<MatsimInterfaceProject>

Deliverable Three Report

<02-06-14>

<Edmund Kemsley>
s3281702

1. Introduction

This report accompanies the third and final deliverable of the MatsimInterface project. The general goal of the project was to develop an API between a BDI system and an Agent Based Model(ABM). The BDI system implemented in this project is Gorite, while the ABM is Matsim, a traffic simulator. While the bdimatsimIntegration application started as an implementation that connected Gorite and Matsim, this project centered on developing an adaptable generic framework to connect any BDI system to Matsim.

1.1 Project Overview

In the first stage, the project focused on adapting the application to use the data structures and communication interfaces specified in the bdsim.data package. The most important data structure is the AgentDataContainer which holds all information that is used to communicate between the BDI and ABM sides of the application. In this stage the application was developed to use the AgentDataContainer in a more purposeful manner.

In the second stage the project focused on separating functionality into three levels: a BDI/ABM level, a Matsim level and a taxi agent level or application dependent level.

This final stage built upon the last two iterations and continued to develop an API framework for a Matsim-BDI integration application by grouping functionality into cohesive objects. Like the second stage, most of the new design and implementation has been added on the ABM side of the application.

1.2 Purpose

The purpose of this report to explain and describe what has been developed during the project lifespan, the concepts behind the development and the details of the final deliverable.

<MatsimInterfaceProject>

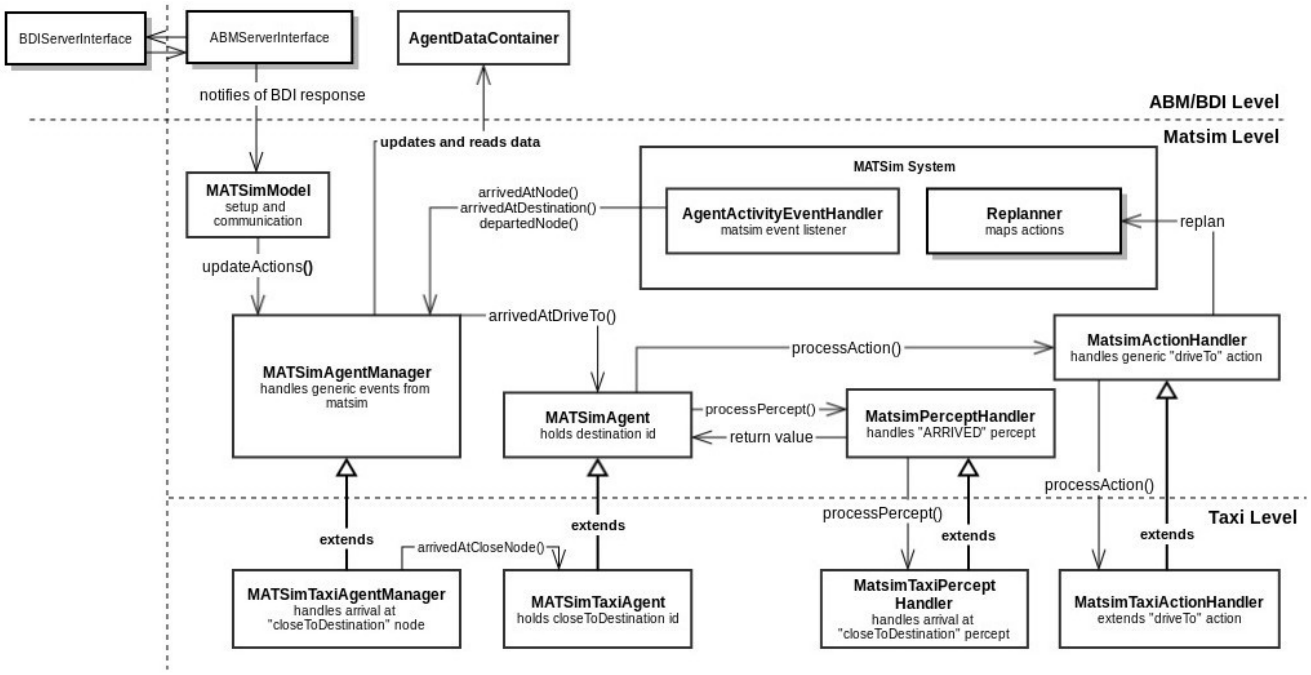
2. General Description

Overall Framework

The project built upon the developments of the last stage, splitting functionality on the ABM side into specific areas. The overall framework of the Matsim ABM module is shown in the diagram below.

The MatsimAgentManager has the most central role on the ABM side and handles new events from the Matsim system by using the MatsimAgent class to (a) check the validity of the event, such as check the destination location, and (b) pass the events to the MatsimPerceptHandler. The MatsimAgentManager also handles new actions from the BDI side, notified by MATSimModel, by passing them to a MATSimAgent's MatsimActionHandler.

There is a MatsimAgent object for every BDI agent in the Matsim system, saved in MatsimAgentManager. MatsimAgent is not a BDI agent, its role is to store Matsim state information about the agent that the MatsimAgentManager can query when needed. MatsimAgent also has a PerceptHandler and an ActionHandler, as mentioned previously, as well as a reference to its ActionContainer and PerceptContainer.



This framework can be extended to multiple levels of functionality. In this application the taxi-application level is an extension of the Matsim functionality. The MatsimAgentManager, MatsimAgent, MatsimAgentPerceptHandler and MatsimActionHandler classes are extended to hold taxi application level functionality, as shown in the diagram. When taxi-level functionality is involved, the extended classes,

MatsimTaxiAgentManager, MatsimTaxiAgent, MatsimTaxiActionHandler and MatsimTaxiPerceptHandler, will handle the processing. When Matsim level functionality is involved, the Matsim-level equivalents handle the processing.
Using this method of extension, the functionality of a higher level BDI Matsim application can be extended with little coding and design effort.

Agent Manager
The MatsimAgentManager and MatsimTaxiAgentManager are new classes introduced in this stage of development. From a design perspective, agent managers are designed to handle information coming from the both Matsim system and the BDI system. The MatsimAgentManager sits on the MatsimLevel and MatsimTaxiAgentManager sits on the taxi application level.

The agent manager checks for new actions from the BDI system when control is returned to the ABM side. The agent manager then uses the data in the AgentDataContainer to call the correct agents on the ABM side to process the actions. If it is an action on the Matsim level, then the MatsimActionHandler will be used to process the new action. If it is an action on the taxi application level, then the MatsimTaxiActionHandler will be used to process the new action and the outcome will be slightly different.

The MatsimAgentManager is notified of new events from Matsim when the AgentActivityEventHandler class calls one one of the three Matsim events in the MatsimAgentManger. The events are: arrivedAtNode, arrivedAtDestination and departedNode. The MatsimAgentManager implements Matsim level code such as processing arrivedAtDestination by updating the action in the AgentDataContainer to *PASSED* so it can be processed by the BDI side. The MatsimTaxiAgentManager extends the functionality of this class by implementing “close to destination” functionality by overriding the arrivedAtNode method.

The Matsim agent state information is stored in the MatsimAgent class and its taxi application extended class, MatsimTaxiAgent. Each also holds the related action and percept handler. For more information see the class descriptions section.

Action and Percept Handling
PerceptHandlers and ActionHandlers are designed to process information to and from the BDI side. A perceptHandler takes information from the ABM side and processes the information into something that can be sent to the BDI side. While an ActionHandler does the opposite, processing new actions from the BDI side into information that the ABM side uses to update itself. Each contains one method each, processPercept and processAction, respectively.
The MatsimPerceptHandler, MatsimAgentPerceptHandler and MatsimActionHandler all sit on the Matsim level. While the MatsimTaxiAgentPerceptHandler and MatsimTaxiActionHandler are extensions of the Matsim functionality and sit on the taxi application level.

When an Agent is added to the AgentManager, the applicable type of PerceptHandler and ActionHandler is saved in the Agent object to be called by the AgentManager.

The agents percepts handled by MatsimAgentPerceptHandler on a Matsim level are: "*Request Location*" and "*Arrived*". The only percept handled by MatsimTaxiAgentPerceptHandler is "*close to destination*". The agent action "*drive to*" is handled by the MatsimActionHandler and MatsimTaxiActionHandler. Though, each has slightly different functionality.

MATSimModel

MATSimModel is the first class that is called by the main method, its main functions are: initialising the Matsim system, creating the agent manager and facilitating communiation to and from the BDI side. The type of agent manager (currently either MatsimAgentManager or MatsimTaxiAgentManager) is set by the main method during initialisation.

Extendibility

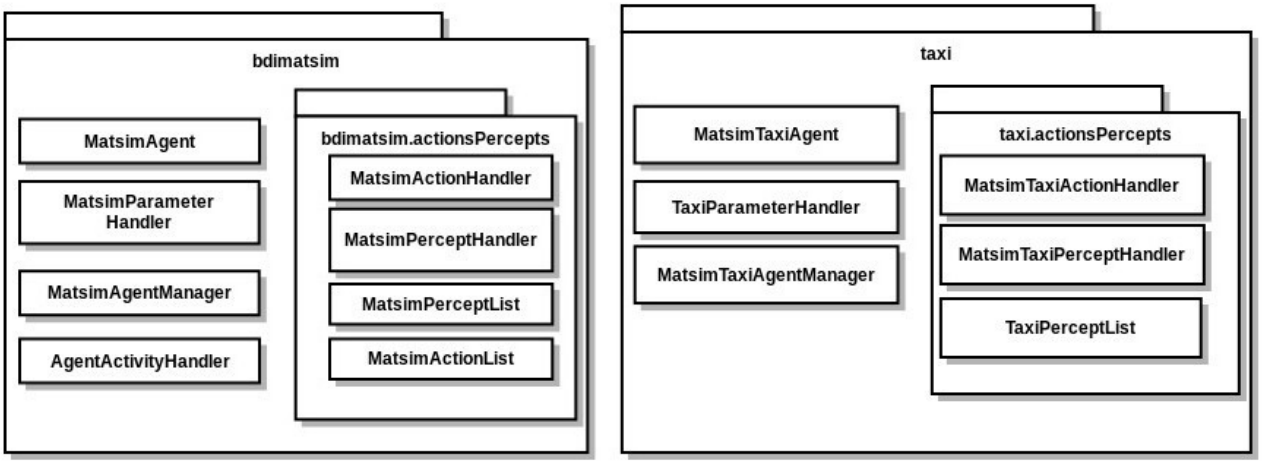
The MatsimAgentManager, MatsimActionHandler and MatsimPerceptHandler are designed to be extended to create new functionality in a Matsim related application. The extendibility has been demonstrated by the taxi level classes.

<MatsimInterfaceProject>

2.1 Application Specifics

Java Packages

The classes added to BDIMATSimIntegration in this project are placed in two root packages, bdimatsim and taxi. All packages are shown below.



Running and Building the Application

The application will need Java 1.7 or newer and the following jars in the classpath, *goritev11c04.jar*, *log4j-1.2.17.jar* and *MATSim.jar*, to be successfully built.

The program arguments are:
./input/simulation_parameters.txt ./input/config_withinday.xml
The first txt file is the application config file where simulaton parameters can be set. The second parameter is a MATSim config file.

The application is run by calling the main method in the SingleMain class. Three things are done in the main method to set up the application:

1. Create new matSimModel
2. Set the correct parameter handler
3. Set the ABMServer

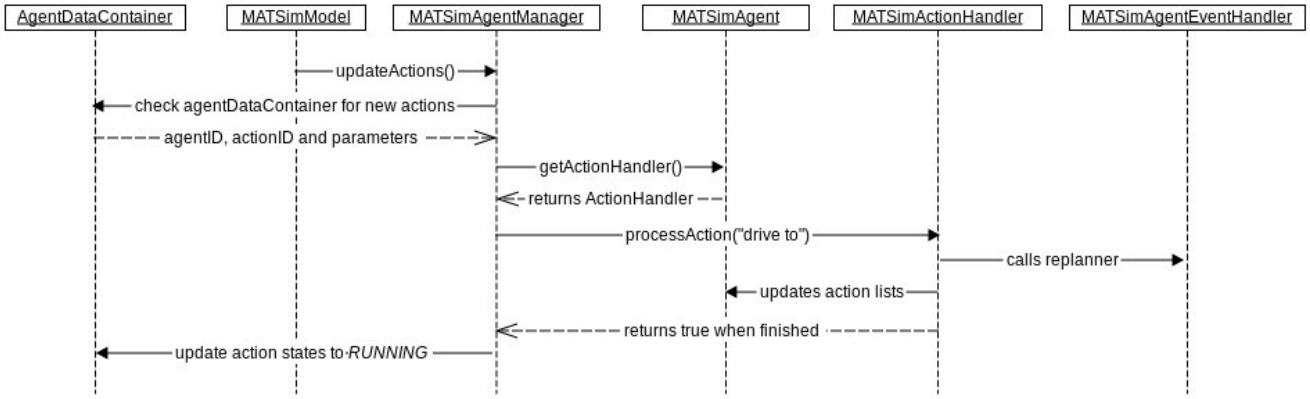
When declaring a new matsimModel object, the type of agent manager is passed as the second parameter of the constructor.
Example:

```
	MATSimModel matsimManager = new MATSimModel(args2, new
	MatsimTaxiAgentManager());
or
	MATSimModel matsimManager = new MATSimModel(args2, new MatsimAgentManager());
```

3. Example Events in the Taxi Application

Sample Action – drive to (Matsim level)

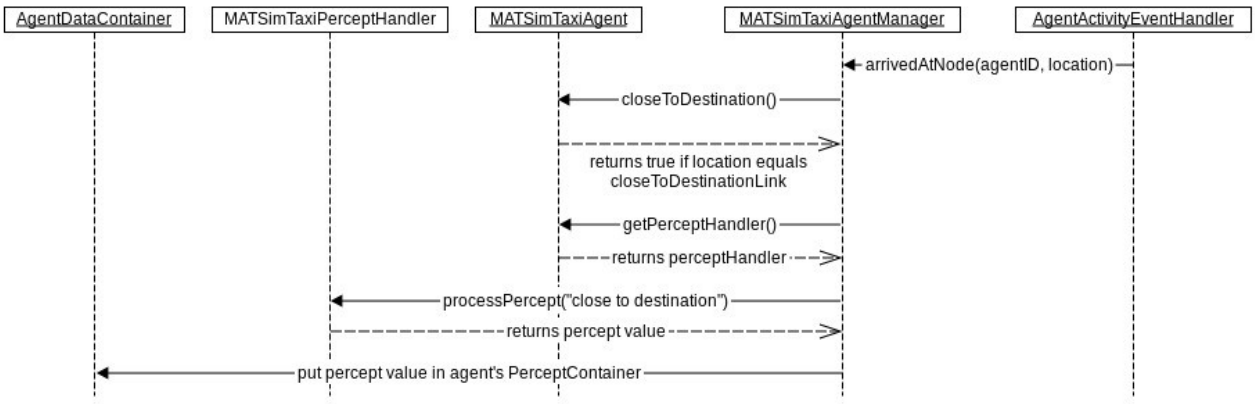
An action is added to the AgentDataContainer on the BDI side and the BDI side calls takeControl to return control back to the ABM side. When control is returned to the ABM side, MATSimModel calls the method *updateActions* in the MatsimAgentManager. The MatsimAgentManager finds all actions in the AgentDataContainer with the state *INITIATED* and calls the MatsimActionHandler method processAction which processes an action by updating the Matsim system and then updating the action lists in MatsimAgent. Finally the MatsimAgentManager changes the action state of the particular action in the AgentDataContainer to *RUNNING*.



Sample Percept – close to destination (Taxi level)

A percept starts as an event in the Matsim system. The event starts as a “link enter” event in AgentActivityEventHandler class. The AgentActivityEventHandler class calls the MatsimTaxiAgentManager method *arrivedAtNode,* passing the agentID and location as parameters. The agent Manager must check the close to destination link by calling the *closeToDestination* method in the MatsimTaxiAgent object. If the location passed by to the *arrivedAtNode* method matches the closeToDestinationLink in the MatsimTaxiAgent object then the *closeToDestination* method returns true and the MatsimTaxiAgentManager calls the MatsimTaxiAgentPerceptHandler which returns a value that the agent manager puts in the respective agent's PerceptContainer.

<MatsimInterfaceProject>



Logging Information

Logging is carried out on both the ABM side, in the MATSimModel class, and BDI side, in the BDIAgentModel class, to diagnose problems during communication between the two sides. There are three events that are recorded in logs:

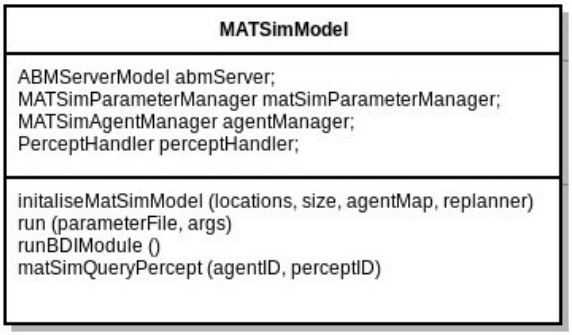
1. Before MATSimModel gives control to the BDIServerInterface it logs all the new percepts. After control is returned to MATSimModel it logs all the actions with the state INITIATED.
2. The BDI side does the opposite, when control is passed to the BDI side, BDIAgentModel logs all the percepts. After the BDI uses the percepts to create new actions, BDIAgentModel logs the new actions.
3. The BDI side calls the *queryPercept* method in ABMServerInterface, passing the perceptID *REQUESTLOCATION,* when it requests the location of a specific agent in Matsim. The Matsim side logs the request.

If the application is working correctly, the logs from the first two events, in the BDI side log and the ABM side log, will match.
The logging in BDIAgentModel is written to the *BDIAgentModel.log* file.
The logging in MATSimModel is written to the *MATSimServer.log* file.
All logged events use the *info* level.

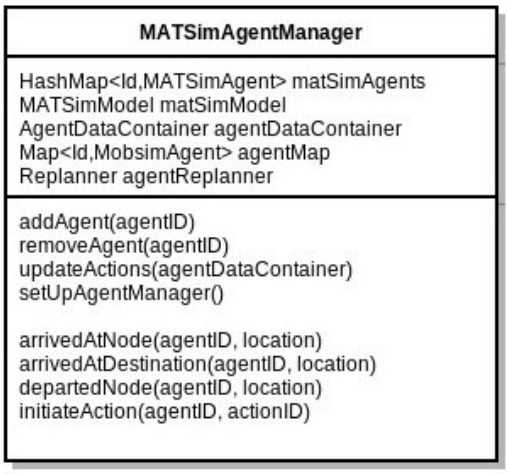
<MatsimInterfaceProject>

3.1 Class Descriptions

Matsim Level Classes

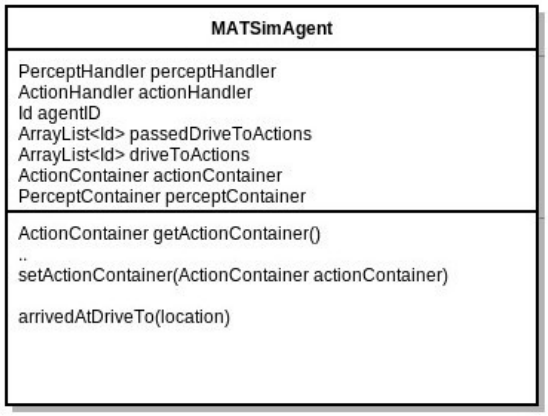


Class Name: MATSimModel
Package: bdimatsim
Description: The *run* method is called by the main method to set up the Matsim system. The bdimatsimWithinDayControlerListener class calls *initialiseMATSimModel* to return information from Matsim once it has been initialised. The *initialiseMATSimModel* method uses this information to set up MatsimAgentManager and the BDI module. *RunBDIModule* is called by the bdimatsimWithinDayEngine for every simulation step and passes control to the BDI side via the BDIServerInterface and then waits for a response before calling *updateActions* when a response is made from the BDI side.

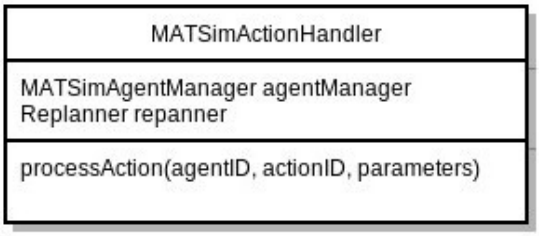


Class Name: MatsimAgentManager
Package: bdimatsim
Description: MatsimAgentManager holds a hash map of MatsimAgents and the AgentDataContainer. *SetUpAgentManager* is called by *initialiseMATSimModel* collects important information from MATSimModel about the Matsim system. The addAgent method is called by the bdimatsimWithinDayEngine before the simulation starts and creates a MatsimAgent object for every agent in the Matsim simulation. *AddAgent* also add every MatsimAgent to the hash map. The *updateActions* method calls *initiateAction* for every action with the state *INITIATED* in the AgentDataContainer.
ArrivedAtNode, arrivedAtDestination and *departedNode* are generic Matsim events called by AgentActivityEventHandler and are designed to be overridden to extend functionality. On the ABM side, MatsimAgentManager is the central object that adds to and reads the AgentDataContainer.

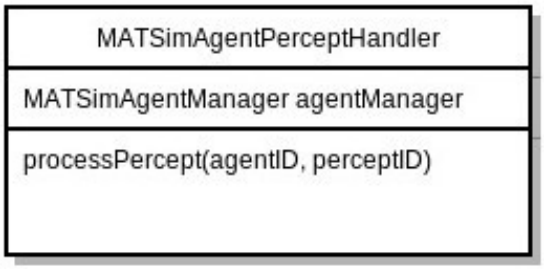
<MatsimInterfaceProject>



Class Name: MatsimTaxiAgentManager
Package: bdimatsim
Description: MatsimAgent holds Matsim state information in the action lists, driveToActions and passedDriveToActions. This class also holds a reference to its ActionContainer and PerceptContainer as well as references to the related action handler and percept handler. The MatsimAgentManager calls *arrivedAtDriveTo* from the *arrivedAtDestination* method. The *arrivedAtDriveTo* uses the list driveToActions to find the related action in the AgentDataContainer.



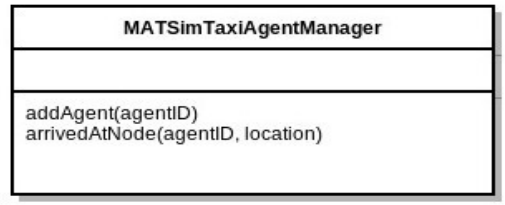
Class Name: MatsimActionHandler
Package: bdimatsim.actionsPercepts
Description: The MatsimActionHandler is held in every MatsimAgent object and is called by *initiateAction* in MatsimAgentManager to process a new action from the BDI side. The method *processAction* uses the replanner to update the Matsim system with a new agent action and adds the action to an action list in MatsimAgent.



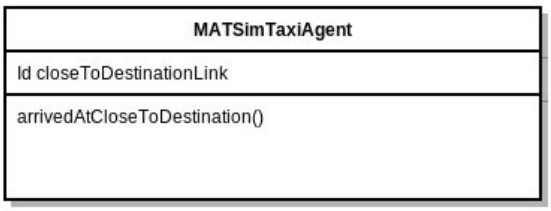
Class Name: MatsimAgentPerceptHandler
Package: bdimatsim.actionsPercepts
Description: The method processPercept in MatsimAgentPerceptHandler is called by one of the Matsim event methods in MatsimAgentManager, a*rrivedAtNode, arrivedAtDestination* and *departedNode. ProcessPercept* information that the MatsimAgentManager adds to an agent's PerceptContainer to be sent to the BDI side.

<MatsimInterfaceProject>

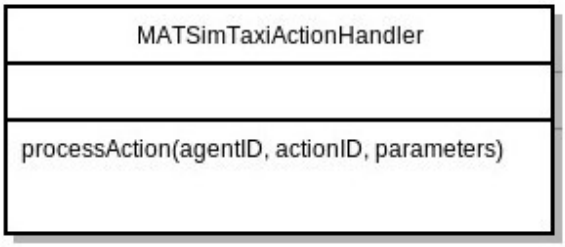
Taxi Level Classes



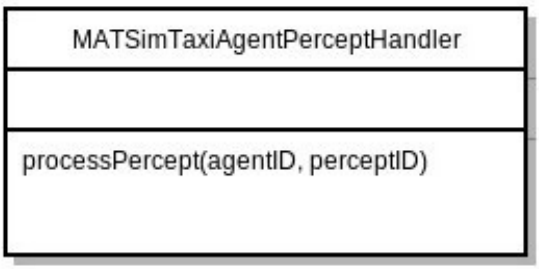
Class Name: MatsimTaxiAgentManager
Package: taxi
Extends: MatsimAgentManager
Description: MatsimTaxiAgentManager extends the functionality of its parent class by overridding two methods, *addAgent* and *arrivedAtNode*. The *addAgent* method is similar to the parent method except in that it creates a new MatsimTaxiAgent rather than a MatsimAgent. *ArrivedAtNode* is called by the ActivityEventHandler and implements the “close to destination” functionality as part of the taxi application.



Class Name: MatsimTaxiAgent
Package: taxi
Extends: MatsimAgent
Description: MatsimTaxiAgent extends the MatsimAgent functionality by holding a closeToDestinationLink. The *arrivedAtCloseToDestination* method compares this Id to the location passed by the *arrivedAtNode* method in the MatsimAgentManager and returns true if they match.



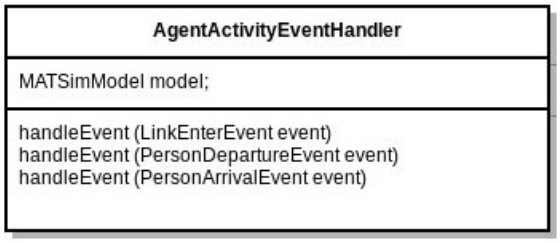
Class Name: MatsimTaxiActionHandler
Package: taxi.actionsPercepts
Extends: MatsimActionHandler
Description: MatsimTaxiActionHanlder extends the functionality of its parent class by overridding the *processAction* method. In this class a new “drive to” action adds the closeToDestinationLink to the MatsimTaxiAgent as well as updates the Matsim system.



Class Name: MatsimTaxiAgentPerceptHandler
Package: taxi.actionsPercepts
Extends: MatsimAgentPerceptHandler
Description: The MatsimTaxiAgentPerceptHandler overrides the parent's *processPercept* method. It is called by the *arrivedAtNode* method in the MatsimTaxiAgentManager class. The extended functionality processes the “close to destination” percept.

<MatsimInterfaceProject>

Other



Class Name: AgentActivityEventHandler
Package: bdimatsim
Description: The AgentActivityEventHandler listens for Matsim events and calls either the methods *ArrivedAtNode, arrivedAtDestination or departedNode* in the MatsimAgentManager or MatsimTaxiAgentManager (depending on the application).