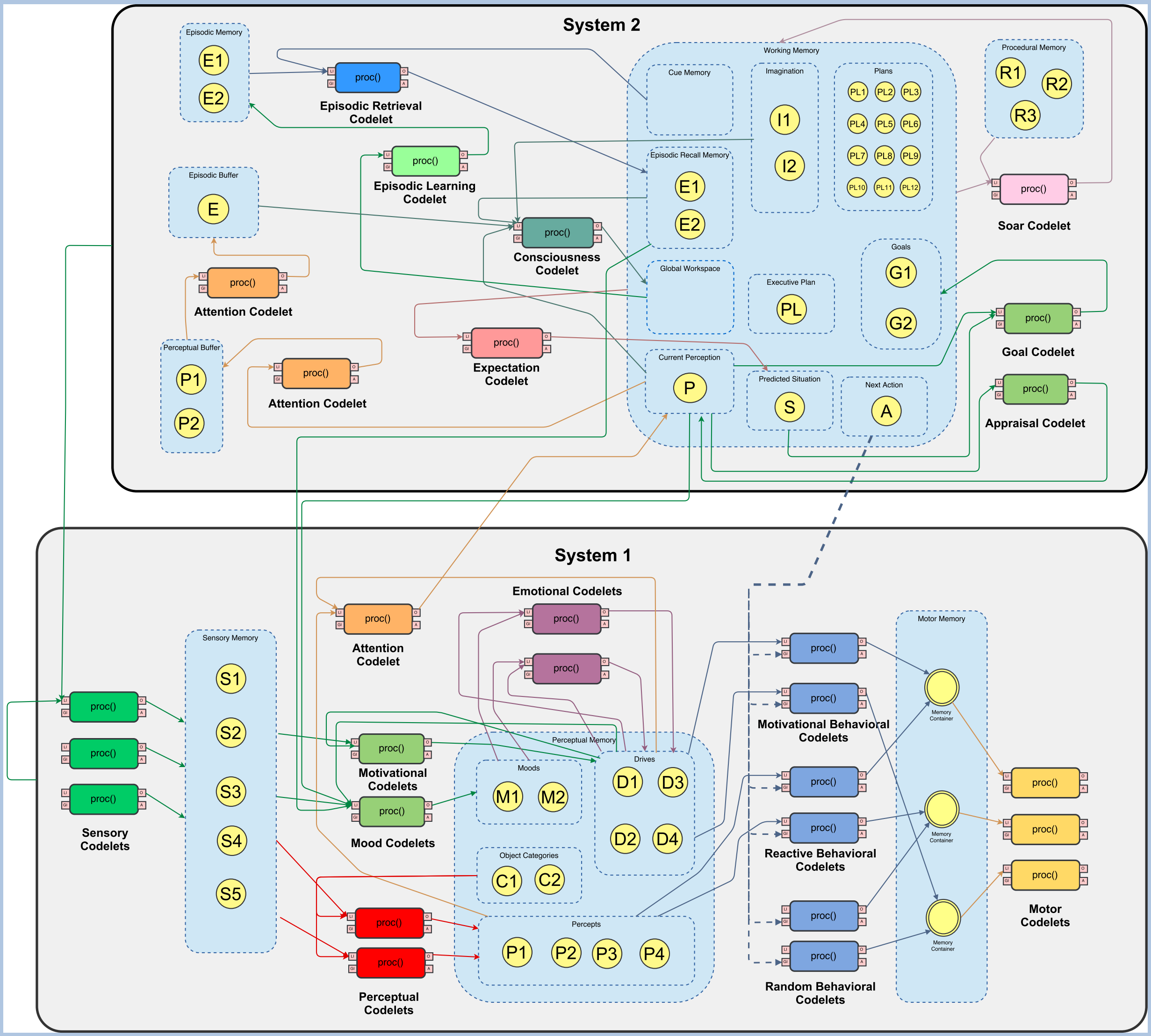
Research Project Technical Report

The MECA Technical User Manual

Andre Paraense & Ricardo Gudwin



DCA-FEEC-UNICAMP

First printing, September 2017

Table of Contents

1 Overview 5

1.1 What is MECA? 5

1.2 What are we going to build? 5

1.3 What will you learn? 6

1.4 What will you need? 6

1.5 Experience 6

2 Get and run the sample code 7

2.1 Get the sample code 7

2.2 Run the sample code 7

3 Development of the sample app 11

3.1 Preparing the start project 11

3.1.1 App architecture 12

3.2 Adding sensors and connecting the MECA mind 14

3.3 Building System 1 codelets 19

3.4 Building System 2 codelets 25

3.5 Conclusion 31

4 References 33

1. Overview ↰

This Technical User Manual will teach you how to build an application using the Multipurpose Enhanced Cognitive Architecture (MECA) through a hands-on codelab.

1.1 What is MECA? ↰

MECA (R. Gudwin et al., 2017) is a cognitive architecture (Kotseruba et al., 2016) implemented in the Java language. MECA was designed based on many ideas coming from Dual Process Theory (Osman, 2004), Dynamic Subsumption (Nakashima and Noda, 1998), Conceptual Spaces (Gärdenfors, 2014), Grounded Cognition (Barsalou, 2010), and constructed using CST (Paraense et al., 2016), a toolkit for the construction of cognitive architectures in Java[1]. Basically, MECA promotes a hybridism of SOAR (Laird, 2012), used to implement rule-based processing and space-state exploration in System 2 modules, with a Dynamic Subsumption Motivational System performing the role of System 1, using a representational system based on conceptual spaces and grounded cognition. R. R. Gudwin, (2017a) goes into detail of the MECA foundational background, concepts behind System 1, System 2 and its subsystems and explains the Software Architecture.

1.2 What are we going to build? ↰

When you have completed this codelab, you will have a MECA app which will be able to control traffic light junctions in a computer simulation using SUMO (Krajzewicz et al., 2012) and a specific urban network model of choice. The app will autonomously control many different traffic light junctions concomitantly through Cognitive Managers for Urban Mobility, which detailed explanation can be found in R. R. Gudwin, (2017b).

[1] http://cst.fee.unicamp.br/

1.3 What will you learn? ↰

- How to add MECA lib to a sample app;
- How to add sensors and actuators that connect the MECA mind with the SUMO simulation;
- How to build System 1 codelets;
- How to mount the MECA mind;
- How to run the MECA app;
- How to built System 2 codelets;
- How to analyze and interpret MECA visualization tools.

1.4 What will you need? ↰

- The Java 8 SDK (or later).
- The SUMO 0.30 version (or later).
- The latest JTraci library[2].
- The latest MECA library.
- A Java IDE of your choice (Eclipse is used as the guidance example in this codelab).

1.5 Experience ↰

- You will need to have previous Java development knowledge.
- You will need to have previous CST development knowledge[3].
- You will need to have previous basic knowledge of SUMO[4] and TraCI[5] (R. R. Gudwin, 2016).
- You will also need to have read the theory of MECA (R. R. Gudwin, 2017a; R. Gudwin et al., 2017).

[2]A Java library to communicate with SUMO through its TraCI protocol. Found in https://github.com/CST-Group/JTraCI

[3]http://cst.fee.unicamp.br/tutorials

[4]http://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/

[5]http://www.sumo.dlr.de/wiki/TraCI

2. Get and run the sample code ↰

2.1 Get the sample code ↰

Just clone the Git repository from the command line.

```
git clone https://github.com/CST-Group/codelab-meca.git
```

This codelab repository contains two sample-projects:

- app-start - The starting code that you'll build upon in this codelab.
- app-done - The complete code for the finished sample app.

2.2 Run the sample code ↰

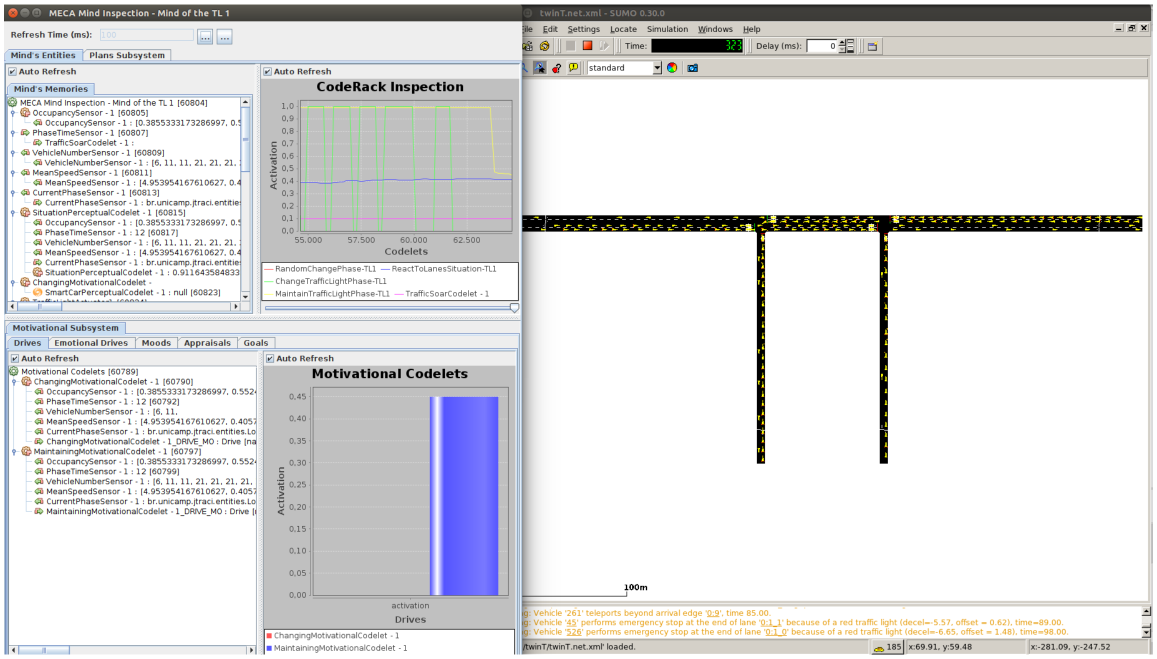
First, let's see what the completed sample app looks like. The following instructions will consider you are using the Eclipse Mars Java IDE, but you can actually use any Java IDE of your choice and perform the same actions analogously.

With the code downloaded, the following instructions describe how to open and run the completed sample app in Eclipse IDE:

1. Select the File->New->Project->Java Project... menu option.
2. Select the 'app-done' directory from the sample code folder as the location and give the project a name.
3. Make sure the Java 8 JRE System Library as well as all the JAR files in the '/lib' directory are selected on the build path, which are:
    - commons-math3-3.0.jar

- cst.jar
- google-collections-1.0.jar
- gson-2.7.jar
- jcommon-1.0.20.jar
- jfreechart-1.0.15.jar
- jsoar-core.jar
- jsoar-debugger.jar
- JTraci.jar
- MECA.jar
- slf4j-api-1.8.0-alpha2.jar

4. Select Run->Run configurations... menu options.
5. In 'Java application', create a "New Launch configuration' pointing to the Main Class $'br.unicamp.MECA_Demo.Main'$. In the 'Arguments' tab, insert '127.0.0.1 5001 -R -S' as program arguments and click 'Run'.



```
MECA Mind Inspection - Mind of the TL 1 [60804]
  OccupancySensor - 1 [60805]
    OccupancySensor - 1 : [0.385533173286997, 0.5
  PhaseTimeSensor - 1 [60807]
    TrafficSoarCodelet - 1 :
  VehicleNumberSensor - 1 [60809]
    VehicleNumberSensor - 1 : [6, 11, 11, 21, 21, 21,
  MeanSpeedSensor - 1 [60811]
    MeanSpeedSensor - 1 : [4.953954167610627, 0.4
  CurrentPhaseSensor - 1 [60813]
    CurrentPhaseSensor - 1 : br.unicamp.jtraci.entitie
  SituationPerceptualCodelet - 1 [60815]
    OccupancySensor - 1 : [0.385533173286997, 0.5
    PhaseTimeSensor - 1 : 12 [60817]
    VehicleNumberSensor - 1 : [6, 11, 11, 21, 21, 21,
    MeanSpeedSensor - 1 : [4.953954167610627, 0.4
    CurrentPhaseSensor - 1 : br.unicamp.jtraci.entitie
    SituationPerceptualCodelet - 1 : 0.911643584833
  ChangingMotivationalCodelet -
    SmartCarPerceptualCodelet - 1 : null [60823]
```

```
Motivational Codelets [60789]
  ChangingMotivationalCodelet - 1 [60790]
    OccupancySensor - 1 : [0.385533173286997, 0.552
    PhaseTimeSensor - 1 : 12 [60792]
    VehicleNumberSensor - 1 : [6, 11,
    MeanSpeedSensor - 1 : [4.953954167610627, 0.4057
    CurrentPhaseSensor - 1 : br.unicamp.jtraci.entities.Lo
    ChangingMotivationalCodelet - 1_DRIVE_MO : Drive [na
  MaintainingMotivationalCodelet - 1 [60797]
    OccupancySensor - 1 : [0.385533173286997, 0.552
    PhaseTimeSensor - 1 : 12 [60799]
    VehicleNumberSensor - 1 : [6, 11, 11, 21, 21, 21, 21,
    MeanSpeedSensor - 1 : [4.953954167610627, 0.4057
    CurrentPhaseSensor - 1 : br.unicamp.jtraci.entities.Lo
    MaintainingMotivationalCodelet - 1_DRIVE_MO : Drive [
```

```
g: Vehicle '261' teleports beyond arrival edge '0:9', time 85.00.
g: Vehicle '45' performs emergency stop at the end of lane '0:1_1' because of a red traffic light (decel=-5.57, offset = 0.62), time=89.00.
g: Vehicle '526' performs emergency stop at the end of lane '0:1_0' because of a red traffic light (decel=-6.65, offset = 1.48), time=98.00.
/twinT/twinT.net.xml' loaded.   185   x:69.91, y:59.48   x:-281.09, y:-247.52
```

Figure 2.1: **Sample code running**

If everything works, you will be able to see the SUMO simulation running, with the MECA controller acting autonomously to control the traffic light junctions, and also the debugger MECA mind viewer's showing the codelet's and memory object's values.

Take a minute to observe the controller in action. Try inserting 1000 ms delay in the SUMO user interface to better watch the traffic jam and how the controller autonomously change the junction phases. Moreover, take some minutes exploring the "Mind inspection" windows, which will show the contents of the mind of each cognitive manager, one for each junction, depending on the urban network model running. Under the "Mind's Entities" tab, you will be able to inspect the contents of the mind's memories, the codelet's activations in real time and also the motivational system drives. Under the "Plan's Subsystem" tab, you will be able to inspect the inner workings of the SOAR Codelet and its decisions and plans being constructed. Finally, try hitting the 'stop' red square button in the SUMO user interface, which will pause the SUMO simulation. What do you see? Notice in the mind inspection windows that the controller do not stop, it keeps deciding the next step even though the simulation is paused, emphasizing the complete asynchronous nature of the cognitive managers built on top of MECA. Hit the play button again in the SUMO user interface and immediately the cognitive managers in each junction are again changing the traffic light phases.

3. Development of the sample app ↰

3.1 Preparing the start project ↰

We will need to add Cognitive Managers capable of autonomously controlling each junction of the urban network model to the starter project. In this beginning, all that we are going to have is an app connected to the SUMO simulation. Source code for sensors, actuators, System 1 codelets and System 2 codelets will also be available, but not connected to a MECA mind. Our job will be to make these connections, inspecting the source code of the codelets in order to learn how to build them and how to connect them, a process called "mounting a MECA mind".

Now you are ready to build on top of the starter project using your Java IDE. The process of configuring the starter project is very much the same as the complete project:

1. Select the File->New->Project->Java Project... menu option.
2. Select the 'app-start' directory from the sample code folder as the location and give the project a name.
3. Make sure the Java 8 JRE System Library as well as all the JAR files in the '/lib' directory are selected on the build path, which are:
    - commons-math3-3.0.jar
    - cst.jar
    - google-collections-1.0.jar
    - gson-2.7.jar
    - jcommon-1.0.20.jar
    - jfreechart-1.0.15.jar
    - jsoar-core.jar
    - jsoar-debugger.jar
    - JTraci.jar
    - MECA.jar

   - slf4j-api-1.8.0-alpha2.jar
4. Select Run->Run configurations... menu options.
5. In 'Java application', create a "New Launch configuration' pointing to the Main Class $'br.unicamp.MECA_{D}emo.Main'$. In the 'Arguments' tab, insert '127.0.0.1 5001 -R -S' as program arguments and click 'Run'.

You will notice that this time the app will run without UIs to representing the mind of the Cognitive Managers. The reason is that there is not any cognitive manager so far: all we have is an app, connected to the SUMO simulation, which pass on through the next steps of the simulation without actuating on the traffic lights. If you spend sometime watching the simulation in the sumo UI, you will be able to notice that the traffic lights are running a fixed time program, which repeats itself regardless of the traffic situation. This is going to change soon.

3.1.1 App architecture

In this beginning, all the app has is a Main Class $'br.unicamp.MECA_{D}emo.Main'$, which instantiates a Thread passing a Runnable object of the type $'br.unicamp.MECA_{D}emo.SimulationRunnable'$. If you take sometime to inspect the SimulationRunnable class, you will be able to learn that its role is to open a connection to the SUMO simulation through the JTraci library (which implements SUMO JTraCI protocol) and controls the simulation, running through each step until all expected cars have been loaded, run and left the urban network. This is very much all that is running in the starter project.

Even though the starter project is very simple, the project already has all the source code we need to get to the final project in terms of sensors, actuators and codelets. The main reason for this arrangement is so we can connect all this pieces into a MECA mind in this codelab faster, and learn through an out of the box example. So, before jumping to adding code, let's take a look in the rest of the source code already available in the starter project and how it is organized in packages. But first, it will be instructive if we learn the app architecture in terms of how System 1 and System 2 Codelets were modeled, according to the urban traffic control context the Cognitive Managers will be inserted. Let's start examining the big picture in figure 3.1.

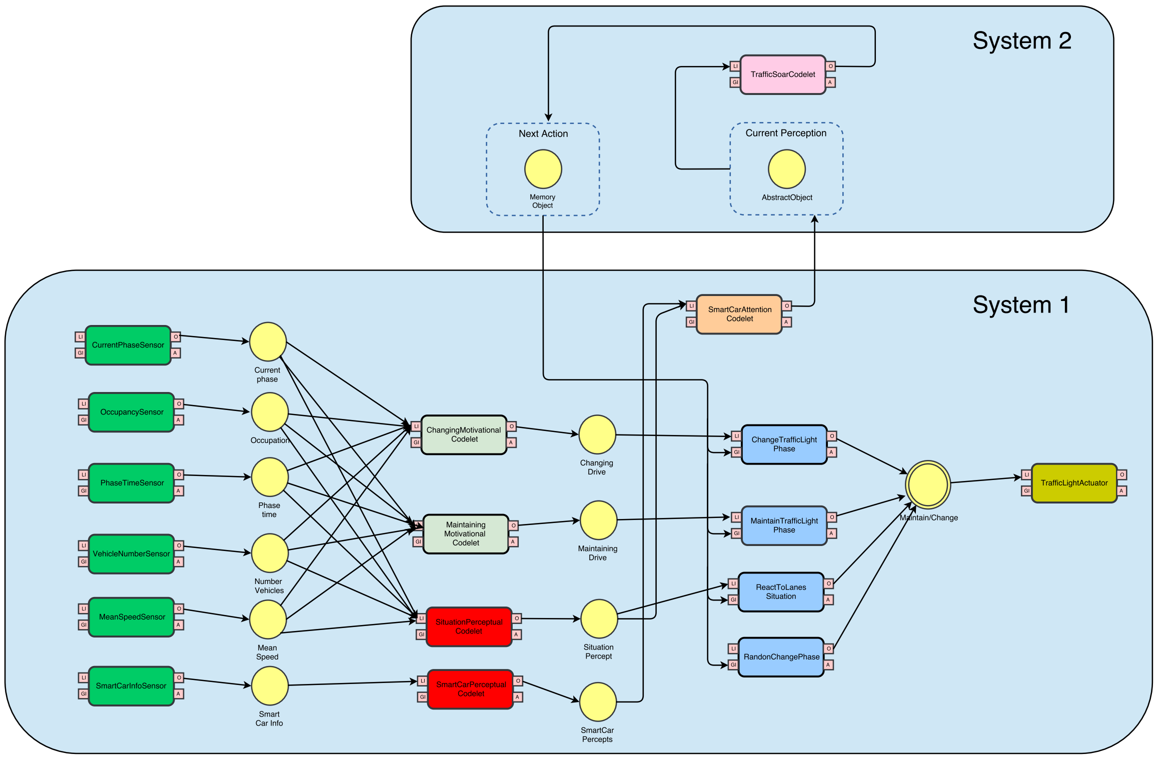


Figure 3.1: **App Architecture**

Figure 3.1 shows the final version of the Junction Cognitive Manager you have seen in the report "A Cognitive Manager for Urban Mobility using MECA (R. R. Gudwin, 2017b). Keeping this App Architecture in mind, let's examine the source code available:

- *br.unicamp.MECA*$_D$*emo.codelets.sensory*: in this package, you will find the six different sensory sensor codelets. Take sometime to understand how each of them is implemented extending sensory codelets in order to acquire an specific type of information from the SUMO simulation, through TraCI protocol using JTraci library methods. Remembering the types of information gathered:
  - Occupation: The average of the occupation of all lanes in the junction, ranging from 0 to 100%. The occupation of each lane depends on the length of the lane, the number of cars waiting in the queue and the size of the cars.
  - Phase Time: The amount of time since the last phase change.
  - Number of Vehicles: The number of vehicles in all lanes
  - Average Speed: The average speed of all vehicles crossing the junction.
  - Smart Car Info: This input receives messages from Smart Cars, informing their current position. If there is no Smart Car in the system, this input remains silent all the time.
  - Current phase: the last chosen phase of the traffic light junction.
- *br.unicamp.MECA*$_D$*emo.codelets.motivational*: in this package, you will find the source code to understand how the motivational codelets are implemented so as to

propose changes and/or to maintain the current traffic lights phase depending on the Drives which are used as inputs to them.
- *br.unicamp.MECA*$_D$*emo.codelets.perceptual*: you will find to implementations of perception codelets, responsible for computing abstractions on the input sensory information:
  - Situation Perceptual Codelet: responsible for computing a traffic condition index for an specific junction, regardless of its geometry
  - SmartCar Perceptual Codelet: responsible for packaging the smart cars information when they approach an specific junction.
- *br.unicamp.MECA*$_D$*emo.codelets.behavioral.random*: implementation of a random behavior, which based on a random variable proposes changes to the current traffic light phase.
- *br.unicamp.MECA*$_D$*emo.codelets.behavioral.reactive*: implementation of a codelet which based on the current traffic situation delivered by the Situation Perceptual codelet proposes to change or maintain the current traffic light phase.
- *br.unicamp.MECA*$_D$*emo.codelets.behavioral.motivational*: implementation of codelets which propose changes or maintenance of the current traffic light phase based on the output of the respective motivational codelets.
- *br.unicamp.MECA*$_D$*emo.codelets.motor*: implementation of a motor codelet responsible for actually performing the operation of changing the traffic light phase in the SUMO simulation, using the TraCI protocol through the JTraci library methods.

Now that you have invested sometime to study the implementation of the codelets which together for the Junction Cognitive Manager, let's start wiring them together as modeled in figure 3.1. This wiring will be done step by step, so as to give you a chance to run the controller in small steps and understand better how the codelets work asynchronously together.

3.2 Adding sensors and connecting the MECA mind

In this step, we are going to connect a MECA Mind (Cognitive Manager) for each junction in the urban traffic network model being simulated and attach five of the six sensors modeled in figure 3.1: occupancy, phase time, vehicle number,mean speed and current phase, as shown in figure 3.2. By the time we have finished this step, we will be able to run the simulation again, but this time we are going to have Cognitive Managers for each junction sensoring the incoming lanes of each junction (but still not actuating on the traffic lights). We will use the CST visualization tool in order to acknowledge that the mind of our Cognitive Manager are getting the values coming from the sensors we are implementing.

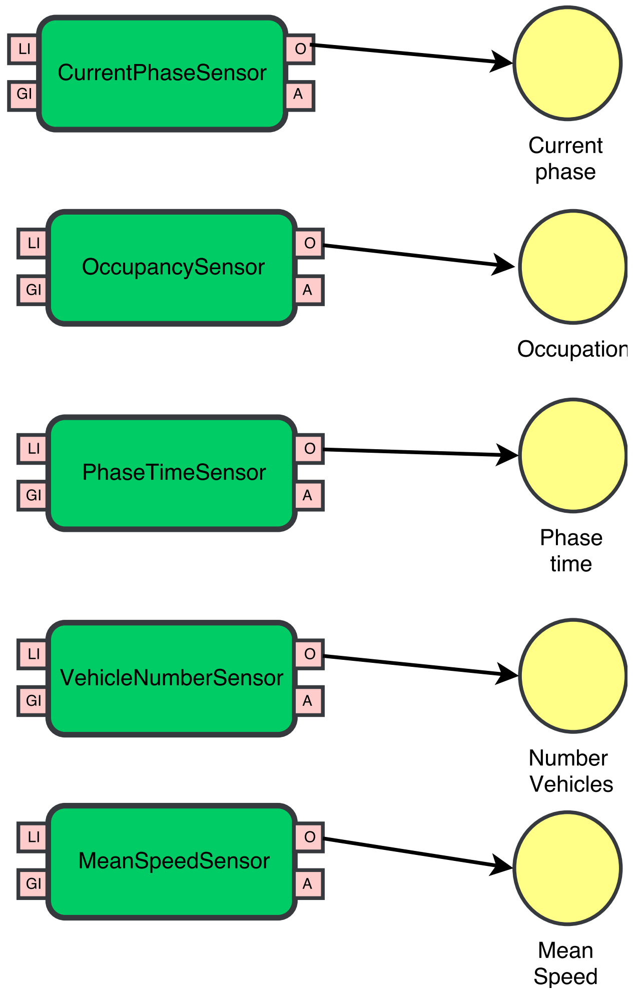


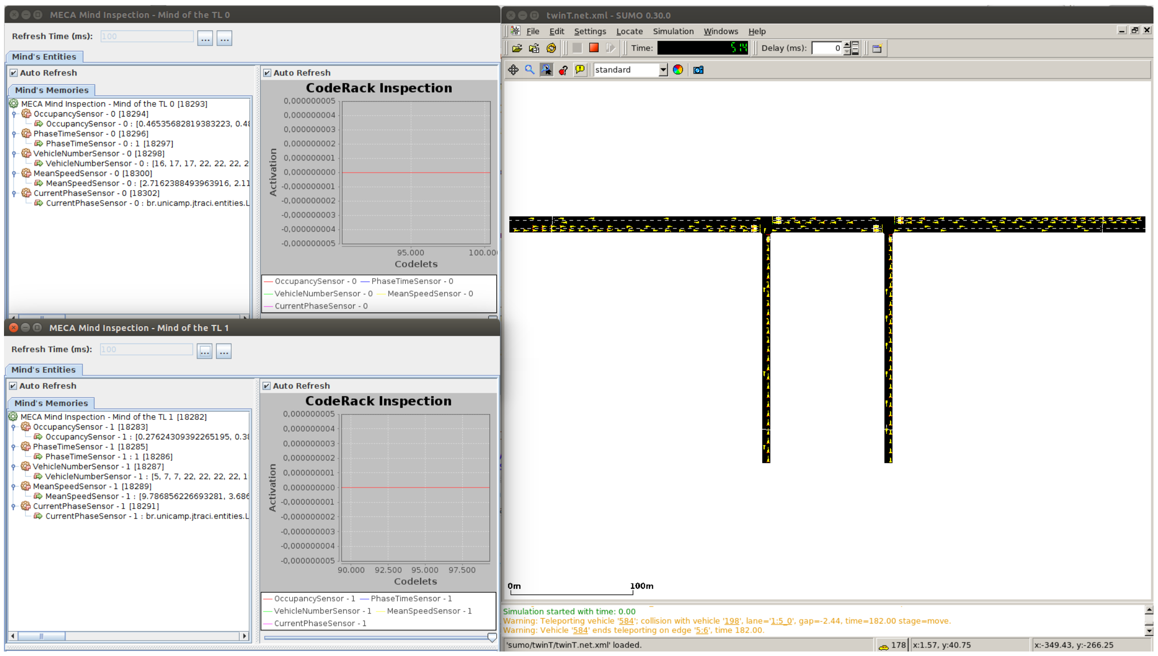
Figure 3.2: **Five sensors implemented in the first step**

In your starter project, change the content of the constructor method $br.unicamp.MECA_Demo.Main.Main(..$ to the following (the explanation of each structure can be found in the format of in line comments in the source code):

```
public Main(String ipServidor, int port, boolean sumoGui, boolean sumoConnect){

 SimulationRunnable simulationRunnable = new SimulationRunnable(ipServidor,
 port, sumoGui, sumoConnect);

 /* The list of Meca Minds (or Cognitive Managers) that will be created - one
 * for each junction
 */
 List<MecaMind> mecaMinds = new ArrayList<>();

 /* Looping through all Traffic Lights junctions in the simulation scenario,
 * in order to capture the elements in each one of them
 */
 for (TrafficLight trafficLight : simulationRunnable.getTrafficLights()) {

  /* Sensory codelets we are about to create for this Cognitive Manager in
  * this junction
  */
  List<SensoryCodelet> sensoryCodelets = new ArrayList<>();

  /* Hold the reference to each incoming lane of this junction.
  * Each sensory codelet will have this reference in order to be
  * able to read values from this incoming lane.
  */
  ArrayList<Lane> incomingLanes = new ArrayList<>();

  /* Hold the reference to each induction loop positioned in the
  * incoming lanes of this junction. Each sensory codelet will have this
  * reference in order to be able to read values from this induction loops.
  */
  ArrayList<InductionLoop> inductionLoops = new ArrayList<>();

  /*
  * In order to get the reference for incoming lanes and induction loops in
  * this specific junction, we will loop through structures called "Links"
  * in the TraCI protocol, which are positioned inside "Controlled Links",
  * which are positioned inside Traffic Lights (all objects in JTraci,
  * implementing the TraCI protocol)
  */
  ControlledLinks controlledLinks = trafficLight.getControlledLinks();
  br.unicamp.jtraci.entities.Link[][] links = controlledLinks.getLinks();
  for (int i = 0; i < links.length; i++) {
   for (int j = 0; j < links[i].length; j++) {

    br.unicamp.jtraci.entities.Link link = links[i][j];
    Lane incomingLane = link.getIncomingLane();
    incomingLanes.add(incomingLane);
    inductionLoops.add(simulationRunnable.getInductionLoop(incomingLane));
   }
  }
  /*
  * For this junction, we are going to create five sensors for now
  * - occupancy, phase time, vehicle number, mean speed and current
  * phase. They are all going to receive references for the incoming
  * lanes and induction loops they read from, and also the Traffic
  * Light (junction) they are associated with.
  */
  OccupancySensor occupancySensor = new OccupancySensor("OccupancySensor - " +
  trafficLight.getID(), inductionLoops, incomingLanes, trafficLight);
  sensoryCodelets.add(occupancySensor);

  PhaseTimeSensor phaseTimeSensor = new PhaseTimeSensor("PhaseTimeSensor - " +
  trafficLight.getID(), trafficLight);
  sensoryCodelets.add(phaseTimeSensor);

  VehicleNumberSensor vehicleNumberSensor = new VehicleNumberSensor(
  "VehicleNumberSensor - " + trafficLight.getID(), inductionLoops,
  incomingLanes);
  sensoryCodelets.add(vehicleNumberSensor);

  MeanSpeedSensor meanSpeedSensor = new MeanSpeedSensor("MeanSpeedSensor - " +
  trafficLight.getID(), inductionLoops, incomingLanes);
  sensoryCodelets.add(meanSpeedSensor);

  CurrentPhaseSensor currentPhaseSensor = new CurrentPhaseSensor(
  "CurrentPhaseSensor - " + trafficLight.getID(), trafficLight);
  sensoryCodelets.add(currentPhaseSensor);

  /*
  * Then, we create the MecaMind for this Cognitive Manage in this
  * junction, and pass references to the sensory codelets in a list.
  */
  MecaMind mecaMind = new MecaMind("Mind of the TL "+trafficLight.getID());
  mecaMind.setSensoryCodelets(sensoryCodelets);

  /*
  * After passing references to the codelets, we call the method
  * 'MecaMind.mountMecaMind()', which is responsible for wiring
  * the MecaMind altogether according to the reference architecture,
  * includingthe creation of memory objects and containers which glue
  * them together. This method is of pivotal importance and inside it
  * resides all the value from the reference architecture created - the
  * idea is that the user only has to create the codelets, put them
  * inside lists of differente types and call this method, which
  * transparently glue the codelets together accordingly to the MECA
  * reference architecture.
  */
  mecaMind.mountMecaMind();

  /*
   * Adding this Cognitive Manager, which lives in this specific junction,
   * to the list of Cognitive Managers of our controller.
   */
  mecaMinds.add(mecaMind);

 }
```

```
 /*
 * Starting the engine of each Cognitive Manager
 */
 for (MecaMind mecaMind : mecaMinds) {

  mecaMind.start();

 }

 /*
  * Starting the simulation thread
  */
 Thread simulationThread = new Thread(simulationRunnable);
 simulationThread.start();

 /*
  * Starting the CST visualization tool in order to observe and debug the
  * contents of the Cognitive Manager mind.
  */
 if(sumoGui){

  for (MecaMind mecaMind : mecaMinds) {

   List<Codelet> listOfCodelets = new ArrayList<>();
   listOfCodelets.addAll(mecaMind.getSensoryCodelets());

   MindViewer mv = new MindViewer(mecaMind,
   "MECA Mind Inspection - "+mecaMind.getId(),listOfCodelets);
   mv.setVisible(true);
  }
 }
}
```

When you run this new source code, you will be able to visualize to contents of the Cognitive Managers minds and the values coming from the sensors changing at each time step, as shown in figure 3.3. In the CodeRack inspection window, you will see all five sensory codelets always showing an activation level of zero, since they do not vary in activation in this app implementation.



```
MECA Mind Inspection - Mind of the TL 0 [18293]
  OccupancySensor - 0 [18294]
    OccupancySensor - 0 : [0.46535682819383223, 0.48
  PhaseTimeSensor - 0 [18296]
    PhaseTimeSensor - 0 : 1 [18297]
  VehicleNumberSensor - 0 [18298]
    VehicleNumberSensor - 0 : [16, 17, 17, 22, 22, 22, 2
  MeanSpeedSensor - 0 [18300]
    MeanSpeedSensor - 0 : [2.7162388493963916, 2.11
  CurrentPhaseSensor - 0 [18302]
    CurrentPhaseSensor - 0 : br.unicamp.jtraci.entities.L
```

```
MECA Mind Inspection - Mind of the TL 1 [18282]
  OccupancySensor - 1 [18283]
    OccupancySensor - 1 : [0.27624309392265195, 0.38
  PhaseTimeSensor - 1 [18285]
    PhaseTimeSensor - 1 : 1 [18286]
  VehicleNumberSensor - 1 [18287]
    VehicleNumberSensor - 1 : [5, 7, 7, 22, 22, 22, 22, 1
  MeanSpeedSensor - 1 [18289]
    MeanSpeedSensor - 1 : [9.786856226693281, 3.686
  CurrentPhaseSensor - 1 [18291]
    CurrentPhaseSensor - 1 : br.unicamp.jtraci.entities.L
```

```
Simulation started with time: 0.00
Warning: Teleporting vehicle '584'; collision with vehicle '198', lane='1:5_0', gap=-2.44, time=182.00 stage=move.
Warning: Vehicle '584' ends teleporting on edge '5:6', time 182.00.
'sumo/twinT/twinT.net.xml' loaded.    178    x:1.57, y:40.75    x:-349.43, y:-266.25
```

Figure 3.3: **Contents of the sensors inside Cognitive Managers minds**

In the next step, we are going to connect all System 1, and you will be able to watch the Cognitive Managers controlling the Traffic Lights autonomously in a reactive/motivational way.

3.3 Building System 1 codelets ↰

In this step, we are going to change the app source code in order to have Cognitive Managers controlling the traffic light junctions implementing a complete System 1, as shown in figure 3.4.

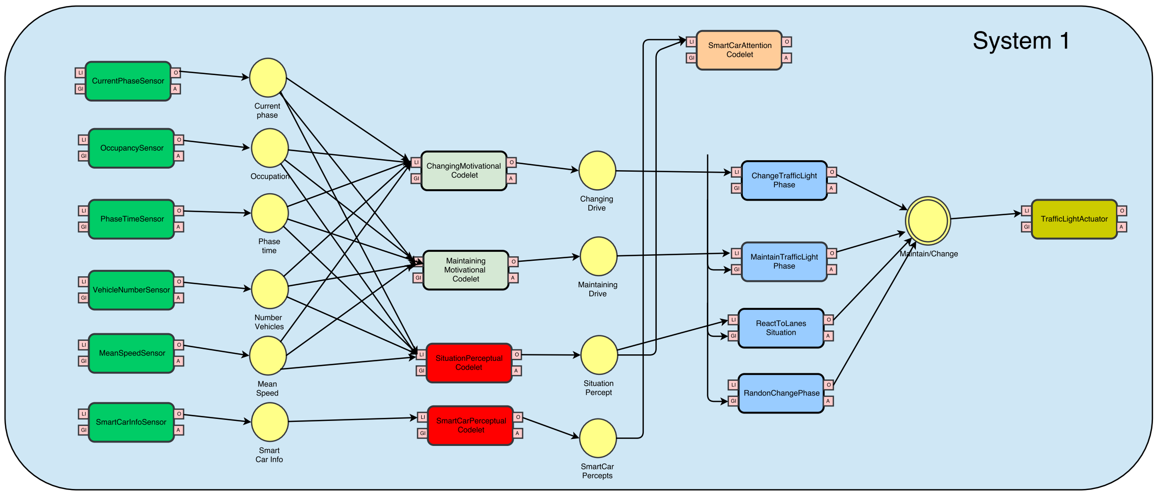


Figure 3.4: **System 1 model**

In your already changed project, append the following source code the content of the constructor method $br.unicamp.MECA_Demo.Main.Main(...)$:

```
...
...
CurrentPhaseSensor currentPhaseSensor = new CurrentPhaseSensor(
"CurrentPhaseSensor - " + trafficLight.getID(), trafficLight);
sensoryCodelets.add(currentPhaseSensor);

/* Lists that will hold the codelets ids. This is important
 * for the MECA mind mounting algorithm be able to glue the
 * codelets according to the reference architecture
 * */
ArrayList<String> sensoryCodeletsIds = new ArrayList<>();
ArrayList<String> perceptualCodeletsIds = new ArrayList<>();
ArrayList<String> changingMotivationalCodeletIds = new ArrayList<>();
ArrayList<String> maintainMotivationalCodeletsIds = new ArrayList<>();

/*
 * Holding sensory codelets ids in order to be able to mount
 * the MECA mind later
 */
sensoryCodeletsIds.add(occupancySensor.getId());
sensoryCodeletsIds.add(phaseTimeSensor.getId());
sensoryCodeletsIds.add(vehicleNumberSensor.getId());
sensoryCodeletsIds.add(meanSpeedSensor.getId());
sensoryCodeletsIds.add(currentPhaseSensor.getId());

/* Perceptual codelets we are about to create for this Cognitve Manager
 * in this junction
 */
List<PerceptualCodelet> perceptualCodelets = new ArrayList<>();
/* Motivational codelets we are about to create for this Cognitve Manager
 * in this junction
 */
List<MotivationalCodelet> motivationalCodelets = new ArrayList<>();
/* Random Behavioral codelets we are about to create for this Cognitve
 * Manager in this junction
 */
List<RandomBehavioralCodelet> randomBehavioralCodelets = new ArrayList<>();
/* Reactive Behavioral codelets we are about to create for this Cognitve
 * Manager in this junction
 */
List<ReactiveBehavioralCodelet> reactiveBehavioralCodelets = new ArrayList<>();
/* Motivational Behavioral codelets we are about to create for this Cognitve
 * Manager  in this junction
 */
List<MotivationalBehavioralCodelet> motivationalBehavioralCodelets
= new ArrayList<>();
/* Motor codelets we are about to create for this Cognitve Manager
* in this junction
*/
List<MotorCodelet> motorCodelets = new ArrayList<>();
```

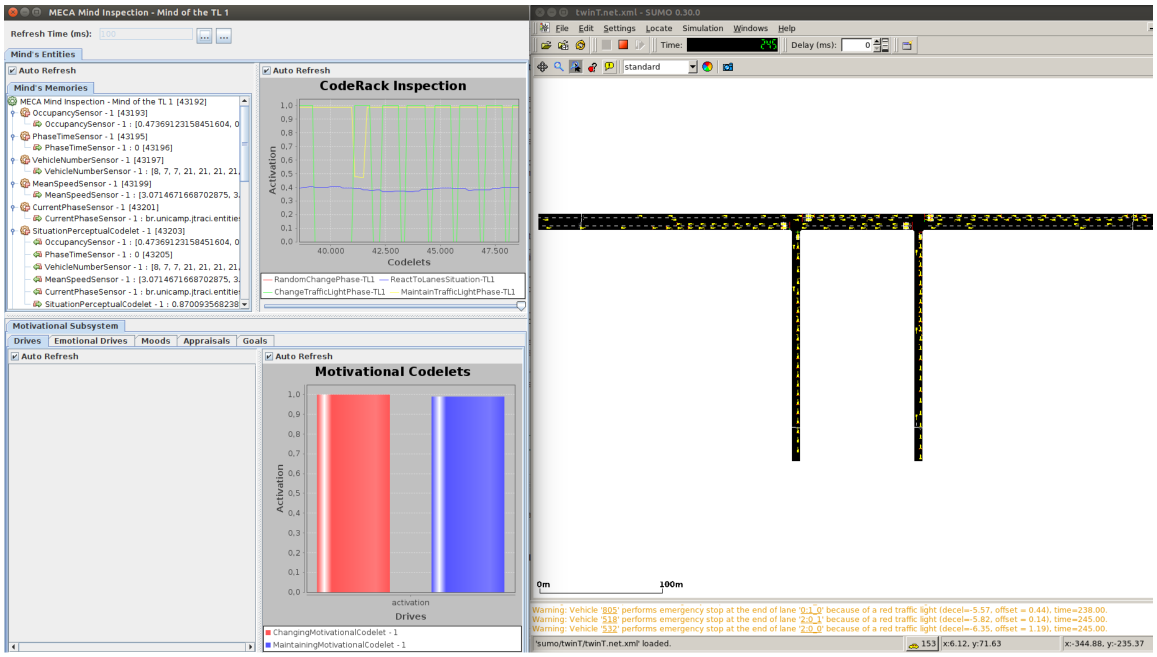
```
/* Get the possible phases the controller can choose for this junction*/
ArrayList<Phase> phases = simulationRunnable.getPhases(trafficLight);
/* Create the motor codelet that will implement the chosen phase by our
 * Cognitive Manager
 */
TrafficLightActuator trafficLightActuator = new TrafficLightActuator(
"TrafficLightActuator" + trafficLight.getID(), trafficLight, phases);
/* Add the motor codelet to the list to be put inside MECA mind*/
motorCodelets.add(trafficLightActuator);

/*
 * Next step is to create the motivational codelets.
 * This codelets must receive the ids of the sensory codelets,
 * in order to be glued to them, receiving  their inputs.
 */
ChangingMotivationalCodelet changingMotivationalCodelet;
MaintainingMotivationalCodelet maintainingMotivationalCodelet;

try {
 changingMotivationalCodelet = new ChangingMotivationalCodelet(
 "ChangingMotivationalCodelet - " + trafficLight.getID(), 0, 0.5,
 0.3, sensoryCodeletsIds, new HashMap<String, Double>());
 maintainingMotivationalCodelet = new MaintainingMotivationalCodelet(
 "MaintainingMotivationalCodelet - " + trafficLight.getID(), 0, 0.49,
 0.9677, sensoryCodeletsIds, new HashMap<String, Double>());

 changingMotivationalCodeletIds.add(changingMotivationalCodelet.getId());
 maintainMotivationalCodeletsIds.add(maintainingMotivationalCodelet.getId());

 motivationalCodelets.add(changingMotivationalCodelet);
 motivationalCodelets.add(maintainingMotivationalCodelet);

} catch (CodeletActivationBoundsException e) {
 e.printStackTrace();
}

/*
 * Then, we create the Situation Perceptual codelet.
 * This codelet must receive the ids of the sensory codelets,
 * in order to be glued to them, receiving  their inputs.
 */
SituationPerceptualCodelet situationPerceptualCodelet = new
SituationPerceptualCodelet("SituationPerceptualCodelet - "
+ trafficLight.getID(), sensoryCodeletsIds);
perceptualCodeletsIds.add(situationPerceptualCodelet.getId());
perceptualCodelets.add(situationPerceptualCodelet);
```

```
/*
 * One of the last steps is to create the behavioral codelets,
 * all three random, reactive and motivational types.
 * They receive the ids of the perceptual codelets and
 * motor codelets, in order to be glued to them, according
 * to the reference architecture.
 * For now, we will pass an empty string "" as the id of the
 * Soar Codelet, as we will not have any System 2 structure.
 */
RandomChangePhase openRandomPhase = new RandomChangePhase(
"RandomChangePhase-TL" + trafficLight.getID(),
trafficLightActuator.getId(),"");
randomBehavioralCodelets.add(openRandomPhase);

ReactToLanesSituation reactToLanesSituation = new ReactToLanesSituation(
"ReactToLanesSituation-TL" + trafficLight.getID(), perceptualCodeletsIds,
trafficLightActuator.getId(),"");
reactiveBehavioralCodelets.add(reactToLanesSituation);

ChangeTrafficLightPhase changeTrafficLightPhase = new ChangeTrafficLightPhase(
"ChangeTrafficLightPhase-TL" + trafficLight.getID(),
trafficLightActuator.getId(), changingMotivationalCodeletIds,"");
motivationalBehavioralCodelets.add(changeTrafficLightPhase);

MaintainTrafficLightPhase maintainTrafficLightPhase = new
MaintainTrafficLightPhase("MaintainTrafficLightPhase-TL" +
trafficLight.getID(), trafficLightActuator.getId(),
maintainMotivationalCodeletsIds,"");
motivationalBehavioralCodelets.add(maintainTrafficLightPhase);

/*
 * Then, we create the MecaMind for this Cognitive Manage in this junction,
 * and pass references to the sensory codelets in a list.
 */

MecaMind mecaMind = new MecaMind("Mind of the TL "+trafficLight.getID());
mecaMind.setSensoryCodelets(sensoryCodelets);

/*
 * Inserting the System 1 codelets inside MECA mind
 */
mecaMind.setPerceptualCodelets(perceptualCodelets);
mecaMind.setMotivationalCodelets(motivationalCodelets);
mecaMind.setRandomBehavioralCodelets(randomBehavioralCodelets);
mecaMind.setReactiveBehavioralCodelets(reactiveBehavioralCodelets);
mecaMind.setMotivationalBehavioralCodelets(motivationalBehavioralCodelets);
mecaMind.setMotorCodelets(motorCodelets);
...
...
```

In another section of the method, insert the following changes:

```java
...
for (MecaMind mecaMind : mecaMinds) {

 List<Codelet> listOfCodelets = new ArrayList<>();

 /*
  * Instead of inserting the sensory codelets in the
  * CST visualization tool, let's insert the behaviroal
  * codelets, which activation has a pivotal role.
  */
 listOfCodelets.addAll(mecaMind.getRandomBehavioralCodelets());
 listOfCodelets.addAll(mecaMind.getReactiveBehavioralCodelets());
 listOfCodelets.addAll(mecaMind.getMotivationalBehavioralCodelets());

 MindViewer mv = new MindViewer(mecaMind, "MECA Mind Inspection - "
 + mecaMind.getId(), listOfCodelets);
 mv.setVisible(true);
}
...
```

When you run the modified code, you will be able to see the SUMO simulation with multiple Cognitive Managers autonomously controlling the traffic light junctions in a reactive way! This means the junctions will be controlled according to the traffic situation in real time, as shown in figure 3.5. Take sometime again to inspect the codelets source code and the CST visualization tool in order to acknowledge that the Cognitive Managers are controlling the traffic lights junctions as mentioned. You will be able to see the codelets activations fighting for the memory container winner-takes-all dynamic subsumption mechanism.



```
MECA Mind Inspection - Mind of the TL 1 [43192]
  OccupancySensor - 1 [43193]
    OccupancySensor - 1 : [0.47369123158451604, 0
  PhaseTimeSensor - 1 [43195]
    PhaseTimeSensor - 1 : 0 [43196]
  VehicleNumberSensor - 1 [43197]
    VehicleNumberSensor - 1 : [8, 7, 7, 21, 21, 21, 21,
  MeanSpeedSensor - 1 [43199]
    MeanSpeedSensor - 1 : [3.0714671668702875, 3.
  CurrentPhaseSensor - 1 [43201]
    CurrentPhaseSensor - 1 : br.unicamp.jtraci.entitie
  SituationPerceptualCodelet - 1 [43203]
    OccupancySensor - 1 : [0.47369123158451604, 0
    PhaseTimeSensor - 1 : 0 [43205]
    VehicleNumberSensor - 1 : [8, 7, 7, 21, 21, 21, 21,
    MeanSpeedSensor - 1 : [3.0714671668702875, 3.
    CurrentPhaseSensor - 1 : br.unicamp.jtraci.entitie
    SituationPerceptualCodelet - 1 : 0.870093568238
```

```
Warning: Vehicle '805' performs emergency stop at the end of lane '0:1_0' because of a red traffic light (decel=-5.57, offset = 0.44), time=238.00.
Warning: Vehicle '518' performs emergency stop at the end of lane '2:0_1' because of a red traffic light (decel=-5.82, offset = 0.14), time=245.00.
Warning: Vehicle '532' performs emergency stop at the end of lane '2:0_0' because of a red traffic light (decel=-6.35, offset = 1.19), time=245.00.
'sumo/twinT/twinT.net.xml' loaded.   153   x:6.12, y:71.63   x:-344.88, y:-235.37
```

Figure 3.5: **System 1 running**

3.4 Building System 2 codelets ↰

In this final step, we will complete the app source code in order to have a System 2 capable of deliberately giving a free way to special vehicles we are going to call "smart vehicles". This simulation can be understood, for instance, as a so called “blue corridor”, giving priority to emergency vehicle. We will have to add System 2 Codelets in the implementation in order to complete our Cognitive Manager with structures as shown in figure 3.6.

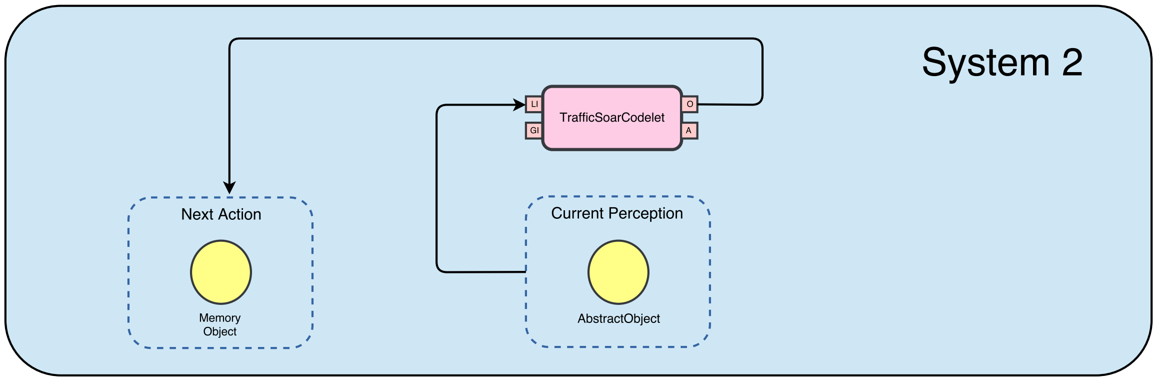


Figure 3.6: **System 2 model**

First, let’s change the code of the "static main" method to the following, in order to be able to pass as arguments a list of smart car ids, which will be considered by the System 2:

```java
public static void main(String[] args) {

System.out.println("Usage: AppSumoECA <P1> <P2> <P3> <P4>
<P5>*<P6>* ... <Pn>*");
System.out.println("<P1> = SUMO Server IP");
System.out.println("<P2> = SUMO Server port");
System.out.println("<P3> = Start SUMO (-R)/ SUMO already running (-L)");
System.out.println("<P4> = SUMO Gui / Meca Viewer (-S) / None (-N)");
System.out.println("<P5>*<P6>*...<Pn>* = List of Smart Cars IDs (optional)");

if (args.length < 4) {
 return;
}

String ipServidor = args[0];
int port = Integer.valueOf(args[1]);
ArrayList<String> smartCarsIDs = null;

boolean sumoGui = false;
boolean sumoConnect = false;

if(args[2].equals("-R")){
 sumoConnect = true;
}

if(args[3].equals("-S")){
 sumoGui = true;
}

if(args.length > 4){

 smartCarsIDs = new ArrayList<>();

 for(int i = 4; i < args.length; i++){
  smartCarsIDs.add(args[i]);
 }

}

Main app = new Main(ipServidor, port, sumoGui, sumoConnect,smartCarsIDs);

}
```

You will notice that a small change in the signature of the method *br.unicamp.MECA*$_{Demo}$*.Main.Main*(...) will also be necessary:

```
...
...

public Main(String ipServidor, int port, boolean sumoGui, boolean sumoConnect,
ArrayList<String> smartCarsIDs) {

...
...
```

Next, append the following code to the method *br.unicamp.MECA$_{D}$emo.Main.Main*(...), in the appropriate position (from now on, the explanation for the changes will be found as in line comments in the source code):

```
...
...
ArrayList<InductionLoop> inductionLoops = new ArrayList<>();

/*
 * Hold the incoming lanes ids to be used by the smart cars
 * sensors
 *
 */
ArrayList<String> incomingLaneIds = new ArrayList<>();

/*
 * In order to get the reference for incoming lanes and induction loops in this
 * specific junction, we will loop through structures called "Links" in the TraCI
 * protocol, which are positioned inside "Controlled Links", which are positioned
 * inside Traffic Lights (all objects in JTraci, implementing the TraCI protocol)
 */
ControlledLinks controlledLinks = trafficLight.getControlledLinks();
br.unicamp.jtraci.entities.Link[][] links = controlledLinks.getLinks();
for (int i = 0; i < links.length; i++) {
 for (int j = 0; j < links[i].length; j++) {

   br.unicamp.jtraci.entities.Link link = links[i][j];
   Lane incomingLane = link.getIncomingLane();
   incomingLanes.add(incomingLane);
   inductionLoops.add(simulationRunnable.getInductionLoop(incomingLane));
   incomingLaneIds.add(incomingLane.getID());// add this line here
  }
}
...
...
```

In another position, make the following changes:

```
...
...
CurrentPhaseSensor currentPhaseSensor = new CurrentPhaseSensor(
"CurrentPhaseSensor - " + trafficLight.getID(), trafficLight);
sensoryCodelets.add(currentPhaseSensor);

/*
 * Create the smart car sensory codelet that will read the smart car
 * values when it approaches the junction
 */
ArrayList<String> smartCarSensoryCodeletsIds = new ArrayList<>();
if(smartCarsIDs!=null && smartCarsIDs.size() > 0){

 SmartCarInfoSensor smartCarInfoSensor = new SmartCarInfoSensor(
 "SmartCarInfoSensor - " + trafficLight.getID(), smartCarsIDs ,
 incomingLaneIds,trafficLight);
 smartCarSensoryCodeletsIds.add(smartCarInfoSensor.getId());
 sensoryCodelets.add(smartCarInfoSensor);

}
...
...
```

Next step, some other changes:

```
...
...
perceptualCodelets.add(situationPerceptualCodelet);

/*
 * Create the Smart car perceptual codelet, which will process the
 * smart cars information and create high level abstractions
 * that will be useful for the controller decisions.
 */
ArrayList<String> smartCarPerceptualCodeletsIds = new ArrayList<>();
SmartCarPerceptualCodelet smartCarPerceptualCodelet = new
SmartCarPerceptualCodelet("SmartCarPerceptualCodelet - "
+ trafficLight.getID(), smartCarSensoryCodeletsIds);
smartCarPerceptualCodeletsIds.add(smartCarPerceptualCodelet.getId());
perceptualCodelets.add(smartCarPerceptualCodelet);

/*
 * Create the SOAR codelet, responsible for deliberative decisions
 * in giving priority to smart cars
 */
String soarRulesPath = "soar_rules/soarRules.soar";
TrafficSoarCodelet trafficSoar = new TrafficSoarCodelet(
"TrafficSoarCodelet - " + trafficLight.getID(),
"br.unicamp.MECA_Demo.util.SOAR_commands",
"TrafficSoarCodelet - " + trafficLight.getID(),
new File (soarRulesPath), false);

/*
 * Create the Smart car attention codelet, which will take the
 * smart cars information to System 2
 */
smartCarPerceptualCodeletsIds.add(situationPerceptualCodelet.getId());
SmartCarAttentionCodelet smartAttention = new SmartCarAttentionCodelet(
"SmartCarAttentionCodelet - " + trafficLight.getID(),
smartCarPerceptualCodeletsIds,trafficLight, phases);
...
...
```

Remember to pass the Traffic SOAR codelet id to the behavioral codelets

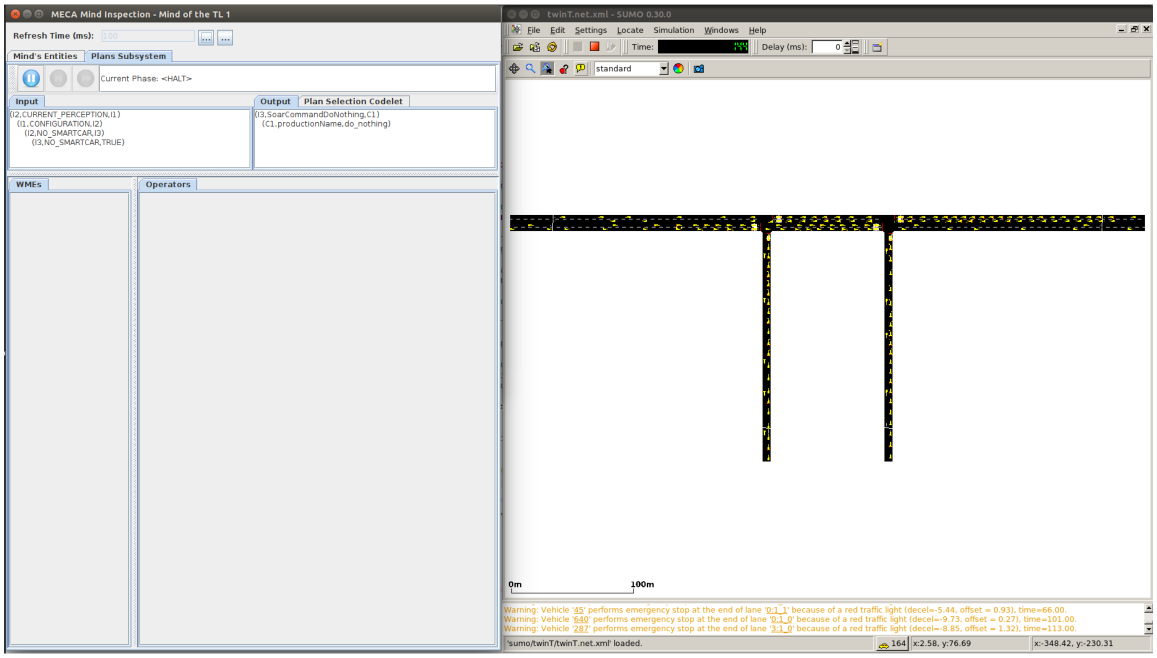
```
...
...
/*
 * One of the last steps is to create the behavioral codelets,
 * all three random, reactive and motivational types.
 * They receive the ids of the perceptual codelets and
 * motor codelets, in order to be glued to them, according
 * to the reference architecture.
 * These codelets must receive the id of the soar codelet as well.
 */
RandomChangePhase openRandomPhase = new RandomChangePhase(
"RandomChangePhase-TL" + trafficLight.getID(),
trafficLightActuator.getId(),trafficSoar.getId());
randomBehavioralCodelets.add(openRandomPhase);

ReactToLanesSituation reactToLanesSituation = new ReactToLanesSituation(
"ReactToLanesSituation-TL" + trafficLight.getID(), perceptualCodeletsIds,
trafficLightActuator.getId(),trafficSoar.getId());
reactiveBehavioralCodelets.add(reactToLanesSituation);

ChangeTrafficLightPhase changeTrafficLightPhase = new ChangeTrafficLightPhase(
"ChangeTrafficLightPhase-TL" + trafficLight.getID(),
trafficLightActuator.getId(), changingMotivationalCodeletIds,
trafficSoar.getId());
motivationalBehavioralCodelets.add(changeTrafficLightPhase);

MaintainTrafficLightPhase maintainTrafficLightPhase = new
MaintainTrafficLightPhase("MaintainTrafficLightPhase-TL" +
trafficLight.getID(), trafficLightActuator.getId(),
maintainMotivationalCodeletsIds,trafficSoar.getId());
motivationalBehavioralCodelets.add(maintainTrafficLightPhase);
...
...
```

Finally, some other small changes:

```
...
...
mecaMind.setMotorCodelets(motorCodelets);

/*
 * Put the System 2 codelets in MECA mind before mouting
 */
mecaMind.setAttentionCodeletSystem1(smartAttention);
mecaMind.setSoarCodelet(trafficSoar);
...
...
```

```
...
...
listOfCodelets.addAll(mecaMind.getMotivationalBehavioralCodelets());

/*
 * Add the SOAR codelet to the CST visualization
 * tool
 */
listOfCodelets.add(mecaMind.getSoarCodelet());
...
...
```

When you run the app after this final step, you will be able to watch the simulation running with both System 1 and System 2. In the CST visualization tool, you will be able to inspect the Plan Subsystem, implemented by System 2 with the Traffic SOAR codelet, as shown in figure 3.7. We have painted the smart car in red so it is easier for you to watch it flow - try to change the delay of the SUMO user interface to 1000 ms so you can see it clearly.


MECA Mind Inspection - Mind of the TL 1
Refresh Time (ms):
Mind's Entities
Plans Subsystem
Input
Output
Plan Selection Codelet
WMEs
Operators
twinT.net.xml - SUMO 0.30.0
File Edit Settings Locate Simulation Windows Help
Time:
0m
100m

Refresh Time (ms): 100

Current Phase: <HALT>

```
(I2,CURRENT_PERCEPTION,I1)
  (I1,CONFIGURATION,I2)
    (I2,NO_SMARTCAR,I3)
      (I3,NO_SMARTCAR,TRUE)
```

```
(I3,SoarCommandDoNothing,C1)
  (C1,productionName,do_nothing)
```

Delay (ms): 0

standard

```
Warning: Vehicle '45' performs emergency stop at the end of lane '0:1_1' because of a red traffic light (decel=-5.44, offset = 0.93), time=66.00.
Warning: Vehicle '640' performs emergency stop at the end of lane '0:1_0' because of a red traffic light (decel=-9.73, offset = 0.27), time=101.00.
Warning: Vehicle '287' performs emergency stop at the end of lane '3:1_0' because of a red traffic light (decel=-8.85, offset = 1.32), time=113.00.
'sumo/twinT/twinT.net.xml' loaded.    164    x:2.58, y:76.69    x:-348.42, y:-230.31
```

Figure 3.7: **System 2 running**

3.5 Conclusion ↰

Congratulations, you have made it! You know now how to build an application implementing Cognitive Managers using MECA.

What about trying to create you own sample application now?

4. References ↰

(Barsalou 2010) Barsalou, L. W. “Grounded cognition: Past, present, and future”. In: *Topics in cognitive science* 2.4 (2010), pp. 716–724 (cit. on p. 5).

(Gärdenfors 2014) Gärdenfors, P. *The geometry of meaning: Semantics based on conceptual spaces*. MIT Press, 2014 (cit. on p. 5).

(R. R. Gudwin 2016) Gudwin, R. R. *Urban Traffic Simulation with SUMO - A Roadmap for the Beginners*. Tech. rep. D3. Campinas-SP, Brazil: University of Campinas, 08/2016 (cit. on p. 6).

(R. R. Gudwin 2017a) Gudwin, R. R. *The MECA Cognitive Architecture*. Tech. rep. D5. Campinas-SP, Brazil: University of Campinas, 01/2017 (cit. on pp. 5, 6).

(R. R. Gudwin 2017b) Gudwin, R. R. *The MECA Cognitive Architecture*. Tech. rep. D7. Campinas-SP, Brazil: University of Campinas, 06/2017 (cit. on pp. 5, 13).

(R. Gudwin et al. 2017) Gudwin, R. et al. “The Multipurpose Enhanced Cognitive Architecture (MECA)”. In: *Biologically Inspired Cognitive Architectures* 22.Supplement C (2017), pp. 20–34. URL: http://www.sciencedirect.com/science/article/pii/S2212683X17301068 (cit. on pp. 5, 6).

(Kotseruba et al. 2016) Kotseruba, I., Gonzalez, O. J. A., and Tsotsos, J. K. “A Review of 40 Years of Cognitive Architecture Research: Focus on Perception, Attention, Learning and Applications”. In: *arXiv preprint arXiv:1610.08602* (2016) (cit. on p. 5).

(Krajzewicz et al. 2012) Krajzewicz, D., Erdmann, J., Behrisch, M., and Bieker, L. “Recent Development and Applications of SUMO - Simulation of Urban MObility”. In: *International Journal On Advances in Systems and Measurements* 5.3&4 (12/2012), pp. 128–138 (cit. on p. 5).

(Laird 2012) Laird, J. *The Soar cognitive architecture*. MIT Press, 2012 (cit. on p. 5).

(Nakashima et al. 1998) Nakashima, H. and Noda, I. “Dynamic Subsumption Architecture for Programming Intelligent Agents”. In: *Proceedings of the 3rd International Confer-*

ence on Multi Agent Systems - ICMAS. IEEE Computer Society. 1998, pp. 190–197 (cit. on p. 5).

(Osman 2004) Osman, M. “An evaluation of dual-process theories of reasoning”. In: *Psychonomic bulletin & review* 11.6 (2004), pp. 988–1010 (cit. on p. 5).

(Paraense et al. 2016) Paraense, A. L. O., Raizer, K., Paula, S. M. de, Rohmer, E., and Gudwin, R. R. “The Cognitive Systems Toolkit and the CST Reference Cognitive Architecture”. In: *Biologically Inspired Cognitive Architectures* 17 (2016), pp. 32–48 (cit. on p. 5).