

1. What is NormLab?

NormLab is a **framework** to support research on norm synthesis for Multi-Agent Systems.

NormLab allows to:

1. **Perform MAS simulations.** It incorporates two different MAS simulators: a traffic simulator, and an on-line community simulator.
2. **Perform on-line norm synthesis on MAS simulations.** *NormLab* incorporates different *state-of-the-art* on-line norm synthesis strategies that can be tested on MAS simulations.
3. **Develop and test custom norm synthesis strategies.** NormLab allows to develop custom on-line norm synthesis strategies to be tested on the MAS simulations.

Tutorial outline

What are the contents of this tutorial?

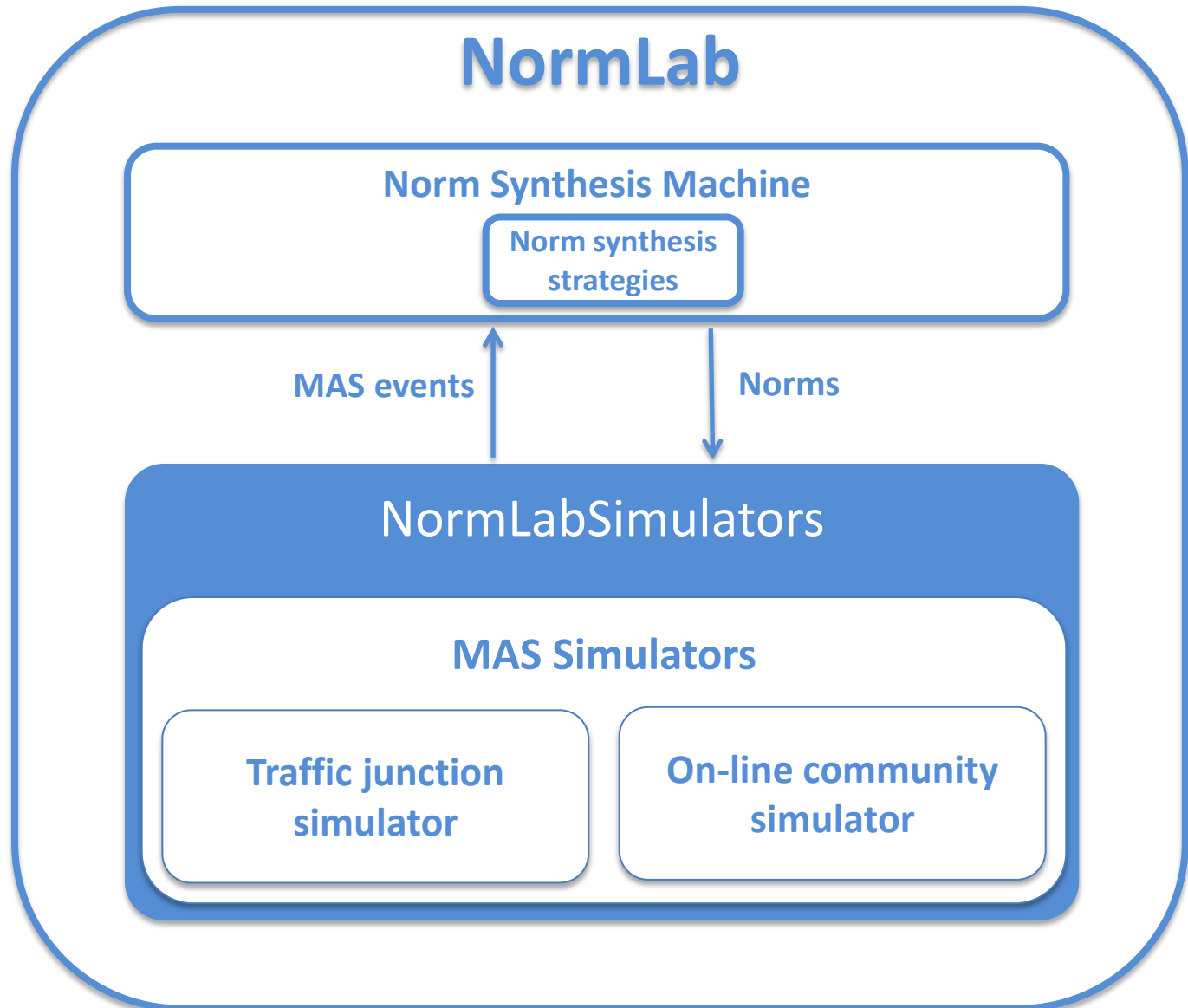
1. An **introduction** to NormLab
 1. The NormLab architecture.
 2. The Norm Synthesis Machine.
 3. The NormLab simulators.
2. **Configuration** of the working environment
 1. NormLab download.
 2. NormLab installation.
3. NormLab **execution**:
 1. Execution examples.
 2. Guided development of different norm synthesis strategies.

Tutorial outline

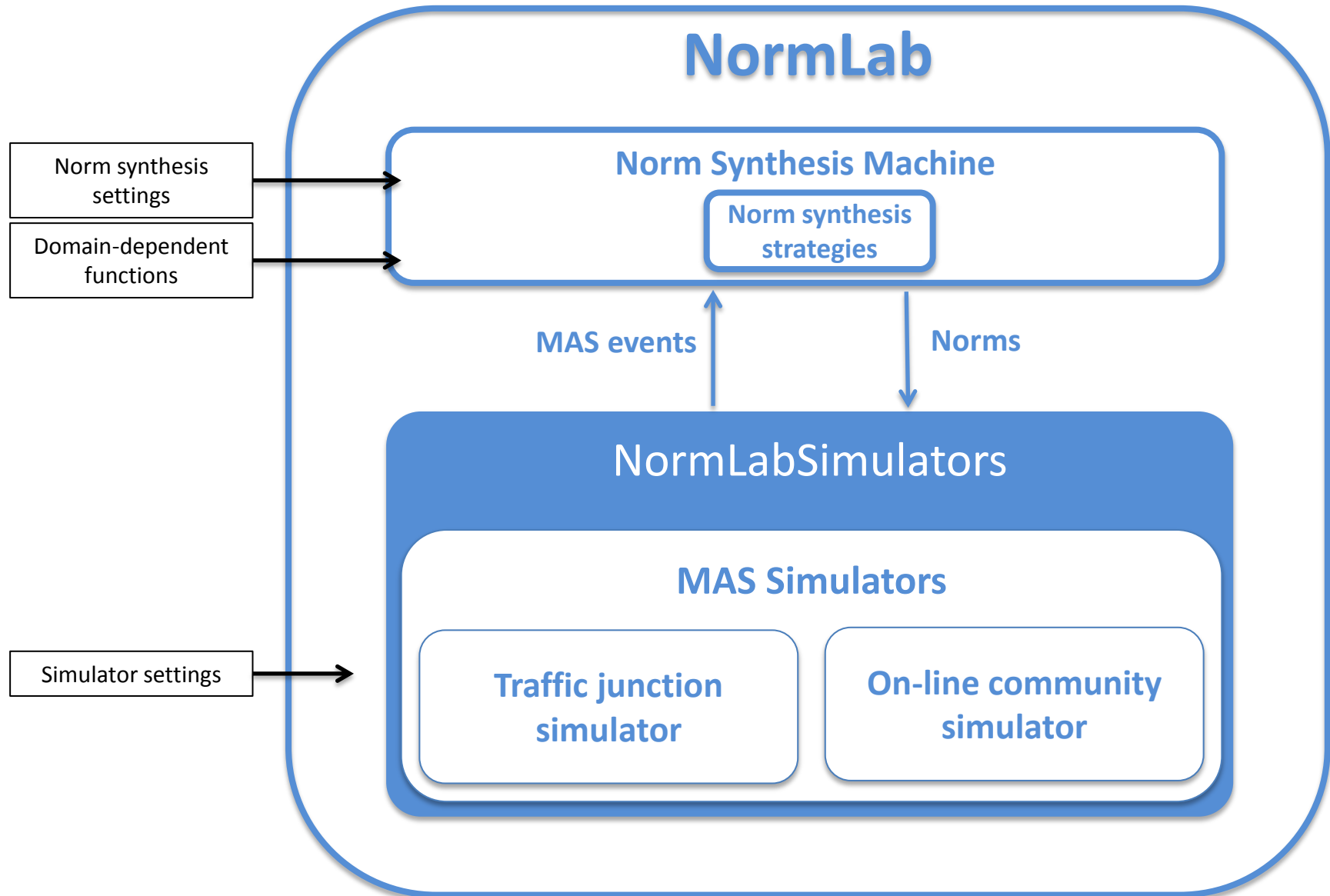
What are the contents of this tutorial?

1. An **introduction** to NormLab
 1. The NormLab architecture.
 2. The Norm Synthesis Machine.
 3. The NormLab simulators.
2. **Configuration** of the working environment
 1. NormLab download.
 2. NormLab installation.
3. NormLab **execution**:
 1. Execution examples.
 2. Guided development of different norm synthesis strategies.

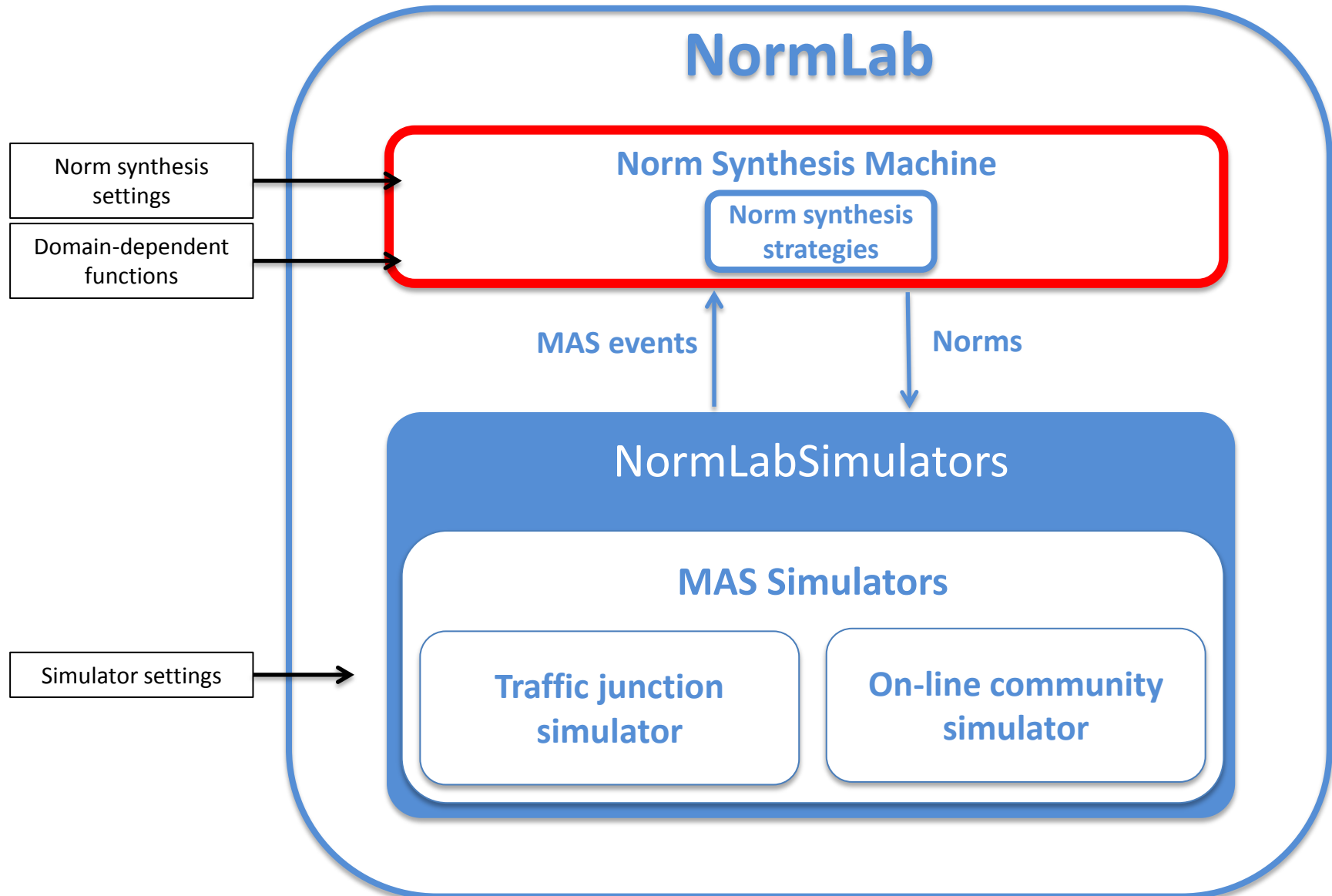
2. NormLab architecture



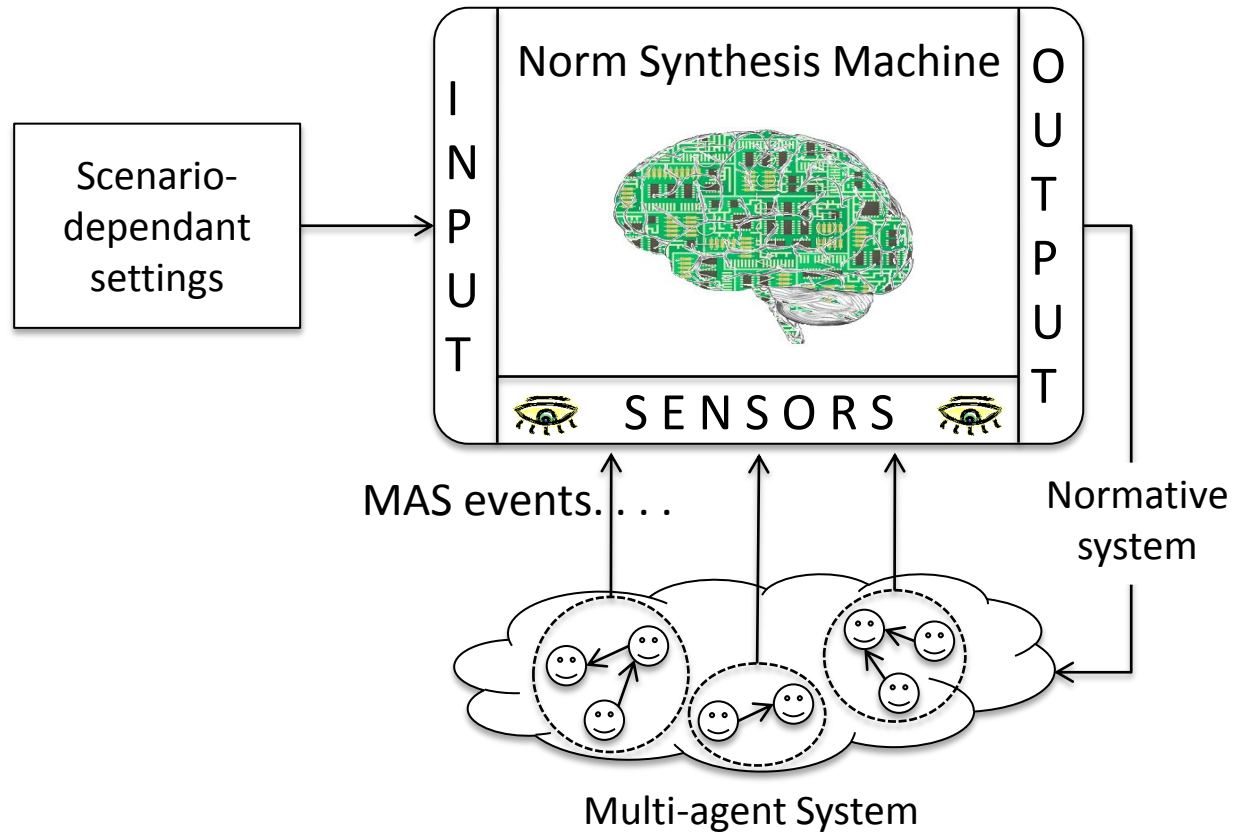
2. NormLab architecture



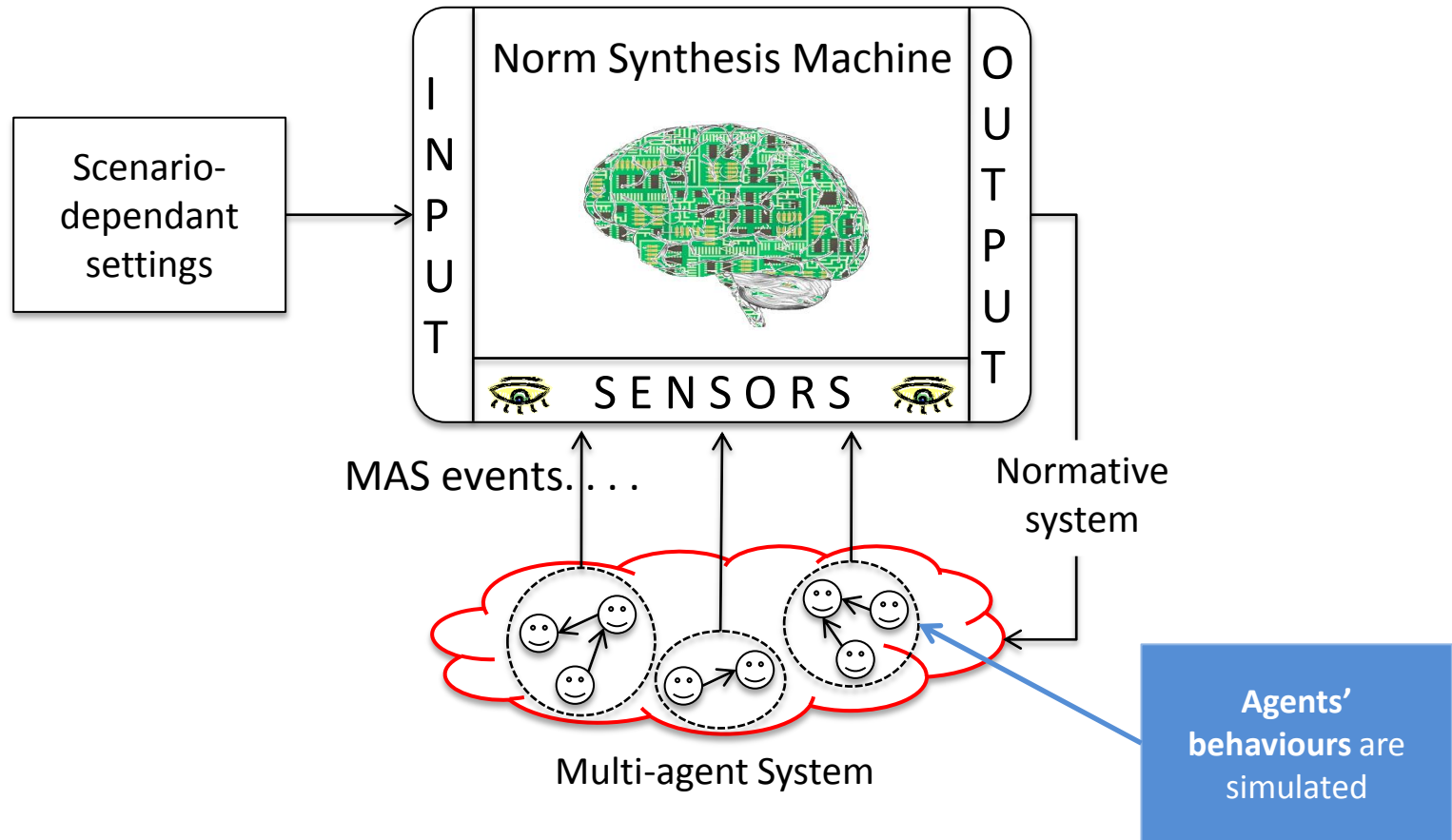
2. NormLab architecture



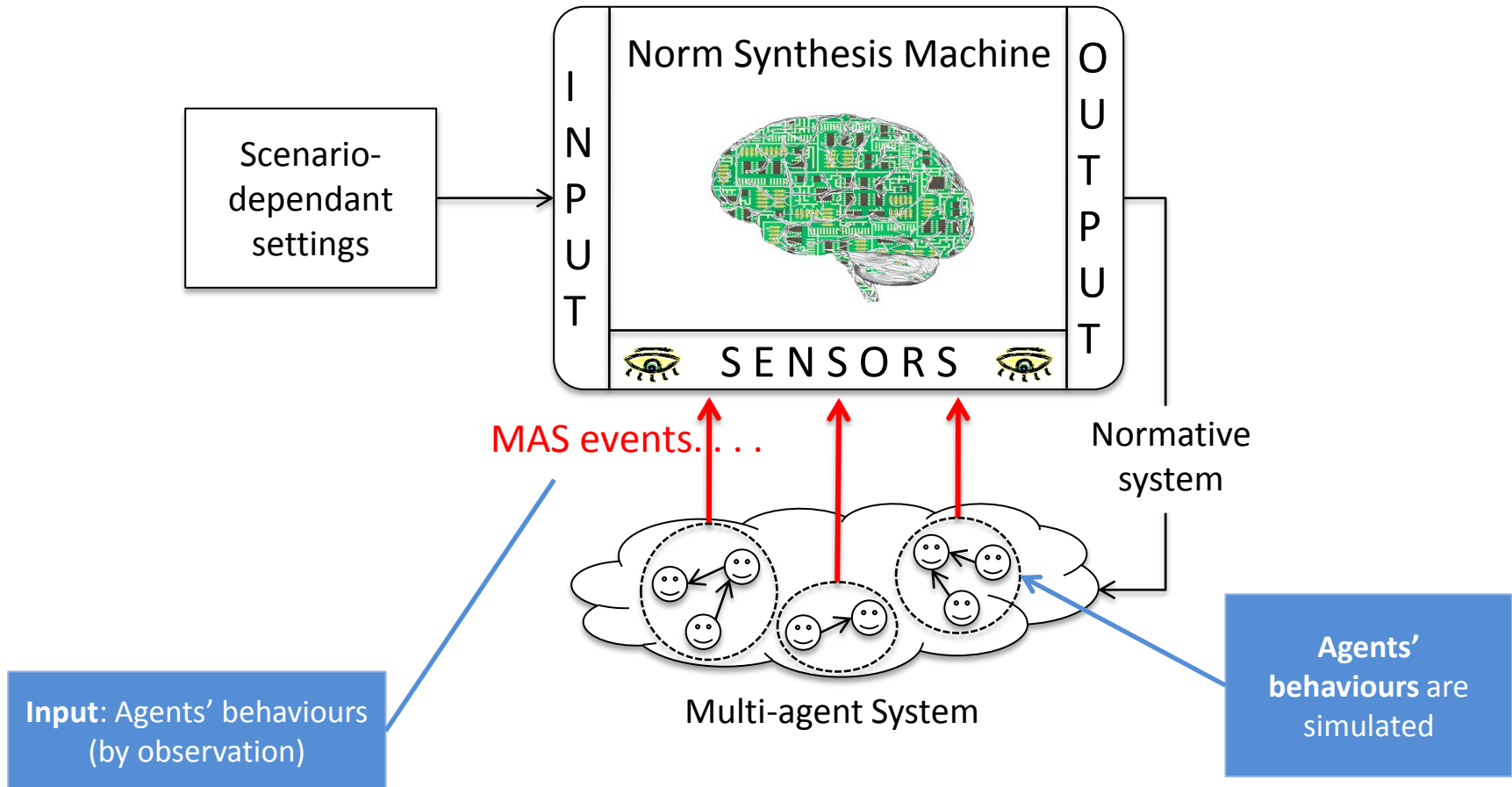
3. The Norm Synthesis Machine



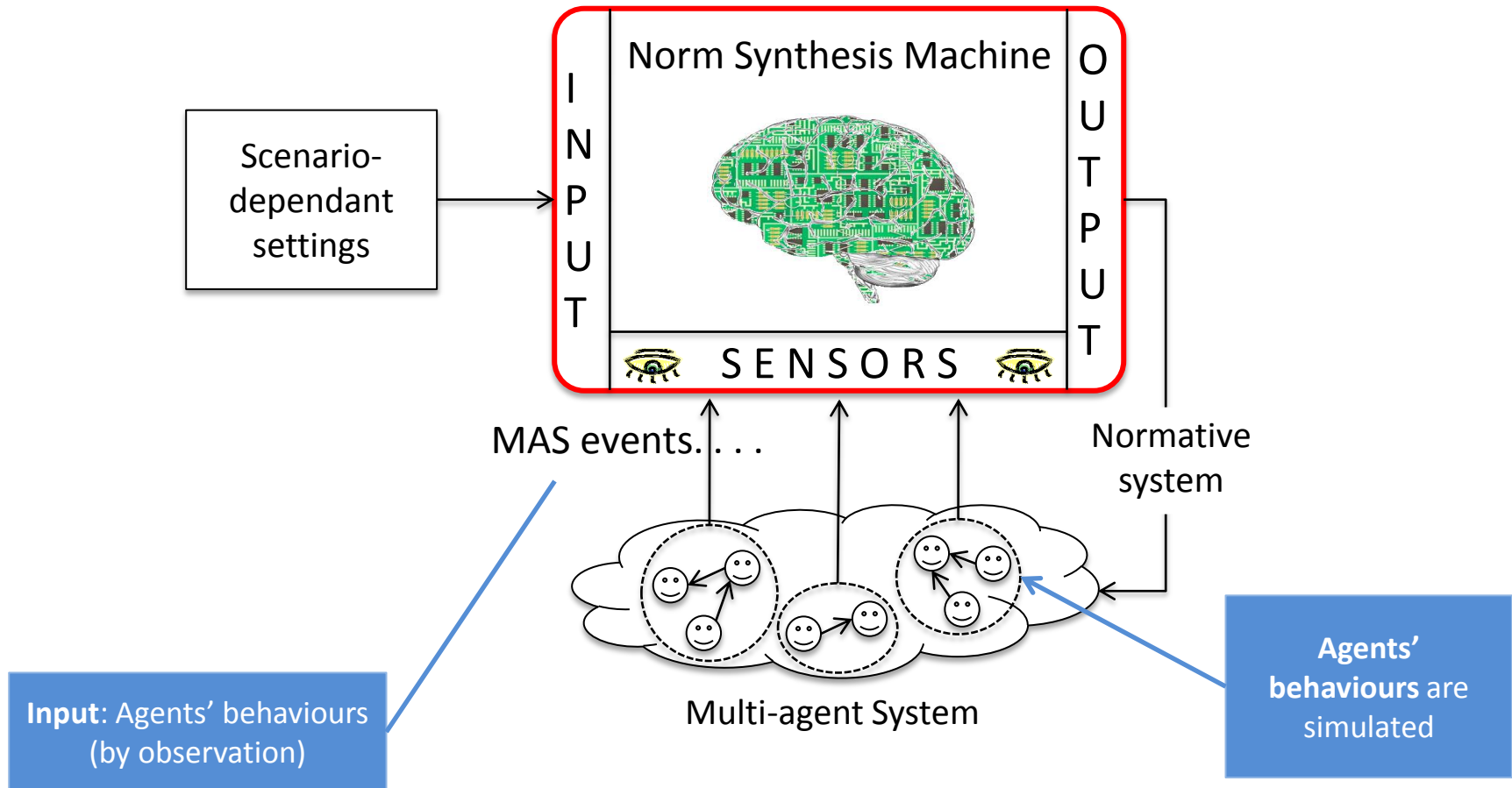
3. The Norm Synthesis Machine



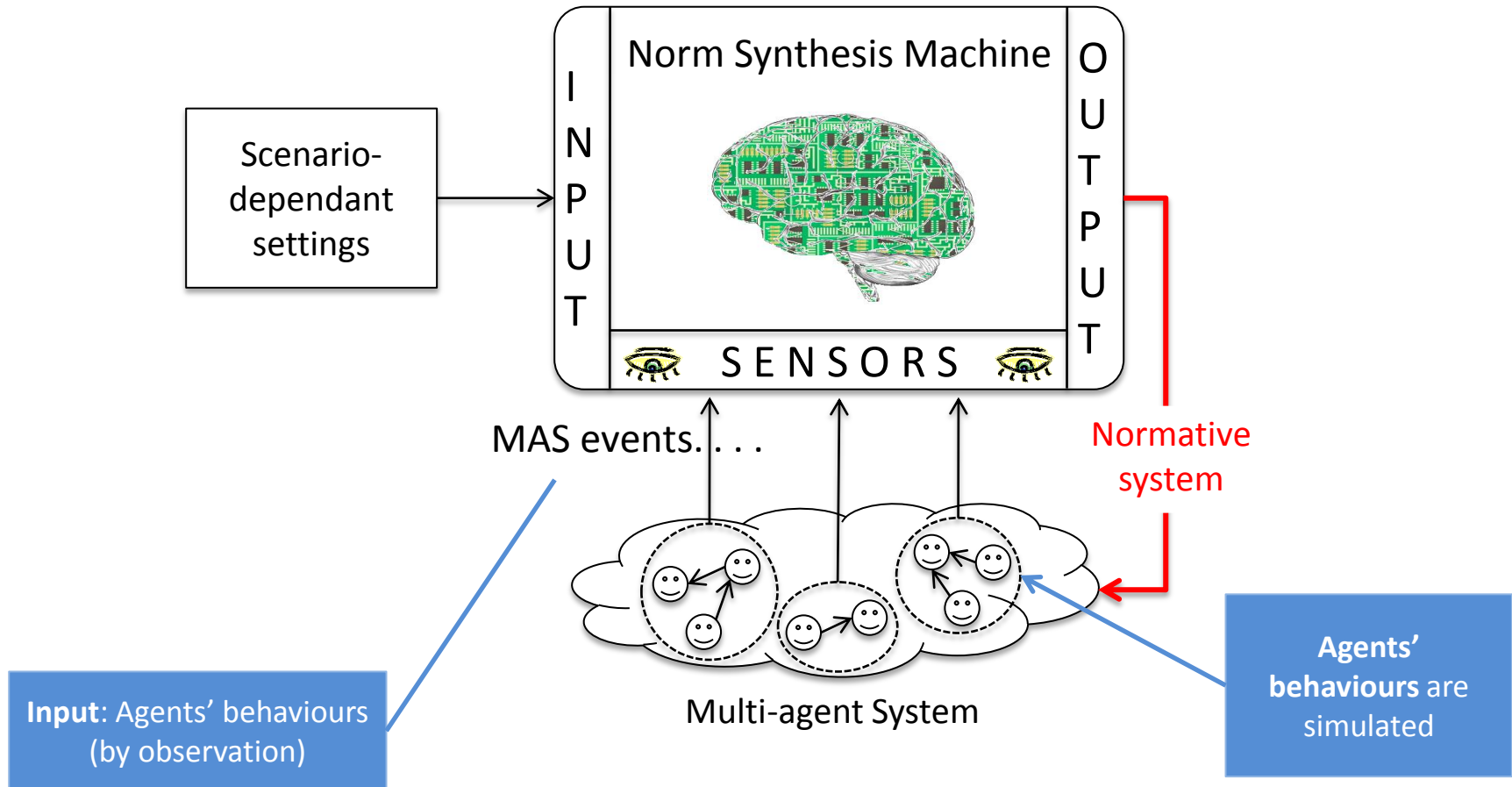
3. The Norm Synthesis Machine



3. The Norm Synthesis Machine

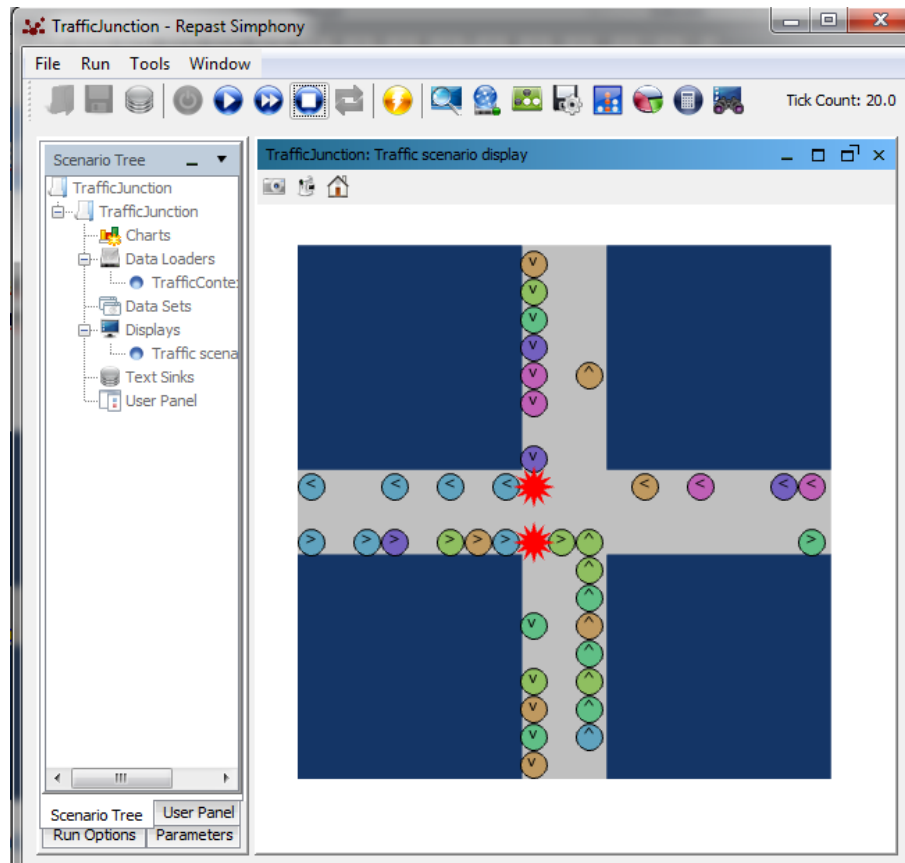


3. The Norm Synthesis Machine



4. The traffic simulator

- Based on Repast Symphony 2.1
- **Agents** are cars, and **conflicts** are collisions among cars.
- **The goal** is to synthesise normative systems that **avoid collisions** between cars.



Tutorial outline

What are the contents of this tutorial?

1. An **introduction** to NormLab
 1. The NormLab architecture.
 2. The Norm Synthesis Machine.
 3. The NormLab simulators.
2. **Configuration** of the working environment
 1. NormLab download.
 2. NormLab installation.
3. NormLab **execution**:
 1. Execution examples.
 2. Guided development of different norm synthesis strategies.

5. NormLab download

NormLab is **multi-platform**. You can use it either in *Windows*, *MacOS* or *Linux*!

Requirements

- **Java JDK 1.6** or greater <http://www.java.com>
- **Eclipse IDE** (just for Linux users) <http://www.eclipse.org/downloads>
- **Repast Symphony 2.1** <http://repast.sourceforge.net>

Downloads

To use *NormLab* you need to download:

- **NormSynthesisMachine:** <http://normsynthesis.github.io/NormSynthesisMachine>
Implements an API that allows to perform norm synthesis for MAS.
- **NormLab:** <http://normsynthesis.github.io/NormLabSimulators>
Contains the code of the two MAS simulators: traffic and on-line community.

Download both projects whether in a **ZIP** or **TAR.GZ** file.

5.1. NormLab installation

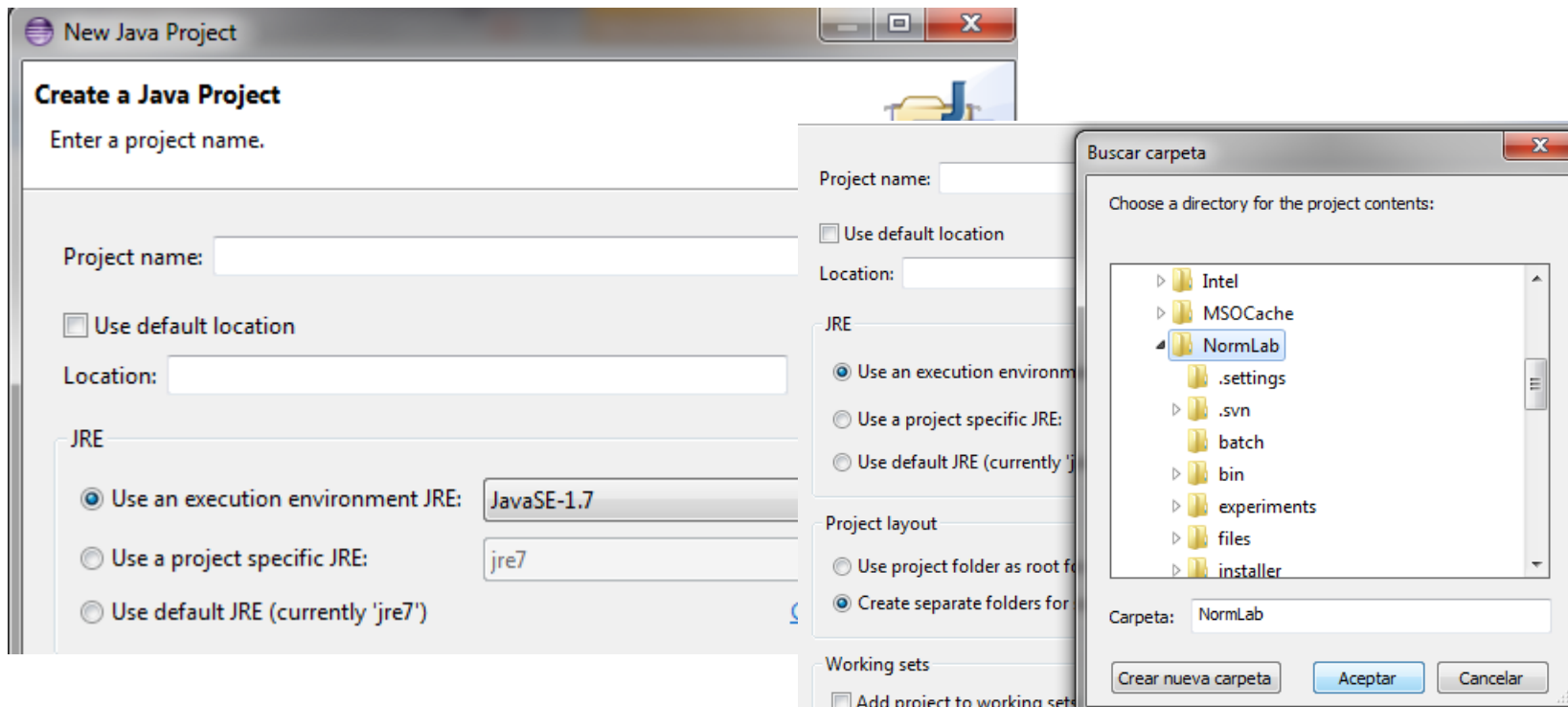
Preparing the working environment

1. Unzip ***NormSynthesisMachine*** and ***NormLabSimulators*** projects to your HOME folder.
 - *For instance... «/Users/Javi/NormLab»*
2. Both projects will be unzipped as *NormSynthesis-«project_name»- «numbers»*. For instance...
 - *NormSynthesis-NormLabSimulators-34d43o*
 - *NormSynthesis-NormSynthesisMachine-1847fje*
3. Rename both projects, removing the «NormSynthesis» part and the numbers. After renaming them they should look like this:
 - *NormLabSimulators*
 - *NormSynthesisMachine*

5.1. NormLab installation

Preparing the working environment

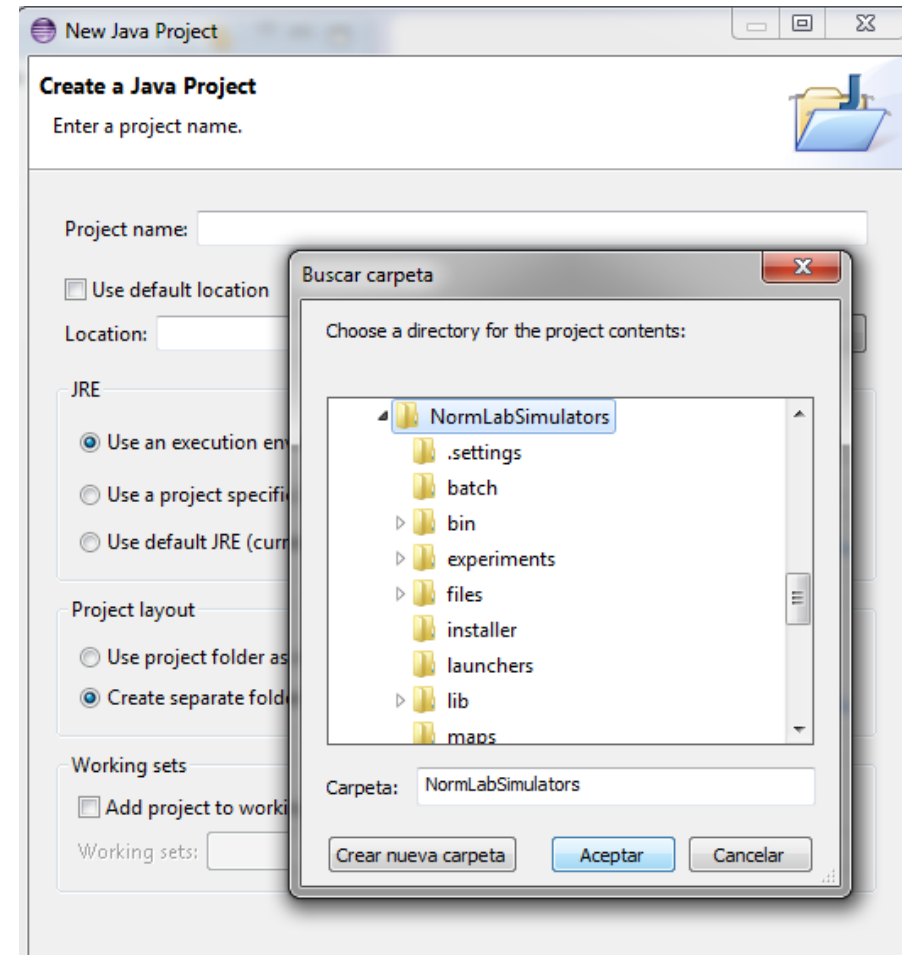
1. Open the **Repast Symphony IDE** (in Linux, open **Eclipse IDE** with Repast installed on it).
2. Import both projects **NormSynthesisMachine** and **NormLabSimulators in Eclipse**.
 1. *File>New>Java Project.*
 2. *Uncheck «Use default location» and click on «Browse».*



5.1. NormLab installation

Preparing the working environment

1. Unzip **NormLabSimulators** and **NormSynthesisMachine** projects to your HOME folder.
 - For instance... «/Users/Javi/NormLab»
2. Open **Eclipse IDE**.
3. Import both projects **NormLabSimulators** and **NormSynthesisMachine** in **Eclipse**:
 1. *File>New>Java Project.*
 2. *Uncheck «Use default location» and click on «Browse».*
1. Import projects **NormLabSimulators** and **NormSynthesisMachine**.



5.2. NormLab structure

Before starting you need to know:

NormLabSimulators project is structured as follows:

src/onlinecomm: The code of the on-line community simulator.

src/traffic: The code of the traffic simulator.

launchers: The launchers that allow to run the two simulators.

repast-settings/OnlineCommunity.rs: Basic Repast settings for the on-line community simulator.

repast-settings/TrafficJunction.rs: Basic Repast settings for the traffic junction simulator.

Tutorial outline

What are the contents of this tutorial?

1. An **introduction** to NormLab
 1. The NormLab architecture.
 2. The Norm Synthesis Machine.
 3. The NormLab simulators.
2. **Configuration** of the working environment
 1. NormLab download.
 2. NormLab installation.
3. NormLab **execution**:
 1. Execution examples.
 2. Guided development of different norm synthesis strategies.

Tutorial outline

NormLab **execution**:

1. Execution examples
 1. **Example** strategy 1: Returns an **empty** set of norms.
 2. **Example** strategy 2: Returns a fixed set of **1 norm**.
 3. **Example** strategy 3: Returns a fixed set of **3 norms**.
2. Guided development of different norm synthesis strategies
 1. **Development** of example strategy 1: **Empty** set of norms.
 2. **Development** of example strategy 2: Fixed set of **1 norm**.
 3. **Studying** example 4: A strategy with norm **generation**.
 4. **Studying** example 5: A strategy with norm **generation** + **evaluation**.
 5. **Studying** SIMON: A strategy with norm **generation** + **evaluation** + **refinement**.

Tutorial outline

NormLab **execution**:

1. Execution examples

1. **Example** strategy 1: Returns an **empty** set of norms.
2. **Example** strategy 2: Returns a fixed set of **1 norm**.
3. **Example** strategy 3: Returns a fixed set of **3 norms**.

2. Guided development of different norm synthesis strategies

1. **Development** of example strategy 1: **Empty** set of norms.
2. **Development** of example strategy 2: Fixed set of **1 norm**.
3. **Studying** example 4: A strategy with norm **generation**.
4. **Studying** example 5: A strategy with norm **generation** + **evaluation**.
5. **Studying** SIMON: A strategy with norm **generation** + **evaluation** + **refinement**.

6. Example 1: Executing NormLab

TrafficJunction norm synthesis example 1

We are going to execute the ***TrafficJunction*** simulator with a very simple norm synthesis strategy, which is as follows:

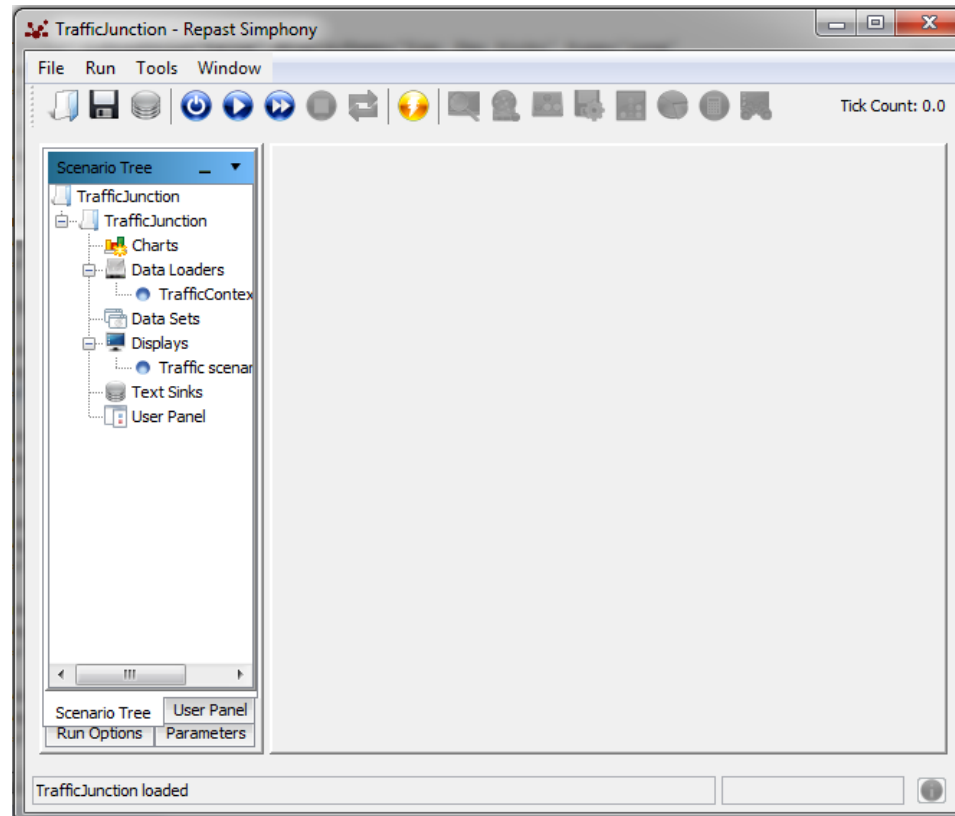
→ *Everytime the strategy is executed, return an **empty** normative system.*

Consequences: No norms are given to the agents → collisions are never removed.

6. Example 1: Executing NormLab


TrafficJunction norm synthesis example 1

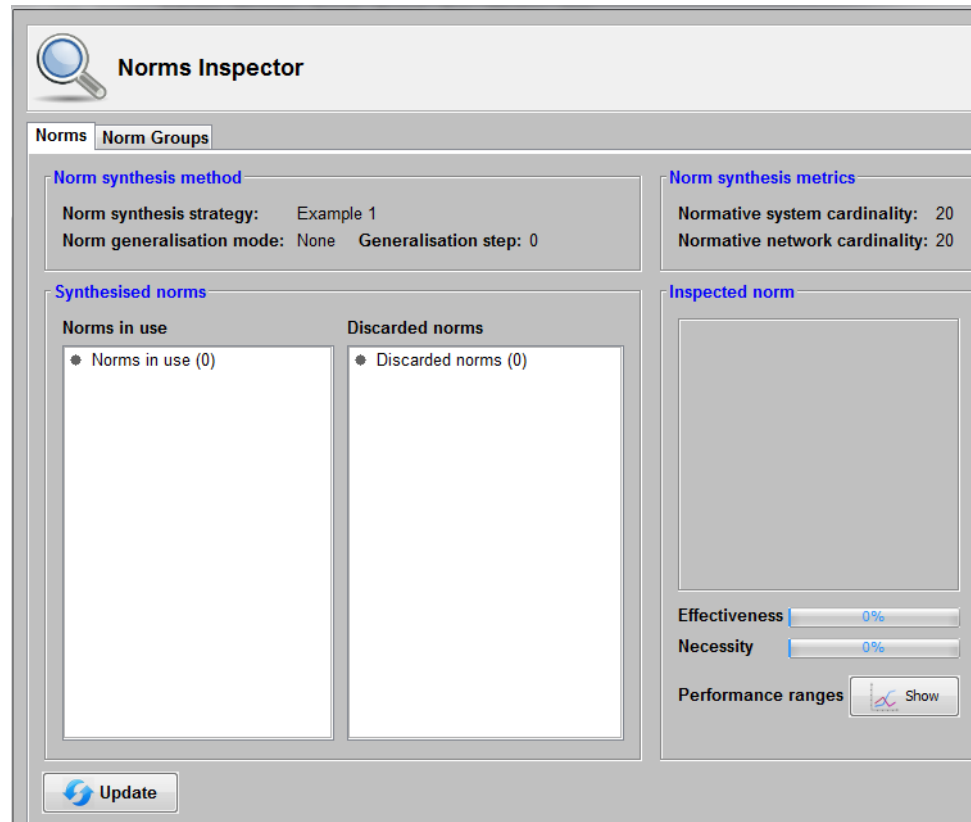
1. In Eclipse, in NormLabSimulators project, go to directory **launchers/**
2. Do right click on the file **TrafficJunctionSimulator.launch**.
3. Click on «Run As» > «TrafficJunctionSimulator».



6. Example 1: Executing NormLab



TrafficJunction norm synthesis example 1

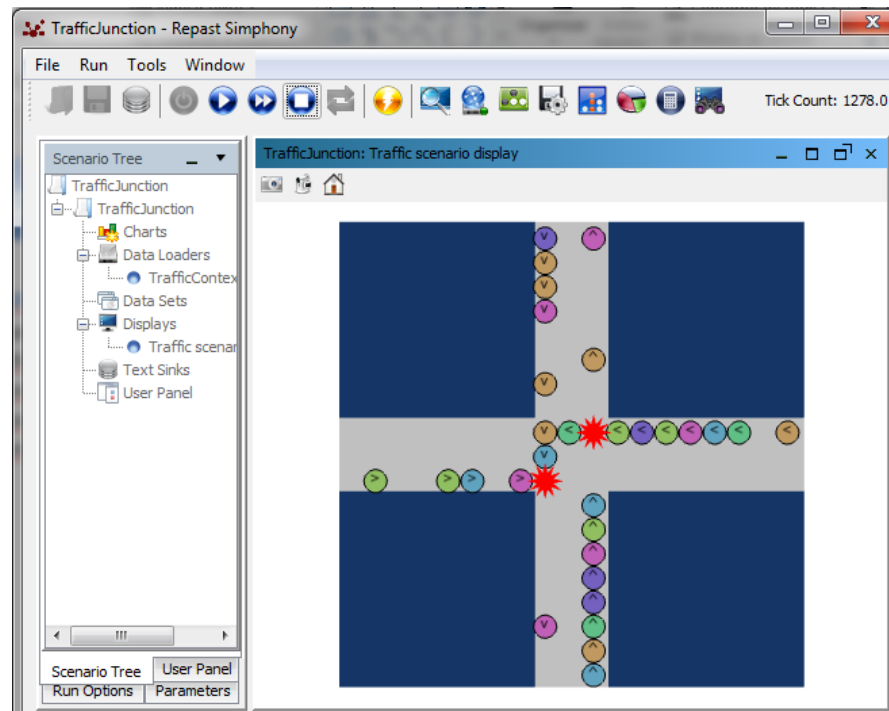
1. In Eclipse, in NormLabSimulators project, go to directory **launchers/**
2. Do right click on the file **TrafficJunctionSimulator.launch**.
3. Click on «Run As» > «TrafficJunctionSimulator».
4. Click on button  to initialise the simulator.



6. Example 1: Executing NormLab





TrafficJunction norm synthesis example 1

1. In Eclipse, in NormLabSimulators project, go to directory **launchers/**
2. Do right click on the file **TrafficJunctionSimulator.launch**.
3. Click on «Run As» > «TrafficJunctionSimulator».
4. Click on button  to initialise the simulator.
5. Click on button  to start the simulator. Cars will appear as coloured balls. Collisions will appear as red stars. Cars will start to drive and they will collide.



6. Example 1: Executing NormLab

TrafficJunction norm synthesis example 1

1. In Eclipse, in NormLabSimulators project, go to directory **launchers/**
2. Do right click on the file **TrafficJunctionSimulator.launch**.
3. Click on «Run As» > «TrafficJunctionSimulator».
4. Click on button  to initialise the simulator.
5. Click on button  to start the simulator. Cars will appear as coloured balls. Collisions will appear as red stars. Cars will start to drive and they will collide.
6. You can pause the simulation with button  and stop it with button 

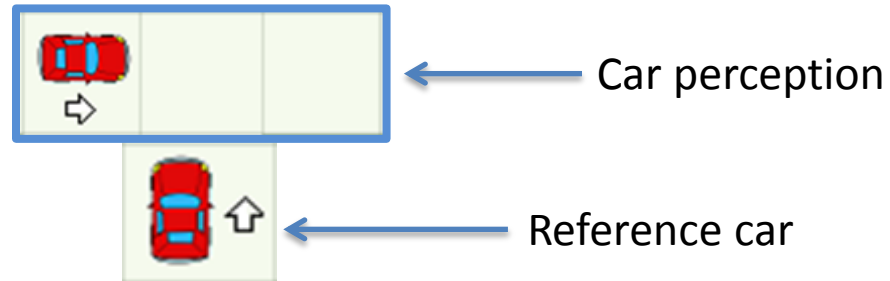
Tutorial outline

NormLab **execution**:

1. Execution examples
 1. **Example** strategy 1: Returns an **empty** set of norms.
 2. **Example** strategy 2: Returns a fixed set of **1 norm**.
 3. **Example** strategy 3: Returns a fixed set of **3 norms**.
2. Guided development of different norm synthesis strategies
 1. **Development** of example strategy 1: **Empty** set of norms.
 2. **Development** of example strategy 2: Fixed set of **1 norm**.
 3. **Studying** example 4: A strategy with norm **generation**.
 4. **Studying** example 5: A strategy with norm **generation** + **evaluation**.
 5. **Studying** SIMON: A strategy with norm **generation** + **evaluation** + **refinement**.

7. Example 2: Using norms

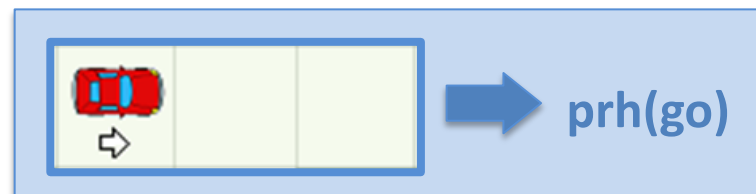
In the traffic simulator, cars perceive the scenario by means of the three cells in front of them:



Norms are...

- **IF ... THEN...** rules.
- Norm precondition: Set of **predicates** with one **term** each.
 - Three different predicates (**left, front, right**).
 - Six different terms (**<, ^, >, v, -, w, ***) representing cars with different headings, term «-» stands for «nothing», «w» for «wall» and «*» for «anything».
- Norm postcondition: A **modality**.

Graphical representation



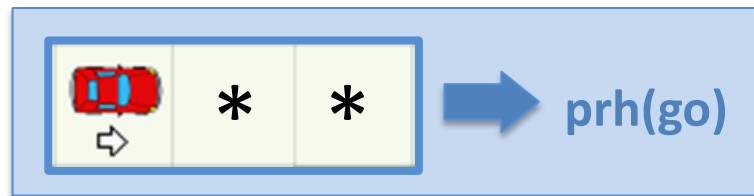
IF left(>) & front(-) & right(-) **THEN** prohibition(go)

7. Example 2: Using norms

TrafficJunction norm synthesis example 2

We are now going to execute the *TrafficJunction* simulator with a norm synthesis strategy that will avoid some (but not all) collisions between cars. With this aim, the strategy always returns a normative system with only one **left-side-priority** norm:

Norm 1




IF left(>) & front(*) & right(*) **THEN** prohibition(*go*)

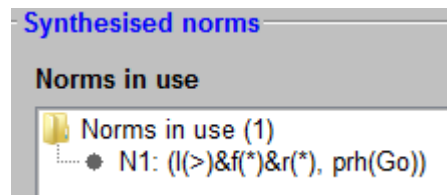
7. Example 2: Using norms

TrafficJunction norm synthesis example 2

1. In Eclipse, in NormLabSimulators project, go to directory **repast-settings/TrafficJunction.rs**
2. Open file **parameters.xml** by doing right click > *Open with* > *Text Editor*. This file defines the traffic simulator setting parameters.
3. Search for the parameter «NormSynthesisExample».
4. Set the field «defaultValue» with the value «2». This will indicate NormLab to launch example 2, which uses a norm synthesis strategy that always returns a normative system with the left-side-priority norm.

```
<parameter name="NormSynthesisExample" readOnly="false" displayName="NSM: Norm synthesis example" type="int"
converter="repast.simphony.parameter.StringConverterFactory$IntConverter"
defaultValue="2" />
```

5. Save the file.
6. Do right click on the file **launchers/TrafficJunctionSimulator.launch**.
7. Click on «Run As» > «TrafficJunctionSimulator».
8. Run the simulation with button 
9. Update the norm synthesis inspector. Observe how now the normative system contains one norm, and now cars occasionally stop to apply norm 1.



Car applying norm 1

Tutorial outline

NormLab **execution**:

1. Execution examples
 1. **Example** strategy 1: Returns an **empty** set of norms.
 2. **Example** strategy 2: Returns a fixed set of **1 norm**.
 3. **Example** strategy 3: Returns a fixed set of **3 norms**.
2. Guided development of different norm synthesis strategies
 1. **Development** of example strategy 1: **Empty** set of norms.
 2. **Development** of example strategy 2: Fixed set of **1 norm**.
 3. **Studying** example 4: A strategy with norm **generation**.
 4. **Studying** example 5: A strategy with norm **generation** + **evaluation**.
 5. **Studying** SIMON: A strategy with norm **generation** + **evaluation** + **refinement**.

8. Example 3: Removing collisions

TrafficJunction norm synthesis example 3

We are now going to execute the ***TrafficJunction*** simulator with a norm synthesis strategy that avoids all possible collisions. With this aim, it always returns the following normative system:

IF left(*) & front(^) & right(*)	THEN prohibition(<i>go</i>)
IF left(>) & front(-) & right(*)	THEN prohibition(<i>go</i>)
IF left(<) & front(<) & right(*)	THEN prohibition(<i>go</i>)

To execute this example, you just have to **follow the steps in section 7**, but setting **defaultValue=«3»** of the NormSynthesisExample parameter (again in NormLabSimulators project, directory **repast-settings/TrafficJunction.rs** , file **parameters.xml**)

Tutorial outline

NormLab **execution**:

1. Execution examples
 1. **Example** strategy 1: Returns an **empty** set of norms.
 2. **Example** strategy 2: Returns a fixed set of **1 norm**.
 3. **Example** strategy 3: Returns a fixed set of **3 norms**.
2. Guided development of different norm synthesis strategies
 1. **Development** of example strategy 1: **Empty** set of norms.
 2. **Development** of example strategy 2: Fixed set of **1 norm**.
 3. **Studying** example 4: A strategy with norm **generation**.
 4. **Studying** example 5: A strategy with norm **generation** + **evaluation**.
 5. **Studying** SIMON: A strategy with norm **generation** + **evaluation** + **refinement**.

9. Developing your own strategy

How are implemented all these examples? Let's implement one of the examples!

We are now going to develop our own norm synthesis strategy. In particular, we are going to implement the norm synthesis strategy of example 1, which returns an **empty normative system**.

The first thing we must do is to indicate *NormLab* that we are going to use a custom norm synthesis strategy. With this aim, follow these steps:

1. In Eclipse, in NormLabSimulators project, go to directory **repast-settings/TrafficJunction.rs**
2. Open file **parameters.xml** by doing right click > *Open with* > *Text Editor*. This file defines the traffic simulator setting parameters.
3. Search for the parameter «NormSynthesisExample» and set the field **defaultValue=«0»**. This will indicate NormLab that we do not want to load a pre-designed example.
4. Search for the parameter «NormSynthesisStrategy» and set the field **defaultValue=«0»**. This will indicate *NormLab* that we will give it a custom norm synthesis strategy.

```
<parameter name="NormSynthesisExample" isReadOnly="false" displayName="NSM: Norm synthesis example" type="int"
  converter="repast.simphony.parameter.StringConverterFactory$IntConverter"
  defaultValue="0" />

<parameter name="NormSynthesisStrategy" isReadOnly="false" displayName="NSM: Norm synthesis strategy (CUSTOM/IRON/SIMON/XSIMON)" type="int"
  converter="repast.simphony.parameter.StringConverterFactory$IntConverter"
  defaultValue="0" />
```

9. Developing your own strategy

Now, to create your norm synthesis strategy, just follow these steps:

1. In Eclipse (NormLabSimulators project), go to **package es.csic.iiaa.normlab.traffic.custom**.
2. There, create a new Java class *MyFirstStrategy.java* that implements the interface **es.csic.iiaa.nsm.strategy.NormSynthesisStrategy**.
3. The interface will require you to implement two methods:
 1. **execute()**: Executes the norm synthesis strategy
 2. **getNonRegulatedConflictsThisTick()**: Returns a data structure containing the conflicts that the strategy has detected during the current tick

```
package es.csic.iiaa.normlab.traffic.custom;

import java.util.List;
import java.util.Map;

import es.csic.iiaa.nsm.config.Goal;
import es.csic.iiaa.nsm.norm.NormativeSystem;
import es.csic.iiaa.nsm.norm.generation.Conflict;

/**
 *
 */
public class MyFirstStrategy implements es.csic.iiaa.nsm.strategy.NormSynthesisStrategy {

    @Override
    public NormativeSystem execute() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public Map<Goal, List<Conflict>> getNonRegulatedConflictsThisTick() {
        // TODO Auto-generated method stub
        return null;
    }
}
```

9. Developing your own strategy

We will create our strategy:

1. Create a new attribute **private Map<Goal, List<Conflict>> conflicts**.
2. Create a constructor for the class and, there, create the structure conflicts.
3. Make method **getNonRegulatedConflictsThisTick()** to return the attribute **conflicts**.
4. Your code should look like this:

```
/**
 *
 */
public class MyFirstStrategy implements es.csic.iiia.nsm.strategy.NormSynthesisStrategy {

    private Map<Goal, List<Conflict>> conflicts; // to save conflicts

    /**
     *
     */
    public MyFirstStrategy() {
        this.conflicts = new HashMap<Goal, List<Conflict>>();
    }

    @Override
    public NormativeSystem execute() {
        return null;
    }

    @Override
    public Map<Goal, List<Conflict>> getNonRegulatedConflictsThisTick() {
        return conflicts;
    }
}
```

9. Developing your own strategy

Now, let's implement the **execute()** method, which implements the norm synthesis strategy. This method must return an object *NormativeSystem*, that contains the norms that will be given to the agents.

There are a couple things that we must take into account:

- The Norm Synthesis Machine keeps synthesised norms in a **normative network**.
- To be able to access to the normative network, and the different elements of the Norm Synthesis Machine, we must receive the *NormSynthesisMachine* as a parameter in our strategy:

Follow now these steps:

1. In the constructor of the class, add the parameter ***es.csic.iiia.nsm.NormSynthesisMachine nsm***.
2. Now we can access the different elements of the Norm Synthesis Machine in our strategy.
3. Let's obtain the Normative Network! Add the following attribute to your class:
private NormativeNetwork normativeNetwork;
4. Now, in your constructor, add the following code line to obtain the (initially empty) normative network:
this.normativeNetwork = nsm.getNormativeNetwork();
5. Finally, we will now return an empty normative system at the end of the strategy execution. Add the following line of code at the end of method `execute()`:
return this.normativeNetwork.getNormativeSystem();

9. Developing your own strategy

Congratulations! You have created your first norm synthesis strategy, which returns an empty normative system Your code should now look like this:

```
package es.csic.iiia.normlab.traffic.custom;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

import es.csic.iiia.nsm.config.Goal;
import es.csic.iiia.nsm.net.norm.NormativeNetwork;
import es.csic.iiia.nsm.norm.NormativeSystem;
import es.csic.iiia.nsm.norm.generation.Conflict;

public class MyFirstStrategy implements es.csic.iiia.nsm.strategy.NormSynthesisStrategy {

    private Map<Goal, List<Conflict>> conflicts; // to save conflicts
    private NormativeNetwork normativeNetwork;

    public MyFirstStrategy(es.csic.iiia.nsm.NormSynthesisMachine nsm) {
        this.conflicts = new HashMap<Goal, List<Conflict>>();

        /* Get norm synthesis elements */
        this.normativeNetwork = nsm.getNormativeNetwork();
    }

    @Override
    public NormativeSystem execute() {
        return normativeNetwork.getNormativeSystem();
    }

    @Override
    public Map<Goal, List<Conflict>> getNonRegulatedConflictsThisTick() {
        return conflicts;
    }
}
```

10. Executing your implemented strategy

And now... how to tell *NormLab* to use your norm synthesis strategy?

We need to create an agent in the Traffic Simulator, which:

1. **Creates** and **configures** the Norm Synthesis Machine.
2. Adds **sensors** to the Norm Synthesis Machine to perceive the scenario.
3. **Creates** and **configures** the norm synthesis **strategy**.
4. **Executes** your strategy at every simulation step.

The traffic simulator incorporates a default Traffic Norm Synthesis Agent, which is implemented in class *DefaultTrafficNormSynthesisAgent* of package **es.csic.iiia.normlab.traffic.agent**.

Let's take a look at it...

10. Executing your implemented strategy

Observe the constructor **DefaultTrafficNormSynthesisAgent()**. It performs these tasks:

1. Creates the norm synthesis machine with a given configuration.
2. Adds a set of sensors to the norm synthesis machine in order to perceive the scenario.
3. Sets the norm synthesis strategy.

```
public DefaultTrafficNormSynthesisAgent(List<TrafficCamera> cameras,
    PredicatesDomains predDomains) {

    this.normativeSystem = new NormativeSystem();
    this.addedNorms = new ArrayList<Norm>();
    this.removedNorms = new ArrayList<Norm>();
    this.nsmSettings = new TrafficNormSynthesisSettings();
    this.dmFunctions = new TrafficDomainFunctions();

    /* 1. Create norm synthesis machine */
    this.nsm = new NormSynthesisMachine(nsmSettings, predDomains,
        dmFunctions, true);

    /* 2. Add sensors to the monitor of the norm synthesis machine */
    for(TrafficCamera camera : cameras) {
        this.nsm.addSensor(camera);
    }

    /* 3. Set the norm synthesis strategy */
    this.setNormSynthesisStrategy();
}
```

4. Executes the norm synthesis strategy at every simulation step.

```
public void step() throws IncorrectSetupException {
    this.addedNorms.clear();
    this.removedNorms.clear();

    /* Execute strategy and obtain new normative system */
    NormativeSystem newNormativeSystem = nsm.executeStrategy();
}
```


10. Executing your implemented strategy

To create the NormSynthesisMachine, it needs to create:

1. **NormSynthesisSettings:** The settings for the norm synthesis machine.
2. **PredicatesDomains:** Information about the agents' language. That is, the predicates and terms the agents employ to describe the scenario from their local point of view.
3. **DomainFunctions:** Some domain-dependent functions that the Norm Synthesis Machine requires to synthesise norms (e.g., conflict detection, norm applicability).

```
public DefaultTrafficNormSynthesisAgent(List<TrafficCamera> cameras,
    PredicatesDomains predDomains) {

    this.normativeSystem = new NormativeSystem();
    this.addedNorms = new ArrayList<Norm>();
    this.removedNorms = new ArrayList<Norm>();
    this.nsmSettings = new TrafficNormSynthesisSettings();
    this.dmFunctions = new TrafficDomainFunctions();

    /* 1. Create norm synthesis machine */
    this.nsm = new NormSynthesisMachine(nsmSettings, predDomains,
        dmFunctions, true);

    /* 2. Add sensors to the monitor of the norm synthesis machine */
    for(TrafficCamera camera : cameras) {
        this.nsm.addSensor(camera);
    }

    /* 3. Set the norm synthesis strategy */
    this.setNormSynthesisStrategy();
}
```

10. Executing your implemented strategy

NormSynthesisSettings: An interface to be implemented (located in package `es.csic.iiia.nsm.config`)

1. **getNormSynthesisStrategy()**: Returns the norm synthesis strategy to use.
2. **getSystemGoals()**: A list of system goals. In traffic, the only goal is “to avoid collisions”.
3. **getNormsDefaultUtility()**: Norms’ default utility (0.5 by default).
4. **getNormEvaluationLearningRate()**: The α rate to evaluate norms (0.1 is ok).
5. **getNormsPerformanceRangesSize()**: The size of the window to compute norms’ performance ranges.
6. **getNormGeneralisationMode()**: SIMON’s norm generalisation mode (Shallow/Deep).
7. **public int getNormGeneralisationStep()**: SIMON’s norm generalisation step, namely the number of norm predicates that can be simultaneously generalised.
8. **getGeneralisationBoundary(Dimension dim, Goal goal)**: Returns the minimum value of Effectiveness/necessity that a norm’s performance must reach to be generalised.
9. **getSpecialisationBoundary(Dimension dim, Goal goal)**: Returns the value of Effectiveness/necessity under which a norm can be specialised.
10. **getNumTicksOfStabilityForConvergence()**: The number of simulation ticks without conflicts or changes to the normative system to converge.

An implementation of these settings for the traffic simulator is located in package `es.csic.iiia.normlab.traffic.normsynthesis`, class *TrafficNormSynthesisSettings*

10. Executing your implemented strategy

PredicatesDomains: Contains the predicates and terms that the agents employ to describe the MAS from their local point of view. Located in project NormSynthesisMachine, package **es.csic.iiia.nsm.agent.language**.

The traffic simulator creates predicates and their domains in (project NormLabSimulators) class **es.csic.iiia.traffic.TrafficSimulator**, method **createPredicatesDomains()**.

- Three different predicates (**l**, **f**, **r**) that represent the left, front and right positions in front of a car.
- Six different terms (**<**, **^**, **>**, **v**, **-**, **w**, *****) representing cars with different headings, term «-» stands for «nothing», «w» for «wall» and «*» for «anything».

10. Executing your implemented strategy

PredicatesDomains: The traffic simulator creates predicates and their domains in class `es.csic.iiia.traffic.TrafficSimulator`, method `createPredicatesDomains()`.

```
private void createPredicatesDomains() {  
    /* Predicate "left" domain */  
    TaxonomyOfTerms leftPredTaxonomy = new TaxonomyOfTerms("l");  
    leftPredTaxonomy.addTerm("*");  
    leftPredTaxonomy.addTerm("<");  
    leftPredTaxonomy.addTerm(">");  
    leftPredTaxonomy.addTerm("-");  
    leftPredTaxonomy.addRelationship("<", "*");  
    leftPredTaxonomy.addRelationship(">", "*");  
    leftPredTaxonomy.addRelationship("-", "*");  
  
    /* Predicate "front" domain*/  
    TaxonomyOfTerms frontPredTaxonomy = new TaxonomyOfTerms("f", leftPredTaxonomy);  
    frontPredTaxonomy.addTerm("^");  
    frontPredTaxonomy.addRelationship("^", "*");  
  
    /* Predicate "right" domain*/  
    TaxonomyOfTerms rightPredTaxonomy = new TaxonomyOfTerms("r", leftPredTaxonomy);  
    rightPredTaxonomy.addTerm("w");  
    rightPredTaxonomy.addRelationship("w", "*");  
  
    this.predDomains = new PredicatesDomains();  
    this.predDomains.addPredicateDomain("l", leftPredTaxonomy);  
    this.predDomains.addPredicateDomain("f", frontPredTaxonomy);  
    this.predDomains.addPredicateDomain("r", rightPredTaxonomy);  
}
```

10. Executing your implemented strategy

DomainFunctions: An interface to be implemented. Located in package **es.csic.iiia.nsm.config** (NormSynthesisMachine project).

1. **isConsistent(SetOfPredicatesWithTerms agentContext)**: Returns true if a set of predicates with terms is consistent with the domain. For instance, (left(>),front(-),right(-)) is consistent. By contrast, (left(>),front(<),right(-)) is not consistent, since two cars can not drive in opposite directions in the same lane.
2. **agentContextFunction(long agentId, View view)**: Returns the local perception of a given agent at a particular system state (received as a View).
3. **agentActionFunction(long agentId,ViewTransition viewTransition)**: Returns a list of the actions that an agent performed in the transition from a state s_t to a state s_{t-1}
4. **getNonRegulatedConflicts(Goal goal,ViewTransition viewTransition)**: Receives a transition between two states, a system goal (e.g., to avoid collisions) and returns the conflicts that have arisen in that transition with respect to the system goal (e.g., returns the collisions).
5. **hasConflict(View view, long agentId, Goal goal)**: Returns true if a given agent is in conflict in a given system state (i.e., View).

An implementation of the domain functions for the traffic simulator is located on NormLabSimulators project, **es.csic.iiia.normlab.traffic.normsynthesis** package, class *TrafficDomainFunctions*.

10. Executing your implemented strategy

Now that we understand how *DefaultTrafficNormSynthesisAgent* works, let's tell it to use your norm synthesis strategy:

1. Open class *DefaultTrafficNormSynthesisAgent* in package **es.csic.iiia.normlab.traffic.agent**. This class implements the agent that «lives» in the traffic simulator, creates the norm synthesis machine and executes the strategy at every simulation tick.
2. Go to method **setCustomNormSynthesisStrategy()**
3. There, tell NormLab to use your norm synthesis strategy. Use this code:

```
/**
 * Sets a custom norm synthesis strategy
 */
protected void setCustomNormSynthesisStrategy() {
    MyFirstStrategy myStrategy = new MyFirstStrategy(this.nsm);
    this.nsm.useStrategy(myStrategy);
}
```

4. It is as simple as creating your norm synthesis strategy and telling the norm synthesis machine to use your strategy.
5. Execute the simulation as you did for example 1.

Congratulations, you are using your own strategy!

Tutorial outline

NormLab **execution**:

1. Execution examples
 1. **Example** strategy 1: Returns an **empty** set of norms.
 2. **Example** strategy 2: Returns a fixed set of **1 norm**.
 3. **Example** strategy 3: Returns a fixed set of **3 norms**.
2. Guided development of different norm synthesis strategies
 1. **Development** of example strategy 1: **Empty** set of norms.
 2. **Development** of example strategy 2: Fixed set of **1 norm**.
 3. **Studying** example 4: A strategy with norm **generation**.
 4. **Studying** example 5: A strategy with norm **generation** + **evaluation**.
 5. **Studying** SIMON: A strategy with norm **generation** + **evaluation** + **refinement**.

11. Adding norms to your strategy

Let's now **add some norms** to our strategy. We will use the same set of norms than used in the **example 2** (with one only left-hand-side priority norm).

1. In package `es.csic.iiia.normlab.traffic.custom` (NormLabSimulators project), **copy** your first norm synthesis strategy (*MyFirstStrategy.java*) as **a new strategy** *MySecondStrategy.java*.
2. To add norms to the normative network we need to know the system goals (in traffic, the only system goal is to avoid collisions). With this aim, add the following attribute to your strategy.
 - **private List<Goal> goals;**
3. Now obtain the system goals in your constructor:
 - **this.goals = nsm.getNormSynthesisSettings().getSystemGoals();**
4. Your code should look like this:

```
private List<Goal> goals;
private Map<Goal, List<Conflict>> conflicts; // to save conflicts
private NormativeNetwork normativeNetwork;

/**
 *
 * @param nsm
 */
public MySecondStrategy(es.csic.iiia.nsm.NormSynthesisMachine nsm) {
    this.conflicts = new HashMap<Goal, List<Conflict>>();

    /* Get norm synthesis elements */
    this.goals = nsm.getNormSynthesisSettings().getSystemGoals();
    this.normativeNetwork = nsm.getNormativeNetwork();
}

/**
 *
 */
@Override
public NormativeSystem execute() {
    return normativeNetwork.getNormativeSystem();
}
```


11. Adding norms to your strategy

1. Let's create the normative system. Norms have four elements: (1) a norm precondition; (2) a modality (in our case, a prohibition); (3) an action to obligate/prohibit. In our implementation, the norm also includes the goal it is aimed to achieve.
2. Now, create a new method **createNormativeSystem()** that will add the norms to the normative network:

```
/**
 * Creates a normative system that allows all possible collisions in the
 * road traffic scenario
 */
private void createNormativeSystem() {

    /* Get system goal (to avoid collisions) */
    Goal goalAvoidCollisions = goals.get(0);

    /* Create norm preconditions */
    SetOfPredicatesWithTerms n1Precondition = new SetOfPredicatesWithTerms();
    n1Precondition.add("l", ">");
    n1Precondition.add("f", "*");
    n1Precondition.add("r", "*");

    /* Create norms */
    Norm n1 = new Norm(n1Precondition,
        NormModality.Prohibition, CarAction.Go, goalAvoidCollisions);

    /* Add the norms to the normative network and activate them */
    this.normativeNetwork.add(n1);
    this.normativeNetwork.activate(n1);
}
```

3. This code first gets the only system goal (to avoid collisions between cars)
4. Then, it creates a norm precondition (set of predicates with terms) and adds the predicates «l» (left), «f» (front) and «r» (right), with its corresponding term.
5. Finally, it creates the norm adding the pre-condition, the modality «Prohibition» over the action «Go», and the goal of the norm (to avoid collisions).

11. Adding norms to your strategy

1. Now, call method **createNormativeSystem()** at the end of your constructor. Your code should look like this:

```
/**
 *
 */
public class MySecondStrategy implements es.csic.iiia.nsm.strategy.NormSynthesisStrategy {

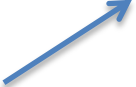
    private List<Goal> goals;
    private Map<Goal, List<Conflict>> conflicts; // to save conflicts
    private NormativeNetwork normativeNetwork;

    /**
     *
     * @param nsm
     */
    public MySecondStrategy(es.csic.iiia.nsm.NormSynthesisMachine nsm) {
        this.conflicts = new HashMap<Goal, List<Conflict>>();

        /* Get norm synthesis elements */
        this.goals = nsm.getNormSynthesisSettings().getSystemGoals();
        this.normativeNetwork = nsm.getNormativeNetwork();

        this.createNormativeSystem();
    }

    /**
     *
     */
    @Override
    public NormativeSystem execute() {
        return normativeNetwork.getNormativeSystem();
    }
}
```



2. At each execution, the strategy will return the norms that are active in the normative network (i.e., the normative system).

11. Adding norms to your strategy

To finish, set the traffic norm synthesis agent to use your new strategy.

1. Open class *DefaultTrafficNormSynthesisAgent* in package **es.csic.iiia.normlab.traffic.agent**. This class implements the agent that «lives» in the traffic simulator, creates the norm synthesis machine and executes the strategy at every simulation tick.
2. Go to method **setCustomNormSynthesisStrategy()**
3. There, tell *NormLab* to use your norm synthesis strategy. Use this code:

```
/**  
 * Sets a custom norm synthesis strategy  
 */  
protected void setCustomNormSynthesisStrategy() {  
    MySecondStrategy myStrategy = new MySecondStrategy(this.nsm);  
    this.nsm.useStrategy(myStrategy);  
}
```

4. You can now execute the Traffic Simulator and see how your second strategy works. Observe that:
 1. The normative system contains now one norm.
 2. The unique norm is never evaluated (click on button *Show* of norms' performance ranges).

Tutorial outline

NormLab **execution**:

1. Execution examples
 1. **Example** strategy 1: Returns an **empty** set of norms.
 2. **Example** strategy 2: Returns a fixed set of **1 norm**.
 3. **Example** strategy 3: Returns a fixed set of **3 norms**.
2. Guided development of different norm synthesis strategies
 1. **Development** of example strategy 1: **Empty** set of norms.
 2. **Development** of example strategy 2: Fixed set of **1 norm**.
 3. **Studying** example 4: A strategy with norm **generation**.
 4. **Studying** example 5: A strategy with norm **generation** + **evaluation**.
 5. **Studying** SIMON: A strategy with norm **generation** + **evaluation** + **refinement**.

12. A strategy with automatic norm generation

Let's see now how can we automatically generate norms on-line.

For this example we are going to use the code of example 4, which is located in the package **es.csic.iiia.normlab.traffic.examples.ex4**.

There, we can find the following classes:

TrafficNSExample4_NSAgent

The agent that creates the Norm Synthesis Machine and executes the strategy.

TrafficNSExample4_NSOperators

Operators to **create**, **add**, **activate** and **deactivate** norms in the normative network.

TrafficNSExample4_NSStrategy

A norm synthesis strategy that generates norms to avoid arisen collisions in the future.

12. A strategy with automatic norm generation

Let's see now how can we automatically generate norms on-line.

For this example we are going to use the code of example 4, which is located in the package **es.csic.iiia.normlab.traffic.examples.ex4**.

There, we can find the following classes:

TrafficNSExample4_NSAgent

The agent that creates the Norm Synthesis Machine and executes the strategy.

TrafficNSExample4_NSOperators

Operators to **create**, **add**, **activate** and **deactivate** norms in the normative network.

TrafficNSExample4_NSStrategy

A norm synthesis strategy that generates norms to avoid arisen collisions in the future.

This agent works along the lines of the DefaultTrafficNormSynthesisAgent

12. A strategy with automatic norm generation

TrafficNSExample4_NSOperators: How do operators work?

Create:

1. Receives a *Conflict* and a system *Goal*.
2. Employs a Case-Based Reasoning (CBR) norm generation approach to generate a norm aimed at avoiding the given conflict in the future.
3. If the norm does not exist in the normative network, then it adds it.
4. If the norm exists in the normative network, then it activates it (since it may be inactive).

Add:

1. Adds a norm to the normative network.
2. Activates the norm in the normative network.

Activate:

1. Sets the state of a norm as «Active» in the normative network

Deactivate:

1. Sets the state of a norm as «Inactive» in the normative network.
 - This operator is not invoked in this example since it does not refine norms (and hence does not deactivate norms).

12. A strategy with automatic norm generation

TrafficNSExample4_NSStrategy: How does the norm synthesis strategy work?

Everytime the strategy is executed, it:

1. **Perceives** the scenario by means of the monitor. It saves perceptions in the form of *ViewTransitions*. A *ViewTransition* describes a part of the scenario at time t-1 and at time t (that is, its transition from the previous to the current tick).

```
/**
 * Calls scenario monitors to perceive agents interactions
 *
 * @return a {@code List} of the monitor perceptions, where each perception
 *         is a view transition from t-1 to t
 */
private void obtainPerceptions(List<ViewTransition> viewTransitions) {
    this.monitor.getPerceptions(viewTransitions);
}
```

2. **Detects conflicts** in perceptions by invoking method **getNonRegulatedConflicts()** of *DomainFunctions*.

```
/**
 * Given a list of view transitions (from t-1 to t), this method
 * returns a list of conflicts with respect to each goal of the system
 *
 * @param viewTransitions the list of perceptions of each sensor
 */
protected Map<Goal, List<Conflict>> conflictDetection(
    List<ViewTransition> viewTransitions) {
    this.conflicts.clear();

    /* Conflict detection is computed in terms of a goal */
    for(Goal goal : this.nsmSettings.getSystemGoals()) {
        List<Conflict> goalConflicts = new ArrayList<Conflict>();

        for(ViewTransition vTrans : viewTransitions) {
            goalConflicts.addAll(dmFunctions.getNonRegulatedConflicts(goal, vTrans));
        }
        conflicts.put(goal, goalConflicts);
    }
    return conflicts;
}
```


12. A strategy with automatic norm generation

3. **Generates norms** (one for each detected conflict) by means of operator **create**.

```
/**
 * Executes the norm generation phase
 */
private void normGeneration() {

    /* Obtain monitor perceptions */
    obtainPerceptions(viewTransitions);

    /* Conflict detection */
    conflicts = conflictDetection(viewTransitions);

    /* Norm generation */
    for(Goal goal : conflicts.keySet()) {
        for(Conflict conflict : conflicts.get(goal)) {
            operators.create(conflict, goal);
        }
    }
}
```

To execute this strategy, follow these steps:

1. In Eclipse, go to directory **repast-settings/TrafficJunction.rs**
2. Open file **parameters.xml** by doing right click > *Open with* > *Text Editor*. This file defines the *NormLab* parameters.
3. Search for the parameter «NormSynthesisExample» and set the field **defaultValue=«4»**.

Execute the simulator and see how, as long as cars collide, it generates norms to avoid those collisions in the future.

Tutorial outline

NormLab **execution**:

1. Execution examples
 1. **Example** strategy 1: Returns an **empty** set of norms.
 2. **Example** strategy 2: Returns a fixed set of **1 norm**.
 3. **Example** strategy 3: Returns a fixed set of **3 norms**.
2. Guided development of different norm synthesis strategies
 1. **Development** of example strategy 1: **Empty** set of norms.
 2. **Development** of example strategy 2: Fixed set of **1 norm**.
 3. **Studying** example 4: A strategy with norm **generation**.
 4. **Studying** example 5: A strategy with norm **generation + evaluation**.
 5. **Studying** SIMON: A strategy with norm **generation + evaluation + refinement**.

13. Automatic norm generation + evaluation

We have seen how to automatically **generate** norms on-line.
Let's see now how can we automatically **evaluate** norms on-line.

For this example we are going to use the code of **example 5**, which is located in the package **es.csic.iia.normlab.traffic.examples.ex5**.

There, we can find the following classes:

TrafficNSExample5_NSAgent

The agent that creates the Norm Synthesis Machine and executes the strategy.

TrafficNSExample5_NSOperators

Operators to **create**, **add**, **activate** and **deactivate** norms in the normative network.

TrafficNSExample5_NSStrategy

A norm synthesis strategy that generates norms to avoid arisen collisions in the future, and continuously **evaluates** them in base of their outcomes in the scenario.

TrafficNSExample5_NSUtilityFunction

A function to **evaluate** norms' utility based on their outcomes whenever agents fulfill/infringe norms.

13. Automatic norm generation + evaluation

We have seen how to automatically **generate** norms on-line.
Let's see now how can we automatically **evaluate** norms on-line.

For this example we are going to use the code of **example 5**, which is located in the package **es.csic.iiia.normlab.traffic.examples.ex5**.

There, we can find the following classes:

TrafficNSExample5_NSAgent

The agent that creates the Norm Synthesis Machine and executes the strategy.

TrafficNSExample5_NSOperators

Operators to **create**, **add**, **activate** and **deactivate** norms in the normative network.

TrafficNSExample5_NSStrategy

A norm synthesis strategy that generates norms to avoid arisen collisions in the future, and continuously **evaluates** them in base of their outcomes in the scenario.

TrafficNSExample5_NSUtilityFunction

A function to **evaluate** norms' utility based on their outcomes whenever agents fulfill/infringe norms.

You know how these things work...

13. Automatic norm generation + evaluation

How does norm evaluation work?

- Norm fulfilled + no conflicts → **Effective** norm (It avoids conflicts).
- Norm fulfilled + conflicts → **Ineffective** norm (It does not avoid conflicts).
- Norm infringed + no conflicts → **Unnecessary** norm (No conflicts arise when it is not fulfilled).
- Norm infringed + conflicts → **Necessary** norm (Conflicts arise when it is not fulfilled).

To evaluate norms at each tick, the norm synthesis strategy requires to retrieve:

1. The norms that have been **fulfilled** and **infringed** during the transition from the previous tick to the current tick.
2. Information about whether norm fulfilments and infringements led to conflicts or not in the current tick.

13. Automatic norm generation + evaluation

TrafficNSExample5_NSStrategy: How does this new norm synthesis strategy work?

Everytime this particular strategy is executed, it performs **norm generation** + **norm evaluation**. You already know norm generation. But... How is norm evaluation implemented?

Norm evaluation consists on the following steps:

1. **Compute norm applicability**, namely to retrieve the norms that applied to each agent in the simulation at time t-1.

```
protected Map<ViewTransition, NormsApplicableInView> normApplicability(
    List<ViewTransition> vTransitions) {

    /* Clear norm applicability from previous tick */
    this.normApplicability.clear();

    /* Get applicable norms of each viewTransition (of each sensor) */
    for(ViewTransition vTrans : vTransitions) {
        NormsApplicableInView normApplicability;
        normApplicability = this.normReasoner.getNormsApplicable(vTrans);
        this.normApplicability.put(vTrans, normApplicability);
    }
    return this.normApplicability;
}
```

- As you can see in the code, for each *ViewTransition* it employs a **NormReasoner** to compute the norms that apply to each agent in the viewTransition.
- The *NormReasoner* employs the *DomainFunctions* to retrieve the norms that apply to each agent.

13. Automatic norm generation + evaluation

2. **Compute norm compliance**, namely to assess if agents **complied or not** with their applicable norms during the transition from the previous tick (time t-1) to the current tick (time t), and if they lead to **conflicts** or not.

```
protected void normCompliance(Map<ViewTransition,
    NormsApplicableInView> normApplicability) {

    /* Check norm compliance in the view in terms of each system goal */
    for(Goal goal : this.nsmSettings.getSystemGoals()) {

        /* Clear norm compliance of previous tick */
        this.normCompliance.get(goal).clear();

        /* Evaluate norm compliance and conflicts in each
         * view transition with respect to each system goal */
        for(ViewTransition vTrans : normApplicability.keySet()) {
            NormsApplicableInView vNormAppl = normApplicability.get(vTrans);

            /* If there is no applicable norm in the view, continue */
            if(vNormAppl.isEmpty()) {
                continue;
            }
            NormComplianceOutcomes nCompliance = this.normReasoner.
                checkNormComplianceAndOutcomes(vNormAppl, goal);

            this.normCompliance.get(goal).put(vTrans, nCompliance);
        }
    }
}
```

13. Automatic norm generation + evaluation

3. Update norms' utilities based on norm compliance.

```
protected void updateUtilitiesAndPerformances(  
    Map<Goal, Map<ViewTransition, NormComplianceOutcomes>> normCompliance) {  
  
    for(Goal goal : this.nsmSettings.getSystemGoals()) {  
        for(ViewTransition vTrans : normCompliance.get(goal).keySet()) {  
            for(Dimension dim : this.nsm.getNormEvaluationDimensions()) {  
                this.utilityFunction.evaluate(dim, goal,  
                    normCompliance.get(goal).get(vTrans), normativeNetwork);  
            }  
        }  
    }  
}
```

Each norm is evaluated in terms of:

- **The system goals.** Are norms useful to achieve system goals?
Example: In the case of traffic, are norms useful to avoid car collisions?
- **Two dimensions**, effectiveness and necessity. Are norms effective to avoid collisions? Are they necessary to avoid collisions?

13. Automatic norm generation + evaluation

Finally, the **normEvaluation()** method puts together norm applicability, norm compliance and update utilities:

```
/**
 * Executes the norm evaluation phase
 */
private void normEvaluation() {

    /* Compute norm applicability */
    this.normApplicability = this.normApplicability(viewTransitions);

    /* Detect norm applicability and compliance */
    this.normCompliance(this.normApplicability);

    /* Update utilities and performances */
    this.updateUtilitiesAndPerformances(this.normCompliance);
}
```

Let's execute this strategy. Follow these steps:

1. In Eclipse, go to directory **repast-settings/TrafficJunction.rs**
2. Open file **parameters.xml** by doing right click > *Open with* > *Text Editor*. This file defines the *NormLab* parameters.
3. Search for the parameter «NormSynthesisExample» and set the field **defaultValue=«5»**. This will indicate NormLab that we do not want to load a pre-designed example.

Execute the simulator and see how now it generates norms and evaluates them. Observe how the effectiveness and necessity of norms change along time.

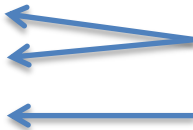
Tutorial outline

NormLab **execution**:

1. Execution examples
 1. **Example** strategy 1: Returns an **empty** set of norms.
 2. **Example** strategy 2: Returns a fixed set of **1 norm**.
 3. **Example** strategy 3: Returns a fixed set of **3 norms**.
2. Guided development of different norm synthesis strategies
 1. **Development** of example strategy 1: **Empty** set of norms.
 2. **Development** of example strategy 2: Fixed set of **1 norm**.
 3. **Studying** example 4: A strategy with norm **generation**.
 4. **Studying** example 5: A strategy with norm **generation + evaluation**.
 5. **Studying** SIMON: A strategy with norm **generation + evaluation + refinement**.

14. SIMON. A complete norm synthesis strategy

We are now going to see how to implement a complete norm synthesis strategy that performs:

1. Norm generation
 2. Norm evaluation
 3. Norm refinement
- 
- You already know these phases
- Let's see how SIMON refines the normative system

With this aim, we will execute the SIMON norm synthesis strategy. First of all, let's tell NormLab that we want to execute SIMON:

1. In Eclipse, go to directory **repast-settings/TrafficJunction.rs**
2. Open file **parameters.xml** by doing right click > *Open with* > *Text Editor*. This file defines the *NormLab* parameters.
3. Search for the parameter «**NormSynthesisExample**» and set the field **defaultValue=«0»**. This will indicate NormLab that we do not want to load a pre-designed example.
4. Search for the parameter «**NormSynthesisStrategy**» and set the field **defaultValue=«2»**. This will indicate *NormLab* that we want to use the **SIMON** norm synthesis strategy.
5. Search for the parameter «**NormGeneralisationMode**» and set the field **defaultValue=«1»**. This will indicate *NormLab* that we want SIMON to use **Deep** norm generalisation.
6. Search for the parameter «**NormGeneralisationStep**» and set the field **defaultValue=«1»**. This will indicate *NormLab* that we want SIMON to generalise just one norm predicate simultaneously in each norm generalisation.

14. SIMON. A complete norm synthesis strategy

Your **parameters.xml** file should look like this:

```
<parameter name="NormSynthesisExample" isReadOnly="false" displayName="NSM: Norm synthesis example" type="int"
converter="repast.simphony.parameter.StringConverterFactory$IntConverter"
defaultValue="0" />

<parameter name="NormSynthesisStrategy" isReadOnly="false" displayName="NSM: Norm synthesis strategy (CUSTOM/IRON/SIMON/XSIMON)" type="int"
converter="repast.simphony.parameter.StringConverterFactory$IntConverter"
defaultValue="2" />

<parameter name="NormGeneralisationMode" isReadOnly="false" displayName="NSM: Norm generalisation mode (SHALLOW/DEEP)" type="int"
converter="repast.simphony.parameter.StringConverterFactory$IntConverter"
defaultValue="1" />

<parameter name="NormGeneralisationStep" isReadOnly="false" displayName="NSM: Norm generalisation step" type="int"
converter="repast.simphony.parameter.StringConverterFactory$IntConverter"
defaultValue="1" />
```

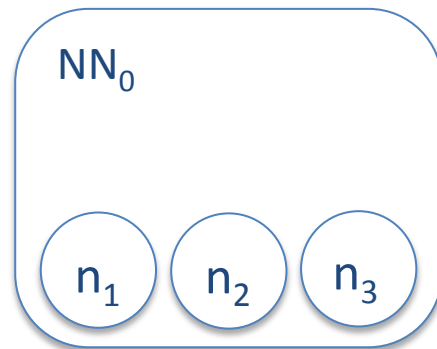
14. SIMON. A complete norm synthesis strategy

Norm refinement: **Generalises** norms when possible, and **specialises** norms when necessary.

- Norm generalisations allow to synthesise compact normative systems by generalising several norms to one unique norms that implicitly represents them.

14. SIMON. A complete norm synthesis strategy

Norm generalisations allow to increase the compactness of the normative system



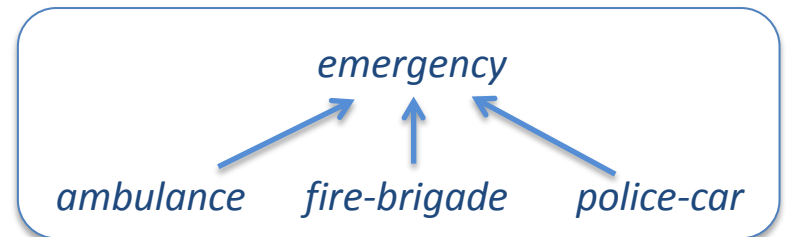
Normative system

$\{n_1, n_2, n_3\}$

n_1 : Give way to **ambulances**

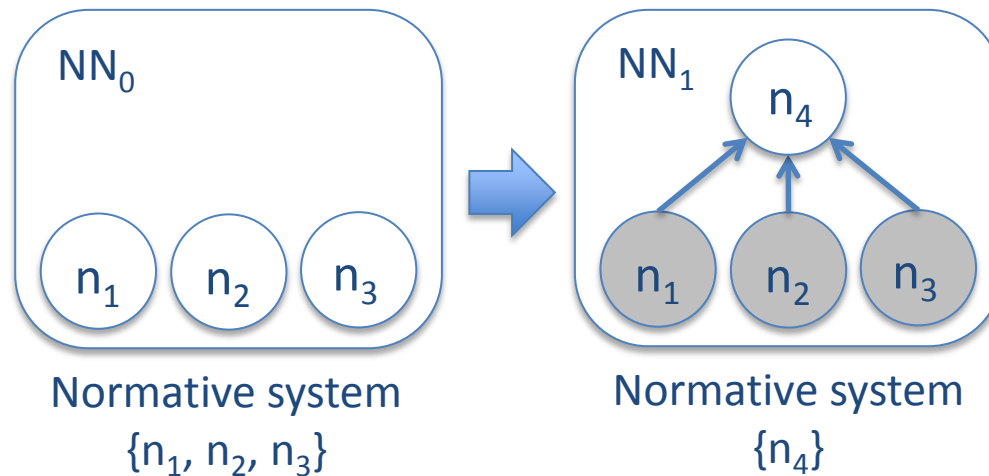
n_2 : Give way to **fire brigade**

n_3 : Give way to **police cars**



14. SIMON. A complete norm synthesis strategy

Norm generalisations allow to increase the compactness of the normative system

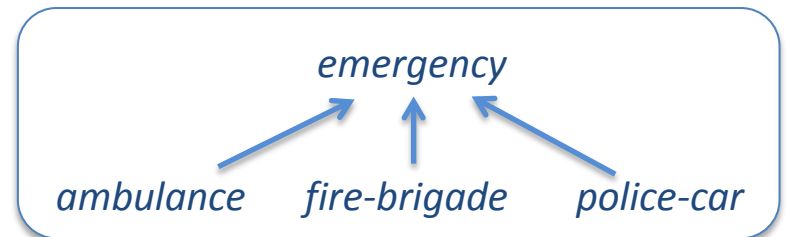


n_1 : Give way to **ambulances**

n_2 : Give way to **fire brigade**

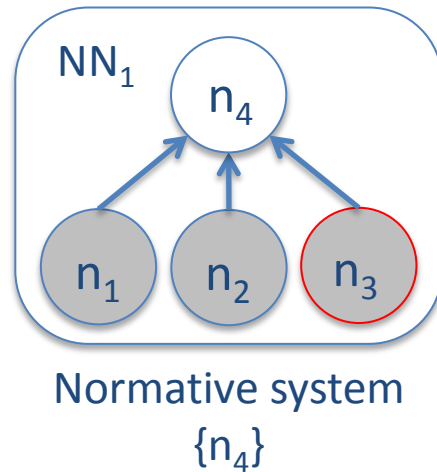
n_3 : Give way to **police cars**

n_4 : Give way to **emergency** vehicles



14. SIMON. A complete norm synthesis strategy

Norm specialisations allow to remove from the normative system those norms that under-perform

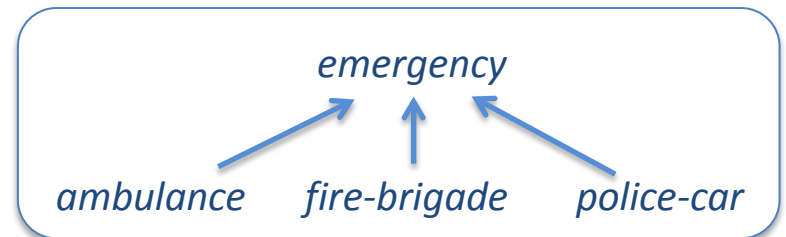


n_1 : Give way to **ambulances**

n_2 : Give way to **fire brigade**

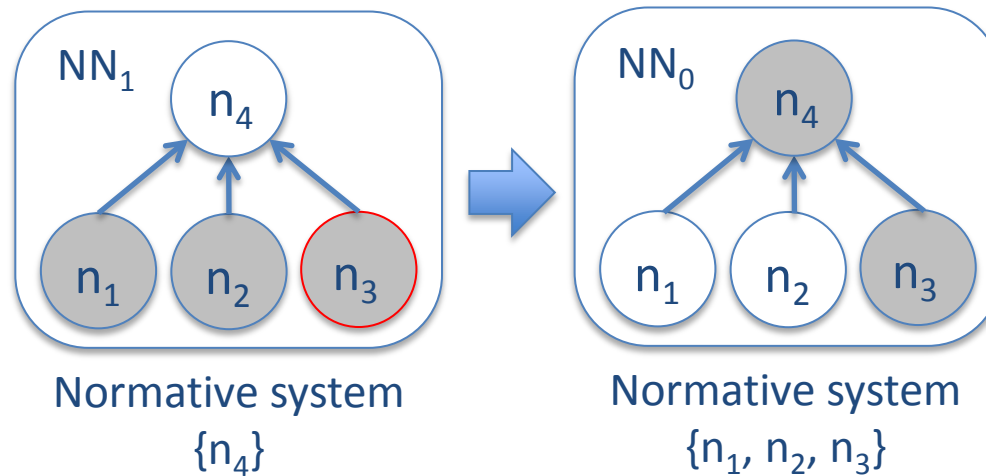
n_3 : Give way to **police cars**

n_4 : Give way to **emergency** vehicles



14. SIMON. A complete norm synthesis strategy

Norm specialisations allow to remove from the normative system those norms that under-perform

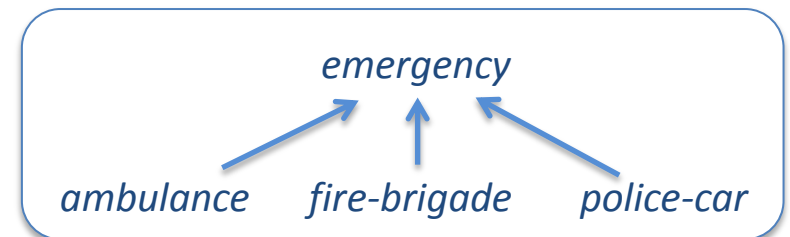


n_1 : Give way to **ambulances**

n_2 : Give way to **fire brigade**

n_3 : Give way to **police cars**

n_4 : Give way to **emergency** vehicles



14. SIMON. A complete norm synthesis strategy

Norm refinement: **Generalises** norms when possible, and **specialises** norms when necessary.

1. Norms are **generalised** whenever their effectiveness **and** necessity are **over** a generalisation threshold.
2. Norms are **specialised** whenever their effectiveness **or** necessity are **under** a specialisation threshold.

```
private void normRefinement() {  
  
    /* Add norms that have been rewarded with a negative value */  
    this.addNegRewardedNorms(negRewardedNorms);  
  
    /* Monitor norm utilities to detect utilities passing thresholds */  
    this.checkThresholds();  
  
    /* Specialise norms that under perform */  
    for(Norm norm : this.specialisableNorms) {  
        specialiseDown(norm);  
    }  
  
    /* Generalise norms that may be generalised */  
    for(Norm norm : this.generalisableNorms) {  
        generaliseUp(norm, genMode, genStep);  
    }  
  
    /* Link new norms that have been generalised to  
     * other potential child norms in the normative network */  
    for(Norm normA : this.createdNorms) {  
        for(Norm normB : this.normativeNetwork.getActiveNorms()) {  
            if(!normA.equals(normB))  
                this.searchRelationships(normA, normB, null, visitedNorms);  
        }  
    }  
}
```