Modeling and Design of an Agent-based Micro-simulation of the Swiss Highway Network

Michael Schumacher[1], Laurent Grangier[2], and Radu Jurca2

[1] University of Applied Sciences Western Switzerland
Institute of Business Information Systems
CH-3960 Sierre
[2] Ecole Polytechnique Fédérale de Lausanne (EPFL)
Artificial Intelligence Laboratory
CH-1015 Lausanne, Switzerland

Abstract. Multiagent simulations can be elegantly modeled and designed by enhancing the role of the environment in which agents evolve. In particular, the environment may have the role of a governing infrastructure that regulates with laws or norms the actions taken by the agents. The focus of modeling and design is thus shifted from a subjective view of agents towards a more objective view of the whole multiagent system. In this paper, we apply the idea of a governing environment to model and design a multi-agent system that micro-simulates the Swiss highway network. The goal of the simulation is to show how traffic jams and accordion phenomena may be handled with appropriate local regulations on speed limits. A natural modeling would give segments the capacity to regulate the speed based on observed local events. We developed the simulation platform from scratch in order to accommodate our design choices and a realistic complexity. This paper presents in details our modeling and design choices, and first experimental results.

1 Introduction

Agent-based micro-simulations are becoming a popular application area of multiagent systems (MAS), in areas such as social sciences, traffic management, biology, geography, or environmental sciences. Agent technology has opened a whole new methodology for studying real-world complex systems by simulating every individual through an autonomous agent. Individual behavior can thus be easily modeled, and the MAS captures the aggregated behavior of the collective. These agent-based *micro-simulations* help understanding better an emergent reality or allows trying virtually some settings that would be very costly to test in reality. Traffic management is a typical example. For instance, a micro-simulation may help to visualize the effect of constructing new roads on the overall traffic.

Some multiagent systems (and a fortiori agent-based micro simulations) may be elegantly modeled and designed by enhancing the role of the environment in which agents evolve. In particular, the environment may have the role of a

governing infrastructure that regulates with laws or norms any action within the system. This has the strong advantage of a flexible modeling and design, where the focus is shifted from a subjective view of agents towards a more objective view of the whole multiagent system.

In this paper, we show first experiments on how we apply the governing environment to the modeling and design of a micro-simulation of the Swiss highway network. The goal of the simulation is to show how accordion phenomena and traffic jams may be handled with appropriate local regulations on the speed limit. For example, adaptive speed limitations my be implemented in order to maximize the throughput of the network.

A natural model gives segments the capacity to regulate the speed based on locally observed events. Therefore, regulating highway segments perfectly captures the design of a governing environment. Because of the complexity of the simulation and our choice in the above described modeling, we developed a simulation platform from scratch. This paper presents in details our modeling choices for the simulation platform. First experimental results of our implementation are also eluded. The adaptive distributed speed regulation will however be the subject of another paper, as it is still under development.

The paper is organized as follows. Section 2 introduces and explains the notion of governing environment. Sec. 3 explains our problematic of traffic simulation in Switzerland. After discussing our global modeling in Sec. 4 following the governing idea, we describe how we model the agent behaviors in Sec. 5. Sec. 6 presents our platform design. In Sec. 7, we discuss experiments. Section 8 concludes the paper.

2 The Governing Environment

Most research in multiagent systems (MAS) has focused on the internal capacities of agents, and not on the medium in which they evolve. This vision is however changing towards enhancing the function of the environment in MAS (see for instance [12,1]). Actually, such a vision was already implicit in the early days of software agent research. This is shown by a definition of an autonomous agent as *a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future* [3]. This description stresses the importance of the environment as the living medium, the condition for an agent to live, or the first entity an agent interacts with. Thus an agent is part of the environment. But it remains autonomous, so that the environment may not "force" the agent's integrity. It is in this environment that an agent (autonomously) senses and acts. The acting of the agent on the environment directly influences its future sensing, because the environment is changed by the agent actions.

Even if the notion of environment was stressed as a main component of MAS, most approaches have viewed it as something being modelled in the "minds" of the agents, thus using a minimal and implicit environment that is not a first-order abstraction, but rather the sum of all data structures within agents that

represent an environment. This is a typical *subjective view* of the multiagent system inherited from distributed artificial intelligence, which contrasts with an *objective view* that deals with the system from an external point of view of the agents [9]. This objective point of view sees the environment as a central component for the modeling and design of MASs. Multiagent simulations belong to the type of systems that most explicitly model the environment.

Whenever a multiagent system is to be implemented and deployed, an underlying *infrastructure* becomes essential [7]. It offers to the MAS basic services to be used by the agents. Example functionalities are agent communication, naming or life-cycle management. The abstractions provided by such infrastructures are essential for agent-oriented software engineering, as they should be as close as possible to the concepts used for analysis and design. Today's infrastructures primarily offer agent-related abstractions for the programming of agent architectures using for instance libraries for BDI agents [4], thus supporting subjective coordination. They only offer implicit support for objective coordination, as they establish the conditions necessary for running agent programs (e.g. life-cycle management) and for setting the basic interaction means (e.g. message-enabled middleware between agents).

An appealing way to exert the necessary level of control out of agents is the use of a *governing infrastructure* to structure and shape the space of actions within a MAS [7]. This governing perspective mainly allows managing agent interactions from an external point-of-view. This has the strong advantage that agents may be defined independently, and that some control is overtaken externally. In the area of virtual organizations, the Electronic Institutions (EI) approach [6] does this by defining so-called *governors* which are middle agents that mediate all (communicative) actions within a MAS [3]. This solution has, however, important disadvantages. Providing each agent with a governor puts a heavy computational burden on the infrastructure. But, more importantly, middle agents do not capture a natural modeling for the functionality they are expected to fulfill, i.e. mediation of communication. The governing or regulating responsibility should be transferred from specialized middle agents to the environment of a MAS, calling for the *environment as a governing infrastructure* [10]. This can be done with the idea of a *programmable coordination medium* [2], which essentially defines *reactions to events* happening in a shared dataspace. This schema has the strong advantage to allow the definition of laws that not only regulate agent interactions, but also any happening within an environment. Overall, we expect that viewing the environment as a governing infrastructure simplifies the design of multiagent systems. We will show this in the area of agent-based micro-simulation applied to traffic management.

3 Micro-Simulation of the Swiss Highway Network

We modeled, designed and implemented an agent-based micro-simulation that captures the ideas of governing environment. The application area is the simu-

[3] All actions that the EI approach accounts for are communicative by nature.

Fig. 1. Swiss national roads with limits of the cantons, as displayed in our simulation platform

lation of the whole highway traffic in Switzerland of about 1700 km (see Fig. 1). We extended the platform also to the national roads, which are roads of national importance (e.g. the Gotthard pass).

The highways are split into segments. This segmentation is given by the real data received from the Federal Office for Spatial Development[4]. Each segment has an arbitrary length, its own speed limit and can have multiple lanes (from one to a maximum of four). On a segment, cars can only drive in one predefined direction. This means that a standard highway part is composed of two segments: one for each direction.

The platform is bound to a geographical information system[5] that allows zooming from the global country view to the local view of each vehicle. We did not develop over an existing platform for agent-based simulation, because the complexity of the problem is much too big. Furthermore, it would be difficult to capture our modeling. We therefore built a new platform from scratch.

Our final goal is to study adaptive and decentralized speed limitation to have an optimal car throughput. Actually, there are some settings in which modern highways perform very poorly. First, the *accordion* is a transient mode in which cars accelerate to a given speed S, only to brake to almost a full stop immediately after reaching the speed S. Secondly, *traffic jams* usually occur in highway segments preceding a bottleneck (e.g. tunnels or accidents). Our goal of the simulation is to show that adaptive distributed speed limitations on the highway segments preceding (and including) the one where problems might appear will drastically decrease the negative effects previously discussed. Therefore we want to investigate whether speeding restrictions can increase the efficiency of highways, and to determine automatic speeding restrictions that optimize highway utilization. As a methodology we decided to develop an agent-based micro-simulation to investigate the above hypothesis and to determine optimal speeding policies. A distributed speed regulation needs to split highways into segments with constant length so that on each segment one speed limitation

[4] http://www.are.admin.ch
[5] http://www.geotools.org

can be imposed. Constraints between the speed limit on neighboring segments have to ensure that the vehicles do not have to break too abruptly.

An adequate modeling of a micro-simulation allowing distributed decision making on segments can elegantly use the paradigm of a governing environment. Actually, the segment naturally build the environment of the MAS. Each segment has a set of rules that regulate the state of the highway segment (number of cars, average speed, etc) and can decide on the speed limitation for that segment. Neighboring segments can propagate events to one another. Each vehicle is modeled by one agent which takes decisions based on a local view: a driver wants to get to the destination as fast as possible and guides her action depending on the traffic in her immediate vicinity. We further assume that drivers respect the speed limits (within certain bounds).

This paper reports the modeling and design of our simulation platform, and *not* the distributed adaptive decision process for optimal speed regulation. Actually, we are currently working on this with the DPOP [8] algorithms for distributed constraint satisfaction. This will be reported in a future paper.

4 Modeling

We describe in this section the modeling of the MAS of the micro-simulation. According to the *governing environment* paradigm [10], laws are defined within the environment. The environment reacts to raised events according to the rules that we define. Unlike the agents, the environment has no behavior and does not act itself: it can just react to events which are intercepted.

Static Model We identified two types of agents that are organized around highway segments that represent the environment (see Fig. 2). The Vehicle class[6] has three state attributes : its position (relative to its current segment), its speed and its lane position. Each vehicle has : a Plan which is an ordered collection of HighwaySegment telling it which way to take; a Behavior which describes its acceleration, deceleration and lane changing behavior. VehicleCreator is a dedicated agent which takes care of creating new agents in the system.

The highway is divided in segments. Each Vehicle lives in a HighwaySegment which can be considered as a continuous space. Each one is connected to its next following segments and its previous preceding segments. Vehicles can only move from their current segment to one of the next segments. HighwaySegment has a few constant attributes (length, numberOfLanes, maxLawSpeedLimit, nationalRoad, slope, curve) and a few variable attributes (currentSpeedLimit, flow, density, meanSpeed). All these attributes are part of the environment and can be perceived by agents.

The choice for a continuous space is given by the better precision and the light implementation that follows, but one could consider a discrete space. Nagel [5] shows how to build a cellular automata simulation with discrete space. It should be easily transposed to our agent-based simulation.

[6] We use the UML profile described in [11], where rounded rectangles are agents

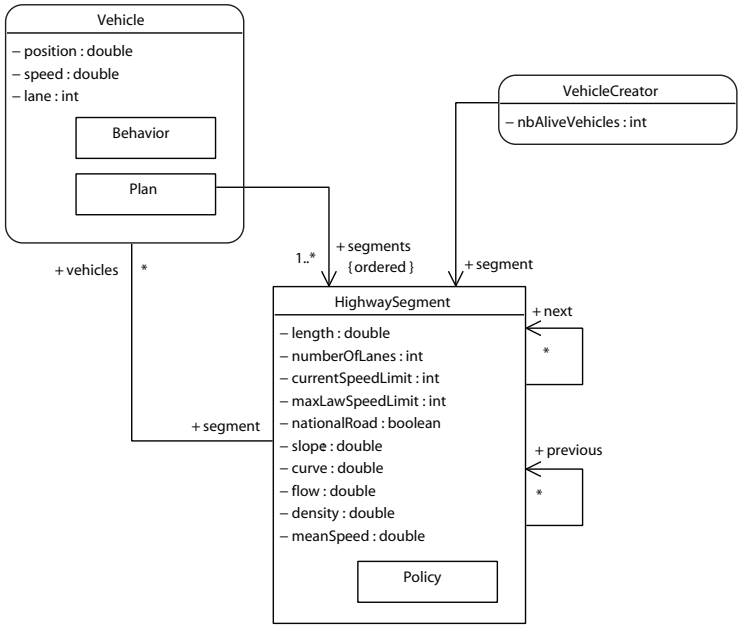

Fig. 2. Agent diagram of the system

Dynamic Description The time of the simulation is discrete. We send a time step message to every agent at each step of the simulation, and they return an action depending on their perception and their internal behavior. The environment has a governor role and can react to some events.

The environment generates events. SpeedPolicyChangedEvent is generated by a segment each time the speed restriction is changed in a segment. The governing environment will tell the neighbor segments to reconsider their current speed limit. StepBeginEvent is an internal event which is generated by the environment itself to warn the segment that a time step has begun. StepEndEvent is the same type of event as StepBeginEvent, but it warns the segment against the end of a time step event. VehicleDestructedEvent is raised by the environment each time a vehicle finished its planning and should die. VehicleDensityChangedEvent is raised by a segment each time the density of the segment has changed. It tells the environment to reconsider its speed limit.

Agent actions generate events. VehicleCreatedEvent is launched by VehicleCreator each time it creates a new vehicle. VehicleChangedLaneEvent is posted by Vehicle every time it changes its lane position. VehicleChangedSegmentEvent is posted by Vehicle every time it leaves a segment and enters a new one.

5 Behavior Models

Vehicle behaviors are described by two different but connected models: i) the *car following model* describes how a car speeds up and brakes, and the ii) the *lane changing model* describes how a driver decides to change lane.

Car Following Model Our model is inspired by the Intelligent-Driver Model (IDM) from Martin Treiber[7], which makes the vehicle accelerate to its speed objective (see Alg. 1). it does not have a constant acceleration. It decreases from the initial acceleration (a) to zero when approaching the speed objective (s_o). The deceleration value increases from b and is not limited in the theoretical model. Because of this, the vehicles can have unrealistic deceleration, but the system is collision free.

Algorithm 1 IDM car following model (acceleration computation)

Require: v, v_f, s, T, v_{limit}, a, b, s_{min}
1: $v_o \leftarrow humanizeSpeed(v_{limit})$
2: $\Delta_v \leftarrow v_f - v$
3: $s^* \leftarrow max\{s_{min}, s_{min} + vT + \frac{v\Delta_v}{2\sqrt{ab}}\}$
4: $a_c \leftarrow a\left[1 - \left(\frac{v}{v_o}\right)^4 + \left(\frac{s^*}{s}\right)^2\right]$
5: **return** $max\{-3b, min\{a_c, a\}\}$

In Alg. 1, T is a the safety time with the ahead vehicle, values can be from 0.8 to 2 seconds. Here we use a normal distribution ($\mu = 1.5$, $\sigma = 0.5$) for this value. a is the maximum acceleration (0.8 m/s^2 for cars, 1.5 m/s^2 for trucks). b is the minimum deceleration (-2.5 m/s^2 for cars and trucks).

This model has interesting advantages since it is not based on the fact that vehicles will always keep a safe distance with the vehicle ahead. On the other hand, deceleration can be high and this can lead to bizarre behaviors, like when cars drive at a high speed and suddenly brake down with a high deceleration because of a traffic slow-down or a slowest car.

Lane Changing Model Each vehicle must at each iteration consider changing lane or not. This decision is based on two main criterions for the agent : *is it safe to go on the other lane?* (safety criterion) and *do I get a reward to go on the other lane?* (incentive criterion). In our model, the safety criterion just says that the car behind would be able to brake in order to avoid a collision. We also check that the car ahead is not to close and that if it brakes, we will have the time to avoid a collision. The incentive criterion is quite simple. Vehicles change lane every time they can increase their speed on the other lane. Furthermore we

[7] http://www.traffic-simulation.de

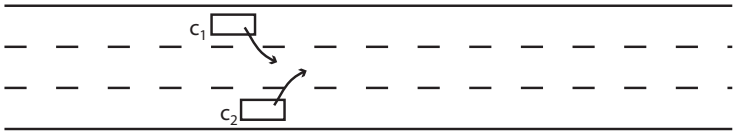

Fig. 3. Problem with +3 lanes highways

add a few biases to make vehicles go to the right lane whenever the highway is going to change from a N lanes to N-1 lanes. Informally, this gives algorithm 2.

Algorithm 2 Basic lane changing model (incentive criterion)

1: **if** lane will end soon and already on the correct lane **then**
2: ← do not change lane
3: **end if**
4: **if** lane will end soon AND not on the correct lane **then**
5: ← go to right lane with an increasing probability when approaching to the end of the lane
6: **end if**
7: **if** already changed lane in the last 10 seconds **then**
8: ← do not change lane
9: **end if**
10: **if** distance to car ahead is more than 200 meters **then**
11: ← do not change lane
12: **end if**
13: ← change lane if we can increase speed on the other lane

The two first conditions make the vehicle go to the right lane if its lane will end soon. The third condition (line 8) avoids an oscillation movement from a lane to another. Imagine five vehicles on the right lane, and no vehicles on the left lane. They all have an incentive to go the left lane, once they changed, there is no vehicles on the right lane, so they all have an incentive to change for the right, and so on. They will all change at each step to the other lane. Condition at line 10 tries to avoid cars going to the left lane when they have no other car in front of them.

Combining the Car Following and the Lane Changing Models Alg. 3 presents how to execute the car following and the lane changing models together. It ensures that all agents will have the same information when taking the decisions. A problem can occur when two vehicles compete for the same lane and think it is safe. They both will have an incentive to change for the target lane and both think that it is safe. Figure 3 shows an example of this. To avoid it, we only change to the right at odd time steps and change to the left at even time steps.

Algorithm 3 Vehicle state update loop

```
1: for all the vehicles do
2:     state ← current state of the environment
3:     decide to change lane or not according to state
4: end for
5: for all the vehicles do
6:     change lane if decided
7: end for
8: for all the vehicles do
9:     newState ← current state of the environment
10:    acceleration ← compute the new acceleration according to newState
11: end for
12: for all the vehicles do
13:    update the speed
14:    update the position
15: end for
```

Generation of Vehicles and plans For the *generation of vehicles*, we used the number of registered vehicles in the canton (swiss regions) where the segment is. We put a defined percentage of vehicles (N) on highways. We also generate trucks on the basis of country statistics. Concerning the starting place, vehicles are created uniformly in the canton. Therefore every canton generates a predefined flow of vehicles in respect to its registered car population. Because official data from the Swiss Federal Roads Authority[8] were not of sufficient granularity, we decided to create cars continuously. A realistic simulation should take into account different timing.

Each generated vehicle immediately has a deterministic assigned route plan, which can not change. This plan is however generated randomly. In future work, we will use demographic statistics and short-path algorithms to generate more realistic plans.

6 Platform Design

We describe in this section the design of our agent-based micro-simulation platform. After presenting the simulation engine interfaces, we discuss the simulation core.

6.1 Simulation Engine Interfaces

Figure 4 shows the class diagrams of the ch.epfl.lia.simengine package. This package intends to provide useful interfaces or classes in order to implement a simulation core with a governing environment. Most of the types are abstract or even just interfaces:

[8] http://www.verkehrsdaten.ch/downloads/AVZ-StandorteStand012005.pdf

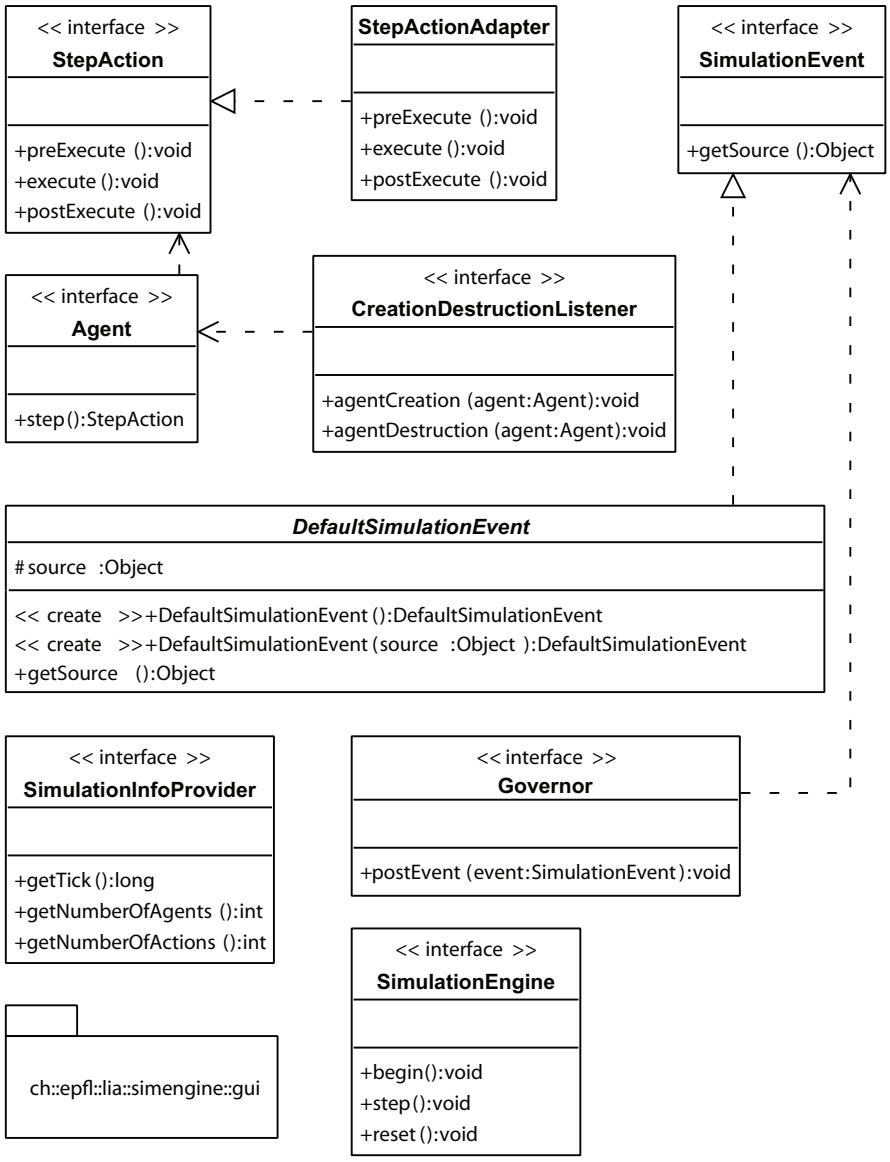

Fig. 4. Class diagram of the ch.epfl.lia.simengine package

StepAction represents an action made by an agent at one step of the simulation. A StepAction provides a way to execute a first piece of code (preExecute()), then execute a second piece of code (execute()) and finally execute a third piece of code (postExecute). preExecute() will be called on every agent before the call to execute(), and execute on every agent before postExecute..

StepActionAdapter is just an empty StepAction provided for convenience.

Agent each agent should implement this interface. step() method is called at each step of the simulation and should return a StepAction instance.

Governor must be implemented by the governing environment classes, and provides a way to post events.

SimulationEngine must be implemented by the core class of the simulation engine.

SimulationInfoProvider is generally implemented by the same class as SimulationEngine. It helps other parts of the software to get a few basic informations about the simulation.

SimulationEvent represents a synchronous event of our event-based simulation engine.

DefaultSimulationEvent is a generic implementation of the SimulationEvent interface.

CreationDestructionListener should be implemented by classes which want to be warned about agent creations and destructions.

6.2 Simulation Core

Static Description Figure 5 presents an elided and simplified class diagram of the simulation's core. Interfaces which are not part of the ch.epfl.lia.ih.sim package (or one of its subpackage) are represented as provided interface (a circle) as the UML specification lets us do it.

IHSimulationEngine gathers the logic of the simulation engine. We find in this class references to all the agents, space objects and the method which executes a step of the simulation.

HighwaySegment and HighwaySegmentImpl are the interface and the default implementation of our space. Each one is connected to a list of previous and next segments, and contains also a list of Vehicle.

SegmentGovernor contains the governing rules and reactions to events posted by other entities. Note that each segment is also a governor, so there are many governors.

Vehicle and VehicleImpl are the interface and the default implementation of the vehicle agent. Each vehicle keeps a reference to its creator and to its current segment.

VehicleCreator and VehicleCreatorImpl are the interface and the default implementation of the vehicle creator agent. Each one keeps a reference to the segment where it creates new vehicles.

Here are some other comments about this diagram :

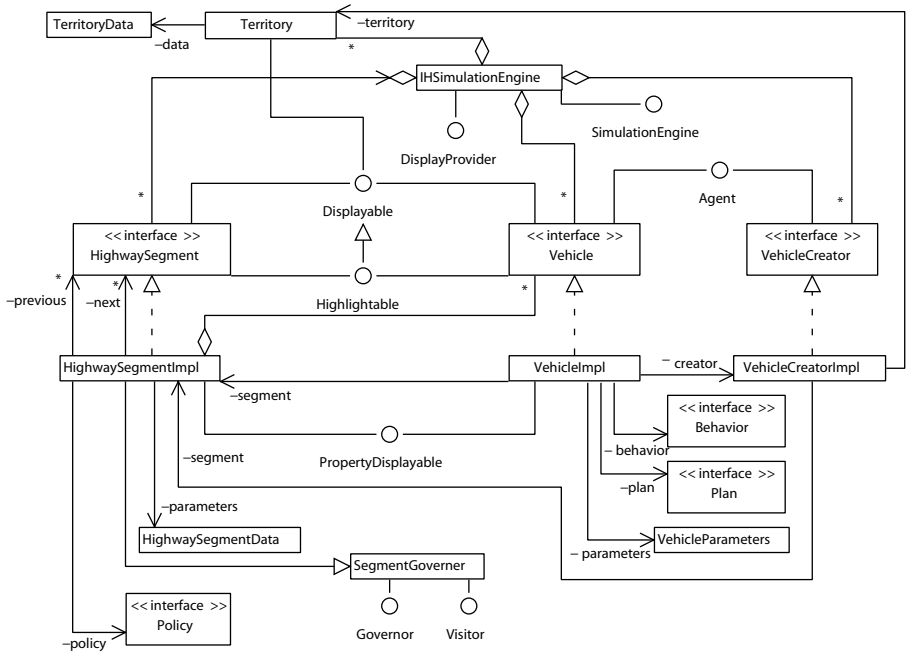

Fig. 5. Elided class diagram of the ch.epfl.lia.ih.sim

– HighwaySegmentImpl, VehicleImpl and VehicleCreatorImpl classes contain our default implementation of the agent's logic.
– HighwaySegment, Vehicle and VehicleCreator interfaces define the method their implementation must follow. We designed it this way in order to let someone change completely the implementation without a huge change elsewhere in the software.
– HighwaySegmentImpl and VehicleImpl externalize a large part of their code. This is done in order to make a very clear separation between the logic of the agent and their intrinsic model or their constant parameters. For instance, VehicleImpl updates at each step of the simulation its speed according to the new acceleration which is computed by an instance of Behavior. This separation has a great advantage since it provides an easy way to change the behavior model or the plan computation of a vehicle. One can even have multiple instance of VehicleImpl with different types of behavior.

Dynamic Description The core of the simulation is the class IHSimulationEngine. This class keeps references to every agent and space. At each step of the simulation, method step() is executed. Figure 6 shows a simplified activity diagram of this method.

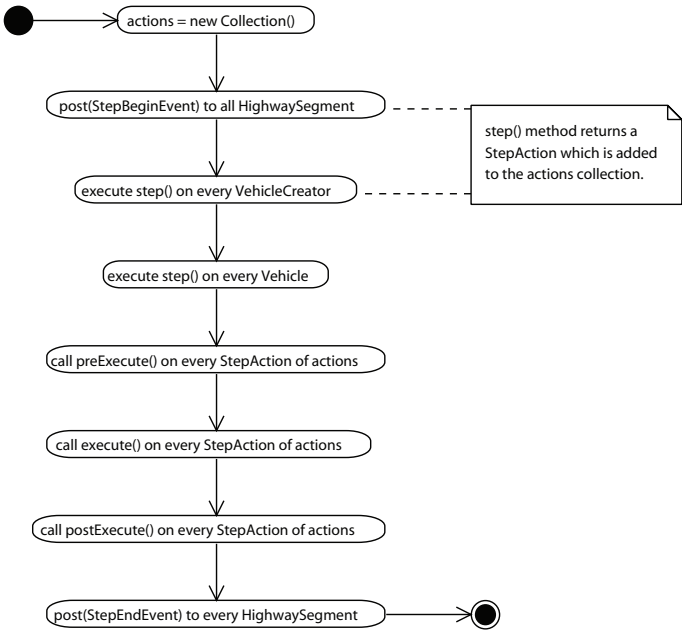

Fig. 6. The IHSimulationEngine.step() method activity diagram

Environment State As we said in Sec. 5, we first need to execute some code on all agents and then execute another piece of code, and so on. This is easily done with StepAction facilities.

We also need to save the state of an agent. We can not simply perceive properties of the agent whenever we want. Even if the step method is theoretically executed simultaneously on every agent. In reality a few agents will change their state before others. Therefore agents need a way to correct state when taking decisions. To do this, we use VehicleState and LaneState which help to keep agent or environment state.

6.3 The Events

Static Description Figure 7 presents the static description of the events system and Fig. 8 shows the attributes of every event.

We use here a visitor design pattern to make the implementation and extensibility easier. We could even imagine a way to generate automatically code of the SimulationEventVisited subclasses and the Visitor. It would be useful for a project with a huge amount of different events. The SimulationEventFactory is used here for performance reason.

Dynamic Behavior Events are launched and treated synchronously (one would need to make a few design changes to make the system asynchronous). Events

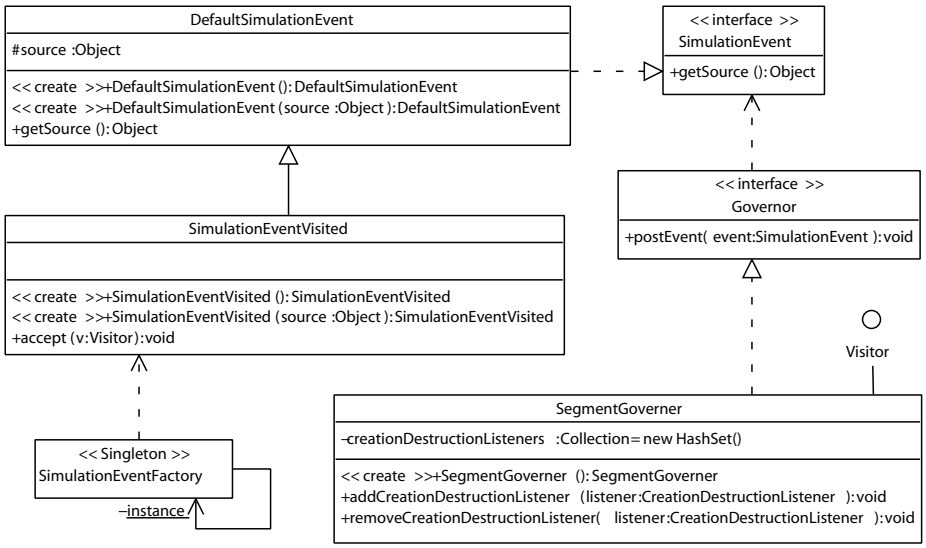

Fig. 7. Class diagram of types related with the event system

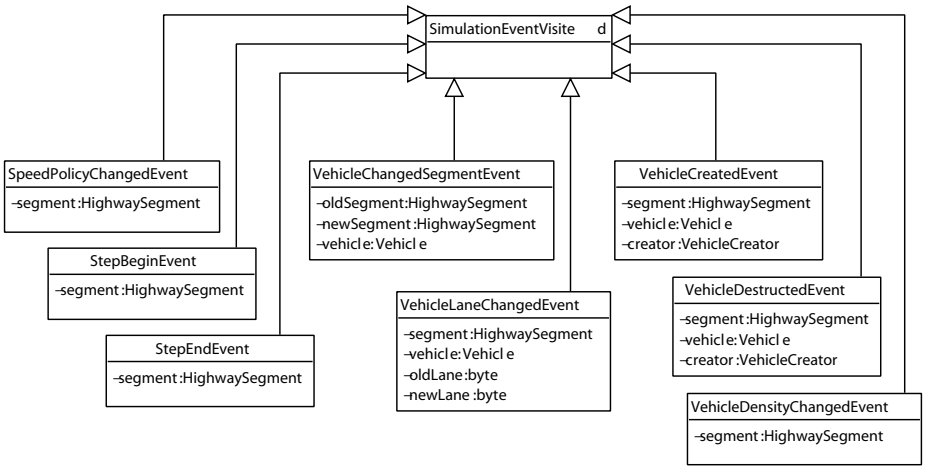

Fig. 8. Class diagram of all the events

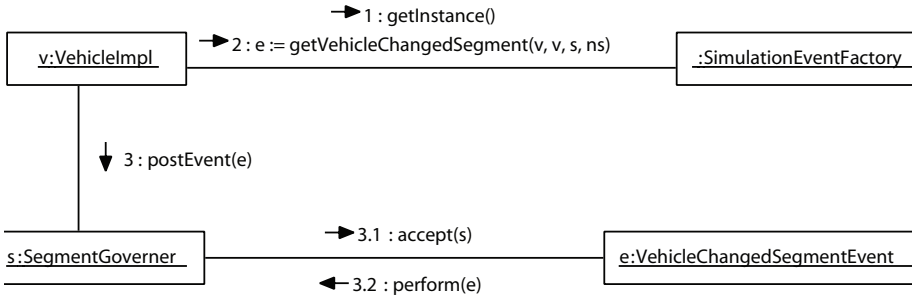

Fig. 9. An event launch example collaboration diagram

are launched each time it is needed. For instance, when a vehicle wants to move from one segment to the next one, it launches a VehicleChangedSegment by posting it to its current governor. Figure 9 shows how this event is posted and treated.

7 Experiments

Vehicle Generation As said in Sec. 5, we can calibrate the simulation to generate a percentage of the registered vehicles. This is difficult since knowing how many vehicles can drive simultaneously on Swiss highways is not obvious.

Swiss highways are composed of 1'855 km of roads. Since these roads have two possible directions and can have multiple lanes, the total length of lanes is about 7'550 km. Supposing a high congestion of 40 vehicles/km everywhere (this means one car each 25 meters on every lane and every highway segment), this leads to an estimation of 302'000 vehicles. It means that $N = 6\%$ of the Swiss vehicles would be on the highways. It can seem very low but, we should not forget that all the cars are never used at the same time and that there is a lot of other roads than highways in Switzerland. And of course, in reality at some place there is much more vehicles than at others, 40 vehicles/km is just an overestimated value of what could be a maximal congestion level.

We have made tests with different values of N (the maximal percentage of alive vehicles at a precise time). Table 1 shows how many vehicles can be simultaneously alive and how much time it costs to simulate a certain time. The first remark is about the theoretical value which is not equal to the practical one. It comes from the way of generating vehicles. Each creator segment has a physical maximum flow of vehicles and depending of the local conditions (i.e. a traffic jam on this segment), it can be lower that what it should be to ensure the theoretical production of cars. Thus it is absolutely normal to have a lower value.

[9] Tests made on a 4 x 3 Ghz 64-bits processors computer with 4 Gb RAM.

N	Theoretical	Practical	Simulated time [h]	Real time [h][9]
10 %	492'230	249'000	1:30	24:00
5 %	246'115	192'000	2:35	60:00
2 %	98'445	95'000	1:00	8:00

Table 1. Maximum number of vehicles with respect to N

Tests of the Models We ran the simulation with different values of N and looked at some randomly chosen place to see if the flow of vehicles we simulate is near reality or not. Vehicles were not always perceiving the current reality and were basing their decision on a partial future state. This was leading to many collisions, but since they are automatically cleared[10], the simulation was realistic. Table 2 shows the measures we found depending on the N value. The simulated time is the value given in table 1.

Place	Real flow	$N = 10\%$	$N = 5\%$
Muttenz	10700	4140	4551
Wuennewil	2342	3318	3533
Grandvaux	5662	3941	3798
Monte Ceneri	3243	3432	3732
Giessbachtunnel	983	2053	2103
Erstfeld	2192	2143	1366
Bardonnex	3656	1834	1783
Oftringen	5928	3304	3625

Table 2. Mean flow measurements with respect to N

Values are very far from reality. However we remark that where there is a high mean flow value in reality, there is also a relative high mean flow in the simulation. This lets us think that even if our vehicle generation method is not realistic, it does not give arbitrary values.

8 Conclusion

We developed a micro-simulation of the Swiss highway network in order to show that the governing environment can be useful for the modeling and design of agent-based micro-simulations. In our simulation platform, the design has shown to be very flexible. Future work will consist in improving the vehicle behavior modeling and the performance of the platform, and in actually implementing the adaptive and distributed speed limit regulations in order to achieve an optimal car throughput. We shortly explain hereafter those points.

The model of a vehicle should become more realistic. Collisions should be avoided when two segments merge in one, including highway entries. We think

[10] The vehicle which causes the collision (the vehicle at the back) is deleted and everything continues as if nothing happened.

this is very tricky to solve since road granularity information is not detailed enough to let us have finer grained models. The lane changing model should also be improved, especially at the end of lanes (when N ways merge to N-1 ways). Our model is not yet very good and produces unrealistic traffic jams.

Running a simulation with hundreds of thousands of agents is not costless. To simulate a real scenario with many vehicles in a reasonable time, we have to make deeper changes in the architecture. A way to do it is to distribute the computation on several computers. We estimate that our architecture should be easily transformable into a distributed one, for instance with segment distribution and asynchronous events.

However, the most important remains the realization and testing of an intelligent distributed speed restriction policy. We are currently working on this using a family of distributed constraint optimization algorithm [8].

References

1. D. Weyns and H. Van Dyke Parunak and F. Michel and T. Holvoet and J. Ferber. Environments for Multiagent Systems, State-of-the-art and Research Challenges. In D. Weyns, H. Parunak, and F. Michel, editors, *Environment for Multi-Agent Systems - Post-proceedings of E4MAS'04*, volume 3374 of *LNCS*. Springer, 2005.
2. E. Denti, A. Natali, and A. Omicini. Programmable Coordination Media. In D. Garlan and D. Le Metayer, editors, *Proceedings of the Second International Conference on Coordination Models, Languages and Applications (Coordination'97)*, number 1282 in LNCS, pages 274–288. Springer Verlag, September 1997.
3. S. Franklin and A. Graesser. Is it an Agent or just a Program? A Taxonomy for Autonomous Agents. In J.P. Muller, M.J. Wooldridge, and N.R. Jennings, editors, *Proceedings of ECAI'96 Workshop (ATAL). Intelligent Agents III. Agent Theories, Architectures, and Languages*, number 1193 in LNAI, pages 21–35, August 1996.
4. M. P. Georgeff and A. S. Rao. The Semantics of Intention Maintenance for Rational Agents. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, number 1202 in LNAI, pages 704–710, 1995.
5. Kai Nagel. Multi-agent transportation simulation (draft). `http://www.vsp.tu-berlin.de/archive/sim-archive/papers/book/book.pdf`, 2004.
6. P. Noriega and C. Sierra. Electronic institutions: Future trends and challenges. In M. Klusch, S. Ossowski, and O. Shehory, editors, *Cooperative Information Agents VI*, volume 2446 of *LNCS*. Springer Verlag, 2002. 6th International Workshop (CIA 2002), Madrid, Spain, September 18-20, 2002. Proceedings.
7. A. Omicini, S. Ossowski, and A. Ricci. Coordination infrastructures in the engineering of multiagent systems. In F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*, chapter 14, pages 273–296. Kluwer Academic Publishers, June 2004.
8. Adrian Petcu and Boi Faltings. Dpop: A scalable method for multiagent constraint optimization. In *IJCAI 05*, pages 266–271, Edinburgh, Scotland, Aug 2005.
9. M. Schumacher. *Objective Coordination in Multi-Agent System Engineering - Design and Implementation*. Number 2039 in LNAI. Springer Verlag, 2001.
10. M. Schumacher and S. Ossowski. The governing environment. In D. Weyns, H. Van Dyke Parunak, and F. Michel, editors, *E4MAS*, volume 3830 of *Lecture Notes in Computer Science*, pages 88–104. Springer, 2005.

11. *A UML Profile for Agent-Oriented Modeling*. Proceedings of Third International Workshop on Agent-Oriented Software Engineering (AOSE-2002), July 2002.
12. D. Weyns, M. Schumacher, A. Ricci, M. Viroli, and T. Holvoet. Environments in multiagent systems. *Knowledge Engineering Revue*, 20(2):127–141, 2005.