Multi-agent Environment for Complex SYstems COsimulation (MECSYCO) - User Guide: Scholar applications

Benjamin Camus[1,2], Julien Vaubourg[2], Yannick Presse[2],
Victorien Elvinger[2], Thomas Paris[1,2], Alexandre Tan[2]
Vincent Chevrier[1,2], Laurent Ciarletta[1,2], Christine Bourjot[1,2]
[1]Universite de Lorraine, CNRS, LORIA UMR 7503,
Vandoeuvre-les-Nancy, F-54506, France.
[2]INRIA, Villers-les-Nancy, F-54600, France.
mecsyco@inria.fr

July 4, 2016

Contents

Introduction

This document is more an explantion of the scholar applications than a User Guide. Scholar applications is a pack of examples that we will present you here. They represent various applications of concepts, methods or functions we talked about in provided *User Guide* .

Each chapter correspond to the package with the same name.

Chapter 1

gettingstarted

This is the model used in the first part of the *Getting Started* section. Here you have the whole model. You can use it to verify your implementation.

To remind you, the first model (MinimalExampleLauncher.java) creates a pool of walkers positioned at the origin. For each step the hidden walkers (out of screen) die and the other ones move in a random direction (Figure 1.1).

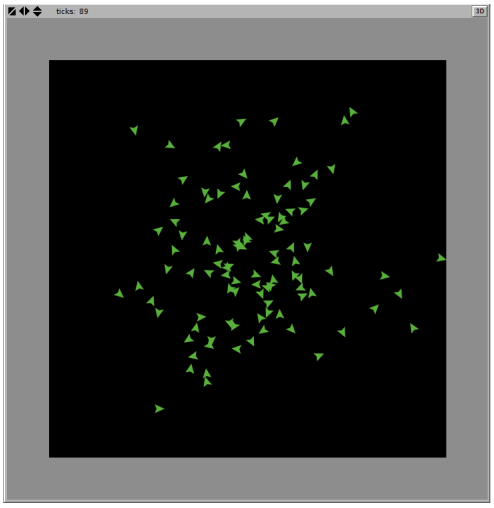

Figure 1.1: NetLogo random walk model view

In a second model (LoopbackExampleLauncher.java), instead of making them disappear, we make the hidden walkers them start at the origin again (Figure 1.2).

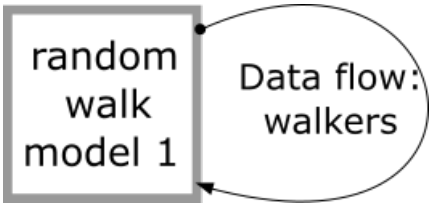

Figure 1.2: Random walk model with loopback

These two show how to create a simple model with only one agent.

The third one (DiscoveringOperationExampleLauncher.java), add the system of data operation and permits the user to distinguish the walkers that came back at least once by changing their color (Figure 1.3).

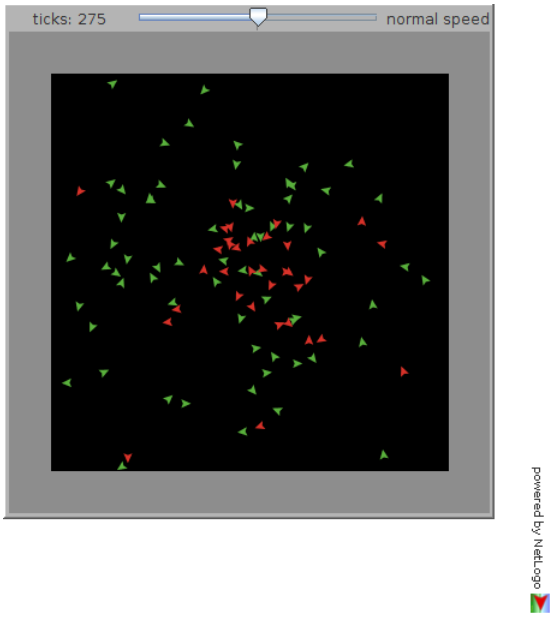

Figure 1.3: Effect of the color setter operation.

The next one (InteractingModelExampleLauncher.java) is the first multi-model, that mean we start to use more than one agent. In this multi-model, we made two pools exchange their walkers randomly. We make sure that a walkers have a color associated to its origin pool (Figure 1.4).

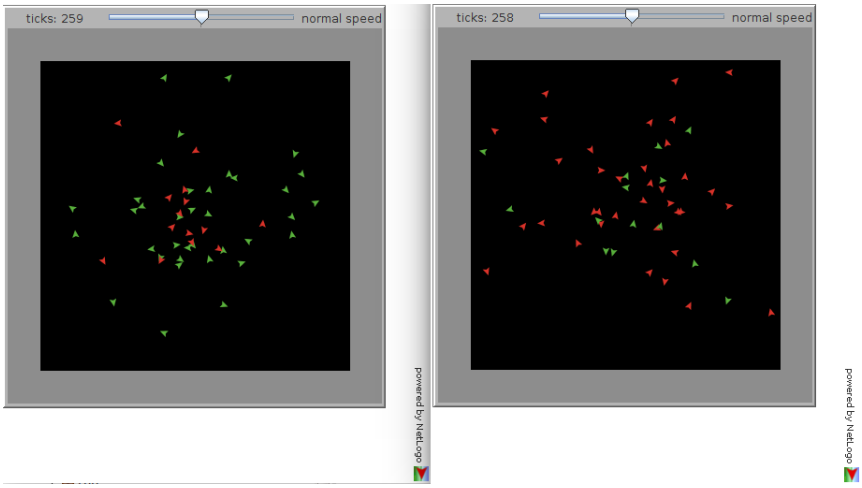

Figure 1.4: View of two interacting random walk models.

The two last model introduce the user to the crafting of his own data operation, then the creation of his own ModelArtifact.
So OperationCreationExampleLauncher.java allows the user to use the new operation created (TurtleSizeSetter.java) in the third model. The new operation will make the walker bigger after being put back at least once (Figure 1.5).

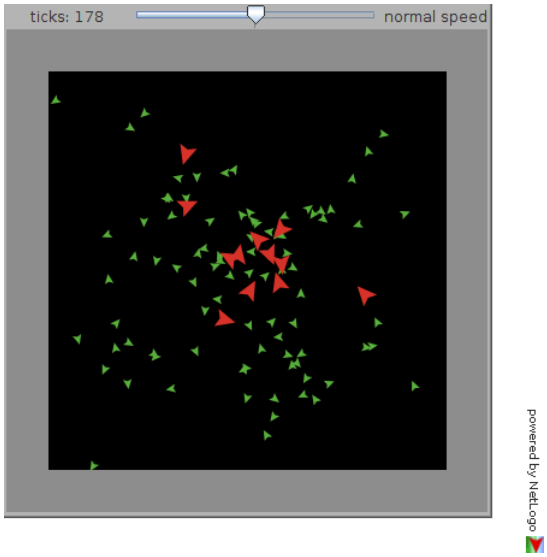

Figure 1.5: Effect of the color and size setter operations.

Finally, ModelArtifactCreationExampleLauncher.java, is the multi model using the new ModelArtifact (RandomWalkModelArtifact.java). The creation of the new model artifact will allow the user to define a new way of treating event by the agent. In this case, they will allows both agent to count, not the number of walker in their pool, but the number of walker in their pool that really belong there (originally created in this pool).

Chapter 2

highway

The highway package, like it name says, contains classes for representing highways in different perspectives of view. For this package, nearly all functionality of MECSYCO were used (operation, model artifact and data type creation, simple multi-model and DDS based multi-model and some functions of the viewing tool too). Some of these examples were also used to test and validate some of MECSYCO's properties.

2.1 equation

The equation launcher (EquationBasedHighwayTrafficLauncher.java) is a multi-model constituted of two agents. Each agent is link to a model of a three ways highway of 5 km long separated in 10 cells (that means a total length of 10 km and 20 cells). This is a macroscopic vision of the highway because we are not watching car by car but average of car per cells. (Figure 2.1)

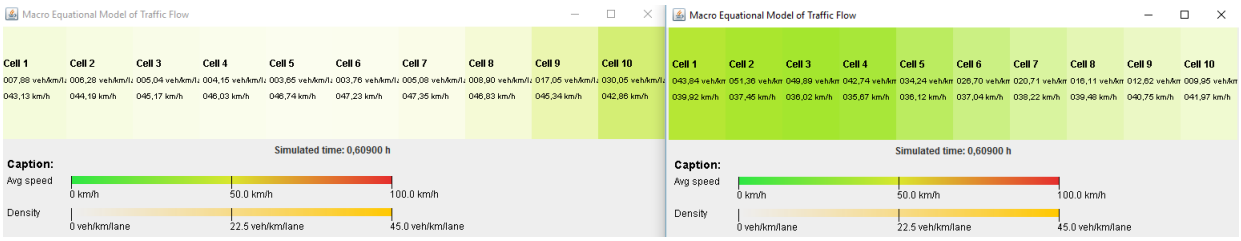

Figure 2.1: the two agents display side by side.

This example contains example of how to create a model (Cell.java) and the *ModelArtifact* to manage it (CellModelArtifact.java). We also provide an example of the creation of a *SimulData* (MacroTrafficData.java). The launcher also use the observing tool: *LiveTXGraphic*.

2.2 event

In this package, we will have a factual view of the highway. We focused on each passing car. This will not be the same part of the highway than the first package, but the interchange (Figure 2.2).

In this multi-model too we use a specific model (TrafficModel.java) and the associated *ModelArtifact* (EventModelArtifact.java) with a new *SimulData* (VehiclesGroup.java). The difference is that the model is not implemented in the *ModelArtifact*, but directly in the launcher, and then link to the *ModelArtifact* as a parameters.
This model was used in order to validate the fact that MECSYCO manages collison problem when synchronizing the agents.

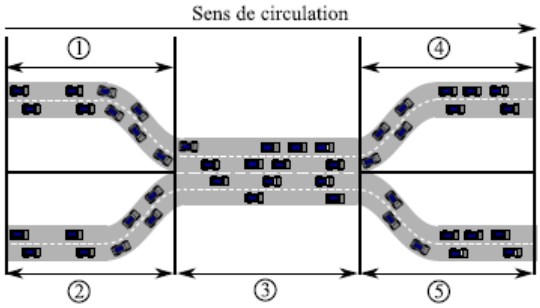

Figure 2.2: Scheme of the interchange.

Note:
Do not try to follow a car in the display, we did not manage the color, as a consequence, when displaying, when a car go to another agent (then it goes to another part of the road), the color is not preserved.

2.3 mixed

This package will combine the two previous models (macro and factual). We then use the previous *ModelArtifacts*, and we create one more (MicroModelArtifact.java) to show the road with actual car on it (Figure 2.3), we have then a micro view added in our multi-model.

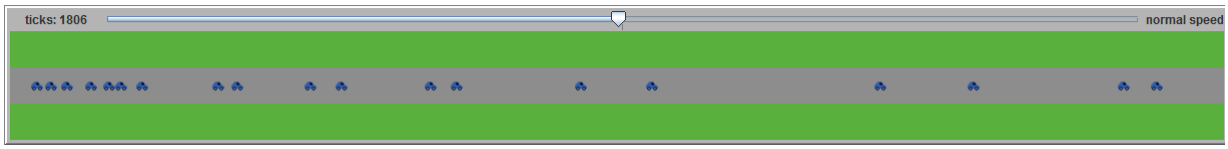

Figure 2.3: Road display with cars

We are now modelling a circuit (the road is a circle) cut in three section. Each section is modelled with one of the three view: one in micro, one in macro and the last in factual view, connected in this same order.
This multi-model was used to show that MECSYCO can really manage different level of abstraction in one simulation. We then use all of MECSYCO functionality:

- **SimulData creation:** As previously said, we created *SimulData* especially for these multi-model: MacroTrafficData and VehiculesGroup.
- **ModelArtifact creation:** All of the *ModelArtifact* use are made for those multi-model. We had the two previous (CellModelArtifact and EventModelArtifact), and the last one (MicroModelArtifact).
- **Model creation:** The model used are created too (Cell and TrafficModel).
- **Data operation:** In order to make the three models interact, we need to adapt the output of one agent to the type expected by the input of another. We created then, data operation:
  - **FlowRate2Group:** macro to event.
  - **Turtles2Flow:** micro to macro.
  - **Group2Turtle:** event to micro.
  - **Density2Cars:** macro to a number of car (for our observing tool).
  - **Turtle2SimulDataDouble** micro to a number of car (for our observing tool).

  - **MacroData2Double** macro to a double depending of which information you need to display with our observing tool (flow, density or speed).

- **Time operation:** we used pre-made operation here. We needed time operation because a macro view has a different time scale than a micro view. As a consequence, we needed one between the micro agent and the macro, then the opposite between the macro and the event one.
- **Observing tool:** In order to get more information, we used our observing tool (here it is the LiveTX).
- **DDS:** To be sure that all properties are not lost when decentralized, we use the same multi-model with DDS.
- **Bonuses:** In the case of MicroModelArtifact, we show that it is possible to use particular simulation data type (here it is the NetLogoCarTutle) inside MECSYCO. You can, like here, convert them in *SimulData* (thanks to the pre-made *SimulData* that are made for being the more generic as possible, see *User Guide section "Simulation data"*). Or you could create your own *SimulData* (see *"User Guide: SimulData manipulation"*)

2.3.1 HybridHighwayTrafficLauncher.java

This is the "normal" launcher of this multi-model. It is composed of three agent associated with the three different model (macro, micro and event), and an observing agent.

2.3.2 distributed

This sub-package provide all separated launchers for the distribution.

- **EquationModelLauncher:** This is the macroscopic model.
- **EventModelLauncher:** The event model is contained in this one.
- **MultiAgentModelLauncher:** The last one, the microscopic model.
- **ObservationLauncher:** It contains all implementations of our observing tool.

Chapter 3

ode

This package contains ordinary derivative equations models and shows how to create and simulate them (with and without distribution). You could see here more examples of model creation (Equation, LorenzSystem, LorenzX, LorenzY, LorenzZ) with their *ModelArtifact* (EquationModelArtifact and NetLogoModelArtifact).

3.1 lorenz

This is the Lorenz example used for the second part of the *Getting Started 2* section.

The Lorenz system is a system of three ordinary differential equations. It solution depends on certain parameter values and initial conditions, and the plotted result often resemble a butterfly. In the Getting Started 2, we make the user construct the whole model nearly from the scratch, we just provide the equational model that defined the general behavior of an equation, then the three different part that precise the equation (LorenzX, LorenzY and LorenzZ). Thank to this tutorial you will be able to try nearly all extension provided in MECSYCO 2.0.0 (observing, DDS...).

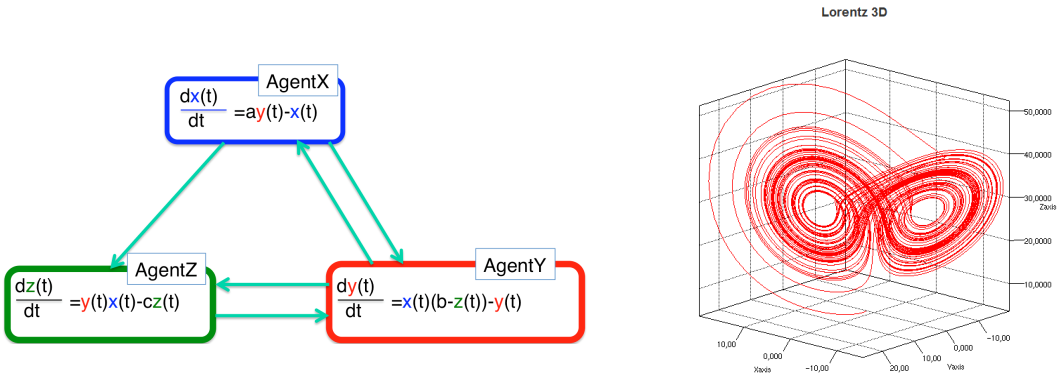

Figure 3.1: Lorenz system in MECSYCO.

3.2 lotkavolerra

As the name suggest, we reproduced the model of prey-predator adapted to MECSYCO. The Lotka-Volterra equations are a couple of first order, non-linear, differential equation that describes the biological system dynamics where preys and predators interacts.

We have the multi-model using DDS (LotkaVolterraPredatorLauncher and LotkaVolterraPreyLauncher) and two simple ones. One is the whole Lockta-Volterra multi-model (LotkaVolterraLauncher, Figure 3.2).

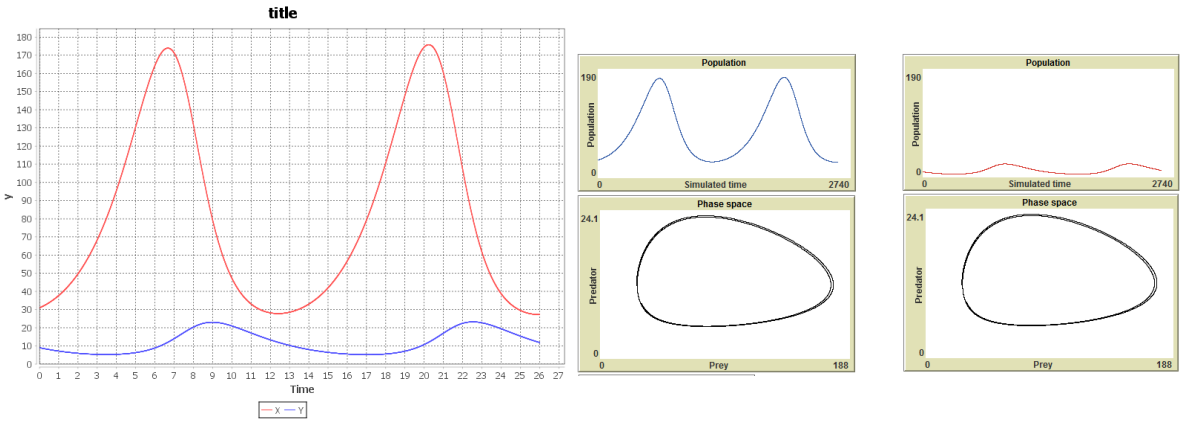

Figure 3.2: Display for Lokta Volterra launcher

The other one shows the evolution of prey (with different time discretization value) without any predators (Figure 3.3).

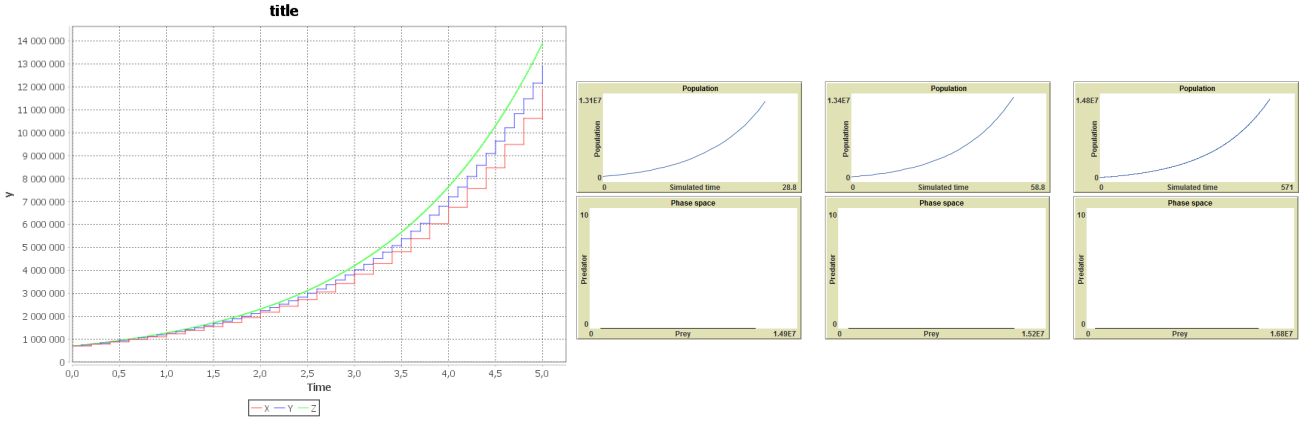

Figure 3.3: Display for prey launcher

Chapter 4

pedestrian

This package contains a simple multi-model that shows that if you have a library for defining a model in Java, you can use it in MECSYCO by creating a *ModelArtifact* in order to have the correct interface.
"Pedestrian" is a multi-model of three agents (Figure 4.1). Each agent has its own way of displaying a pedestrian (shape). These pedestrians are walking in "circle", that is to say that agent one is connected to agent two who is connected to agent three, and agent three is sending the pedestrians back to agent one. Pedestrians have a speed and a pattern generated randomly, so at each launch you could see different thing. What is sure, is that between agents, pedestrians will keep their color, and position (in high).

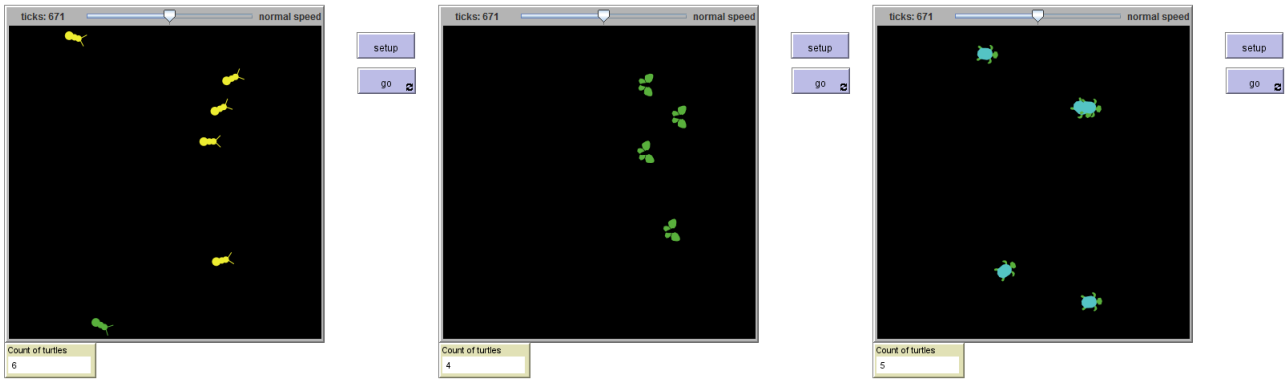

Figure 4.1: the three agents displayed side by side.

Chapter 5

sheepgrass

This package is another way to integrate NetLogo models. As previously, we created an interface thank to a *ModelArtifact*, but we had to create interface for the simulation data type used too. The multi-model contains two agents describing two worlds.
The first world represents grass and sheeps eating it. When a sheep eat a grass, the grass patch disappears and the sheep gains energy. The second world represents shepherds and sheeps. When a sheep has its energy equals to zero, it dies. Each agent is sharing the sheep state (energy) and sheep position.

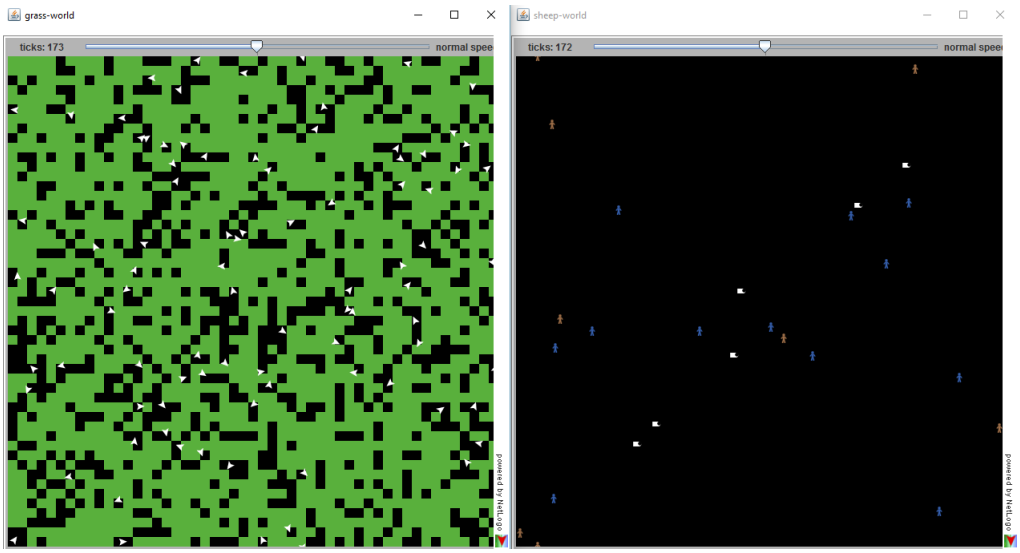

Figure 5.1: The display for sheep grass multi-model.

Chapter 6

trafficgrid

This last package is a simple multi-model that adapts a model from NetLogo thanks to an adapted *ModelArtifact*. This time, the multi-model represent a little "world" decomposed in 4 parts. Each part is a traffic grid and is associated to one agent. It is a little world because the agents are connected and you can imagine the whole as a sphere. Cars will respect "laws", that is to say, they will not go if they have a red light, and they will not collide with a car in front of them.

Figure 6.1: The display for the traffic grid multi-model.

Chapter 7

Remark

The display used in some figure are not provided in MECSYCO, but show that you can also use other observing tools, if you know how they work, and how to extract the data you need from the models.