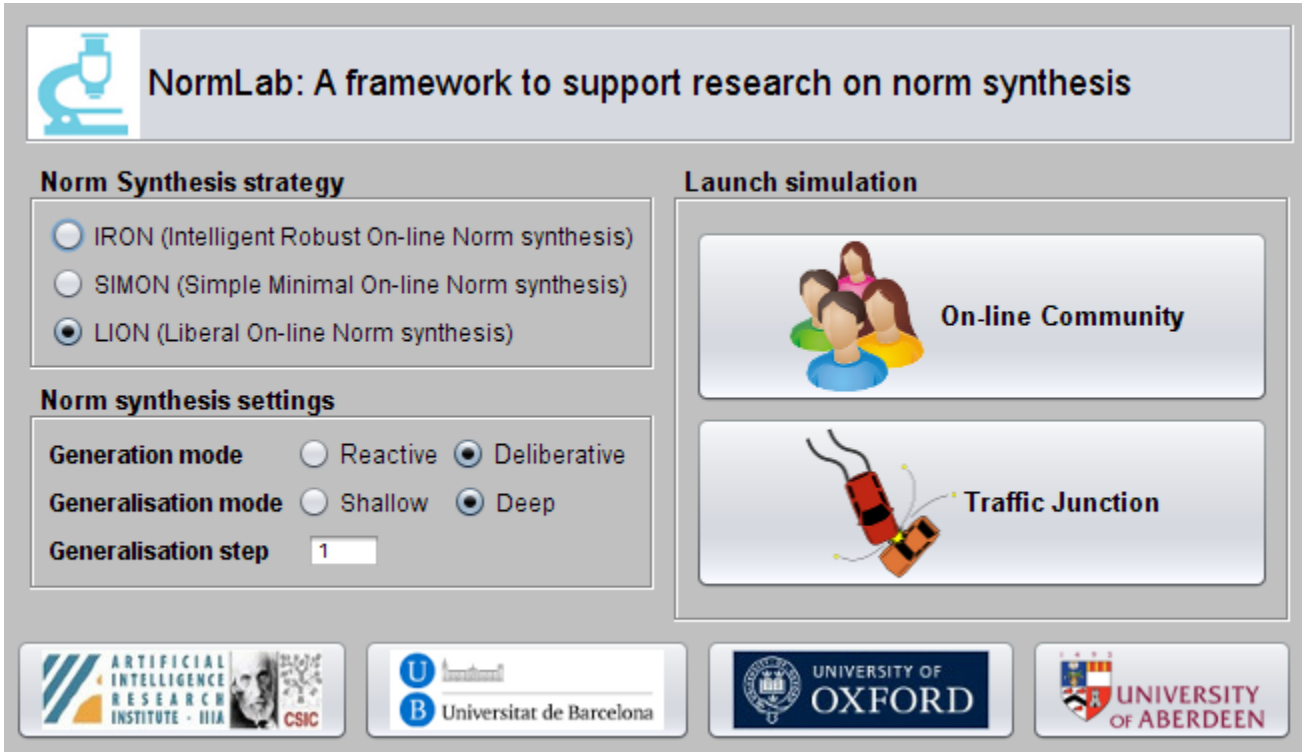
NormLab hands-on Tutorial



Javier Morales (IIIA-UB), Maite López-Sánchez(UB), Juan A. Rodríguez-Aguilar (IIIA-CISC), Michael Wooldridge (UO), Wamberto Vasconcelos (UA)

1. NormLab (Introduction)

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

NormLab allows to:

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

NormLab hands-on tutorial Outline

An **introduction** to NormLab

1. (Introduction to NormLab)
2. NormLab architecture.
3. Norm Synthesis Machine.
4. Traffic simulator.

Configuration of the working environment

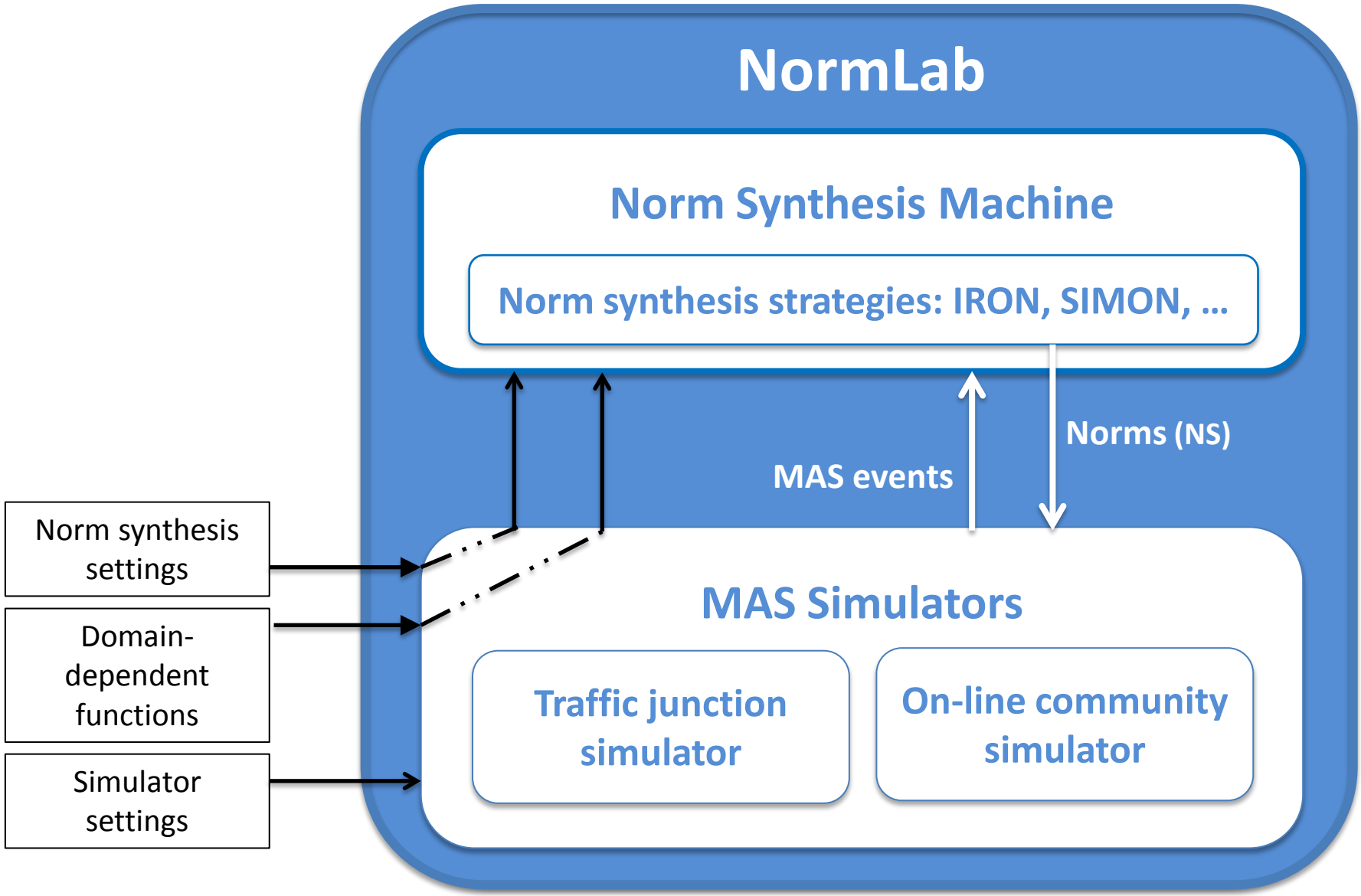
5. NormLab download and installation.

NormLab **execution**:

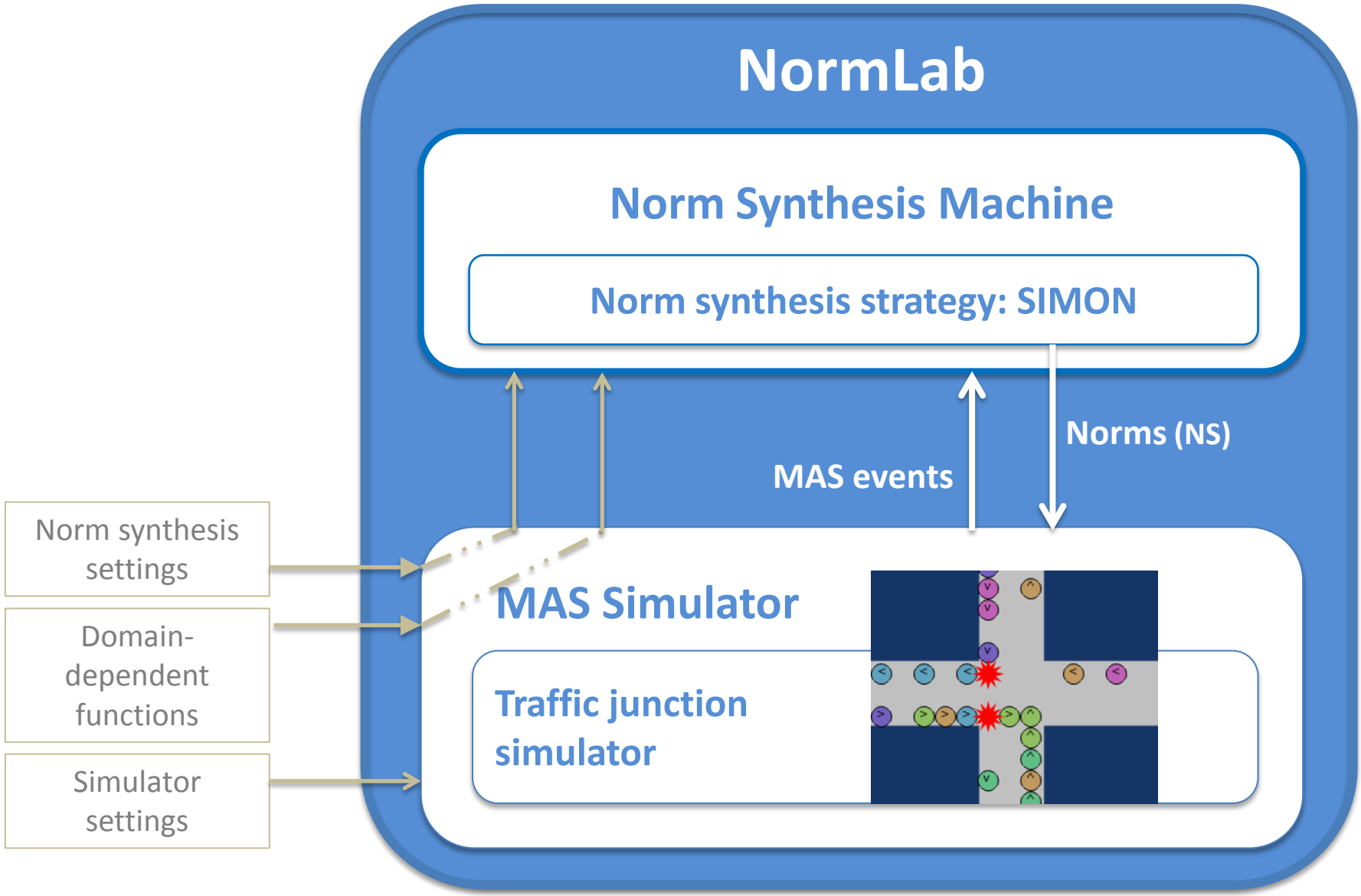
6-8. Execution examples.

9-14. Guided development of different norm synthesis strategies.

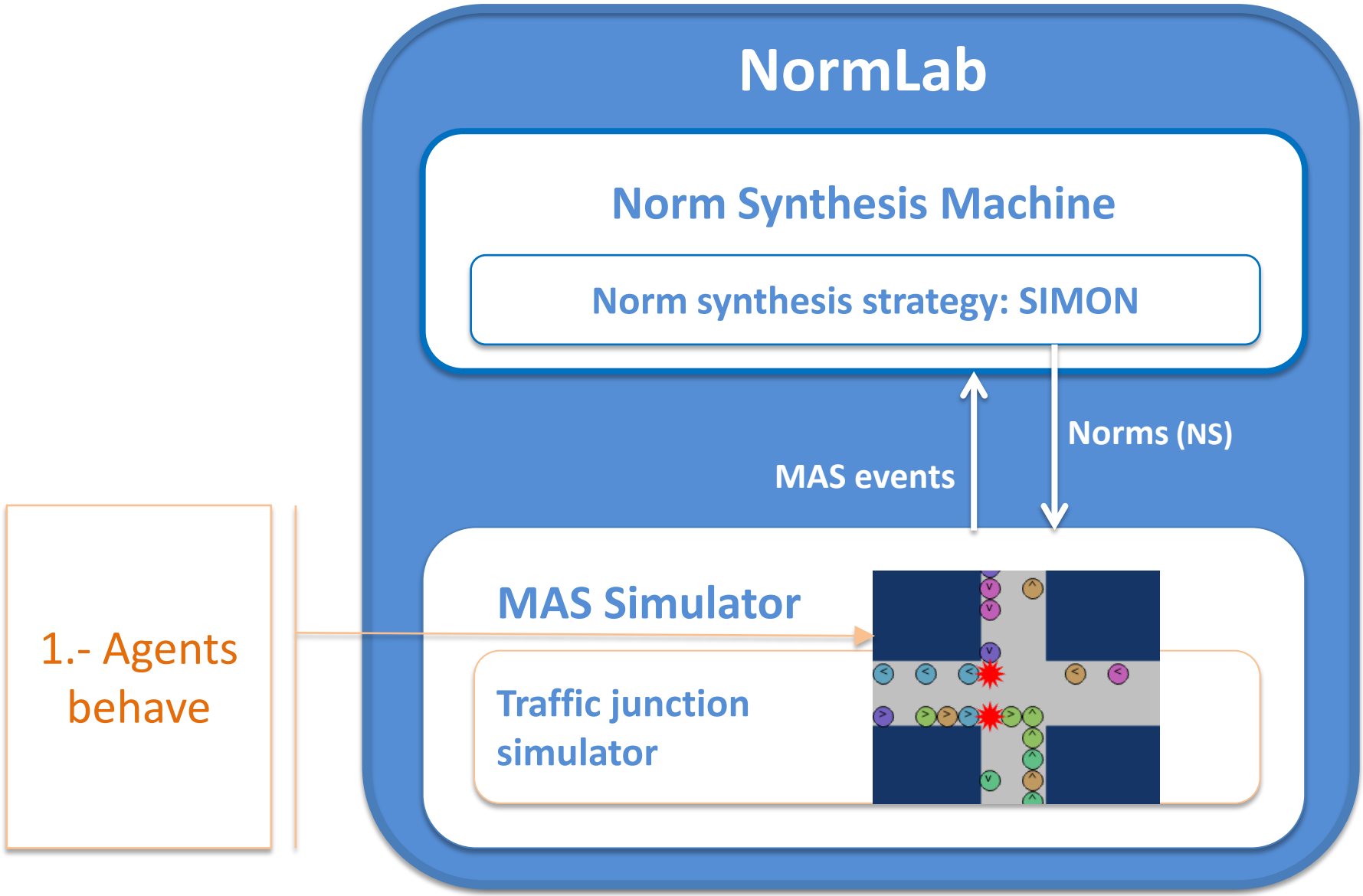
2. NormLab architecture



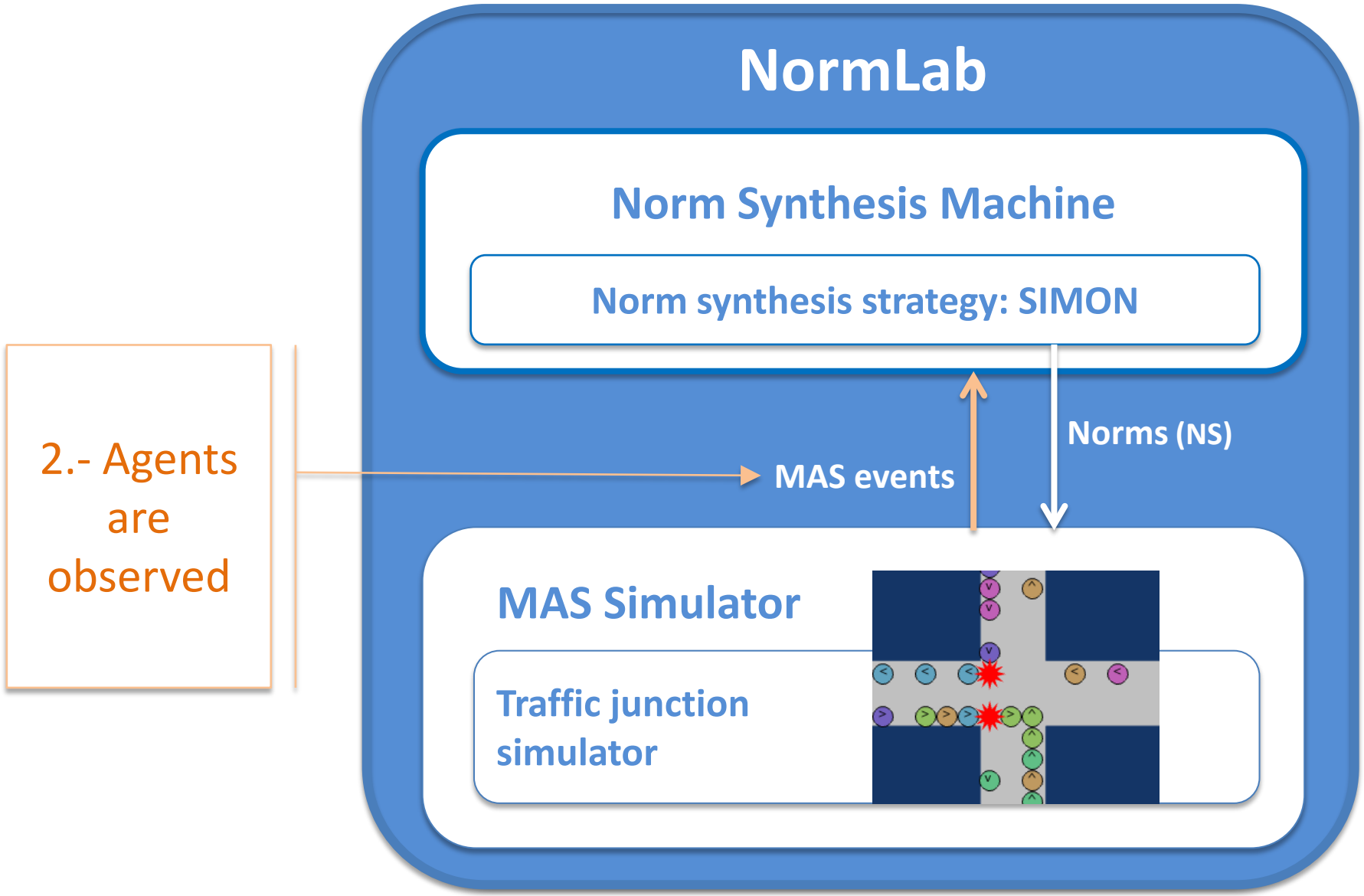
3. Norm Synthesis Machine



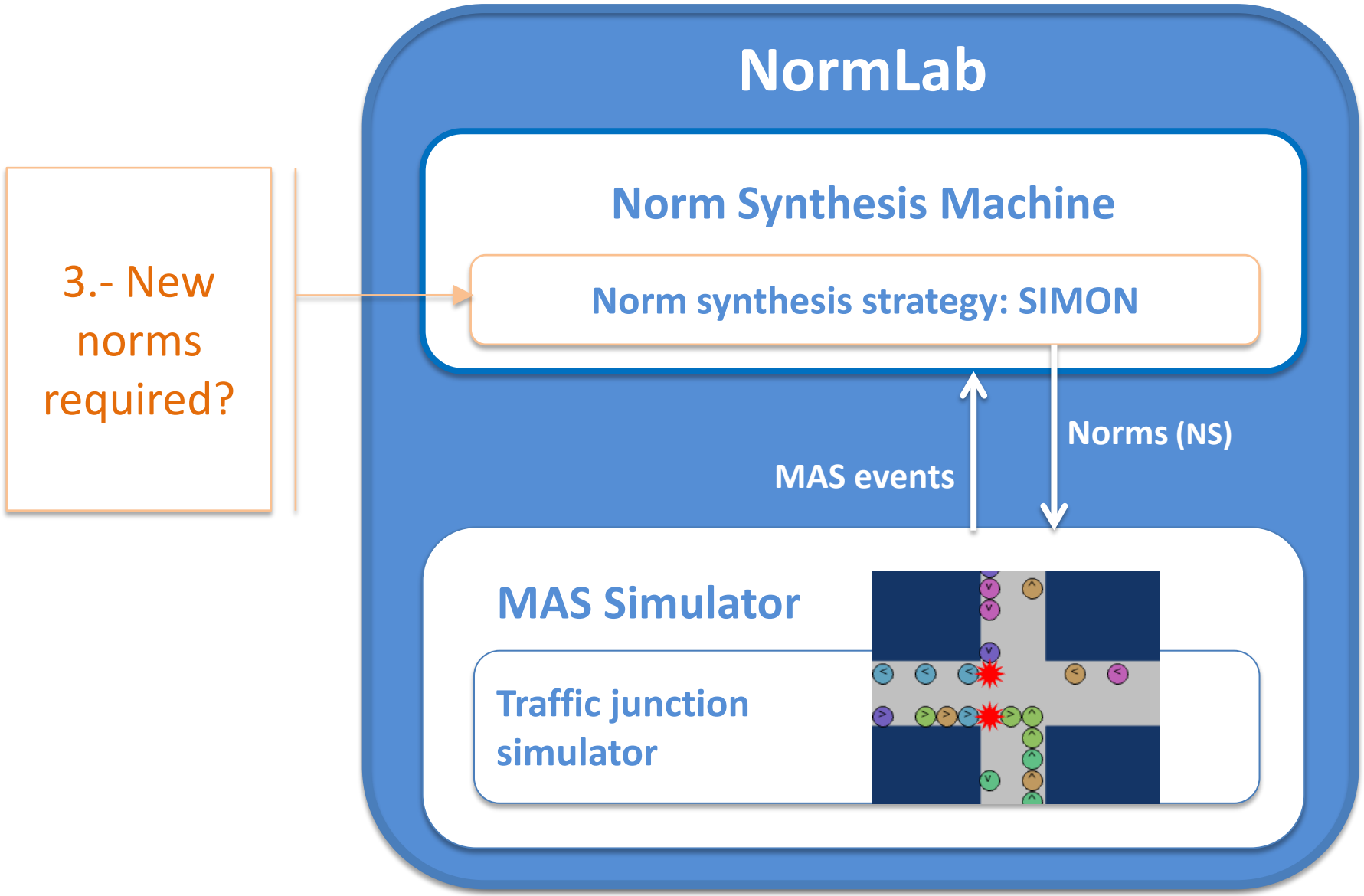
3. Norm Synthesis Machine



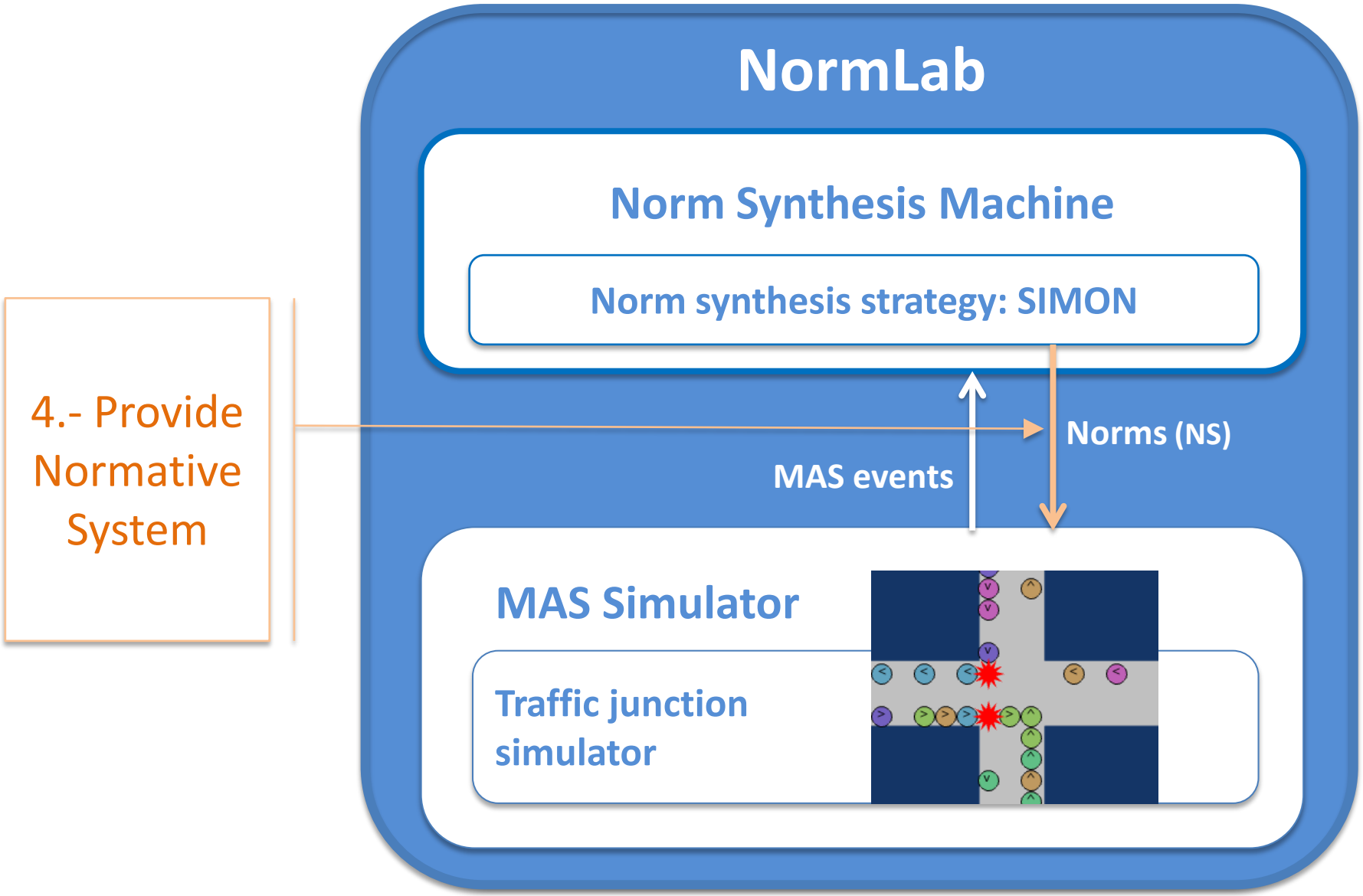
3. Norm Synthesis Machine



3. Norm Synthesis Machine

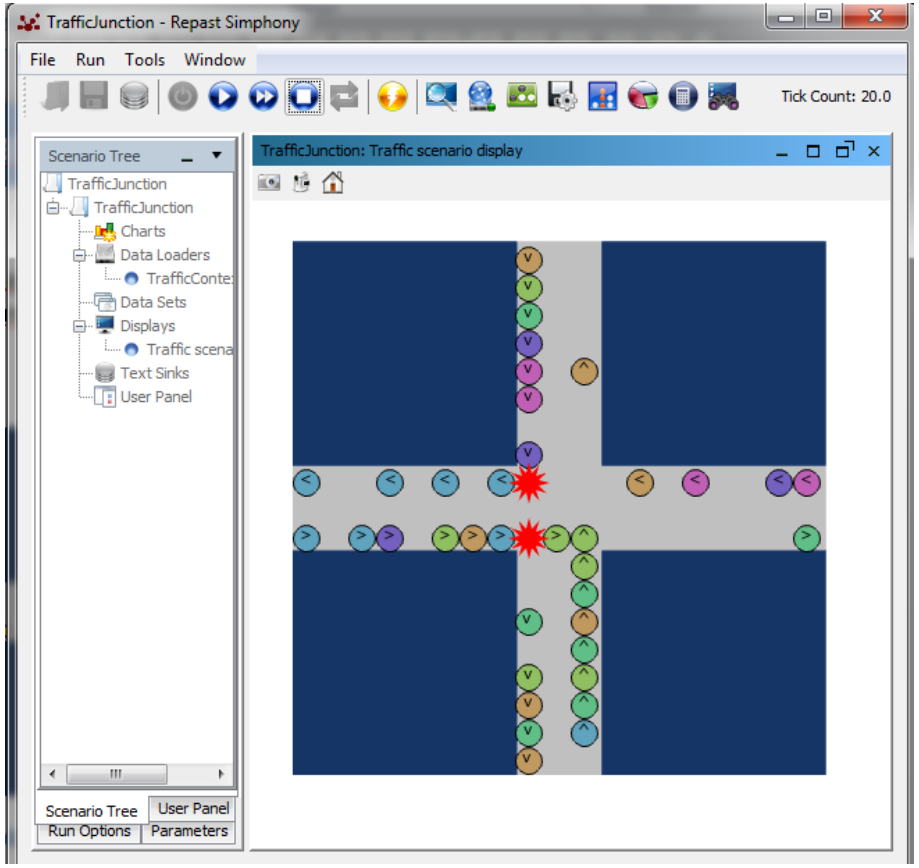


3. Norm Synthesis Machine



4. Traffic simulator

- **Based on Repast Simphony 2.2**
- **Agents** are cars, and **conflicts** are car collisions
- **The goal** is to synthesise normative systems that **avoid collisions** between cars.



NormLab hands-on tutorial Outline

An **introduction** to NormLab

1. (Introduction to NormLab)
2. NormLab architecture.
3. Norm Synthesis Machine.
4. Traffic simulator.

Configuration of the working environment

5. NormLab download and installation.

NormLab **execution**:

6-8. Execution examples.

9-14. 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 later — http://www.java.com
- ***Eclipse IDE*** (just for Linux users) — http://www.eclipse.org/downloads
- ***Repast Simphony 2.2*** — 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.
- ***NormLabSimulators*****: http://normsynthesis.github.io/NormLabSimulators**
  Code of two MAS simulators: traffic and on-line community.

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

5. 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*

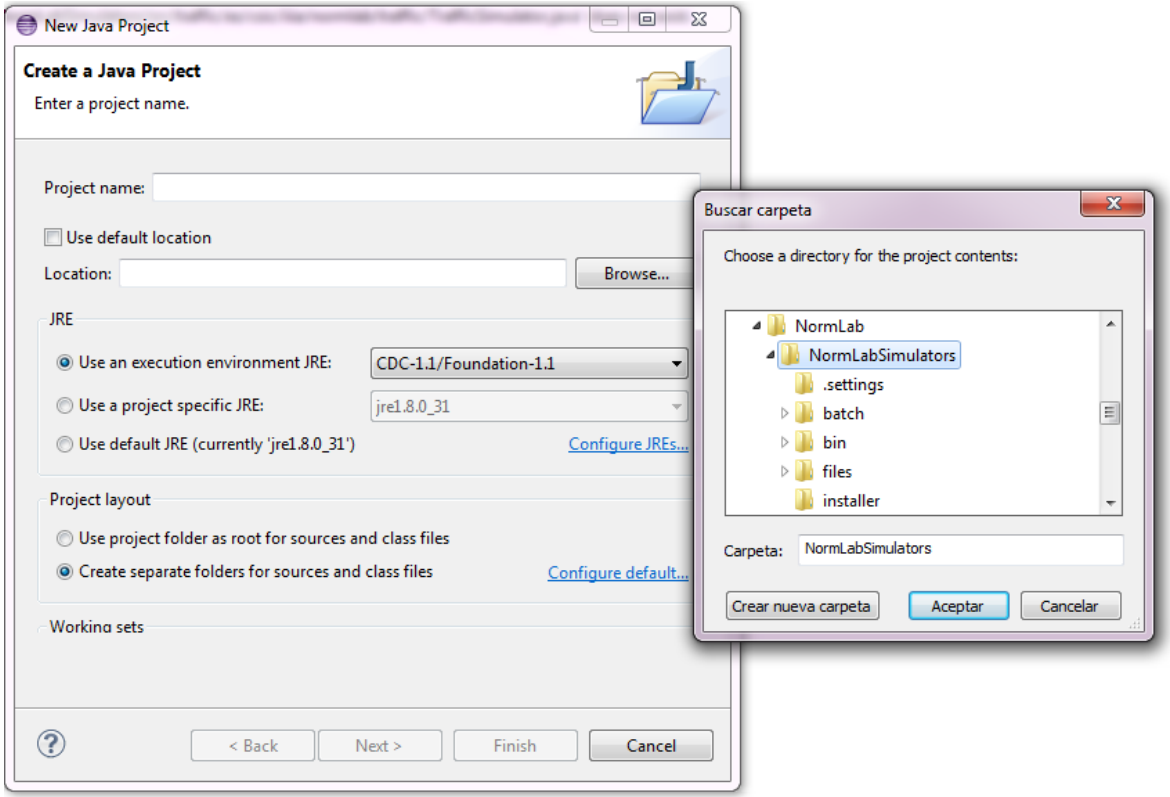
3. Rename both projects, removing the «NormSynthesis» part and the numbers. After renaming them they should look like this:

   - *NormLabSimulators*
   - *NormSynthesisMachine*

5. NormLab installation

Preparing the working environment

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



5. NormLab structure

Before starting you need to know:

NormLabSimulators project is structured as follows:

src/traffic: The code of the traffic simulator.
(src/onlineComm: The code of the on-line community simulator)
launchers: The launchers that allow to run the two simulators.
repast-settings/TrafficJunction.rs: Basic Repast settings for the traffic junction simulator.
(repast-settings/OnlineCommunities.rs: Basic Repast settings for the on-line community simulator)

NormLab hands-on tutorial Outline

An **introduction** to NormLab

1. (Introduction to NormLab)
2. NormLab architecture.
3. Norm Synthesis Machine.
4. Traffic simulator.

Configuration of the working environment

5. NormLab download and installation.

NormLab **execution**:

6-8. Execution examples.

9-14. Guided development of different norm synthesis strategies.

Tutorial outline

NormLab **execution**:

6-8. Execution examples:

6. **Example** strategy 1: Normlab execution: Returns an **empty** set of norms.
7. **Example** strategy 2: Returns a fixed set of **1 norm**.
8. **Example** strategy 3: Returns a fixed set of **3 norms**.

9-14. Guided development of different norm synthesis strategies:

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

Tutorial outline

NormLab **execution**:

6-8. Execution examples

6. **Example** strategy 1: NormLab execution: Returns an **empty** set of norms.
7. **Example** strategy 2: Returns a fixed set of **1 norm**.
8. **Example** strategy 3: Returns a fixed set of **3 norms**.

9-14. Guided development of different norm synthesis strategies

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

6. NormLab Execution: Example 1

TrafficJunction norm synthesis example 1

We are going to execute the ***TrafficJunction*** simulator with the simplest norm synthesis strategy:

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

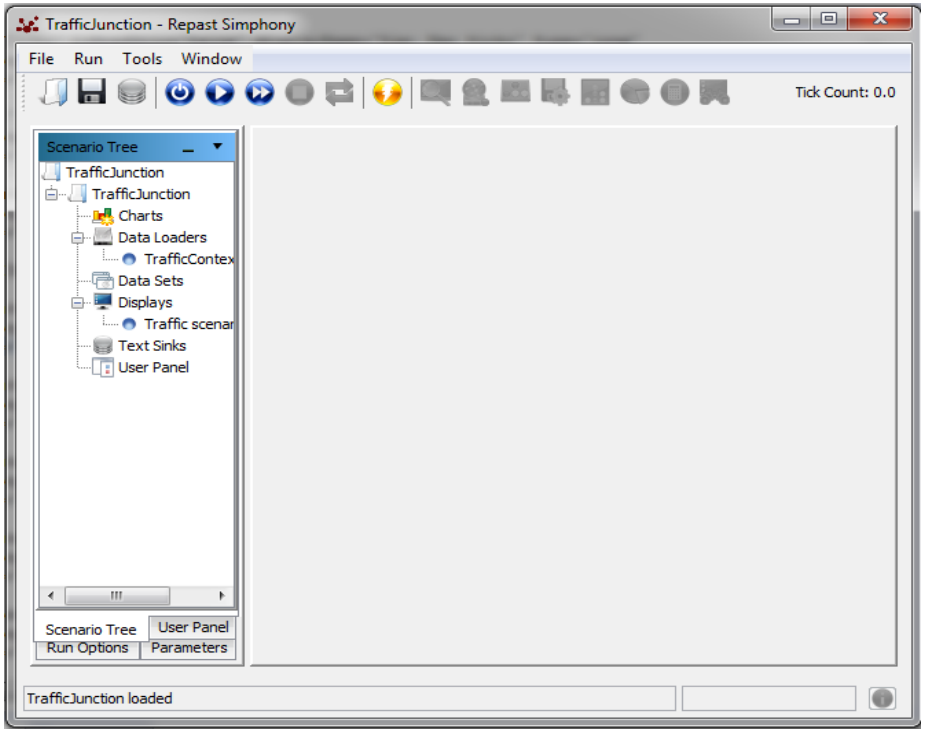
Consequences: No norms are given to the agents → collisions are never avoided.

Note: This execution assumes that file parameters.xml (in directory repast-settings/TrafficJunction.rs within NormLabSimulators project) has parameter «NormSynthesisExample» with field «defaultValue» set to «1»

6. NormLab Execution: Example 1

TrafficJunction norm synthesis example 1

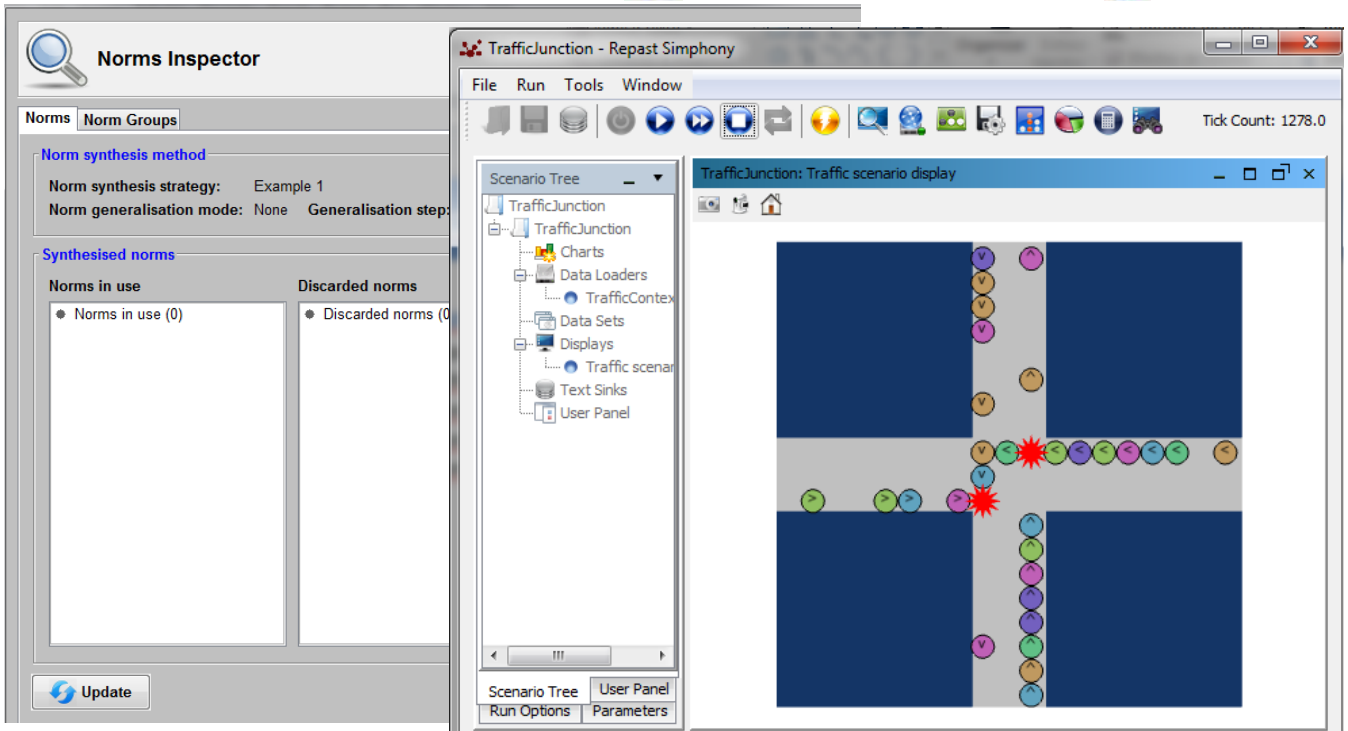
1. In Eclipse, in NormLabSimulators project, go to directory **launchers/**
2. Do right click on the file **TrafficJunctionSimulator.launch**.
3. Click on «Run As» > «TrafficJunctionSimulator».
4. Click on button  to initialise the simulator.



6. NormLab Execution: Example 1

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**:

6-8. Execution examples

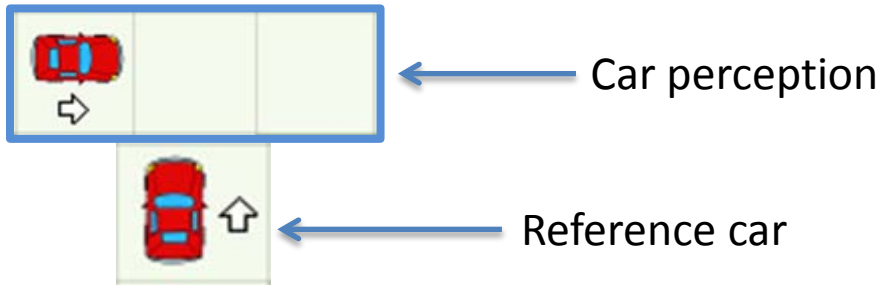
6. **Example** strategy 1: NormLab execution: Returns an **empty** set of norms.
7. **Example** strategy 2: Using norms: Returns a fixed set of **1 norm**.
8. **Example** strategy 3: Returns a fixed set of **3 norms**.

9-14. Guided development of different norm synthesis strategies

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

7. Using norms: Example 2

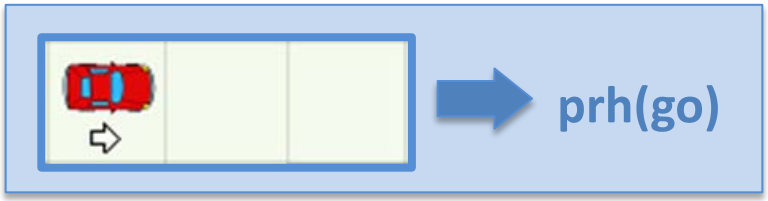
In the traffic simulator, cars' perceptions correspond to the three cells in front of them:



Norms are…

- **IF** … **THEN**… rules.
- Norm precondition: Set of **predicates** with one **term** each.
  - Three predicates **(left, front, right).**
  - Terms **{<, ^, >, v, -, w, *}** represent: cars with {**<**, **^**, **>**, **v**} headings; nothing (-), wall (w) ; and anything (*)
- Norm postcondition: A **modality**.

Graphical representation



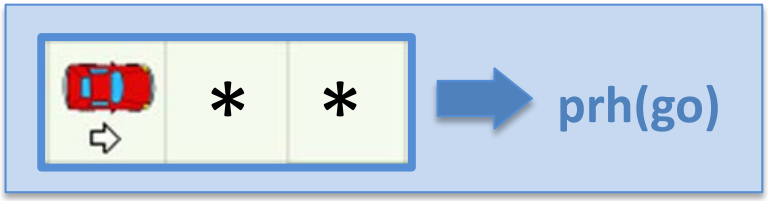
IF left(>) **& front**(-) **& right**(-) **THEN prohibition**(*go*)

7. Using norms: Example 2

TrafficJunction norm synthesis example 2

We will execute the *TrafficJunction* simulator with a norm synthesis strategy that returns a normative system with only **one left-side-priority** norm:

Norm 1



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

It avoids some (but not all) collisions.

7. Using norms: Example 2

TrafficJunction norm synthesis example 2

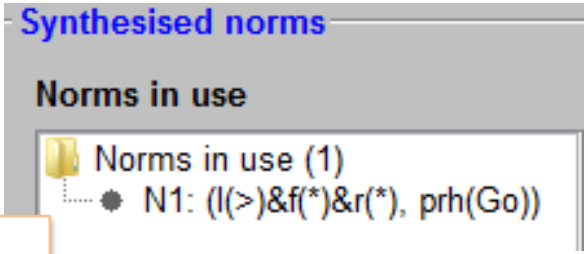
1. In Eclipse, in NormLabSimulators project, go to directory **repast-settings/TrafficJunction.rs**
2. Open file **parameters.xml** by doing right click > *Open with* > *Text Editor.* This file defines the *NormLab* parameters.
3. Search for the parameter «NormSynthesisExample».
4. Set the field «defaultValue»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

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

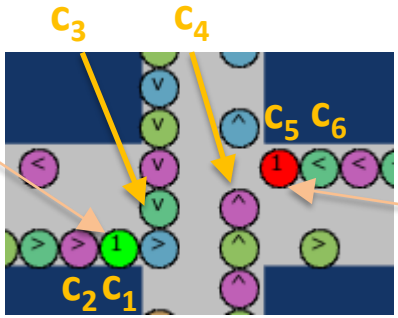
7. Using norms: Example 2

TrafficJunction norm synthesis example 2

6. Do right click on the file **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 norm N1, and cars occasionally stop to conform to it.



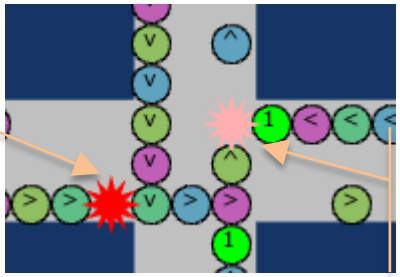
Green cirle: Norm 1 applies and car c_1 stops (c_3 has priority)



Tick i

Red circle: Norm 1 applies but car c_5 does NOT stop

Non regulated collision (between c_1 - c_2)



Tick i +1

Regulated collision (between c_4 - c_5) c_6 complies with N1 (stops)

Tutorial outline

NormLab **execution**:

6-8. Execution examples

6. **Example** strategy 1: NormLab execution: Returns an **empty** set of norms.
7. **Example** strategy 2: Using norms: Returns a fixed set of **1 norm**.
8. **Example** strategy 3: Removing collisions: Returns a fixed set of **3 norms**.

9-14. Guided development of different norm synthesis strategies

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

8. Removing collisions: Example 3

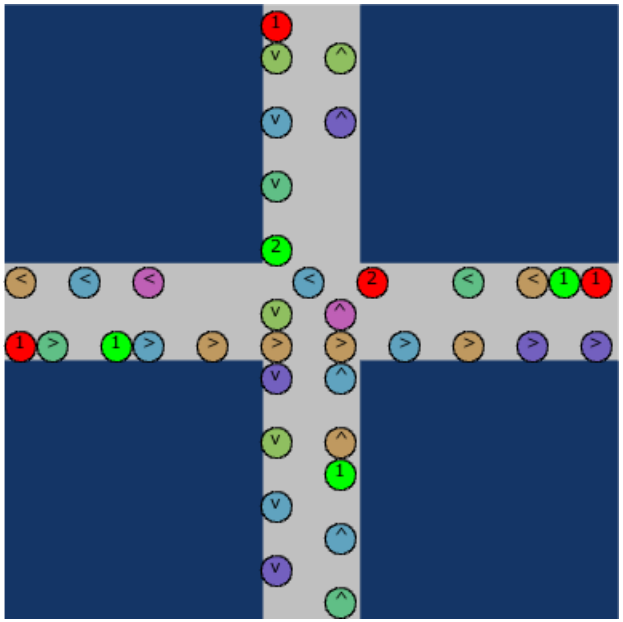
TrafficJunction norm synthesis example 3

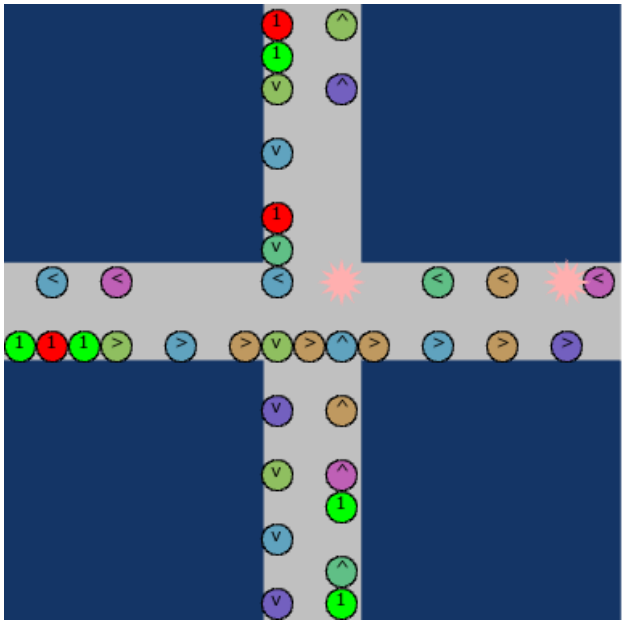
Let's define a norm synthesis strategy that avoids all possible collisions by always returning this Normative System:

N1: IF **left**(*) **&** **front**(^) **&** **right**(*) THEN **prohibition**(*go*)
N2: IF **left**(>) **&** **front**(-) **&** **right**(*) THEN **prohibition**(*go*)
N3: IF **left**(<) **&** **front**(<) **&** **right**(*) THEN **prohibition**(*go*)

Set NormSynthesisExample **defaultValue=«3»** in **parameters.xml** (in NormLabSimulators project, **repast-settings/TrafficJunction.rs**)

Tick i





Tick i+1

Tutorial outline

NormLab **execution**:

6-8. Execution examples

6. **Example** strategy **1**: NormLab execution: Returns an **empty** set of norms.
7. **Example** strategy **2**: Adding norms: Returns a fixed set of **1 norm**.
8. **Example** strategy **3**: Removing collisions: Returns a fixed set of **3 norms**.

9-14. Guided development of different norm synthesis strategies

9. **Development** of example strategy 1: **Empty** set of norms.
10. **Executing** your own strategy
11. **Development** of example strategy 2: **Adding norms** to your strategy (1 norm)
12. **Example 4**: A strategy with norm **generation**.
13. **Example 5**: A strategy with norm **generation** + **evaluation**.
14. **SIMON**: A complete strategy with norm **generation** + **evaluation** + **refinement.**

9. Developing your own strategy

How are all these examples **implemented**? We will now develop our own norm synthesis strategy as the one from example 1, which returns an **empty normative system**.

To do so, we first **parameterise** *NormLab* to use a **custom norm synthesis strategy**:

1. In Eclipse (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 *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=«0».** This will indicate *NormLab* that we will provide a custom norm synthesis strategy.
5. Save the file

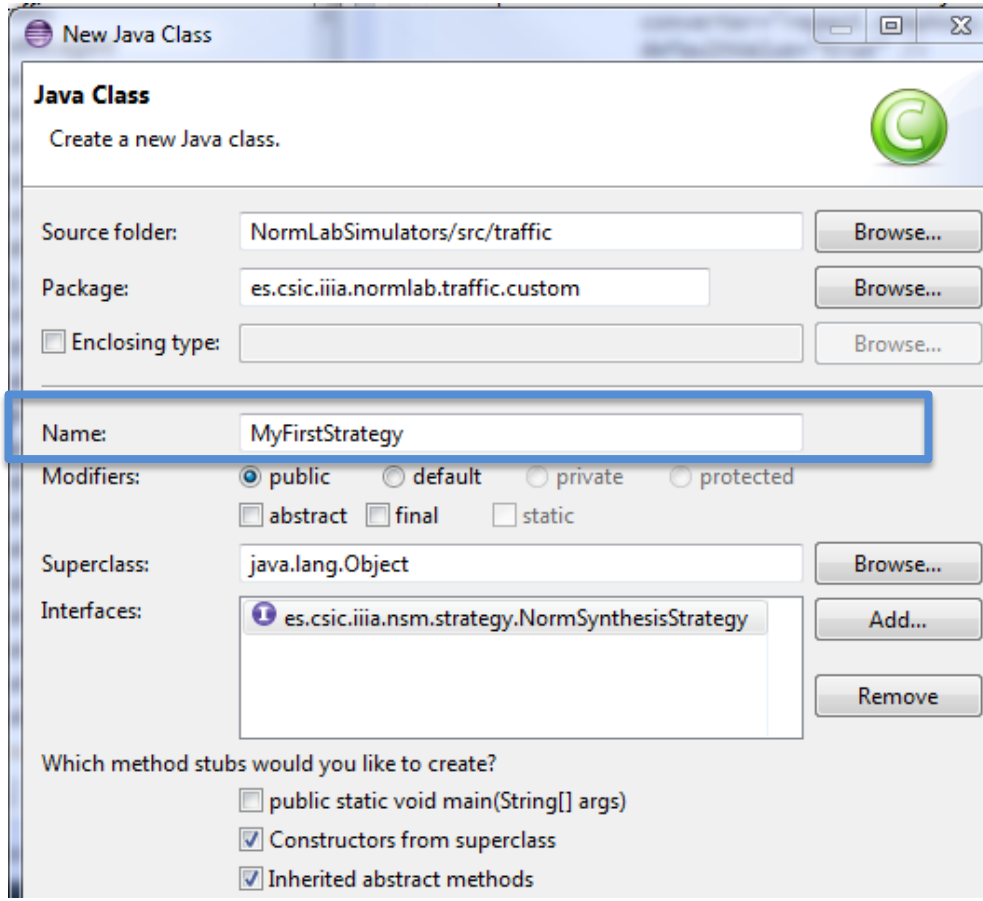
```
<parameter
 name="NormSynthesisExample" isReadOnly="false" displayName="NSM: Norm synthesis example" type="int"
 converter="repast.simphony.parameter.StringConverterFactory$IntConverter"
 defaultValue="0" />
<parameter
 name="NormSynthesisStrategy" isReadOnly="false"
displayName="NSM: Norm synthesis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

9. Developing your own strategy

Now, **create your own norm synthesis strategy** *MyFirstStrategy.java:*

- In NormLabSimulators project, go to package **es.csic.iiia.normlab.traffic.custom** in src/traffic **.**
- There, right-click **New > Class** to create a new Java class *MyFirstStrategy.java* that implements NormSynthesisStragegy interface by:

1.- Naming it MyFirstStrategy



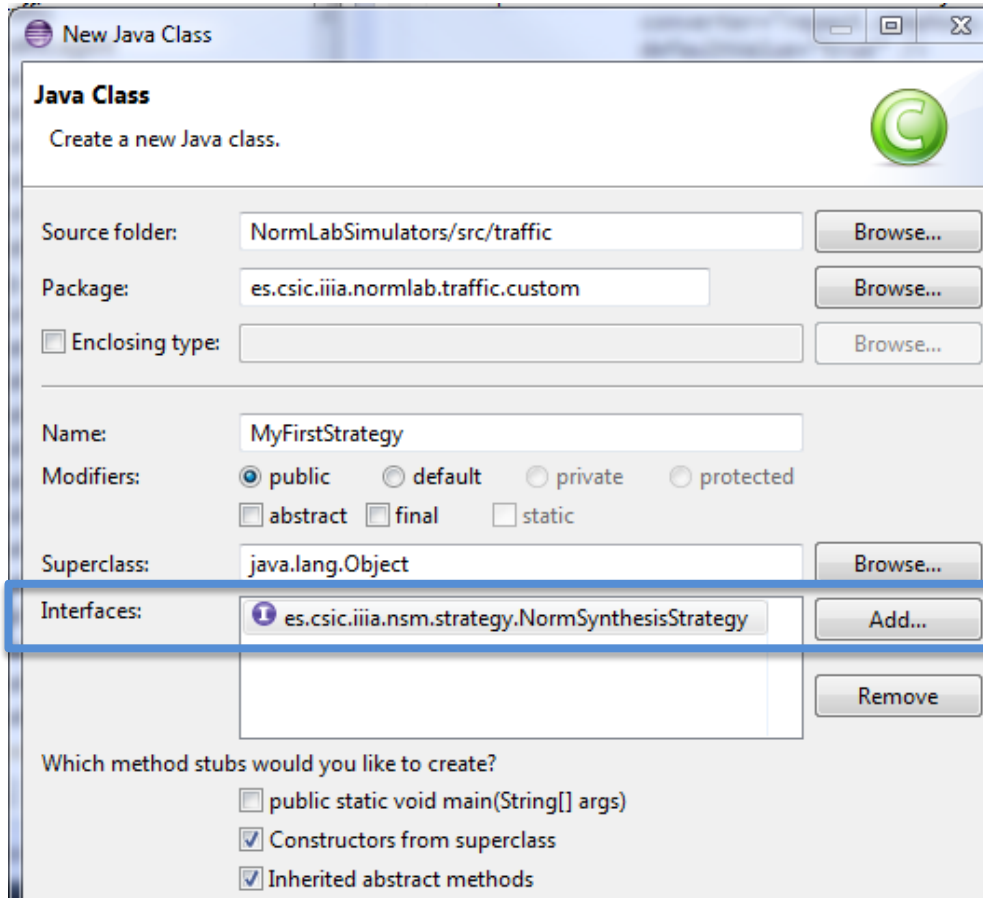
9. Developing your own strategy

Now, **create your own norm synthesis strategy** *MyFirstStrategy.java:*

- In NormLabSimulators project, go to package **es.csic.iiia.normlab.traffic.custom** in src/traffic **.**
- There, right-click **New > Class** to create a new Java class *MyFirstStrategy.java* that implements NormSynthesisStragegy interface by:

1.- Naming it MyFirstStrategy

2.- Adding interface **es.csic.iiia.nsm.strategy.NormSynthesisStrategy**



9. Developing your own strategy

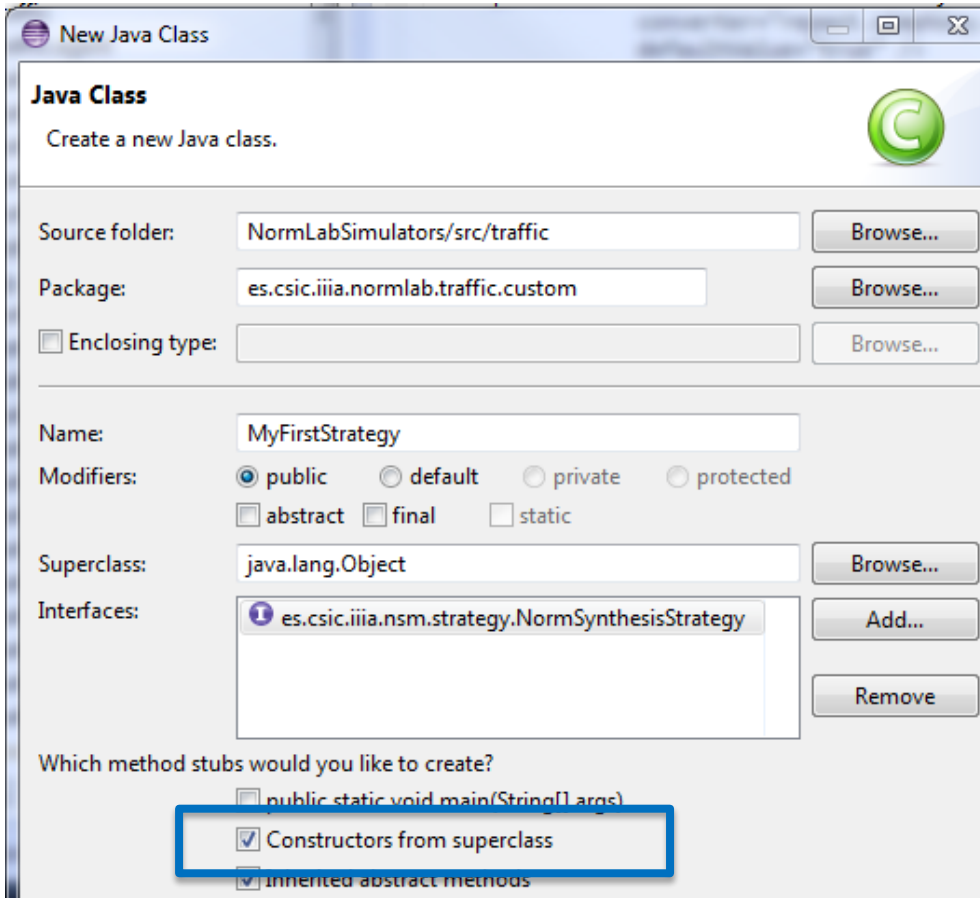
Now, **create your own norm synthesis strategy** *MyFirstStrategy.java:*

- In NormLabSimulators project, go to package **es.csic.iiia.normlab.traffic.custom** in src/traffic **.**
- There, right-click **New > Class** to create a new Java class *MyFirstStrategy.java* that implements NormSynthesisStragegy interface by:

1.- Naming it MyFirstStrategy

2.- Adding interface
es.csic.iiia.nsm.strategy.NormSynthesisStrategy

3.- Cheking the constructor creation



9. Developing your own strategy

Now, **create your own norm synthesis strategy** *MyFirstStrategy.java:*

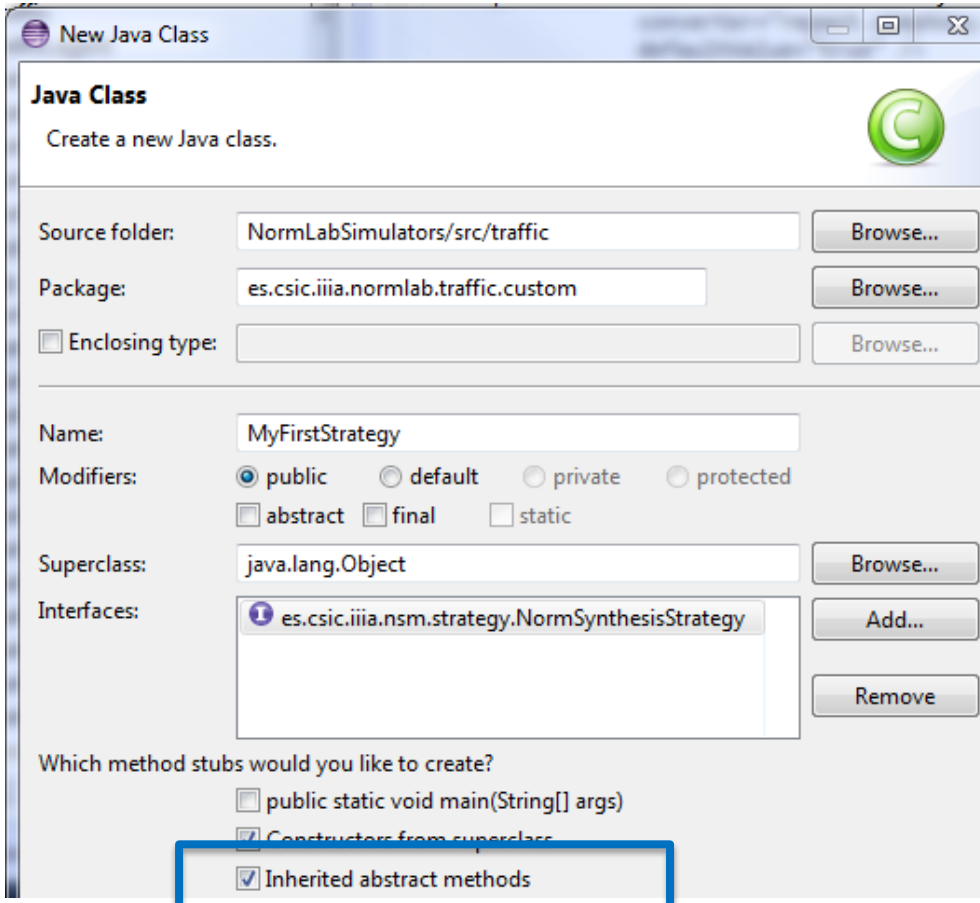
- In NormLabSimulators project, go to package **es.csic.iiia.normlab.traffic.custom** in src/traffic **.**
- There, right-click **New > Class** to create a new Java class *MyFirstStrategy.java* that implements NormSynthesisStragegy interface by:

1.- Naming it MyFirstStrategy

2.- Adding interface **es.csic.iiia.nsm.strategy.NormSynthesisStrategy**

3.- Cheking the constructor creation

4.- Creating inherited abstract method **execute()** (check "Inherited abstract methods")




```java
package es.csic.iiia.normlab.traffic.custom;
import es.csic.iiia.nsm.norm.NormativeSystem;
import es.csic.iiia.nsm.strategy.NormSynthesisStrategy;

public class MyFirstStrategy implements NormSynthesisStrategy {

  public MyFirstStrategy() {
    // TODO Auto-generated constructor stub
  }

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

9. Developing your own strategy

And **implement** the **norm synthesis strategy** class:

1. In the class, add a Normative Network attribute :
   private NormativeNetwork normativeNetwork;

The Norm Synthesis Machine contains the **Normative Network** which includes the Normative System:

- Normative Network: contains all synthesised norms.
- Normative System: set of (active) norms given to the agents.

```
package es.csic.iiia.normlab.traffic.custom;
import es.csic.iiia.nsm.net.norm.NormativeNetwork;
import es.csic.iiia.nsm.norm.NormativeSystem;
import es.csic.iiia.nsm.strategy.NormSynthesisStrategy;

public class MyFirstStrategy implements NormSynthesisStrategy {

  /* Normative Network: a data structure to keep synthesised norms*/
  private NormativeNetwork nomativeNetwork;

  /** Constructor of the strategy
   * @param nsm*/
  public MyFirstStrategy(es.csic.iiia.nsm.NormSynthesisMachine nsm) {
    /* Get Normative Network*/
    this.nomativeNetwork=nsm.getNormativeNetwork();
  }

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

9. Developing your own strategy

And **implement** the **norm synthesis strategy** class:

1. In the class, add a Normative Network attribute :
   private NormativeNetwork normativeNetwork;
2. In the constructor, add the parameter ***es.csic.iiia.nsm.NormSynthesisMachine nsm*** and use it to initialize (to empty) the Normative Network attribute:
   this.normativeNetwork = nsm.getNormativeNetwork();

The Norm Synthesis Machine contains the **Normative Network** which includes the Normative System:

- Normative Network: contains all synthesised norms.
- Normative System: set of (active) norms given to the agents.

```
package es.csic.iiia.normlab.traffic.custom;
import es.csic.iiia.nsm.net.norm.NormativeNetwork;
import es.csic.iiia.nsm.norm.NormativeSystem;
import es.csic.iiia.nsm.strategy.NormSynthesisStrategy;

public class MyFirstStrategy implements NormSynthesisStrategy {

  /* Normative Network: a data structure to keep synthesised norms*/
  private NormativeNetwork nomativeNetwork;

  /** Constructor of the strategy
   * @param nsm*/
  public MyFirstStrategy(es.csic.iiia.nsm.NormSynthesisMachine nsm) {
    /* Get Normative Network*/
    this.nomativeNetwork=nsm.getNormativeNetwork();
  }

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

9. Developing your own strategy

And **implement** the **norm synthesis strategy** class:

1. In the class, add a Normative Network attribute :
   private NormativeNetwork normativeNetwork;
2. In the constructor, add the parameter ***es.csic.iiia.nsm.NormSynthesisMachine nsm*** and use it to initialize (to empty) the Normative Network attribute:
   this.normativeNetwork = nsm.getNormativeNetwork();

The Norm Synthesis Machine contains the **Normative Network** which includes the Normative System:

- Normative Network: contains all synthesised norms.
- Normative System: set of (active) norms given to the agents.

3. Strategy execution: return the empty **Normative System** in method **execute()**:

return this.normativeNetwork.getNormativeSystem();

```
package es.csic.iiia.normlab.traffic.custom;
import es.csic.iiia.nsm.net.norm.NormativeNetwork;
import es.csic.iiia.nsm.norm.NormativeSystem;
import es.csic.iiia.nsm.strategy.NormSynthesisStrategy;

public class MyFirstStrategy implements NormSynthesisStrategy {

  /* Normative Network: a data structure to keep synthesised norms*/
  private NormativeNetwork nomativeNetwork;

  /** Constructor of the strategy
   * @param nsm*/
  public MyFirstStrategy(es.csic.iiia.nsm.NormSynthesisMachine nsm) {
    /* Get Normative Network*/
    this.nomativeNetwork=nsm.getNormativeNetwork();
  }

  /* Execute the strategy*/
  @Override
  public NormativeSystem execute() {
    return this.nomativeNetwork.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 es.csic.iiia.nsm.net.norm.NormativeNetwork;
import es.csic.iiia.nsm.norm.NormativeSystem;

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

  /* The normative network, a data structure to keep track of norms */
  private NormativeNetwork normativeNetwork;

  /**
   * Constructor of the strategy
   *
   * @param nsm
   */
  public MyFirstStrategy(es.csic.iiia.nsm.NormSynthesisMachine nsm) {

    /* Get normative network */
    this.normativeNetwork = nsm.getNormativeNetwork();
  }

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

Tutorial outline

NormLab **execution**:

6-8. Execution examples

6. **Example** strategy **1**: NormLab execution: Returns an **empty** set of norms.
7. **Example** strategy **2**: Adding norms: Returns a fixed set of **1 norm**.
8. **Example** strategy **3**: Removing collisions: Returns a fixed set of **3 norms**.

9-14. Guided development of different norm synthesis strategies

9. **Development** of example strategy 1: **Empty** set of norms.
10. **Invoking** your strategy
11. **Development** of example strategy 2: **Adding norms** to your strategy (1 norm)
12. **Example 4**: A strategy with norm **generation**.
13. **Example 5**: A strategy with norm **generation** + **evaluation**.
14. **SIMON**: A complete strategy with norm **generation** + **evaluation** + **refinement.**

10. Invoking your strategy

But, how does *NormLab* invoke our new norm synthesis strategy?
The Traffic Simulator includes (in package **es.csic.iiia.normlab.traffic.agent**) an agent **DefaultTrafficNormSynthesisAgent** whose:

A. **Constructor** creates the **Norm Synthesis Machine** and configures it to use **our strategy**
B. **step()** method invokes our strategy at every simulation **tick**.

```
public DefaultTrafficNormSynthesisAgent(List<TrafficCamera> cameras,
    PredicatesDomains predDomains, DomainFunctions dmFunctions,
    NormSynthesisSettings nsSettings, long randomSeed) {
```

A

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

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

B

10. Invoking your strategy (A)

Specifically, the constructor **(A) DefaultTrafficNormSynthesisAgent()** is in charge of:

1. Creating the norm synthesis machine.
2. Adding a set of sensors to the norm synthesis machine in order to perceive the scenario.
3. Setting the norm synthesis strategy.

```
public DefaultTrafficNormSynthesisAgent(List<TrafficCamera> cameras,
    PredicatesDomains predDomains, DomainFunctions dmFunctions,
    NormSynthesisSettings nsSettings, long randomSeed) {

  this.nsSettings = nsSettings;

  this.normativeSystem = new NormativeSystem();
  this.addedNorms = new ArrayList<Norm>();
  this.removedNorms = new ArrayList<Norm>();

  /* 1. Create norm synthesis machine */
  this.nsm = new NormSynthesisMachine(nsSettings, predDomains,
      dmFunctions, !RunEnvironment.getInstance().isBatch(), randomSeed);

  /* 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();
}
```

1
2
3
A

10. Invoking your strategy (A.1)

The invocation to the constructor of the NormSynthesisMachine (**A.1**) requires :

i. **NormSynthesisSettings**: The settings for the norm synthesis machine.
ii. **PredicatesDomains**: Agents' language: predicates and terms describing the scenario from the agents' local point of view.
iii. **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, DomainFunctions dmFunctions,
    NormSynthesisSettings nsSettings, long randomSeed) {

  this.nsSettings = nsSettings;

  this.normativeSystem = new NormativeSystem();
  this.addedNorms = new ArrayList<Norm>();
  this.removedNorms = new ArrayList<Norm>();

  /* 1. Create norm synthesis machine */
  this.nsm = new NormSynthesisMachine(nsSettings, predDomains,
      dmFunctions, !RunEnvironment.getInstance().isBatch(), randomSeed);

  /* 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();
}
```

i ii iii 1 A

10. Invoking your strategy (A.1.i)

NormSynthesisSettings (A.1.i) : An interface to be implemented (located in package **es.csic.iiia.nsm.config** in **NormSynthesisMachine** project)

- **getNormSynthesisStrategy()**: Returns the norm synthesis strategy to use.
- getSystemGoals(): A list of system goals. In traffic, the only goal is "to avoid collisions".
- isNormGenerationReactiveToConflicts(): True if NSM tries to add a new norm upon the detection of each non-regulated conflict. False if it creates the nom but does not add it to the Normative System immediately.
- getNormsDefaultUtility(): Norms' default utility (0.5 by default).
- getNormEvaluationLearningRate(): The α rate in IRON and SIMON to evaluate norms (0.1 recom.).
- getNormsPerformanceRangesSize(): The size of the window to compute norms' performance ranges.
- getNormGeneralisationMode(): SIMON's norm generalisation mode (Shallow/Deep).
- public int getNormGeneralisationStep(): SIMON's norm generalisation step: number of norm predicates that can be simultaneously generalised.
- getGeneralisationBoundary(Dimension dim, Goal goal): Minimum value of effectiveness/necessity that a norm's performance must reach to be generalised. It corresponds to the threshold α_{gen} in [1].
- getSpecialisationBoundary(Dimension dim, Goal goal): Value of Effectiveness/necessity under which a norm can be specialised. It corresponds to the threshold α_{spec} described in [1].
- getSpecialisationBoundaryEpsilon(Dimension dim, Goal goal): LION's epsilon to create, together with the specialisation boundaries, a norm deactivation band.
- getNumTicksOfStabilityForConvergence(): Number of simulation ticks without conflicts nor changes in the normative system to converge.

An **implementation** of these settings for the traffic simulator is located in (NormLabSimulators project, src/traffic) package **es.csic.iiia.normlab.traffic.normsynthesis,** in class *TrafficNormSynthesisSettings*

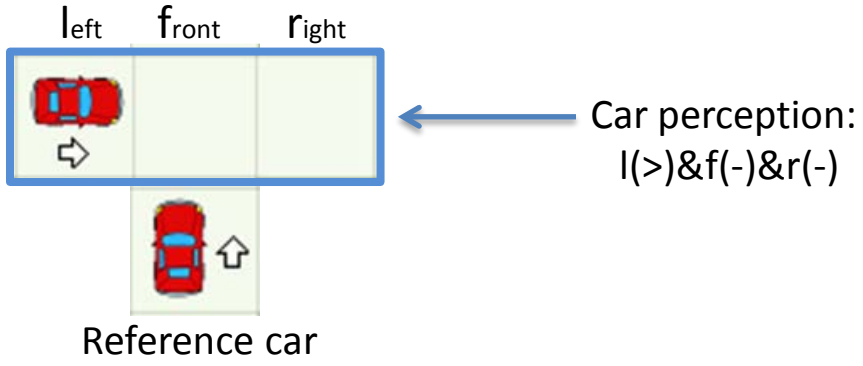
[1] Minimality and Simplicity in the On-line Automated Synthesis of Normative Systems. Javier Morales; Maite López-Sánchez; Juan A. Rodríguez-Aguilar; Michael Wooldridge; Wamberto W. Vasconcelos. AAMAS '14:, p.109-116 (2014)

10. Invoking your strategy (A.1.ii)

PredicatesDomains (A.1.ii) : Contains the predicates and terms that the agents employ to describe the MAS from their local point of view. Located in package **es.csic.iiia.nsm.agent.language** (NormSynthesisMachine project, src/)**.**

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

- Three different predicates **(l, f, r)** that represent the left, front and right positions in front of a car.
- Seven different terms {**<, ^, >, v, -, *, w**} representing: cars with different headings **{<, ^, >, v}**, nothing (-), anything (*), and wall (**w**).



10. Invoking your strategy (A.1.ii)

PredicatesDomains (A.1.ii) : class 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. Invoking your strategy (A.1.iii)

DomainFunctions (A.1.iii) : An interface to be implemented. Located in package **es.csic.iiia.nsm.config** (NormSynthesisMachine project, src/).

- **isConsistent(**SetOfPredicatesWithTerms agentContext**):** Returns true if a set of predicates with terms is consistent with the domain scenario. E.g.: (left(>),front(-),right(-)) is consistent (possible) but (left(>),front(<),right(-)) is not consistent, since two cars can not drive in opposite directions in the same lane.
- **agentContextFunction(**long agentId, View view**):** Returns the local perception of a given agent (i.e., its context) from the observation (view) of the state of the simulated scenario.
- **agentActionFunction(**long agentId,ViewTransition viewTransition**):** Returns a list of actions performed by an agent in the transition from a state $\mathbf{s_t}$ to a state $\mathbf{s_{t-1}}$
- **getConflicts(**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).
- **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, src/traffic) **es.csic.iiia.normlab.traffic.normsynthesis**, *TrafficDomainFunctions* class.

10. Invoking your strategy (recap)

The Traffic Simulator includes **DefaultTrafficNormSynthesisAgent** agent whose:

A. Constructor

1. Creates the Norm Synthesis Machine (NSM).
2. Adds a set of sensors to SNM to perceive the scenario.
3. Sets the norm synthesis strategy in the NSM.

B. step() method invokes our strategy at every simulation **tick**.

```
public DefaultTrafficNormSynthesisAgent( ···) {
  ...
  /* 1. Create norm synthesis machine */
  this.nsm = new NormSynthesisMachine(nsSettings, predDomains,
      dmFunctions, !RunEnvironment.getInstance().isBatch(), randomSeed);

  /* 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();
}
```

1 2 3 — A

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

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

B

10. Invoking your strategy (A.3, B)

The Traffic Simulator includes **DefaultTrafficNormSynthesisAgent** agent whose:

A. Constructor

1. Creates the Norm Synthesis Machine (NSM).
2. Adds a set of sensors to SNM to perceive the scenario.
3. Sets the norm synthesis strategy in the NSM: Method SetNormSynthesisStrategy() invokes method **createCustomNormSynthesisStrategy()**
   (located in the same class *DefaultTrafficNormSynthesisAgent*):
   - **Implement** this method by creating and returning your norm synthesis strategy:

```
/**
 * Sets a custom norm synthesis strategy
 */
protected NormSynthesisStrategy createCustomNormSynthesisStrategy() {
  return new MyFirstStrategy(nsm);
}
}
```

B. step() method invokes our strategy at every simulation **tick**.

- Execute the simulation as you did for examples 1, 2 and 3 (NormLabSimulators project, launchers/: TrafficJunctionSimulator.launch > Run As …)

Congratulations! You are using your own strategy!

Tutorial outline

NormLab **execution**:

6-8. Execution examples

6. **Example** strategy **1**: NormLab execution: Returns an **empty** set of norms.
7. **Example** strategy **2**: Adding norms: Returns a fixed set of **1 norm**.
8. **Example** strategy **3**: Removing collisions: Returns a fixed set of **3 norms**.

9-14. Guided development of different norm synthesis strategies

9. **Development** of example strategy 1: **Empty** set of norms.
10. **Executing** your own strategy
11. **Development** of example strategy 2: **Adding norms** to your strategy (1 norm)
12. **Example 4**: A strategy with norm **generation**.
13. **Example 5**: A strategy with norm **generation** + **evaluation**.
14. **SIMON**: A complete strategy with norm **generation** + **evaluation** + **refinement.**

11. Adding norms to your strategy

Let's now **add some norms**. We will add the left-side-priority norm from **example 2**.

1. **Crate** a new norm synthesis strategy *MySecondStrategy.java* by **Copying** (cut&paste+rename) your first strategy *MyFirstStrategy.java*
   Your code should look like this:

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

  /* The normative network, a data structure to keep track of norms */
  private NormativeNetwork normativeNetwork;

  /**
   * Constructor of the strategy
   *
   * @param nsm the norm synthesis machine
   */
  public MySecondStrategy(es.csic.iiia.nsm.NormSynthesisMachine nsm) {
    this.normativeNetwork = nsm.getNormativeNetwork();
  }

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

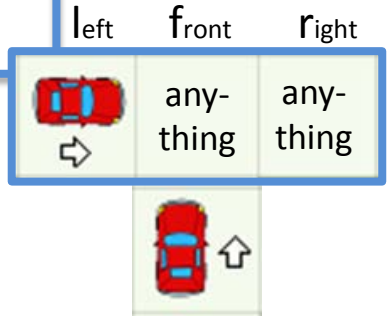
11. Adding norms to your strategy

2. Implement a method **createNormativeSystem()** in *MySecondStrategy.java* to create norms with:

- Preconditions: a set of predicate-term pairs and
- Postconditions: a modality (prohibition/obligation) over an action

i. Create a new norm precondition: **IF l(>) & f(*) & r(*)**

```
private void createNormativeSystem() {
  /* 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);

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



11. Adding norms to your strategy

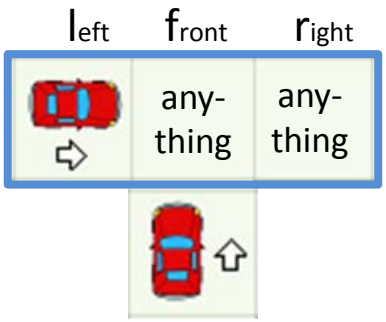
2. Implement a method **createNormativeSystem()** in *MySecondStrategy.java* to create norms with:

- Preconditions: a set of predicate-term pairs and
- Postconditions: a modality (prohibition/obligation) over an action

i. Create a new norm precondition: **IF l(>) & f(*) & r(*)**

ii. Create a **new norm n1** with this precondition and as postcondition: **THEN Prohition(Go)**

```
private void createNormativeSystem() {

  /* 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);

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



11. Adding norms to your strategy

2. Implement a method **createNormativeSystem()** in *MySecondStrategy.java* to create norms with:

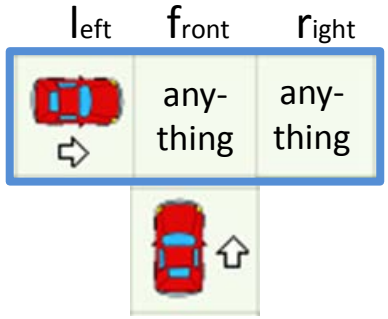
- Preconditions: a set of predicate-term pairs and
- Postconditions: a modality (prohibition/obligation) over an action

i. Create a new norm precondition: **IF l(>) & f(*) & r(*)**

ii. Create a **new norm n1** with this precondition and as postcondition: **THEN Prohition(Go)**

iii. Add norm n1 to the Normative Network and activate it so it becomes part of the Normative System

```
private void createNormativeSystem() {

  /* 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);

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



11. Adding norms to your strategy

3. Invoke method **createNormativeSystem()** at the end of *MySecondStrategy* constructor

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

  /* The normative network, a data structure to keep track of norms */
  private NormativeNetwork normativeNetwork;

  /**
   * Constructor of the strategy
   *
   * @param nsm the norm synthesis machine
   */
  public MySecondStrategy(es.csic.iiia.nsm.NormSynthesisMachine nsm) {
    this.normativeNetwork = nsm.getNormativeNetwork();

    this.createNormativeSystem(); // Create a default normative system
  }

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

  /**
   * Creates a normative system to give way to the cars on the left
   */
  private void createNormativeSystem() {
    /* 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);

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

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

11. Adding norms to your strategy

4. Change method **createCustomNormSynthesisStrategy()** from *DefaultTrafficNormSynthesisAgent* (in package **es.csic.iiia.normlab.traffic.agent**, NormLabSimulators project, src/traffic) to use your new strategy.

```
/**
 * Sets a custom norm synthesis strategy
 */
protected NormSynthesisStrategy createCustomNormSynthesisStrategy() {
  return new MySecondStrategy(nsm);
}
```

   - Recall that the traffic norm synthesis agent in the traffic simulator creates the norm synthesis machine and executes the strategy at every simulation tick.

5. Execute the Traffic Simulator (NormLabSimulators project, launchers/: TrafficJunctionSimulator.launch > Run As …) to observe that this second strategy works as example 2.
   - The normative system contains a single norm N1.

Tutorial outline

NormLab **execution**:

6-8. Execution examples

6. **Example** strategy **1**: NormLab execution: Returns an **empty** set of norms.
7. **Example** strategy **2**: Adding norms: Returns a fixed set of **1 norm**.
8. **Example** strategy **3**: Removing collisions: Returns a fixed set of **3 norms**.

9-14. Guided development of different norm synthesis strategies

9. **Development** of example strategy 1: **Empty** set of norms.
10. **Executing** your own strategy
11. **Development** of example strategy 2: **Adding norms** to your strategy (1 norm)
12. **Example 4**: A strategy with automatic norm **generation**.
13. **Example 5**: A strategy with norm **generation** + **evaluation**.
14. **SIMON**: A complete strategy with norm **generation** + **evaluation** + **refinement.**

12. Your strategy with automatic norm generation

How can we automatically generate norms on-line?

Example 4 (***TrafficNSExample4_NSStrategy*** in package **es.csic.iiia.normlab.traffic.examples.ex4,** NormLabSimulators project) uses **operators** (methods defined in *TrafficNSExample4_NSOperators*) to **create**, **add** and **activate** norms the Normative Network:

- **Activate (norm):** sets the state of norm to «Active»

- **Add (norm):** adds norm into the Normative Network and activates it.

- **Create (Conflict, Goal):**
  - Applies Case-Based Reasoning (CBR) to create a norm aimed at avoiding future conflicts.
  - If the norm does not exist in the Normative Network, then it adds (and activates) it. Otherwise, if the norm is not active (nor represented) in the NN , then it activates it.

12. Your strategy with automatic norm generation

TrafficNSExample4_NSStrategy uses operators to synthesize norms :

Everytime the strategy is executed, it:

1. Generates norms

2. Returns the Normative System.

```
/**
 * Executes IRON's strategy
 * @return the normative system resulting from the norm synthesis cycle
 */
public NormativeSystem execute() {
  this.normAdditions.clear();
  this.normDeactivations.clear();
  this.createdNorms.clear();
  this.activatedNorms.clear();

  /*-------------------
   *  Norm generation
   *-------------------*/

  this.normGeneration();

  /* Return the current normative system */
  return normativeNetwork.getNormativeSystem();
}

/**
 * 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);
    }
  }
}
```

12. Your strategy with automatic norm generation

TrafficNSExample4_NSStrategy uses operators to synthesize norms :

Everytime the strategy is executed, it:

1. Generates norms
    1. Perceives the scenario

2. Returns the Normative System.

```
/**
 * Executes IRON's strategy
 * @return the normative system resulting from the norm synthesis cycle
 */
public NormativeSystem execute() {
  this.normAdditions.clear();
  this.normDeactivations.clear();
  this.createdNorms.clear();
  this.activatedNorms.clear();

  /*-------------------
   *  Norm generation
   *-------------------*/

  this.normGeneration();

  /* Return the current normative system */
  return normativeNetwork.getNormativeSystem();
}
```

```
/**
 * 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);
    }
  }
}
```

ViewTransition: description of partial scenario transition from time t-1 to time t (current tick)

12. Your strategy with automatic norm generation

TrafficNSExample4_NSStrategy uses operators to synthesize norms :

Everytime the strategy is executed, it:

1. Generates norms
    1. Perceives the scenario
    2. Detects non regulated conflicts

2. Returns the Normative System.

Conflict detection through getConflicts() domain function
Each conflict has a ViewTransition with a conflict at tick t and an involved (responsible) agent.

```
/**
 * Executes IRON's strategy
 * @return the normative system resulting from the norm synthesis cycle
 */
public NormativeSystem execute() {
  this.normAdditions.clear();
  this.normDeactivations.clear();
  this.createdNorms.clear();
  this.activatedNorms.clear();

  /*-------------------
   *  Norm generation
   *-------------------*/

  this.normGeneration();

  /* Return the current normative system */
  return normativeNetwork.getNormativeSystem();
}

/**
 * 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);
    }
  }
}
```

12. Your strategy with automatic norm generation

TrafficNSExample4_NSStrategy uses operators to synthesize norms :

Everytime the strategy is executed, it:

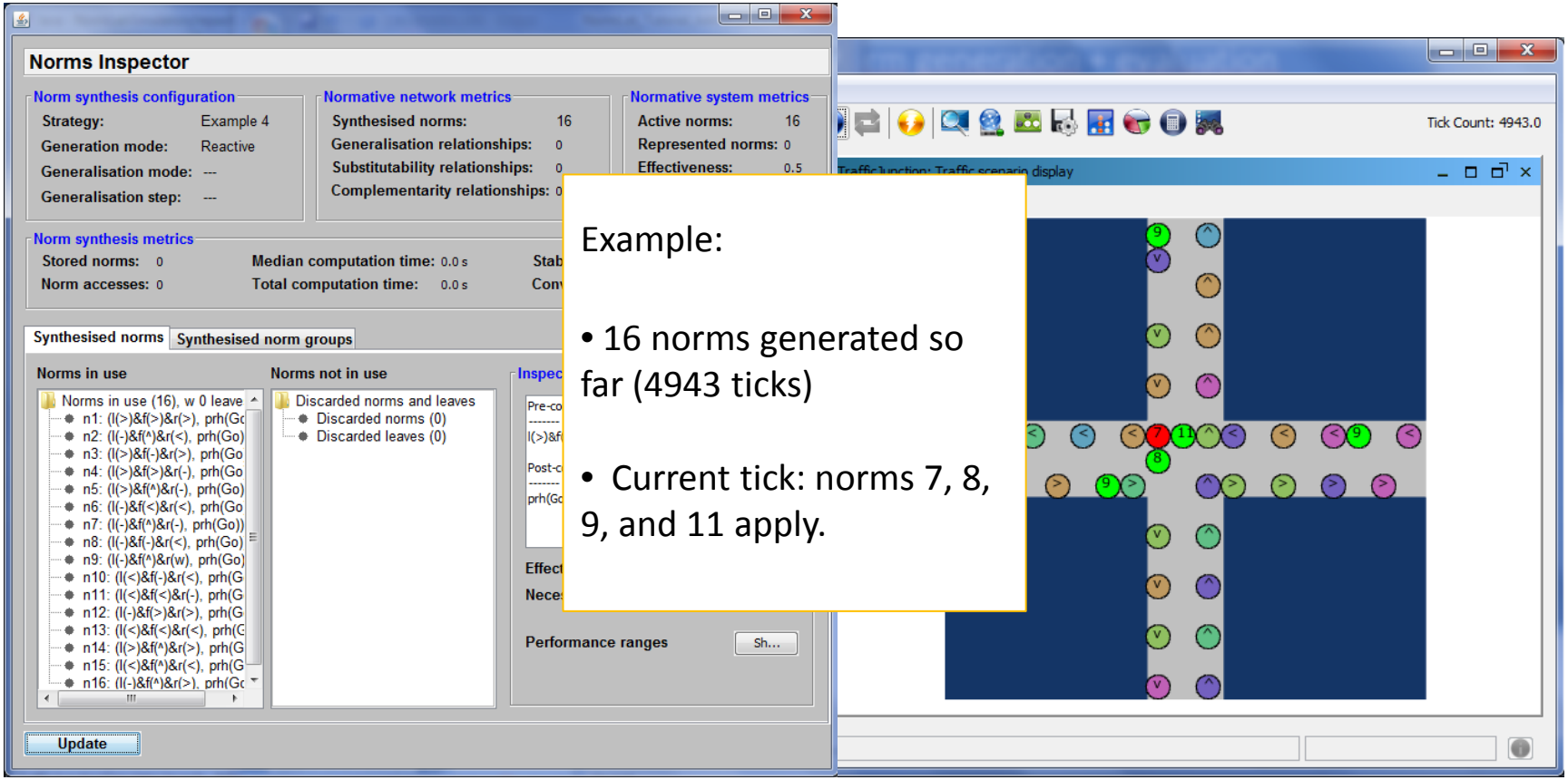
1. Generates norms
    1. Perceives the scenario
    2. Detects non regulated conflicts
    3. **Creates** norms for each conflict.
2. Returns the Normative System.

```
/**
 * Executes IRON's strategy
 * @return the normative system resulting from the norm synthesis cycle
 */
public NormativeSystem execute() {
  this.normAdditions.clear();
  this.normDeactivations.clear();
  this.createdNorms.clear();
  this.activatedNorms.clear();

  /*-------------------
   *  Norm generation
   *-------------------*/

  this.normGeneration();

  /* Return the current normative system */
  return normativeNetwork.getNormativeSystem();
}

/**
 * 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);
    }
  }
}
```

12. Your strategy with automatic norm generation

Execute the strategy:

1. Set NormSynthesisExample **defaultValue=«4»** in **parameters.xml** (in NormLabSimulators project, **repast-settings/TrafficJunction.rs**) and save the file.
2. Execute the simulator
   - NormLabSimulators project, launchers/: TrafficJunctionSimulator.launch > Run As …
3. Observe how, as long as cars collide, it generates norms to avoid these collisions
   - Norms are never evaluated (select a norm and click on button *Show* performance ranges).



Tutorial outline

NormLab **execution**:

6-8. Execution examples

6. **Example** strategy **1**: NormLab execution: Returns an **empty** set of norms.
7. **Example** strategy **2**: Adding norms: Returns a fixed set of **1 norm**.
8. **Example** strategy **3**: Removing collisions: Returns a fixed set of **3 norms**.

9-14. Guided development of different norm synthesis strategies

9. **Development** of example strategy 1: **Empty** set of norms.
10. **Executing** your own strategy
11. **Development** of example strategy 2: **Adding norms** to your strategy (1 norm)
12. **Example 4**: A strategy with automatic norm **generation**.
13. **Example 5**: A strategy with norm **generation** + **evaluation**.
14. **SIMON**: A complete strategy with norm **generation** + **evaluation** + **refinement.**

13. Automatic norm generation + evaluation

Are generated norms good enough?

Let's see example 5: ***TrafficNSExample5_NSStrategy*** (in NormLabSimulators project, src/traffic **es.csic.iiia.normlab.traffic.examples.ex5** package) :

Whenever the strategy is **executed:**

- It generates norms (as example 4)
- It **evaluates norms:** how?

```
public NormativeSystem execute() {
  this.normAdditions.clear();
  this.normDeactivations.clear();
  this.createdNorms.clear();
  this.activatedNorms.clear();

  this.normGeneration();

  this.normEvaluation();

  /* Return the current normative system */
  return normativeNetwork.getNormativeSystem();
}
```

13. Automatic norm generation + evaluation

Norm Evaluation (*TrafficNSExample5_NSStrategy*) :

```
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);
}
```

1. **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;
}
```

For each viewTransition, normReasoner computes the norms that apply to each agent by using DomainFunctions

13. Automatic norm generation + evaluation

Norm Evaluation (*TrafficNSExample5_NSStrategy*):

```
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);
}
```

2. **Norm compliance**: Did agents **complied** with their applicable norms? Did that lead to conflicts?

```
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);
    }
  }
}
```

normReasoner.
checkNormComplianceAndOutcomes

13. Automatic norm generation + evaluation

Norm Evaluation (*TrafficNSExample5_NSStrategy*):

```
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);
}
```

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);
      }
    }
  }
}
```

evaluate(…) method in
TrafficNSExample5_NSUtilityFunction
(in NormLabSimulators project, src/traffic
es.csic.iiia.normlab.traffic.examples.ex5 package)

13. Automatic norm generation + evaluation

Norm Evaluation (*TrafficNSExample5_NSStrategy*):

```
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);
}
```

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);
      }
    }
  }
}
```

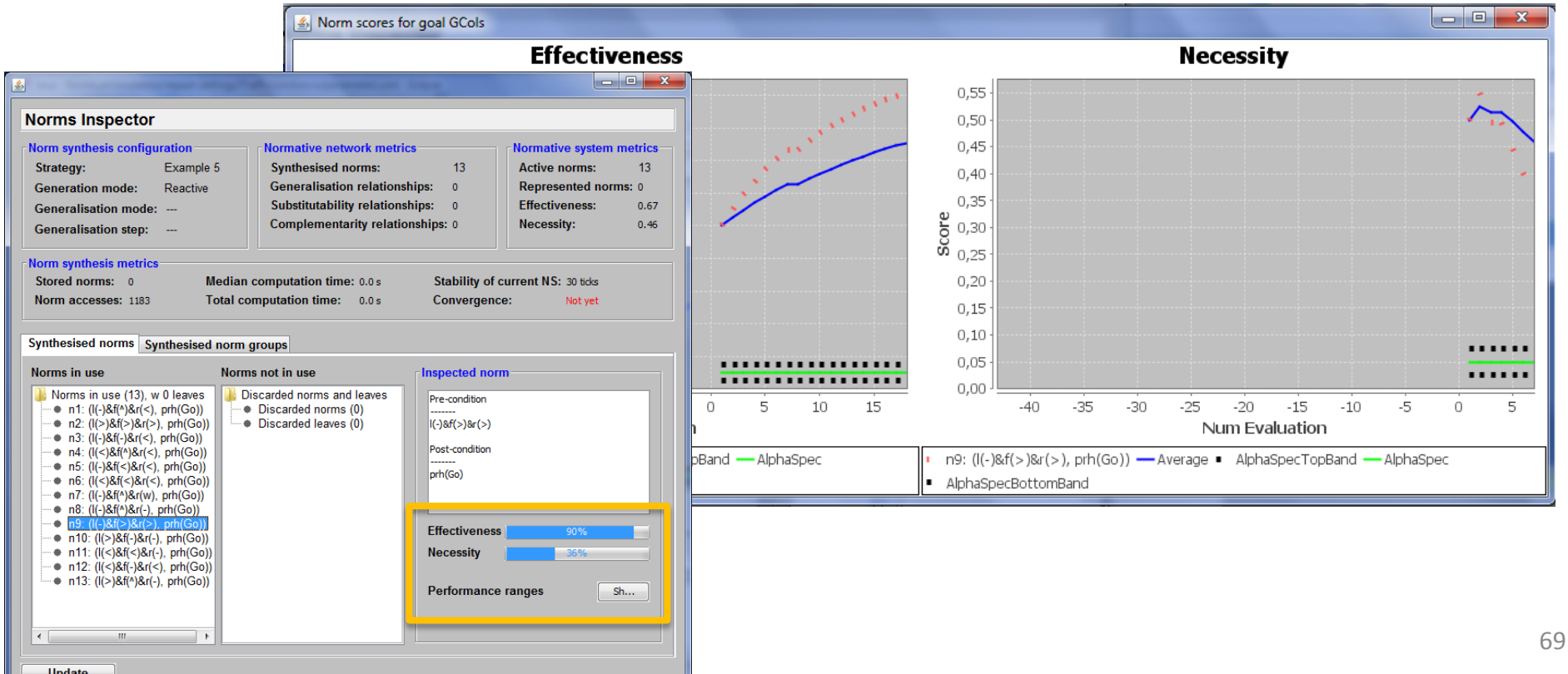
Evaluates each norm in terms of system goals: Is it useful to **avoid conflicts**?
(e.g. traffic: avoids car collisions?). Two dimensions:

- **Effectiveness**: when complied, is it effective to avoid conflicts?
  - If complied + no conflicts → ***Effective***
  - If complied + conflicts → ***Ineffective***
- **Necessity**: when infringed, did some conflicts actually arise?
  - If infringed + no conflicts → ***Unnecessary***
  - If infringed + conflicts → ***Necessary***

13. Automatic norm generation + evaluation

Execute the strategy:

1. Set NormSynthesisExample **defaultValue=«5»** in **parameters.xml** (in NormLabSimulators project, **repast-settings/TrafficJunction.rs**) and save the file.
2. Execute the simulator
   - NormLabSimulators project, launchers/: TrafficJunctionSimulator.launch > Run As …
3. Observe how it generates norms and evaluates them.
   - Effectiveness and necessity of each norm change along time (select a norm and click on button *Show* performance ranges).



Tutorial outline

NormLab **execution**:

6-8. Execution examples

6. **Example** strategy **1**: NormLab execution: Returns an **empty** set of norms.
7. **Example** strategy **2**: Adding norms: Returns a fixed set of **1 norm**.
8. **Example** strategy **3**: Removing collisions: Returns a fixed set of **3 norms**.

9-14. Guided development of different norm synthesis strategies

9. **Development** of example strategy 1: **Empty** set of norms.
10. **Executing** your own strategy
11. **Development** of example strategy 2: **Adding norms** to your strategy (1 norm)
12. **Example 4**: A strategy with automatic norm **generation**.
13. **Example 5**: A strategy with norm **generation** + **evaluation**.
14. **SIMON**: A complete strategy with norm **generation** + **evaluation** + **refinement.**

14. SIMON: generation + evaluation + refinement

SIMON is a **complete norm synthesis strategy** that uses norm evaluation to refine norms

SIMONStrategy (in NormSynthesisMachine project, src **es.csic.iiia.nsm.strategy.simon** package) :

Whenever the strategy is **executed:**

- It generates norms
- It evaluates norms
- It **refines them :** how?

```
public NormativeSystem execute() {
  this.nsMetrics.resetNonRegulatedConflicts();
  this.visitedNorms.clear();

  /* Norm generation */
  List<Norm> normsActivated = this.normGenerator.step(viewTransitions, conflicts);

  /* Norm evaluation */
  this.normEvaluator.step(viewTransitions, normApplicability,
      normCompliance, normGroupCompliance);

  /* Norm refinement */
  this.normRefiner.step(normApplicability, normsActivated);

  /* Manage lists that control new additions to the normative network,
   * normative system, as well as norms that have been removed */
  this.manageNormControlLists();

  /* Return the current normative sys
  return normativeNetwork.getNormativ
}
```

step(…) method in **SIMONNormRefiner** (in NormSynthesisMachine project, src es.csic.iiia.nsm.strategy.simon package)

14. SIMON: generation + evaluation + refinement

Norm refinement:

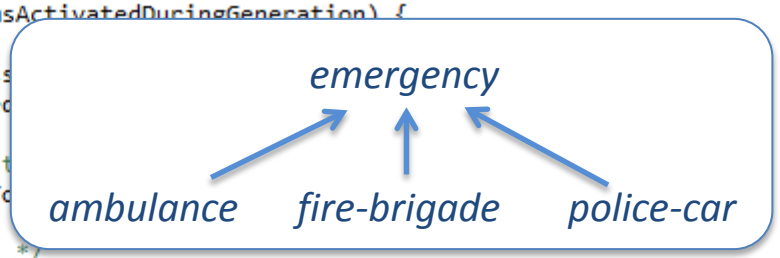
1. Norms are **generalised** if their (effectiveness **and** necessity) **> threshold.**

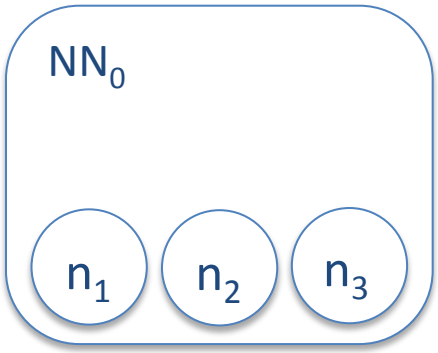
```
public void step(Map<ViewTransition, NormsApplicableInView> normApplicability,
    List<Norm> normsActivatedDuringGeneration) {

  List<Norm> processed = new ArrayList<Norm>();
  List<Norm> visited = new ArrayList<Norm>();

  /* Compute norms that must be revised */
  List<Norm> normsToRevise = this.checkNormsToRevise(normApplicability);

  /* Classify norms */
  this.normClassifications = this.normClassifier.step(normsToRevise);

  /* Refine norms based on norm classifications */
  for(Norm norm : normClassifications.keySet()) {
    if(processed.contains(norm)) {
      continue;
    }
    List<NormAttribute> attributes = normClassifications.get(norm);

    boolean isIneffective = attributes.contains(NormAttribute.INEFFECTIVE);
    boolean isUnnecessary = attributes.contains(NormAttribute.UNNECESSARY);
    boolean isGeneralisable = attributes.contains(NormAttribute.GENERALISABLE);

    /* If the norm is whether ineffective or unnecessary, then deactivate
     * it (specialise it into its children) */
    if(isIneffective || isUnnecessary) {
      visited.clear();
      specialiseDown(norm, NetworkNodeState.DISCARDED, visited);
    }
    /* If the norm has enough utility to be generalised,
     * then try to generalise it */
    else if(isGeneralisable) {
      generaliseUp(norm, genMode, genStep);
    }
    /* Update complexities metrics */
    this.nsMetrics.incNumNodesVisited();
  }
}
```

14. SIMON: generation + evaluation + refinement

Norm refinement:

1. Norms are **generalised** if their (effectiveness **and** necessity) ≥ gen. **threshold**.





Normative system
$NS_0=\{n_1, n_2, n_3\}$

n_1: Give way to ambulances
n_2: Give way to fire brigade
n_3: Give way to police cars

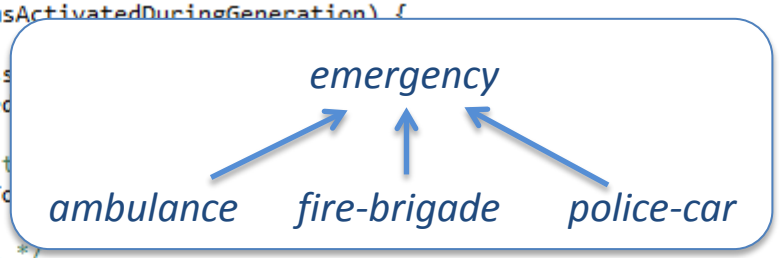
```
public void step(Map<ViewTransition, NormsApplicableInView> normApplicability,
    List<Norm> normsActivatedDuringGeneration) {

  List<Norm> process
  List<Norm> visited

  /* Compute norms t
  List<Norm> normsTo

  /* Classify norms */
```

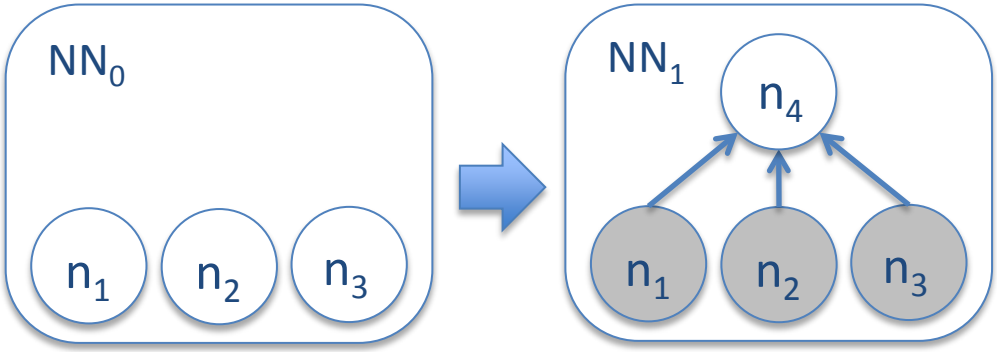
```
  /* If the norm has enough utility to be generalised,
   * then try to generalise it */
  else if(isGeneralisable) {
    generaliseUp(norm, genMode, genStep);
  }

  /* Update complexities metrics */
  this.nsMetrics.incNumNodesVisited();
 }
}
```

14. SIMON: generation + evaluation + refinement

Norm refinement:

1. Norms are **generalised** if their (effectiveness **and** necessity) ≥ gen. **threshold.**





New Normative system $NS_1=\{n_4\}$

Increases Compactness

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

```
/* If the norm has enough utility to be generalised,
 * then try to generalise it */
else if(isGeneralisable) {
  generaliseUp(norm, genMode, genStep);
}
```

14. SIMON: generation + evaluation + refinement

Norm refinement:

1. Norms are **generalised**
2. Norms are **specialised** if their (effectiveness **or** necessity) **<** esp. **threshold**

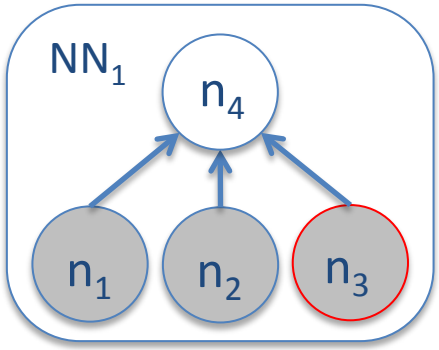
```
public void step(Map<ViewTransition, NormsApplicableInView> normApplicability,
    List<Norm> normsActivatedDuringGeneration) {

  List<Norm> processed = new ArrayList<Norm>();
  List<Norm> visited = new ArrayList<Norm>();

  /* Compute norms that must be revised */
  List<Norm> normsToRevise = this.checkNormsToRevise(normApplicability);

  /* Classify norms */
  this.normClassifications = this.normClassifier.step(normsToRevise);

  /* Refine norms based on norm classifications */
  for(Norm norm : normClassifications.keySet()) {
    if(processed.contains(norm)) {
      continue;
    }
    List<NormAttribute> attributes = normClassifications.get(norm);

    boolean isIneffective = attributes.contains(NormAttribute.INEFFECTIVE);
    boolean isUnnecessary = attributes.contains(NormAttribute.UNNECESSARY);
    boolean isGeneralisable = attributes.contains(NormAttribute.GENERALISABLE);

    /* If the norm is whether ineffective or unnecessary, then deactivate
     * it (specialise it into its children) */
    if(isIneffective || isUnnecessary) {
      visited.clear();
      specialiseDown(norm, NetworkNodeState.DISCARDED, visited);
    }

    /* If the norm has enough utility to be generalised,
     * then try to generalise it */
    else if(isGeneralisable) {
      generaliseUp(norm, genMode, genStep);
    }

    /* Update complexities metrics */
    this.nsMetrics.incNumNodesVisited();
  }
}
```

14. SIMON: generation + evaluation + refinement

Norm refinement:

1. Norms are **generalised**
2. Norms are **specialised** if their (effectiveness **or** necessity) **<** esp. **threshold**

Normative System $NS_1 = \{n_4\}$



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

```
public void step(Map<ViewTransition, NormsApplicableInView> normApplicability,
    List<Norm> normsActivatedDuringGeneration) {

  List<Norm> processed = new ArrayList<Norm>();
  List<Norm> visited = new ArrayList<Norm>();

  /* Compute norms that must be revised */
  List<Norm> normsToRevise = this.checkNormsToRevise(normApplicability);

  /* Classify norms */
                                                                        ;
                                                                        ;
                                                                    BLE);

  /* If the norm is whether ineffective or unnecessary, then deactivate
   * it (specialise it into its children) */
  if(isIneffective || isUnnecessary) {
    visited.clear();
    specialiseDown(norm, NetworkNodeState.DISCARDED, visited);
  }

  /* If the norm has enough utility to be generalised,
   * then try to generalise it */
  else if(isGeneralisable) {
    generaliseUp(norm, genMode, genStep);
  }

  /* Update complexities metrics */
  this.nsMetrics.incNumNodesVisited();
 }
}
```

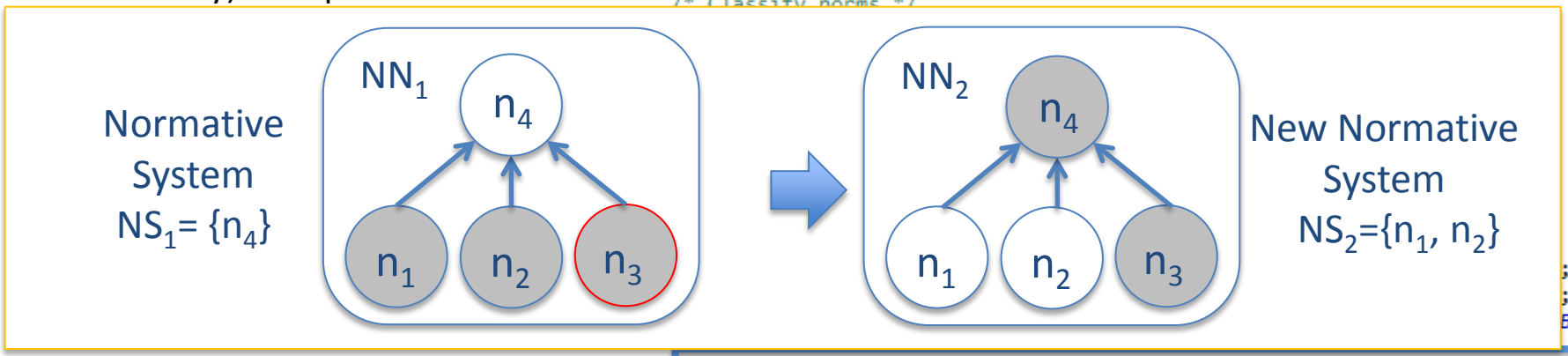
Increases Compactness

14. SIMON: generation + evaluation + refinement

Norm refinement:

1. Norms are **generalised**
2. Norms are **specialised** if their (effectiveness **or** necessity) **<** esp. **threshold**

Removes Under-performing norms



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

```
public void step(Map<ViewTransition, NormsApplicableInView> normApplicability,
    List<Norm> normsActivatedDuringGeneration) {

  List<Norm> processed = new ArrayList<Norm>();
  List<Norm> visited = new ArrayList<Norm>();

  /* Compute norms that must be re
  List<Norm> normsToRevise = this.

  /* Classify norms */
```

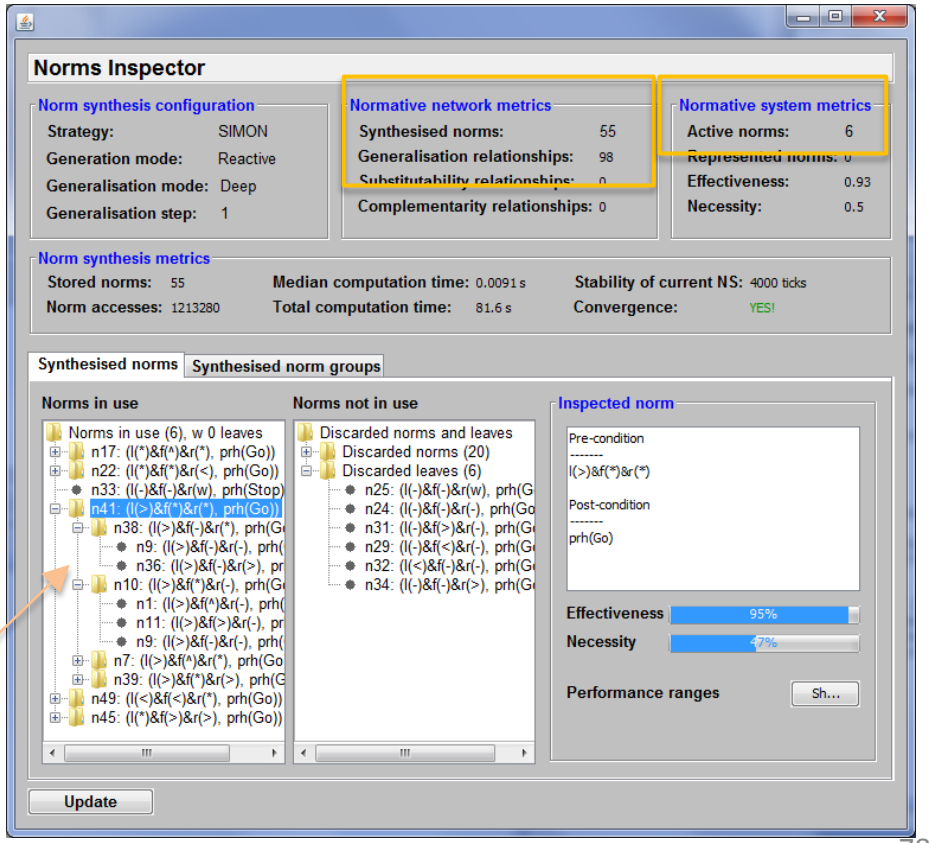
```
;
;
BLE);
```

```
  /* If the norm is whether ineffective or unnecessary, then deactivate
   * it (specialise it into its children) */
  if(isIneffective || isUnnecessary) {
    visited.clear();
    specialiseDown(norm, NetworkNodeState.DISCARDED, visited);
  }

  /* If the norm has enough utility to be generalised,
   * then try to generalise it */
  else if(isGeneralisable) {
    generaliseUp(norm, genMode, genStep);
  }

  /* Update complexities metrics */
  this.nsMetrics.incNumNodesVisited();
 }
}
```

14. SIMON. A complete norm synthesis strategy

Execute SIMON strategy:

1. In **parameters.xml** (in NormLabSimulators project, repast-settings/TrafficJunction.rs) set:
   - **NormSynthesisExample** defaultValue=**«0»**
   - **NormSynthesisStrategy** defaultValue=**«2»** (2 stands for SIMON strategy)
   - **NormGeneralisationMode** defaultValue=**«1»** (***Deep*** norm generalisation)
   - **NormGeneralisationStep** defaultValue=**«1»** (generalises 1 predicate at a time)
   - Save the file.
2. Execute the simulator
   - NormLabSimulators project, launchers/: TrafficJunctionSimulator.launch > Run As …
3. Observe how it generates norms, evaluates, and refines them.
   - Compact Normative System.



Normative System: 6 norms
Normative Network: 55 norms
Generalisations: 98 relationships
- Ex: n41 generalises n38, n10, n7 and n39

Covergence at tick 9428