ZEUS; AgentBuilder is a commercial software package. However their is a low-cost package which facilitates academic development.

4. Framsticks [5][7] is a software toolkit which is implemented by using multi-agent systems as the core concept. This toolkit focuses on modeling artificial life. We can use this toolkit to create 3D simulations on topics such as modeling the relationships between species and ecosystems in a 3D environment with virtual reality concepts. Further this toolkit can be also used to implement GIS (Geographic Information Systems) simulations based on a multi-agent paradigm. Agents can be programmed into the Framsticks toolkit by using an in-built high level scripting language called FramScript provided by the toolkit itself.
5. Breve [6][7] is another software package which can be used to model artificial life in a 3D simulation environment with the use of multi-agent systems. This package includes 3D articulated body physical simulation with collision detection and response. Developers can simulate realistic creatures within an OpenGL display engine to visualize simulated worlds. Each and every creature can be simulated as an agent. Agent programs can be written in python and can be configured to the toolkit runtime which supports python scripting.

References

[1] Multi-Agent Systems – Wikipedia
http://en.wikipedia.org/wiki/Multi-agent_system
[2] JADE – Java Agent Development Framework
http://jade.tilab.com/
[3] ZEUS: A Toolkit for Building Distributed Multi-Agent Systems by Hyacinth S. Nwana, Divine T. Ndumu, Lyndon C. Lee & Jaron C. Collis
[4] Agent Builder
http://www.agentbuilder.com/
[5] Framsticks
http://www.framsticks.com/
[6] Breeve
http://www.spiderland.org/node/2617
[7] Comparison of Agent-Based Software
http://en.wikipedia.org/wiki/Comparison_of_agent-based_modeling_software

Appendix A (Source Code)

RoadAgent.java

```
package assignment;

import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.TickerBehaviour;
import jade.lang.acl.ACLMessage;

import java.util.Random;

public class RoadAgent extends Agent {

	private AID nearestJunctionAgentID;
	private int reportingPeriod;

	protected void setup() {

		System.out.println("Setting up a new road agent!");
		System.out.println("Local-name is "+ getAID().getLocalName());
		System.out.println("GUID is "+ getAID().getName());

		// Retrieve the nearest junction agent
		Object[] args = getArguments();
		if (args != null && args.length > 1) {

			// load settings for nearest junction and reporting period
			String nearestJunctionName = (String)args[0];
			String reportingPeriodAsString = (String)args[1];
			this.reportingPeriod = Integer.parseInt(reportingPeriodAsString);
			this.nearestJunctionAgentID = new AID(nearestJunctionName,
AID.ISLOCALNAME);

			// add a behaviour to report incoming vehicle rate periodically
			addBehaviour(new TickerBehaviour(this, this.reportingPeriod) {

				@Override
				protected void onTick() {
					// get latest statistics on incoming vehicle rate
					int latestIncomingVehicleRate =
this.getLatestIncomingVehicleRate();

					try {

						// send the latest statistics to nearest junction
						ACLMessage msg = new
ACLMessage(ACLMessage.INFORM);
						msg.addReceiver(nearestJunctionAgentID);
						msg.setLanguage("English");
						msg.setOntology("road-ontology");

	msg.setContent(Integer.toString(latestIncomingVehicleRate));
						myAgent.send(msg);

						System.out.println("Road-agent " +
getAID().getLocalName() +
								" is sending latest incoming vehicle
rate " + latestIncomingVehicleRate +
								" to junction-agent " +
nearestJunctionAgentID.getLocalName());

					} catch (Exception ex) {
						ex.printStackTrace();
					}
				}

				// this is a dummy method returning the latest vehicle rate from a
camera
				// simulates reading inputs from the percepts of the agent
				private int getLatestIncomingVehicleRate() {
					Random rand = new Random();
					int next = rand.nextInt(10);
					return next;
				}

			});
		} else {
			System.out.println("Wrong configuration for the road agent : " +
getAID().getLocalName());
		}
	}
}
```

NWayJunctionAgent.java

```
package assignment;

import jade.core.Agent;
import java.util.Random;

import jade.core.AID;
import jade.core.behaviours.TickerBehaviour;
import jade.lang.acl.ACLMessage;

public abstract class NWayJunctionAgent extends Agent {

	protected AdjacentAgentKey[] adjacentAgentKeys;
	protected int reportingPeriod;
	protected int trafficCyclePeriod;

	protected int degreeOfJunction;

	protected NWayJunctionAgent(int degreeOfJunction) {
		this.degreeOfJunction = degreeOfJunction;
		this.adjacentAgentKeys = new AdjacentAgentKey[this.degreeOfJunction];
	}

	protected void setup() {

		System.out.println("Initializing a new " + this.degreeOfJunction + "-way agent 
with local name " +
				getAID().getLocalName() + " and GUID " + getAID().getName());

		// Retrieve the nearest junction agent
		Object[] args = getArguments();
		if (args != null && args.length >= (this.degreeOfJunction + 2)) {

			// load settings for adjacent junctions
			int currArgIndex = 0;
			for(; currArgIndex < this.degreeOfJunction; currArgIndex++) {
				adjacentAgentKeys[currArgIndex] = new 
AdjacentAgentKey((String)args[currArgIndex]);
```

```
			}

			// load settings for traffic cycle period
			String trafficCyclePeriodAsString = (String)args[currArgIndex];
			this.trafficCyclePeriod = Integer.parseInt(trafficCyclePeriodAsString);

			currArgIndex++;

			// load settings for reporting period
			// a multiple (>= 1) of trafficCyclePeriod is better
			// otherwise no-use
			String reportingPeriodAsString = (String)args[currArgIndex];
			this.reportingPeriod = Integer.parseInt(reportingPeriodAsString);

			// add a behaviour to report outgoing vehicle rate for the adjacent
junctions periodically
			addBehaviour(new TickerBehaviour(this, this.reportingPeriod) {

				@Override
				protected void onTick() {

					for(int currAdjAgentIdx = 0; currAdjAgentIdx <
adjacentAgentKeys.length; currAdjAgentIdx++) {

						AdjacentAgentKey currAdjAgent =
adjacentAgentKeys[currAdjAgentIdx];

						if(currAdjAgent != null &&
currAdjAgent.isRelatedWithAJunction()) {

							int latestOutgoingVehicleRateForCurrAgent
=

	getLatestOutgoingVehicleRate(currAdjAgentIdx);


	currAdjAgent.notifyLatestOutgoingVehicleRate(

	latestOutgoingVehicleRateForCurrAgent, myAgent);
						}
					}
				}
```

```
                        });

                        // add a behaviour to prepare the next schedule based on incoming
messages from
                        // adjacent agents
                        addBehaviour(new TickerBehaviour(this, this.trafficCyclePeriod) {

                                @Override
                                protected void onTick() {

                                        boolean messageProcessingDoneForThisRound = false;
                                        int messagesProcessedForThisRound = 0;
                                        while(!messageProcessingDoneForThisRound) {

                                                ACLMessage nextMessage = myAgent.receive();

                                                if(nextMessage != null &&
                                                                messagesProcessedForThisRound <=
adjacentAgentKeys.length + 2) {

        updateAdjacentAgentPointers(nextMessage);

                                                        messagesProcessedForThisRound++;
                                                } else {
                                                        messageProcessingDoneForThisRound =
true;
                                                }
                                        }

                                        generateTrafficSchedule();
                                }
                        });

                } else {
                        System.out.println("Wrong configuration for the " +
this.degreeOfJunction + "-way agent : " + getAID().getLocalName());
                }
        }

        // this is a dummy method returning the latest vehicle rate from a camera
        // simulates reading inputs from the percepts of the agent
        protected int getLatestOutgoingVehicleRate(int adjacentAgentIndex) {
                Random rand = new Random();
                int next = rand.nextInt(10);
                return next;
        }

        private void updateAdjacentAgentPointers(ACLMessage nextMessage) {

                AID senderAgentID = nextMessage.getSender();
                AdjacentAgentKey messageRelatedAgent = null;

                for(int currAdjAgentIdx = 0; currAdjAgentIdx < adjacentAgentKeys.length;
currAdjAgentIdx++) {

                        AdjacentAgentKey currAdjAgent = adjacentAgentKeys[currAdjAgentIdx];

                        if(currAdjAgent != null) {

                                if(currAdjAgent.getNeighbourId().getLocalName().
                                                equalsIgnoreCase(senderAgentID.getLocalName()))
{

                                        messageRelatedAgent = currAdjAgent;
                                        break;
                                }
                        }

                }

                if(messageRelatedAgent != null) {
                        String latestIncomingVehicleRateAsStr = nextMessage.getContent();
                        int latestIncomingVehicleRate =
Integer.parseInt(latestIncomingVehicleRateAsStr);

        messageRelatedAgent.setLatestIncomingVehicleRate(latestIncomingVehicleRate);

                        System.out.println("junction-agent " + getAID().getLocalName() + " is
receiving latest vehicle incoming rate " +
                                        latestIncomingVehicleRate + " from " +
senderAgentID.getLocalName());

                } else {
```

```
                              System.out.println("[ERROR] Message from a non-adjacent agent " +
senderAgentID.getLocalName() +
                                          " is received by junction-agent " +
getAID().getLocalName());
            }

      }

      protected abstract void generateTrafficSchedule();
}
```

FourWayJunctionAgent.java

```
package assignment;


public class FourWayJunctionAgent extends NWayJunctionAgent {

      public FourWayJunctionAgent() {
            super(4);
      }

      @Override
      protected void generateTrafficSchedule() {

            int maxVehicleRate1 =
this.getMaxTrafficTimeForRoad(this.adjacentAgentKeys[0],
this.adjacentAgentKeys[2]);
            int maxVehicleRate2 =
this.getMaxTrafficTimeForRoad(this.adjacentAgentKeys[1],
this.adjacentAgentKeys[3]);

            float timeForRoad1 = 0.0f;
            float timeForRoad2 = 0.0f;
            if(maxVehicleRate1 < 0 || maxVehicleRate2 < 0) {
                  //schedule equal time
                  timeForRoad1 = this.trafficCyclePeriod / 2.0f;
                  timeForRoad2 = this.trafficCyclePeriod / 2.0f;
            } else {
                  //schedule time based on proportion
                  timeForRoad1 = this.trafficCyclePeriod * (maxVehicleRate1 /
(float)(maxVehicleRate1 + maxVehicleRate2));
                  timeForRoad2 = this.trafficCyclePeriod * (maxVehicleRate2 /
(float)(maxVehicleRate1 + maxVehicleRate2));
            }

            System.out.println("Generating next traffic schedule by 4-way
junction-agent " + getAID().getLocalName() + " ...");
```

```
            System.out.println("4-way junction-agent " +
getAID().getLocalName() + " allocating " +
                        timeForRoad1 + " out of the total traffic cycle of "
+ this.trafficCyclePeriod +
                        " for the road from " +
this.adjacentAgentKeys[0].getLocalName() + " to " +
                        this.adjacentAgentKeys[2].getLocalName());

            System.out.println("4-way junction-agent " +
getAID().getLocalName() + " allocating " +
                        timeForRoad2 + " out of the total traffic cycle of "
+ this.trafficCyclePeriod +
                        " for the road from " +
this.adjacentAgentKeys[1].getLocalName() + " to " +
                        this.adjacentAgentKeys[3].getLocalName());
      }

      private int getMaxTrafficTimeForRoad(AdjacentAgentKey agent1Pointer,
AdjacentAgentKey agent2Pointer) {
            return Math.max(
                        agent1Pointer.getLatestIncomingVehicleRate(),
                        agent2Pointer.getLatestIncomingVehicleRate());
      }
}
```

ThreeWayJunctionAgent.java

```
package assignment;

public class ThreeWayJunctionAgent extends NWayJunctionAgent {

      public ThreeWayJunctionAgent() {
            super(3);
      }

      @Override
      protected void generateTrafficSchedule() {

            int maxVehicleRate1 =
this.getMaxTrafficTimeForRoad(this.adjacentAgentKeys[0],
this.adjacentAgentKeys[2]);
            int maxVehicleRate2 =
this.adjacentAgentKeys[1].getLatestIncomingVehicleRate();

            float timeForRoad1 = 0.0f;
            float timeForRoad2 = 0.0f;
            if(maxVehicleRate1 < 0 || maxVehicleRate2 < 0) {
                  //schedule equal time
                  timeForRoad1 = this.trafficCyclePeriod * (2.0f / 3.0f);
                  timeForRoad2 = this.trafficCyclePeriod * (1.0f / 3.0f);
            } else {
                  //schedule time based on proportion
                  timeForRoad1 = (2.0f / 3.0f) * this.trafficCyclePeriod *
```

```
                        (maxVehicleRate1 / (float)(maxVehicleRate1 +
maxVehicleRate2));
                  timeForRoad2 = this.trafficCyclePeriod - timeForRoad1;
            }

            System.out.println("Generating next traffic schedule by 3-way
junction-agent " + getAID().getLocalName() + " ...");

            System.out.println("3-way junction-agent " +
getAID().getLocalName() + " allocating " +
                        timeForRoad1 + " out of the total traffic cycle of "
+ this.trafficCyclePeriod +
                        " for the main road from " +
this.adjacentAgentKeys[0].getLocalName() + " to " +
                        this.adjacentAgentKeys[2].getLocalName());

            System.out.println("3-way junction-agent " +
getAID().getLocalName() + " allocating " +
                        timeForRoad2 + " out of the total traffic cycle of "
+ this.trafficCyclePeriod +
                        " for the road from " +
this.adjacentAgentKeys[1].getLocalName() +
                        " which is connected with the main road running from
" + this.adjacentAgentKeys[0].getLocalName() +
                        " to " + this.adjacentAgentKeys[2].getLocalName());
      }

      private int getMaxTrafficTimeForRoad(AdjacentAgentKey agent1Pointer,
AdjacentAgentKey agent2Pointer) {
            return Math.max(
                        agent1Pointer.getLatestIncomingVehicleRate(),
                        agent2Pointer.getLatestIncomingVehicleRate());
      }
}
```