Utrecht University

Master's Thesis Artificial Intelligence

(45 EC)

Norm-based distributed controllers in a Multi-Agent System

Authors:
J. T. Baumfalk
B. W. A. Poot

Supervisor:
dr. M. M. Dastani
Second assessor:
prof. dr. J-J.Ch. Meyer
Advisor:
B. J. G. Testerink

July 2015

Universiteit Utrecht

Abstract

Autonomous vehicles will most likely participate in traffic in the near future. These autonomous vehicles have private goals they want to achieve, which might conflict with the goals of the traffic system designers. To ensure that the goals of the system are met, the vehicle behavior needs to be influenced. Since the vehicles are autonomous, their behavior cannot be controlled directly. One way of influencing behavior indirectly is through norms. A norm is a violable rule that describes correct behavior. Vehicles who do not comply with the norm may receive a sanction, such as a monetary fine. The norms are issued to the vehicles by traffic control systems, which monitor the vehicles and sanction them when norm violations are detected.

In this thesis, we present a formal model and a corresponding implementation for a norm-based multi-agent system for traffic. We created a formal model of vehicles, traffic regulations and traffic controllers and implemented this in an extension of a state-of-the-art traffic simulator named SUMO. The features of this extension are presented in several demonstrative experiments.

Acknowledgements

We would like to express our gratitude to our supervisor Mehdi Dastani for the useful comments, feedback and cooperation during the writing of our paper and this master thesis. We are also grateful to John-Jules Meyer for reading and grading our thesis. Furthermore we would like to thank everybody who provided useful tips, feedback and support: Lies Dijkstra, Ted Poot and Minke Brandenburg. But most of all we would like to thank Bas Testerink, who always quickly provided feedback, was ready to give advice, could share in our frustration, and whose support greatly improved the quality of this thesis.

Work Distribution

This thesis is a joint effort of both Barend Poot and Jetze Baumfalk. We have worked together on the model and its implementation for more than nine months. During that time, we both profited from each others insights, comments and corrections. It is therefore impossible to exactly determine who is the author of what. However, we did keep track of who wrote the main part of each chapter. For the sake of clarity and transparency, you will find the 'main' author of each chapter in the table below. However, it is important to stress that we are both the spiritual fathers of each chapter and the application accompanying this thesis.

Table 1: Work distribution in this thesis

Chapter	Title	Main author
1	Introduction	Barend
2	Background	Barend
3	Model	Jetze
4	Implementation	Barend
5	Experimentation	Jetze
6	Conclusion	Jetze

Contents

1 Introduction **1**
1.1 Methodology . . . 2
1.2 Research Questions . . . 3
1.3 Overview . . . 4

2 Background and Related Work **5**
2.1 Agent Systems . . . 5
2.1.1 Agents . . . 5
2.1.2 Norms . . . 8
2.1.3 Norm-based controllers . . . 12
2.2 Simulation . . . 13
2.3 Related Work . . . 15
2.4 Chapter Summary . . . 16

3 A Multi-Agent Model for Traffic **17**
3.1 Environment . . . 17
3.2 Norms and Controllers . . . 20
3.2.1 Example Norm . . . 23
3.3 Agents . . . 25
3.3.1 Profile . . . 26
3.3.2 Actions . . . 27
3.3.3 Directive distance measure . . . 28
3.3.4 Action Selection . . . 29
3.3.5 Deliberation Example . . . 30
3.4 Chapter Summary . . . 32

4 Implementation **33**
4.1 SUMO . . . 34
4.2 Global control structure . . . 35

4.3 Framework Structure 37
4.3.1 Flow of control 39
4.3.2 Road Network 41
4.3.3 Norms 41
4.3.4 Norm-based controllers 43
4.3.5 Agents 44
4.4 Chapter Summary 44

5 Experiments **47**
5.1 Merge scenario 48
5.2 Experiment 1: SUMO and our extension 49
5.3 Experiment 2: Simple norms and Advanced norms 51
5.4 Experiment 3: Violating norms 52
5.5 Experiment 4: Observation sharing between controllers 54
5.6 Chapter Summary 56

6 Conclusions & Future Research **57**
6.1 Answering the research questions 57
6.2 Discussion 58
6.3 Future research 59

Bibliography **61**

Appendix: User Guide **65**

Chapter 1

Introduction

Automation takes place in all our lives. More and more devices around us display autonomous behavior and many applications run with minimal or reduced human intervention. These devices range from your typical household thermostat to your smartphone. One of the most recent and interesting developments in the field of automation is that of the autonomous vehicle. Companies like Google [21] and Nissan [11] are creating self-driving cars capable of driving autonomously, without intervention from a human driver. These vehicles pose both a challenge and an opportunity for our current traffic regulations. Currently, the Google self-driving vehicles have already driven over a million miles in the United States with just a few incidents[1]. They have proven to be able to adapt to other vehicles and the traffic regulations currently in place. However, current traffic control systems might not be optimal, since autonomous vehicles are inherently different from human drivers and the current system is tailored to human drivers. For example, these autonomous vehicles are able to cope with precise and more frequently changing directives, while for humans speed limits are given in multiples of 10 and do not change in quick succession.

These new vehicles open up an array of possibilities for new traffic control systems. These driverless cars are autonomous and may have private goals which can be in conflict with the goals of the traffic system designer. For example, like human drivers these autonomous vehicles will most likely aim to arrive at their point of destination as quickly as possible. Hence, any directives to decrease their velocity will be in conflict with their private goals.

[1] Google now posts monthly reports on their driverless vehicle on the project website: `http://www.google.com/selfdrivingcar/`

It is important therefore, that like vehicles with human drivers, these autonomous vehicles are monitored and controlled in some way while respecting the autonomy of the vehicle. To be able to handle these autonomous vehicles in future traffic networks, a system is required that preserves the autonomy of the vehicles and is relatively similar to our current traffic regulation system in order to preserve control over human drivers, allowing for mixed populations of human drivers and autonomous vehicles on the road.

In this thesis we present a prototype of such a system. We created a model of future traffic and their regulations, as well a prototype implementation of that model. This implementation enables research on the effect of new traffic regulations on future traffic scenarios with autonomous vehicles.

In the next section, we introduce the methods used to realize this implementation. Next, we pose our research questions and their correlation to the chapters. Finally, an overview of the remainder of this thesis is presented.

1.1 Methodology

Both human drivers and self-driving vehicles display autonomous behavior. Because of this defining property a natural way to represent them is by using *agents*. While the term agent remains difficult to define, for now it suffices to say that it is some autonomous entity with the ability to observe and and act upon its environment. A multi-agent system (MAS) is a system that consists of multiple intelligent agents that can interact with each other and their environment [40]. We feel that the multi-agent system perspective is a natural way to look at the development of autonomous cars in traffic. Not only does the concept of self-driving cars correspond with intelligent agents described in the literature [41], the notion of traffic regulations has been linked before with normative systems in MAS [18]. Furthermore, a multi-agent system model is able to cope with a population of heterogeneous agents. This means that besides modeling different types of artificial agents, this system can be used for modeling human drivers, since they too can be seen as some sort of intelligent agent [10].

A system that is a widely proposed way of regulating the behavior of agents, is to employ *norm-based controllers* [17]. These controllers issue norms to the agents, which tell them how (not) to behave. Through monitoring, the controllers determine whether the agents comply with the norms. Through sanctioning, the controllers can punish the noncomplying agents. In this way, agents can be coaxed (but not forced, since they are autonomous)

to change their behavior towards a desired form of behavior needed for achieving the goals of the system. This system is similar to how vehicles are currently controlled, as traffic regulations are similar to norms. Therefore, we think that is a good choice to use this norm-based controller approach in a traffic setting.

It would be expensive to verify such a multi-agent system in practice, since a large traffic network and a decent number of vehicles are needed in order to collect the necessary data. Therefore, we employ a simulator to model traffic as a multi-agent system. Simulations are often used in traffic situations as an inexpensive way to test out new policies [24]. We built our framework on top of SUMO, an open source, mature traffic simulator [8]. We will introduce this traffic simulator in Chapter 4.

1.2 Research Questions

In this thesis we will describe our framework which aims to simulate future traffic scenarios with the use of norm-based controllers in a multi-agent system. Our aim is not to represent an entirely realistic image of future traffic featuring autonomous vehicles, instead we aim to show the merit of norm-based controllers in these future traffic scenarios. The main research question of this thesis is:

> *How can future traffic be modeled and implemented using norm-based controllers within the MAS paradigm, and can this paradigm be used to improve traffic safety and efficiency?*

For the modeling part, we turn to existing theories. There is already a wide body of work regarding the monitoring and control of agents in norm-based systems. We will discuss the relevant articles and see how we can use existing and proven theories for our traffic scenario to create a formal specification for traffic from a MAS standpoint. This part will answer the first subquestion:

> *How can we model future traffic within the MAS paradigm?*

After the formal specification, we will present our implementation of the specification. A large part of the domain-specific work is already done by the SUMO software package.[2] We will explain what SUMO is and then show

[2]See the website http://sumo-sim.org for the software package.

how we implemented our specification by building on top of the simulation package. This will answer our second subquestion:

> *How can we implement the MAS model of future traffic?*

Finally, we will present several scenarios that illustrate how norm-based controllers in conjunction with the MAS paradigm can improve on future traffic in terms of safety and efficiency. This will answer our final subquestion:

> *Can norm-based controllers in conjunction with the MAS paradigm improve on traffic safety and efficiency?*

1.3 Overview

The remainder of this thesis is structured as follows. In Chapter 2 we introduce the reader to the concepts used in this thesis and provide some background. In Chapter 3 we discuss how we can construct a formal multi-agent model for traffic. This will answer the first subquestion. In Chapter 4, we discuss how to implement the formal framework developed in Chapter 2. We look at a state-of-the-art traffic simulator, SUMO and see how we can integrate this with our multi-agent system, thus answering the second subquestion. In Chapter 5, we will validate our framework through a number of experiments. Through the discussion of the results, we will answer subquestion three. After this, we will answer the main research question, summarize the findings and contributions for this thesis and point into directions for future research in Chapter 6. Finally a user guide for our extension is provided in the appendix.

Chapter 2

Background and Related Work

While the notion of agents, controllers and norms are well-discussed terms within computer science and artificial intelligence, it is far from trivial to provide an accurate definition to either of these concepts. The goal of this chapter is to provide the reader with a broad overview and context surrounding the concepts covered in this thesis. First, we start by describing Agent Systems. We explain the notion of an agent as autonomous entity in Section 2.1.1 and look at how such agents can be combined in a multi-agent system. Next, we look at controllers and norms which can be used to regulate the behavior of agents in Section 2.1.2 and Section 2.1.3. Finally, we describe the notion of agent-based simulation in Section 2.2. After this background overview, we review some related work.

2.1 Agent Systems

2.1.1 Agents

We often ascribe feelings and thoughts to computers and machines, since many of these machines have an active influence on our environment. For example, a simple household thermostat is an active entity since it is able to autonomously influence the room temperature. It shows a sense of agency, the capacity to act in its given environment autonomously. These active entities are what we call "Agents". A popular definition of an agent in computer science is given by Wooldridge [40]:

> "*An agent is a computer system that is situated in some envi-*

ronment, and that is capable of autonomous action in this environment in order to meet its design objectives."

An agent is an entity that observes using sensors and operates within its environment in order to reach a certain goal. It is able to execute actions using actuators and modify the environment in the process. It can (partially) observe the environment which may change dynamically at any time. Agents may apply a *Sense-Plan-Act* [29] cycle to interact with the environment and other agents. First, the sensors on the agent observe, perceiving new information about the environment. The acquired information is stored in the agent's knowledge base and old facts about the environment are updated in accordance with the new data. Secondly, the agent will reason about these revised facts by generating a plan that, if executed, will establish the next state associated with the most desirable environment. Finally, the agent will execute (act) this plan upon the environment using its actuators and changing the environmental state, at which point the cycle will restart. The way in which an agent interacts with its environment and possible other agents or artifacts distinguishes if the agent conforms to a *weak* or *strong* notion of agency as described by Wooldridge [41]. The characteristics for weak agency are:

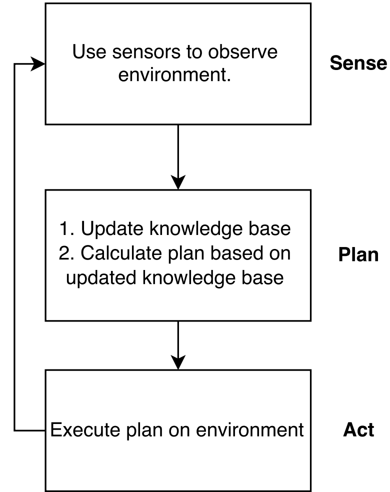

Figure 2.1: A general Sense-Plan-Act cycle

- autonomy: the agent should have some degree of control over the actions it chooses and its internal state.
- social ability: the agent should be able to communicate with other agents, artifacts and/or humans via an agent-communication language.
- reactivity: the agent senses the environment and reacts to changes in this external world.
- pro-activeness: the agent not only reacts to events in the world, but also actively tries to achieve its own goals.

Thus, the simple thermostat mentioned above represents an agent with a weak notion of agency, it autonomously decides to turn on the boiler if its internal state shows it needs to do so, it can communicate by displaying the current temperature and will react to temperature changes in the environment. It continually checks the environment for changes and reacts accordingly.

The stronger notion of agency relies on a *mentalistic* view. In this view, an agent can reason more explicitly about its internal state. The strong notion of agency is characterized by its use of the notions of beliefs, desires and intentions and even emotions [12, 5, 28]. Rao and Cohen elaborate on this Belief, Desire and Intention (BDI) architecture of agents [13, 31]. In this work, the behavior of agents and their decision-making process is determined by these three mental attitudes. These three mentalistic terms define the informational, motivational and deliberative states of the agent respectively. The knowledge base mentioned above represents the beliefs of an agent, it is everything the agent knows about the world, regardless of it being true. The desires of an agent are its goals, that which determines together with its current beliefs which plan or action is to be executed next. The intentions of an agent show what the agent is committed to do, it intends to do something if it has started executing a certain plan.

As stated previously, agents are situated in an environment which they can sense and influence. The environment has a certain *state*. An agent has the capability to (partially) sense the environment and can differentiate between different states in the environment. An agent takes actions based on its perceived state and its internal goals. The performed action can influence the environment, thus changing the environmental state and (perhaps) the perceived state. Often, an agent is not certain if its action succeeds or even what the effect of its action is. Thus, it predicts it is in one of a certain number of states after acting, instead of just the one state in which its action had the desired outcome.

The action which the agent will execute on the environment can be decided in several ways. For example, the agent can be provided with a direct mapping from perceived states and goals to actions. Another often used and intuitive technique for selecting the next course of action is the use of a *utility function* [40]. Such a function calculates the desirability of a future state of affairs, by investigating which goals are satisfied in which states. The agent predicts the future states of the environment following from the possible actions. It then employs the utility function which assigns a real

value to every environmental state and finally decides on the action which leads to the most desirable state, i.e. the state with the highest numerical value.

A system with multiple agents situated in the environment is called a Multi-Agent System (MAS). The interesting aspect of such a system is the interaction between agents. In order to work towards a shared goal, or even just their own goals, these agents benefit from being able to *communicate*, *cooperate*, *coordinate* and *negotiate* with each other. We can discern between first-order and second-order intentional systems [15]. In a first-order intentional system, agents have a mentalist attitude, that is some kind of beliefs and desires. In a second-order intentional system, agents also have beliefs and desires *about* beliefs and desires, either their own, those of others or both.

Since agents are often self-interested entities, next to cooperation MAS also show elements of competition. A well-known example is the *Contract-Net Protocol* [33]. The Contract-Net Protocol is a task-sharing protocol which resembles an electronic marketplace. Here agents can either announce tasks to the network as a manager, or accept tasks as a contractor. Agents recognize tasks that will help realize their goal and bid against each other for these tasks. If they are unable to complete their assigned tasks however, sub-contracts can be generated and awarded to other agents. Agents both cooperate by helping each other to reach their respective goals, while competing by bidding for tasks. This protocol shows that both cooperation and competition can be used in order to solve problems faster and with better results.

2.1.2 Norms

As described in Section 2.1, agents interact in a common environment. However, besides the goals and desires of agents themselves, there may also be globally desired system properties. An example of this are traffic systems, where every agent (vehicle) in the system has a certain set of goals (i.e. get home as soon a possible), the system itself aims to keep every agent safe with the use of globally system properties (i.e. do not drive faster than 120 km/h). In order to ensure these global desired system properties, the behavior of agents needs to be coordinated. Norms are an effective means to coordinate agents' behavior [16]. Norms are used to constrain behavior by laying down rules of how agents ought to behave or should not behave.

Typical norms (regulations) in traffic systems are, “Keep your distance” or “Do not drive through a red light”. If a norm’s specification is violated, then the displayed behavior is undesired.

In this thesis we are interested in using norms to coordinate or influence autonomous vehicles’ behavior, not directly controlling the vehicle themselves. This method requires controlling structures that can both monitor (observe the environment and the agents in the environment) and control (impose sanctions on agents). We will elaborate on these controllers in Section 2.1.3. If a controller in an environment prevents agents from creating a violating state, we talk about *regimentation* [22]. In case of regimentation, while agents may be able to execute an action which would bring about a violating state, these states are not allowed by the controller monitoring the system and can thus not be reached. Simply put, if agent behavior cannot violate norms due to a controller, regimentation is applied. In this case where controllers are designed to prevent violating states, there is no use for violations since norms cannot be violated. Another solution is norm *enforcement* [20]. With norm enforcement, a controller aims to impede violating agents in order to enforce desired behavior. Sanctions are used by the controllers in order to do so.

Norms are statements about the desired behavior agents in the multi-agent system, not the actual behavior in the MAS. For computer science, this is an important distinction that was often disregarded in the past. Formally, such statements can be modeled by *Deontic Logic* [38]. Meyer *et al* [27] illustrate the importance of deontic logic in normative computer science systems with the following argument:

> *“Until recently in specifications of systems in computational environments the distinction between normative behavior (as it should be) and actual behavior (as it is) has been disregarded: mostly it is not possible to specify that some system behavior is non-normative (illegal) but nevertheless possible. Often illegal behavior is just ruled out by specification, although it is very important to be able to specify what should happen if such illegal but possible behaviors occurs! Deontic logic provides a means to do just this by using special modal operators that indicate the status of behavior: that is whether it is legal (normative) or not.”*[27].

A multi-agent system where norms are employed is called a normative multi-agent system. In such a system, the agents are not directly controlled.

However, agents should be able to reason about norms and subsequently modify their behavior. In order to do so, deontic logic is used to formally cope with normative concepts such as *obligations* (an action required to take) and *prohibitions* (an action forbidden to take). Some research has been done in this area by Meneguzzi *et al* [26], considering the impact of norms on the practical reasoning of agents, including an implementation in the AgentSpeak(L) language. Furthermore, Alechina *et al* [1] presented the norm-aware programming language N-2APL, in which agents try to achieve their goals and obligations while respecting the prohibitions as stated by norms. An explanation of Deontic Logic does not fit the scope of this thesis however, since we build on the definition of norms as provided by Tinnemeier [36], who states that full-blown deontic logic is not needed for a norm-based implementation since it is impractical to implement just to make use of the limited possibilities it provides.

A formal definition of norms is provided by Tinnemeier [36] who considers norms as consisting of norm schemes and norm instances. A norm scheme is described as a conditional obligation or prohibition with a certain deadline. These obligations and prohibitions are described by what is called *brute facts*. Tinnemeier uses the definitions of brute and institutional facts as described by Searle [32]. Brute facts represent situations in reality, albeit mental or physical. A common brute fact describing an agent may for example be that agent's position in the world. Institutional facts however are facts that depend on social institutions for their existence; they are an imposition on the brute state of the environment. A brute fact may be that an agent has a piece of paper with some specific print on it. The institutional imposition might be that the agent owns 5 euro which can be exchanged for goods.

A norm scheme as described by Tinnemeier [36] states under which condition a specific obligation or prohibition should be instantiated for an agent. A norm scheme is defined by a condition which establishes when the norm becomes active, an obligation or a prohibition of what the agent must (not) do and a deadline, which defines when the obligation or prohibition ends. An obligation should be met before the deadline and a prohibition may not occur before the deadline. The condition, obligation, prohibition and deadline are all conjunctions of brute and institutional facts that should be seen by the agent as a set of instructions. For example, If the system is in such a state that this conjunction of literals holds for the condition, then there is either a duty or it is forbidden to establish the brute state in which the

obligation or prohibition holds.

A norm instance is created when the condition of a norm scheme is satisfied. It is an active unconditional obligation or prohibition with a deadline. In this manner every norm instance is coupled with a specific norm scheme. Since norms can be violated, undesired agent behavior may still occur. One way to impede future undesirable behavior is to punish agents who do not conform to the norms. The agents can be punished by imposing sanctions on them when a violation occurs.

A description of sanctions can be found in work done by Dastani *et al* [14, 37]. In their work, two constructs related to sanctioning are used: counts-as rules and sanction rules. The counts-as rules state which states of the environment 'count-as' being a violating state, and which type of violation they are. The sanction rules couple these violations to changes in the environment, i.e. a punishment to the violating agent. It is important to stress that although sanctioning is most often associated with a negative effect on an agent, in essence it is nothing more than a change to the model, and therefore could also be used to reward.

Tinnemeier couples every sanction rule to a specific norm by creating an equivalence between the precedent of the sanction rule and a violation on either the obligation or prohibition of the norm instance. If this sanction is imposed on an agent, its brute state may be changed, for example, the amount of currency he owns could be decreased, representing a monetary fine.

To be able to cope with these norms and react to imposed sanctions, our agents are required to be *norm-aware*. We take our definition of norm-aware agents from Alechina *et al* [1]:

> *"(...) it can deliberate on its goals, norms and sanctions before deciding which plan to select and execute. A norm-aware agent is able to violate norms (accepting the resulting sanctions) if it is in the agent's overall interests to do so, e.g., if meeting an obligation would result in an important goal of the agent becoming unachievable."*

This means that agents in a normative environment have a more elaborate reasoning cycle than agents who are not norm-aware. Besides predicting if the state that results from the execution of their actions approximates the goal, they also deliberate if this may result in violating a norm and if the

sanction that will consequently be imposed on them is worth closing in on their goal.

2.1.3 Norm-based controllers

As stated above, to properly enforce norms a controller that monitors these agents is required. Such a controller is a common thing in environments where many agents operate. In a first-order intentional system, agents only deliberate about their own utility and actions. As we have seen in Section 2.1.2, norms can be used to announce the globally desired system properties. However, in order to be effective, these properties also need *enforcement*. A widely proposed and effective mechanism to control individual agents are *norm-based controllers* [9].

Controllers have two tasks, *observing* and *controlling* the agents in the environment. The notion of controllers is based on the norm-based artifacts mentioned in Dastani *et al* [14]. Here, a normative multi-agent system is considered to contain two separate modules, an organization module and an environmental module. The agents move about and perform actions in the environmental module while the organization module specifies the norms and enforces them by monitoring, controlling and imposing sanctions on agents. In this manner, controllers maintain the agents' autonomy while promoting desirable behavior.

The monitoring and controlling of these norm-based controller structures can be implemented in two distinct manners, either *endogenous* or *exogenous*. When the controllers monitor the agents with the use of external sensors and inform them of norms or sanctions by sending messages, they are exogenous. If this is the case, norms are defined explicitly. This allows for both maintenance of current norms and easy implementation of new norms. With endogenous controllers, the norms are intrinsic to the agents design. This means that if a new norm is implemented, a new type of agent should be inserted into the system. While this type of system is more secure since monitoring and controlling is an intrinsic process, it would be impractical to modify or introduce norms at runtime.

Many applications where controllers are employed feature a single all-knowing institution [2, 14]. However, distributed control has various benefits. Distributed systems have an increased robustness, parallel processing of data, less communication of data and modular maintenance [35]. We believe this is also strongly the case for future traffic where sensors and controllers

are geographically distributed and where different parts of road networks are governed by different sets of norms and regulations. For these reasons and since the amount of data generated and processed is big, we aim at distributed controllers monitoring sensor data as well as enforcing norms. While decentralized controlling is more robust in the sense that it avoids the problem of single point of failure, it also introduces new issues. In decentralized monitoring one has to make sure that the sensors are cooperating correctly to detect norm violations in a timely fashion [34].

2.2 Simulation

Of special interest for this thesis is traffic simulation. A traffic simulator is a tool used to model transportation systems in virtual simulations. These simulations are used to, for example, determine bottlenecks in a road network, try out new policies before implementing them on the road or to simulate traffic flows as part of monitoring in a real life, real-time system. It is often much more practical to use a virtual simulation, since real-life experiments would be costly and take a considerable amount of time.

One of the main defining features of a simulator is the amount of detail in which traffic is simulated. The simulation may have different levels of granularity; these levels can be classified as microscopic, mesoscopic and macroscopic. [19, p. 137]. *Microscopic models* study the individual entities in traffic simulations; vehicles, bikes and possibly pedestrians. These models keep track of the properties of every entity, their velocity, position, acceleration and so on. *Mesoscopic models* are less detailed, in these models the vehicles are represented in platoons, (groups of several vehicles) where every vehicle in the platoon is assumed to be identical to another. Finally, *macroscopic models*, these are quite similar to the mesoscopic models, albeit more sophisticated. The platooning of vehicles is also applied, grouping several vehicles together. With this method however, equations regarding the speed, density and flow of the traffic stream are used to provide a more accurate representation.

When simulating a traffic scenario where every vehicle is represented by an agent, a microscopic level of detail is required. The following characteristics are common of a microscopic model.

- Ontological Correspondence. The agents in the model correspond to real world actors.

- Heterogeneous Agents. Agents are not identical in their decision making process.

- Representation of the environment. The real-world environment is modeled.

- Bounded Rationality. The agents in the model are somewhat rational, but may divert from rational behavior. This means that they try to achieve their goals, but may be faced with circumstances that force them to act non-optimal.

In validation one checks the accuracy of the model's representation of a target. A model requires precision and openness in assumptions at the service of a specific goal. Therefore, its validity should be determined for that same goal. In this section we describe three different kinds of models with different goals and therefore a different kind of validation.

The first kind of model is the abstract model. The abstract model is a very simplified model of the real world and is not intended to be a real description of the world. Instead, its purpose is to get some feeling for the problem at hand to determine an appropriate middle range model. Therefore it can be validated by its most important feature: it should portray the macro-level regularities we wish to explain. Moreover, there should be an accurate connection between micro-level behavior and macro-level regularities. Lacking abstract high-level empirical data this validation means comparing (common sense) theoretical propositions in this field of expertise to the model's behavior by a technique called sensitivity analysis.

The second kind of model is the middle-range model. This model is more fleshed out than the abstract model and has a closer link to the problem at hand. However, it is still general enough to be used for different cases. Since it has close ties to reality, it can be validated by using actual empirical data. Although in a qualitative way, a quantitative match is not expected.

The final kind of model is the facsimile model (literally: 'make the same'). The aim of this model is to represent a specific phenomenon as closely as possible, so that predictions can be made about that phenomenon. It can be validated by giving it a situation of the past and checking that it follows the same steps as were taken in the real world.

2.3 Related Work

Our approach has some similarity with that of Baines et al. [2] since they employ autonomous agents and use governing institutions to influence agents to show desirable behavior. However, Baines et al. concentrate on agents' internal architecture, situational awareness, and the communication between agents. The project is set up with realistic maps imported from the Open Street Map foundation and uses real-world data from a highway in the UK, the M25.

While our framework is related to the work done by Baines et al., the aim of our research is different. Our driver model is deliberately kept simple in order to focus on the interaction between traffic controllers on the one hand, and agents and traffic controllers on the other hand. Furthermore, our framework is not developed in order to simulate the existing real-world scenario. Finally, our framework supports decentralized traffic controllers while Baines et al. focus on a single, all knowing, institution.

Another comparable line of research has been done by Balke et al. [3]. In their extended abstract, Balke et al. discuss the difference between off-line and on-line reasoning of institutions (similar to norm-based traffic controllers) governing open multi-agent systems. They state that most research up to that point had been focused on the off-line reasoning of institutions, which can be used to research the static properties of institutions. The on-line reasoning of institutions concerns the monitoring and controlling of agents, observing if norms are being violated and informing agents if this is the case. In this implementation, there is a single institution with the title "The Governor" with which agents can communicate and receive information regarding possible consequences of their actions.

Our approach is most closely related to the on-line reasoning as described by Balke et al. However, communication between the agents and the institution is handled in a different way. Within our framework, the information provided by agents to the traffic controller is acquired via sensors. This is a more realistic representation of traffic situations, since it is often beneficial for the agent to not disclose any information about itself. Furthermore, in our model, multiple traffic controllers are present, creating a more robust and better controlled system through communication between these institutions.

2.4 Chapter Summary

In this chapter, we presented the concepts necessary for understanding the rest of this thesis. We described what agents are, and how they try to achieve their goals. We also discussed that since agents are self-centered, their goals might not align with that of the system. Norms were presented as a solution to impeding undesired agent behavior while preserving the agent autonomy. We also introduced the notion of controllers, which monitor the system and issue norms. Furthermore, we provided a short introduction into simulation, discussing the various kinds of simulation models that are used. Finally, we presented some work that was related to ours by Baines *et al* and Balke *et al* .

Chapter 3

A Multi-Agent Model for Traffic

The goal of this chapter is to introduce our formal model of future traffic. The model is based of the concepts introduced in Chapter 2 where we introduced the concepts of agents, norms and controllers. We will present our implementation of this model in Chapter 4. We will use multi-agent system concepts to formalize the notion of future traffic systems. More concretely, we will define the different parts of such a traffic system in terms of the concepts treated in Chapter 2.

The rest of this chapter is structured as follows. First, we begin with defining the environment, the structure where all the agents reside in. Second, we describe the norms which are used to guide the agent behavior. Third, we specify controllers, which issue the norms to agents and verify if the agents comply to the norms. Finally, we specify the agents, how they choose actions and how they deal with norms. For an overview of how the concepts and definitions throughout this chapter are related to each other, see Table 3.1.

3.1 Environment

We formalize the traffic domain by specifying the environment. The environment is the external world in which agents reside and where they hold some influence. In a traffic setting, the environment consists of the roads and the lanes the vehicles can drive on. The roads are connected to each other using junctions, forming the road network.

Table 3.1: Overview of all symbols used in this chapter and a description of the concept they stand for

Symbol	Description
RN	The road network
N	The set of nodes that represent junctions
N_{in}	The set of entrance nodes of the road network, subset of N
N_{out}	The set of exit nodes of the road network, subset of N
E	The set of edges that represent roads
L	The list of lanes belonging to a road
O	The set of physical objects situated in the road network
O_{veh}	The set of all vehicles in the road network, subset of O
O_{sens}	The set of all sensors situated in the road network, subset of O
I	Set of all norm instances
NS	Set of all norm schemes
C	Set of all controllers
S	The set of all sanctions
IS	The set of all information sources. It is the union of C and O_{sens}
P	The set of all agent profiles
A	The set of all agent actions

A road network consists of all the roads and junctions in the scenario. Every node in a road network must occur as a start or an end node of a road in the the road network. Some nodes have only one edge connected to them, these nodes can be seen as the *entry* and *exit* nodes. These nodes coincide with the entry and exit points of a road network. An intuitive formal representation of the road network is a directed graph. The edges correspond to the roads, and the nodes correspond to the junctions.

Definition 3.1 (Road Network)**.** The road network RN is a directed graph

$$RN = \langle N, E \rangle,$$

where N is a set of nodes that represent junctions and E is a set of edges representing roads connecting the junctions. The set of all entrance/exit nodes is denoted $N_{in} \subseteq N$ and $N_{out} \subseteq N$ respectively. ■

Each road has a certain distance and has a list of lanes. The order of the list corresponds with how lanes are connected to each other. For example, in Figure 3.1 it is possible to traverse in one step from lane 2 to lane 1 or lane 3, but not to lane 4, since lane 2 and lane 4 are not directly connected to each other. Hence, the list of lanes would be $[1, 2, 3, 4]$.

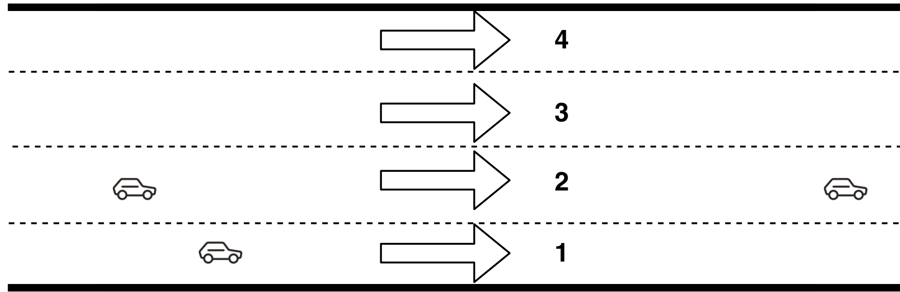

Figure 3.1: A road example with four lanes.

Definition 3.2 (Road)**.** Let RN be a road network. A road r in $RN = \langle N, E \rangle$ is specified as a tuple

$$r = \langle n_{start}, n_{end}, L, length \rangle,$$

where $n_{start}, n_{end} \in N$ are the start and end node of the road respectively, with $(n_{start}, n_{end}) \in E$; L is the list of lanes; *length* is the length of the road. ■

The road network is populated by various *physical objects*. In our formalization, the only physical objects that exists are vehicles, which drive on the road, and road sensors, which are used by traffic control systems to detect vehicles. A physical object has unique identifier. Furthermore, it has a certain position in the road network, which is determined by the road and lane they are on and their position on that road. Finally, they also have a certain velocity (which can be zero) and length.

The location of a physical object o is defined as $loc = \langle r, l, pos \rangle$. Here, r and l are the road and corresponding lane where o is located, with $l \in r.L$, $pos \in \mathbb{R}$ and $0 \leq p < r.length$ is where o is positioned on the road in meters. The properties $id, v, length$ and loc of an object o can be referred to by $o.id$, $o.v$, $o.length$ and $o.loc$ respectively.

Definition 3.3 (Physical object)**.** A physical object o is specified as a tuple

$$o = \langle id, v, length, loc \rangle$$

where $id \in \mathbb{N}$ is the object identifier, $v \in \mathbb{R}$ is the velocity in m/s, $length \in \mathbb{R}$ is the length of the object in meters and loc is the position of the object in the road network. ■

The set of all physical objects is denoted O, the set of all vehicles is denoted $O_{veh} \subseteq O$ and the set of all road sensors is denoted $O_{sens} \subseteq O$, with $O_{veh} \cup O_{sens} = O$ and $O_{veh} \cap O_{sens} = \emptyset$. A particular instance of a physical object is called a physical object state.

3.2 Norms and Controllers

As mentioned in the Chapter 2, a proposed way of regulating traffic is by using norm based distributed controllers. Specifically, norm schemes and norm instances as used by Tinnemeier were discussed in Section 2.1.2. In our model, the norm schemes and norm instances are defined in a more abstract way, allowing for a wide range of possible implementations. In the context of traffic, norm instances should be interpreted as traffic regulations or directives issued to the vehicle.

A norm instance is associated with a vehicle id (as defined in Definition 3.3). Furthermore, it has a function that is used by the controller to check norm compliance. This function takes as input a set of vehicle objects, and returns `true` if the norm is violated and `false` otherwise. In addition, the

norm instance has a vehicle state that serves as the directive, i.e. this is the state the autonomous vehicle should achieve. The vehicle state is used by the agent to determine what it should do to oblige to the norm. Furthermore, a norm instance contains a directive type, which denotes whether the directive is an obligation or a prohibition.

Note that the norm compliance function receives a set of vehicle objects instead of just the one vehicle object associated with the norm instance. The reason for this is that this allows us to model situations in which agents are not punished for non-compliance for reasons beyond their control. Situations may occur where an agent is unable to confirm to a norm, since it is restricted in its behavior because of violating agents around it. For example, suppose a vehicle is supposed to drive 80 km/h, but it cannot do so because of slower vehicle in front of it. By providing the norm compliance function with additional information about vehicles, it can detect such situations and prevent from receiving a sanction unjustly.

The norm instance also contains the sanction that is returned upon violation. In our model, a sanction s is a monetary fine, denoted in natural numbers. Hence, in our model, the set of all possible sanctions S corresponds of the set of the natural numbers $\mathbb{N}$.

Finally, a norm instance has a function used to check if the deadline of the norm instance is met. It takes a set of vehicle objects as input and returns `true` if the deadline condition is met and `false` otherwise. This function also receives multiple vehicle objects, so that a deadline can be determined in a flexible way. For example, a speed directive might normally be removed after a certain point on the road, but may be kept to prevent congestion if the speed of vehicles further down the road is low.

All functions belonging to a norm instance can read the vehicle id property o_{id} of the norm instance.

Definition 3.4 (Norm Instance)**.** A norm instance ni is defined as a tuple

$$\langle o_{id}, c, g, t, s, d \rangle,$$

where $o_{id} \in \mathbb{N}$ is the associated vehicle, $c : 2^{O_{veh}} \mapsto \mathbb{B}$ is the compliance function, $g \in O_{veh}$ is the directive, $t \in \{O, P\}$ is the type of directive, $s \in S$ is the sanction and $d : 2^{O_{veh}} \mapsto \mathbb{B}$ is the deadline function. ■

The set of all norm instances is denoted I. A norm scheme is used to generate the norm instances from Definition 3.4. It is a function that takes a set of vehicle states and returns a set of norm instances.

Definition 3.5 (Norm Scheme)**.** A norm scheme ns is defined as a function of the form $ns : 2^{O_{veh}} \mapsto 2^I$. ■

The set of all norm schemes is denoted NS. A controller uses vehicle states and norm schemes to generate norm instances and to check norm compliance. It receives these vehicle states using information sources. In our model, the two information sources used are the road sensors associated to the controller and other controllers that the controller is subscribed to. We denote the set of all information sources IS, with $IS = C \cup O_{sens}$. We used a subscription mechanism because i) it is a straightforward way of sharing information between specific controllers, without having to share information with all controllers (such as with broadcasting) and ii) it is easy to implement. The formal definition of a controller is as follows:

Definition 3.6 (Controller)**.** Let C be the set of all controllers. A controller $c \in C$ is specified by the following tuple

$$c = \langle C_{sub}, N_s, N_i, Sens, B \rangle,$$

where $C_{sub} \subseteq C$ is the set of controllers c is subscribed to; $N_s \subseteq NS$ is the set of norm schemes; $N_i \subseteq I$ is the set of norms currently instantiated; $Sens \subseteq O_{sens}$ is the set of all road sensors connected to c; and $B \subseteq O_{veh}$ is the set of all vehicle states known to c. ■

As we shall see in Chapter 4, the controller is continuously performing its monitor and control cycle. The cycle of a controller is depicted in Algorithm 3.1. First, a controller receives updates from its information sources and saves them in its knowledge base (line 2 - 5). It uses the $receive : IS \mapsto 2^{O_{veh}}$ function, which takes an external information source and returns a set of vehicle states retrieved from this source.

Next, the controller checks it the vehicles comply to their norm instances and if the deadline of a norm instance is met. If the deadline is met, the norm instance is removed (line 6 - 13). The sanctions and removed instances are sent to the corresponding vehicles using the *sendRemoved* and *sendSanctions* functions (line 14 - 15). Finally, new norm instances are generated using the norm schemes. Each norm scheme can generate a set of norm instances. The generated instances are sent to the corresponding vehicles using the *sendInstances* function (line 16 - 18).

One thing to note is that a norm instance is only removed when its deadline is met. Therefore, a vehicle can be fined multiple timed because

of one instance. This corresponds to the current traffic situation, where for example a vehicle who keeps driving above the speed limit can receive multiple tickets.

Algorithm 3.1 Pseudo code for a controller cycle

1: $c = \langle C_{sub}, N_s, N_i, Sens, B\rangle$ is a controller
2: **for each** $s \in c.Sens$ **do**
3: $\quad c.B \leftarrow c.B \cup receive(s)$
4: **for each** $c_{sub} \in c.C_{sub}$ **do**
5: $\quad c.B \leftarrow c.B \cup receive(c_{sub})$
6: $Sanc_{new} \leftarrow \emptyset$
7: $NI_{rem} \leftarrow \emptyset$
8: **for each** $ni \in c.N_i$ **do**
9: $\quad$ **if** $\neg ni.c(B)$ **then**
10: $\quad\quad Sanc_{new} \leftarrow Sanc_{new} \cup \{ni.s\}$
11: $\quad$ **if** $ni.d(B)$ **then**
12: $\quad\quad c.N_i \leftarrow c.N_i \backslash \{ni\}$
13: $\quad\quad NI_{rem} \leftarrow NI_{rem} \cup \{ni\}$
14: $sendSanctions(Sanc_{new})$
15: $sendRemoved(NI_{rem})$
16: **for each** $ns \in c.N_s$ **do**
17: $\quad c.N_i \leftarrow c.N_i \cup ns(B)$
18: $\quad sendInstances(ns(c.B))$

3.2.1 Example Norm

It might be enlightening for the reader to see an example of a norm at this point. Consider the following traffic regulation:

> As long as an vehicle drives on road A, it should drive 22.2 m/s.
> Else, it receives a fine of 200 euros.

The following norm scheme and norm instance formalize this traffic regulation. In Algorithm 3.2 the norm scheme function written in pseudocode. In Algorithm 3.3 and 3.4, the c and d function of the norm instance are written in pseudocode.

Algorithm 3.2 Pseudo code for a norm scheme

```
1: function NS(B)
2:     I ← ∅
3:     for each o ∈ B do
4:         if (o.loc = (A, _, _)) then
5:             desiredVehState ← (22.2, _, _)
6:             I ← I ∪ (o.id, c, desiredVehState, O, 200, d)
7:     return I
```

The norm scheme works as follows. It cycles through all observations (line 3), and creates a norm instance for every agent that is on road A (line 4-6). Note that the directive type is an obligation. Finally, it returns the set of created norm instances (line 7).

Algorithm 3.3 Pseudo code for checking compliance

```
1: function C(B)
2:     for each o′ ∈ B do
3:         if (o′.id ≡ o_id) & o′.loc = (A, _, _) & (o′.v ≡ 22.2) then
4:             return true
5:         else
6:             return false
```

The compliance function checks if the vehicle occurs in the observed data, if it drives on road A and if it does drives 80km/h (line 3). If this is the case, the function returns `true`. Else, it returns `false`.

Algorithm 3.4 Pseudo code for checking the deadline

```
1: function D(B)
2:     for each o′ ∈ B do
3:         if (o′.id ≡ o_id) & (o′.loc ≠ (A, _, _)) then
4:             return true
5:         else
6:             return false
```

The deadline function returns `true` if the vehicle occurs in the observed data and the vehicle does not drive on road A, and returns `false` otherwise.

3.3 Agents

The concept of autonomous vehicles has a large overlap with autonomous agents as treated in Section 2.1.1. In this section, we will define autonomous vehicles in terms of agents. An overview of all the functions defined and used in this section can be found in Table 3.2.

An agent is situated in the environment as an autonomous vehicle. It can sense its environment and exert influence on this environment using actions. The agent has a certain goal it wants to achieve, namely to arrive on time at a certain destination. The destination is modeled by an exit node in the road network. Agents are able to receive traffic regulations, understand these regulations and decide whether or not to obey them. These deliberations are steered by their agent profile. The agent profile, as explained in more detail in Section 3.3.1, models the differences between various types of autonomous vehicles, such as their dislike for certain sanctions and the intensity of their to be on time. The formal agent structure is defined as the following:

Table 3.2: Overview of the functions used in Section 3.3

Function	Description	Function type
g_s	sanction grading function	$S \to \mathbb{R}$
g_t	arrival time grading function	$\mathbb{N} \to \mathbb{R}$
e	action effect function	$O_{veh} \times A \mapsto O_{veh}$
f	expected arrival time function	$O_{veh} \mapsto \mathbb{N}$
δ	directive distance measure	$O_{veh} \times I \mapsto \mathbb{R}$
u	action utility function	$O_{veh} \times P \times A \times 2^I$
α	action selection function	$O_{veh} \times P \times 2^I \to A$

Definition 3.7 (Agent)**.** An *agent* is specified by the following tuple.

$$A = \langle veh, goal, prof, N_i \rangle.$$

Here, $veh \in O_{veh}$ is a physical vehicle situated in the road network, $goal \in N_{out} \times \mathbb{N}$ is the goal of the agent consisting of a destination (an exit node) and a desired arrival time, $prof \in P$ is the profile of the agent as defined in Definition 3.8 and $N_i \subseteq I$ is a set containing all norms instances known to the agents. ■

In the next sections, we will explain and define the concepts of an *agent profile*. We will also describe what actions agents can take to move around

in the environment, how they deliberate about which action to take and how they take norms into account with these deliberations.

3.3.1 Profile

We expect autonomous vehicles in future traffic to be able to disregard traffic regulations. An autonomous vehicle might disregard a regulation if it thinks it is better off by not obeying. The reasoning about regulations and sanctions can differ between autonomous vehicles. An agent may dislike a certain sanction more than another agent, just as one agent might care more about arriving on time than another agent might.

We model these notions in our model using an *agent profile*. The agent profile consists of a sanction grading function g_s and an arrival time grading function g_t. These functions model the tension between the desire of an agent to evade sanctions and the desire of an agent to arrive on time at their destination.

The function g_s is a function that specifies how unsatisfied an agent will be with a sanction. Since monetary fines are used as sanctions, g_s can for instance model the wealth difference between agents. A wealthy agent will be less unsatisfied with a fine than an impoverished agent. In order to model the dissatisfaction with sanctions, in our model the output of g_s is always negative. That is, an agent is never content with a sanction.

The function g_t is a strictly monotone function that quantifies the satisfaction of an agent of arriving at its destination at a certain time. The value of g_t will be maximal if the agent arrives at its destination on its goal time. Differences between instances of g_t model the different levels of impatience between autonomous vehicles.

Definition 3.8 (Agent Profile)**.** An agent's personal profile $prof$ is specified by:

$$prof = \langle g_s, g_t \rangle$$

here, $g_s : S \rightarrow \mathbb{R}_{\leq 0}$ is the sanction grading function that returns how important it is to avoid a given sanction from S and $g_t : \mathbb{N} \rightarrow \mathbb{R}$ is the arrival time grading function that determines the importance for an agent to arrive at its destination on time. With P we denote the set of all possible personal profiles. ■

3.3.2 Actions

We want the implementation of our model to be able to simulate traffic scenarios with large numbers of autonomous agents. Therefore, we wanted to keep the agents' model simple, so that the deliberation cycle could be executed quickly. Hence, agents have a limited range of actions to deliberate about and their deliberations are kept simple.

In order to show realistic vehicle movement while following a certain route, agents are able to execute four types of actions: change their velocity, change their lane, change the route which they are currently following or change nothing at all. Firstly, agents can change their velocity each time step, either increasing or decreasing it by a set amount, these amounts are specified in our model. However, these changes in velocity are of course limited by the acceleration and deceleration values of the vehicle itself.

Secondly, an agent can decide to change its current lane, going either to the lane right or left of him if there are more lanes on that side. A lane change action is typically taken to take over leading vehicle with a lower average velocity or to merge into another lane in preparation to take a turn onto another road.

Thirdly, the agents are able to change the current route they are following. If an agent happens to receive information about certain roads ahead which are on his route (by observing a variable message sign for example), this new information can cause the agent to deliberate about his situation and calculate a new possible route. This action requires some sort of path-planning algorithm and enough knowledge about the road network.

Finally, agents can decide to change nothing. If they decide this, they simply continue on their current route with their current speed on their current lane.

Agents in this system are homogeneous with respect to the actions they can execute. The formal definition of the set of actions A available to every agent is the following:

Definition 3.9 (Action space)**.**

$$A = \{a_x | x \in \{0.1, -0.1, 1, -1, 5, -5, 10, -10, 20, -20\}\} \cup \{l_{left}, l_{right}, change_{route}, \epsilon\},$$

where a_x stands for an increase/decrease in velocity, l_x stands for changing lane to the left/right and ϵ denotes the action of changing nothing. ■

An agent is able to estimate the next state it will be in after executing a certain action. This prediction is needed to see if any norm instances

will be violated. Agents can reason about the future with the local *action effect* function $e : O_{veh} \times A \mapsto O_{veh}$ which returns the next expected vehicle state, given a vehicle state and an action. For instance: the vehicle's current velocity is 20 m/s and it accelerates by 5 m/s as an action. It would then expect its velocity for the next simulation tick to be at 25 m/s, if this is possible within its acceleration capabilities. It was sufficient for the purposes of our thesis to use the same action effect function for all agents. The function e is used when calculating the utility of a certain action.

Each time a prediction over the next state is made, the agent will also calculate a new expected arrival time. For this the function $f : O_{veh} \mapsto \mathbb{N}$ is used. This function returns an expected arrival time given a vehicle state. This function reflects for instance the GPS planning tools that vehicles have available. The function f is uniform for all agents, but can be parameterized in the future in order to make more optimistic or pessimistic agents. The utility function itself is explained in more detail in the next section.

3.3.3 Directive distance measure

In our model, agents plan only action in advance to keep the deliberation cycle fast. However, some norm instances cannot be met in one step, such as the directive to change multiple lanes. Therefore, an agent needs to be able to reason about whether an action will bring it closer or further away from fulfilling a norm instance. To this end agents use a directive distance measure $\delta : O_{veh} \times I \mapsto \mathbb{R}$ which returns, given a vehicle state and a norm instance, a real number denoting the expectancy of the agent that it can comply with the norm instance in $t_{exit} - t$ steps. A lower number corresponds with a higher expectancy. If the agent needs zero steps to adhere to the norm, the output of δ is zero. Otherwise the distance proportionally moves to 1 given the current time. If the control system will check obligation fulfillment before the agent can achieve compliance (i.e. $t_{exit} - t < \delta_t$), then δ should return 1.

In our model, autonomous vehicles expect that the control system will check whether a directive is fulfilled somewhere between the current time t and the current expected exit time t_{exit} for the vehicle. The minimal amount of steps needed to adhere to the norm is denoted as δ_t. For instance, suppose that a vehicle at time step t receives the directive to drive 25 m/s, and that it can accelerate to this speed in minimally 3 time units. In that case, $\delta_t = 3$.

Definition 3.10 (Directive distance measure)**.** The directive distance measure δ is defined as

$$\delta(\langle o, i\rangle) = \frac{\texttt{min}(\delta_t, t_{exit} - t)}{(t_{exit} - t)},$$

where o is the vehicle state, i is a norm instance, t is the current time, t_{exit} is the expected exit time and δ_t is the minimal number of steps needed for compliance. ■

3.3.4 Action Selection

Agents select actions using an action selection function. This function, denoted α, returns an action given a vehicle state, an agent profile and a set of norm instances. An action is selected based on its utility. The action utility is determined by the action utility function u, as defined in Definition 3.12. The action selection function returns the actions which has the highest action utility.

A tie-break mechanism is used when two or more actions have the highest utility. If such a situation occurs, then the action with the highest priority is chosen. The less an action changes the agent state, the higher its priority is. For example, in a tie-break situation, doing nothing is preferred to increasing velocity to a small amount, which in turn is preferred to increasing velocity to a larger amount, which in turn is preferred to changing lane. We chose for this 'least impactful action' tie-break ordering since we believe this to be intuitive behavior. However, this ordering is not essential to our framework and can be replaced by arbitrary tie-break orderings.

Definition 3.11. The action selection function $\alpha : O_{veh} \times P \times 2^I \rightarrow A$ is defined as

$$\alpha(o, prof, ni) = \texttt{argsmax}_{a \in A}(u(o, prof, ni, a)),$$

where $o \in O_{veh}$ is a vehicle state, $prof \in P$ is an agent profile, $ni \subseteq I$ is a set of norm instances and u is the action utility function. ■

The action utility function is used to calculate the utility of taking an action. The function takes as input an agent profile as defined in Definition 3.8, a set of norm instances and an action. It returns a real value denoting the utility of taking that action with the agent profile and the set of norm instances. The utility function returns the sum of two parts. The first part is the utility from the expected arrival time based on the new vehicle state. The second part consists of the sum of the multiplication of the directive

distance and the sanction severity for each norm instance. Note that the second part is always zero or negative, since δ has a positive codomain and g_s has a negative codomain. Hence, the utility for noncompliance is considered to be negative.

Definition 3.12. The action utility function $u : O_{veh} \times P \times 2^I \times A \rightarrow \mathbb{R}$ is defined as:

$$u(o, \langle g_s, g_t \rangle, ni, a) = g_t(f(o')) + \sum_{(ni \in n)} (\delta(o', ni.g) \cdot g_s(ni.s)),$$

where $o \in O_{veh}$ is a vehicle state, $prof \in P$ is an agent profile, $ni \subseteq I$ is a set of norm instances, $a \in A$ is an action and $o' = e(o, a)$ ■

The action utility function models the tension between being on time and receiving no sanctions. It is able to model situations in which vehicles knowingly ignore directives, because either the profit for being on time is very high, or the dissatisfaction for the fine for disobedience of a directive is very low. It is also able to model the opposite, a situation in which a vehicle knowingly aims for a undesirable arrival time because it is not willing to receive the fines associated with disobeying the directives. Hence, the action utility function can capture the deliberations of the autonomous vehicles we expect to see on the road in the future. We therefore feel that it is a plausible utility function to use in our model.

3.3.5 Deliberation Example

The reader might also benefit to see the agent model in action. Suppose we have two agents, a poor impatient agent, and an affluent impatient agent. Their current speed is 20 m/s, their maximum speed is 30m/s, they can accelerate/decelerate with 10m/s and the distance to their destination is 1080 meters. Since both are in a hurry, their arrival time grading function g_t is defined as

$$g_t(time) = \frac{bestTime}{time}.$$

where $bestTime$ is defined as the minimal travel time in seconds, i.e. the time it would take the agents to travel the distance if they could go their maximum speed all the time. In this case,

$$bestTime = \frac{1080m}{30m/s} = 36s.$$

However, the road the agents travel on has a speed norm, with the maximum speed being 10 m/s. Not obliging to this norm gives a fine of 200 euro's, denoted $fine_{200}$. The poor agent cannot afford this fine, so its sanction grading function g_s with respect to a fine of 200 euros is defined as

$$g_s(200) = -20.$$

The affluent agent can easily afford this fine, so its sanction grading function g_s is defined as

$$g_s(200) = -2.$$

In this example, the action space of the agents is restricted to $A = \{\epsilon, a_{10}, a_{-10}\}$. In Table 3.3, we see the utilities for each of these actions and both agents. The highest rewarded action for the poor agent is to indeed oblige to the norm by decreasing its speed, since it the negative utility of getting the sanction is too high. The affluent agent is in a position to not decrease its speed to oblige to the norm, since it can afford the fine. In fact, it will even increase its speed, since it then maximizes its utility. Hence, the poor agent will select the action a_{-10} and the affluent agent will select the action a_{10}.

Table 3.3: The utilities of a poor and a rich agent with various actions

	Change nothing	Accelerate 10m/s	Decelerate 10m/s
New velocity	20m/s	30m/s	10m/s
Norm speed	10m/s	10m/s	10m/s
Remaining travel time	54s	36s	108s
g_t	0.67	1.00	0.33
δ_t	1	2	0
δ	$\frac{1}{54} = 0.019$	$\frac{2}{0.33} = 0.055$	0
Sanction utility poor agent	$0.019 \cdot -20 = -0.38$	$0.055 \cdot -20 = -1.10$	0.00
Sanction utility rich agent	$0.019 \cdot -2 = -0.04$	$0.055 \cdot -2 = -0.11$	0.00
Utility poor agent	0.67 - 0.37 = 0.29	1.00 - 1.10 = -0.10	0.33
Utility rich agent	0.67 - 0.04 = 0.63	1.00 - 0.11 = 0.89	0.33

3.4 Chapter Summary

In this chapter, we presented a model for future traffic that is based on the concepts of multi-agent systems. We first defined the environment and the objects residing in it as the road network and vehicles and sensors respectively. Next, we modeled traffic regulations and traffic controllers as norms and controllers from the multi-agent system paradigm. Finally, we presented our definition of an autonomous vehicle as an intelligent agent. Different kind of agents can be created using the agent profile, which contains an arrival time grading function and a sanction grading function.

Chapter 4

Implementation

In Chapter 3, we have given a formal specification of a multi-agent system for traffic. In this chapter we describe how we implemented this specification and what choices we made during implementation.

The goal of our application was to simulate traffic as a multi-agent system, using a heterogeneous, norm-aware agent population, as well as norm-based controllers which monitor and influence the agent behavior. We build this normative MAS application as a separate module on top of an existing simulator as an extension. The motivation for our choice of traffic simulator is expanded on in Section 4.1.

While developing, we kept two requirements in mind. First of all, we wanted the implementation to be extensible. That is, we want to allow for easy addition or modification of data sources, simulation packages and classes without having to alter the rest of the program. Furthermore, we wanted our program to be efficient, since some use cases for our framework include a large number of agents.

The framework was developed using the Java language. We chose Java as the implementation language since i) it is the language we have the most experience with, ii) it is a cross-platform language and iii) Java is used often in implementing agent systems[1].

The remainder of this chapter describes how our extension was implemented. We will start by explaining why we chose an existing traffic simulator instead of developing our own, and will motivate our choice for this specific simulator. Following that, we will expand on the model-view-controller

[1] See http://en.wikipedia.org/wiki/Comparison_of_agent-based_modeling_software for a comparison of various agent software implementations.

design pattern used for the global control structure of the program in Section 4.2. Finally in Section 4.3 the framework structure and its relation to the formal framework as explained in the previous chapter is discussed.

4.1 SUMO

In this section we will expand on the traffic simulator compatible with our extension. We chose the **SUMO** (**S**imulation of **U**rban **MO**bility)[23] simulation package. SUMO is an open-source, microscopic, state-of-the-art traffic simulator and allows for complex simulations. It features the modeling of different traffic modalities, such as pedestrians, cars and public transport. Since this is a microscopic simulator, all vehicles are modeled individually, as opposed to modeling traffic as multiple streams.

We chose SUMO as simulation package for the following reasons. First of all, it is very fast. It is possible to simulate thousands of vehicles concurrently in various scenarios. This feature is important since we required our application to be efficient. Secondly, it is a robust and mature traffic simulator. It has been in development for over fourteen years and has become very sophisticated over the years. Thirdly, it has a very active development community. The team behind SUMO is quick in answering questions and giving support. Finally, it is very easy to manually manipulate traffic and individual cars in SUMO using **TraCI** (**Tra**ffic **C**ontrol **I**nterface).

TraCI is an extension to SUMO which allows external programs to influence the simulation via the TCP/IP protocol. For example, one can instruct SUMO to increase the speed of a certain car at runtime. This way, we can model cars as agents in a straightforward fashion and thus easily implement our MAS as a separate module to this simulator. It is also worth mentioning that we have improved the TraCI mechanism during the implementation of our framework, yielding an average speedup of 4.8 times compared to the unmodified version.

This framework was implemented as an extension to SUMO, rather than modifying the source code of SUMO directly. There are several reasons for this. First of all, vehicles in SUMO are only goal-driven in a limited way. Their goal is to follow a certain route, as opposed to having a specific location as their destination. Second, SUMO agents are preprogrammed to follow a specific route. They only respond reactively to their environment, instead of deliberating on what action would best suit them. Finally, vehicles in SUMO are only norm-aware in certain situations. For example, they stop

for a red light and they obey to the right of way. This is because vehicles in SUMO move according to specific car-following and lane-changing models and this car-following model is not created with norm-aware agents in mind. The most commonly used car-following model made by Stefan Krauß is designed purely to create realistic traffic flows on a macroscopic level, since in most traffic simulations individual movement on the microscopic level is not interesting [25]. In contrast, we aim at designing cars with a more fine-grained sense of control and most importantly, the ability to violate norms.

There are also pragmatic reasons to propose a SUMO extension rather than alter its code. For example, we can now support multiple versions of SUMO, starting from SUMO 0.20 and upwards. This also makes our extension more accessible to users, since it eliminates the need for users to recompile SUMO before they can use the framework. Furthermore, by building on top of SUMO, our system is only loosely coupled to it. This makes it easier to switch to another traffic simulator if one so desires.

4.2 Global control structure

One of the requirements was to make the program easy to extend. We did this by using the Model-View-Controller (MVC) design pattern. A design pattern is a solution for a recurring problem in software, in this case the problem of decoupling various parts of a software solution. This design pattern fits our application well, since the MAS part is decoupled from the traffic simulator. The model-view-controller pattern splits responsibilities of the program into three parts, allowing for separation of concerns.

The model part abstracts away from the representation of domain-specific knowledge. Different model instantiations might represent the knowledge in different ways, but can be used in similar fashion. The view part abstracts away from how the user is informed and can influence the rest of the program. Finally, the controller part abstracts away over how the events raised by the user, the view and the model are processed.

In our implementation, we employed a separate data model, simulation model, view and a controller module. Figure 4.1 depicts how the different parts are instantiated. The data model parses the initial values needed for the multi-agent part of the simulation. See Table 4.1 for an overview of the files that are parsed by the data model. Our implementation for the MAS model is `XML` based. That is, it parses a set of `XML`-files to instantiate the

variables needed to run the program. The `XML` implementation was chosen because of its human readability and ease of parsing.

This simulation model is used to define how our framework can use the underlying simulation package SUMO to control cars and coordinate traffic. Thus, our framework is used rather as a layer on top of an existing traffic simulation than a traffic simulation in its own right. We chose to decouple our agent and controller framework from the actual traffic framework for several reasons. First of all, this decoupling allowed us to use an existing traffic simulation package for the actual traffic simulation. We could thus immediately get started with the agent and controller part, without first developing a traffic simulator ourselves. Secondly, the decoupling, as mentioned earlier, allows for quick swapping of traffic simulators. For the usage of a different traffic simulator, only the simulation model should be changed. And finally, by keeping the multi-agent system logic different from the simulation logic, it should be relatively easy to switch from the traffic domain to a different domain without rewriting agent and traffic controller code completely.

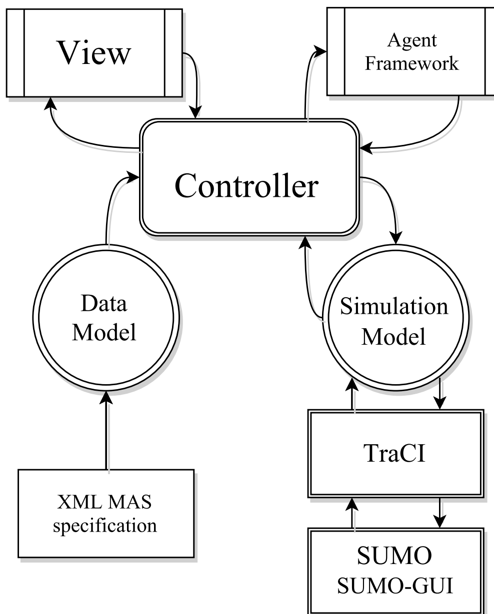

Figure 4.1: The Model-View-Controller structure.

The controller is the part of the framework which binds together the models and the view. The controller retrieves information from the models, and uses that to generate framework responses, such as agent actions. These responses are returned to the simulation model, which uses this information to calculate the next state. Each state is sent to the view part of the software, which visualizes the current state of affairs. In Figure 4.1, the program flow for the controller is depicted.

The view module is employed to show information to the user and let the user interact with the simulation. For our purposes, a simple text-based, non interactive interface was sufficient. However, it is possible to add

interactivity and graphical visualisations, but these features are considered future work.

File	Description
`scenario.net.xml`	Specification of the roads, their connections and the lanes of the roads. Native SUMO file.
`scenario.rou.xml`	Specification of the routes driven by agents. Native SUMO file.
`scenario.prof.xml`	Specification of the different agent types and distribution of agents used in the scenario.
`scenario.org.xml`	Specification of the controllers, along with the sensors and norms associated with the controllers.
`scenario.sens.xml`	Specification of the sensors placed on the lanes. Native SUMO file
`scenario.norms.xml`	Specification of the different norm schemes.

Table 4.1: Files read by the MAS Model

4.3 Framework Structure

In previous sections, we discussed the general outline of our framework and what simulation package we used. In this section the implementation of the model presented in the previous chapter is described in more detail. We start by providing an overview of control flow. Following that, we expand on the different Java classes we implemented. The Java classes are structured in a similar way to the formal model. First of all, there is the road network package, which contains all the code relating to sensors, roads, lanes and their connections. Secondly, there is the agent package. This package consists of the template Agent class, some derived implementations from this class and some auxiliary classes. Thirdly, we have the norm package. In this package, the prototype norm scheme and instances are implemented, as well as some more complex norms. Finally, we discuss the controller package. This package contains the controller class as well as the classes needed for communication between controllers. An UML overview of all packages is provided in Figure 4.2.

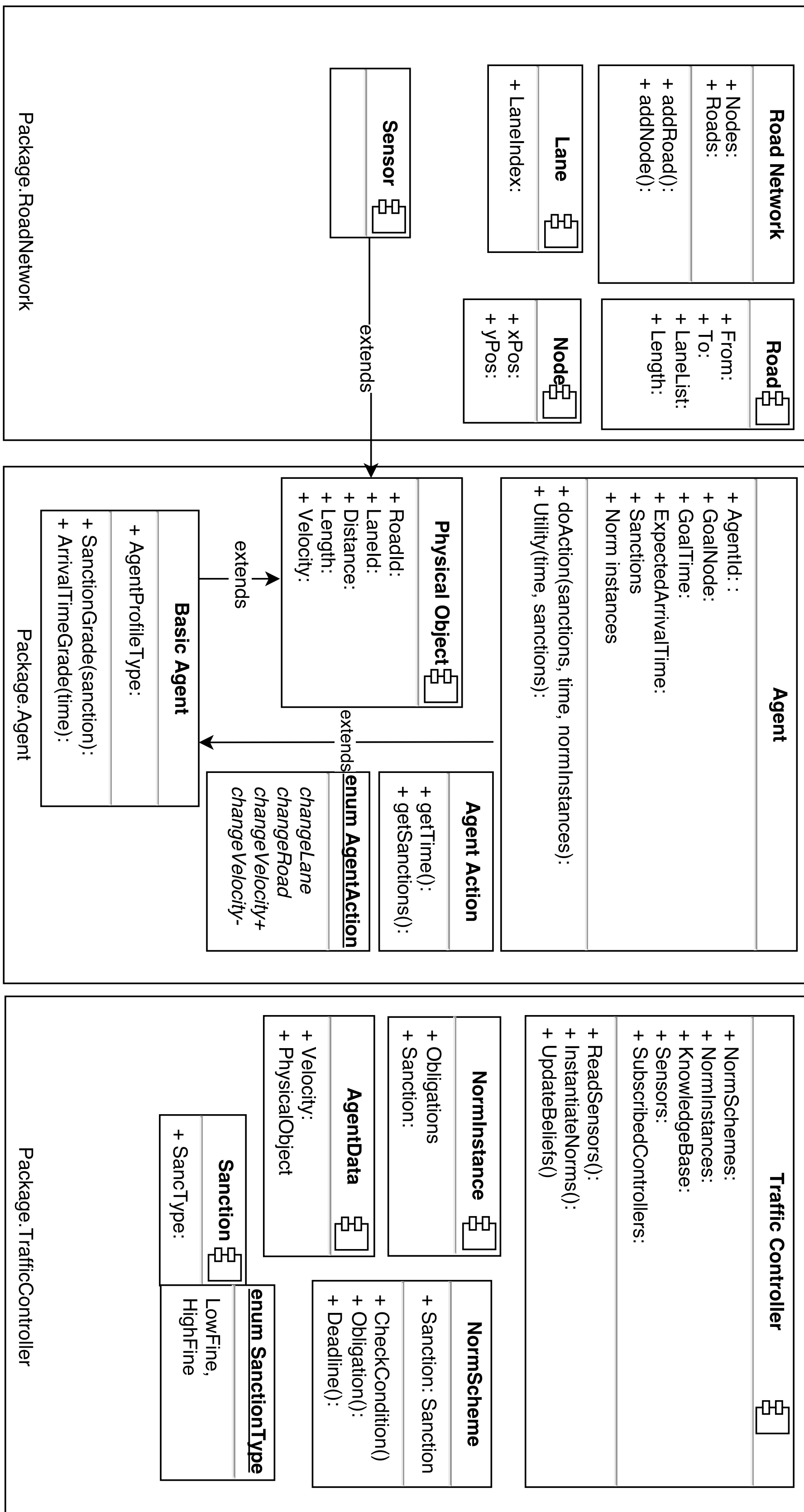

Figure 4.2: An UML overview of our framework

4.3.1 Flow of control

The sequence diagram depicted in Figure 4.3 on page 40 depicts the flow of control of our extension. The program starts by initializing the controller `ctrl`, the controller then acts as a central hub which regulates all information streams between the different modules. First the Datamodel `dataModel` is requested to parse all XML files corresponding to the scenario with the function `getMASData()`. In these files the configuration of the environment, controllers and agents is specified.

After all the initial data is read from the XML files, the environment is set up. All roads, junctions and sensors are initialized in our extension, using the same XML input files SUMO will use to set up its road network. This is executed by the function `setupEnvironment()` which returns a RoadNetwork structure containing all roads, junctions, routes and sensors. Next, the agents can be initialized with the function `instantiateAgents()`. Using the data parsed in the XML file, together with a seed specified as a command-line argument and the road network that was just generated, a list of agents and their respective spawn times is created and then passed back to the controller module. The next step is setting up the traffic control systems with `setupTCs()`, coupling them to specific norm schemes and sensors on the road. Then it is time to start the SUMO traffic controller with the `setupSimulation()` function. Together with several of the initial XML files, our extension provides information about the agents and the spawning times of those agents. Finally the view module is set up with `setupView()`. Since we have not implemented a GUI yet, in our case this module keeps track of statistics about the simulation and outputs those statistics on the console.

After the setup of both our extension and SUMO is completed, the main loop is started. This loop will run until the simulation length specified in one of the initial XML files has been reached. The first function called with every iteration of this loop is `nextSimState()`, this function requests SUMO to execute a single time step and subsequently returns the most recent data about the road network and the vehicles in the simulation. Next is the update cycle of the MAS part of the extension, it is quite similar to setting up and there is thus no need to elaborate. Then the controller calls the `nextMASState()` function, this function uses the updated environment, agent and traffic controller data in order to calculate new actions for the agents to execute. This function returns a map containing all agents with their respective actions to the controller.

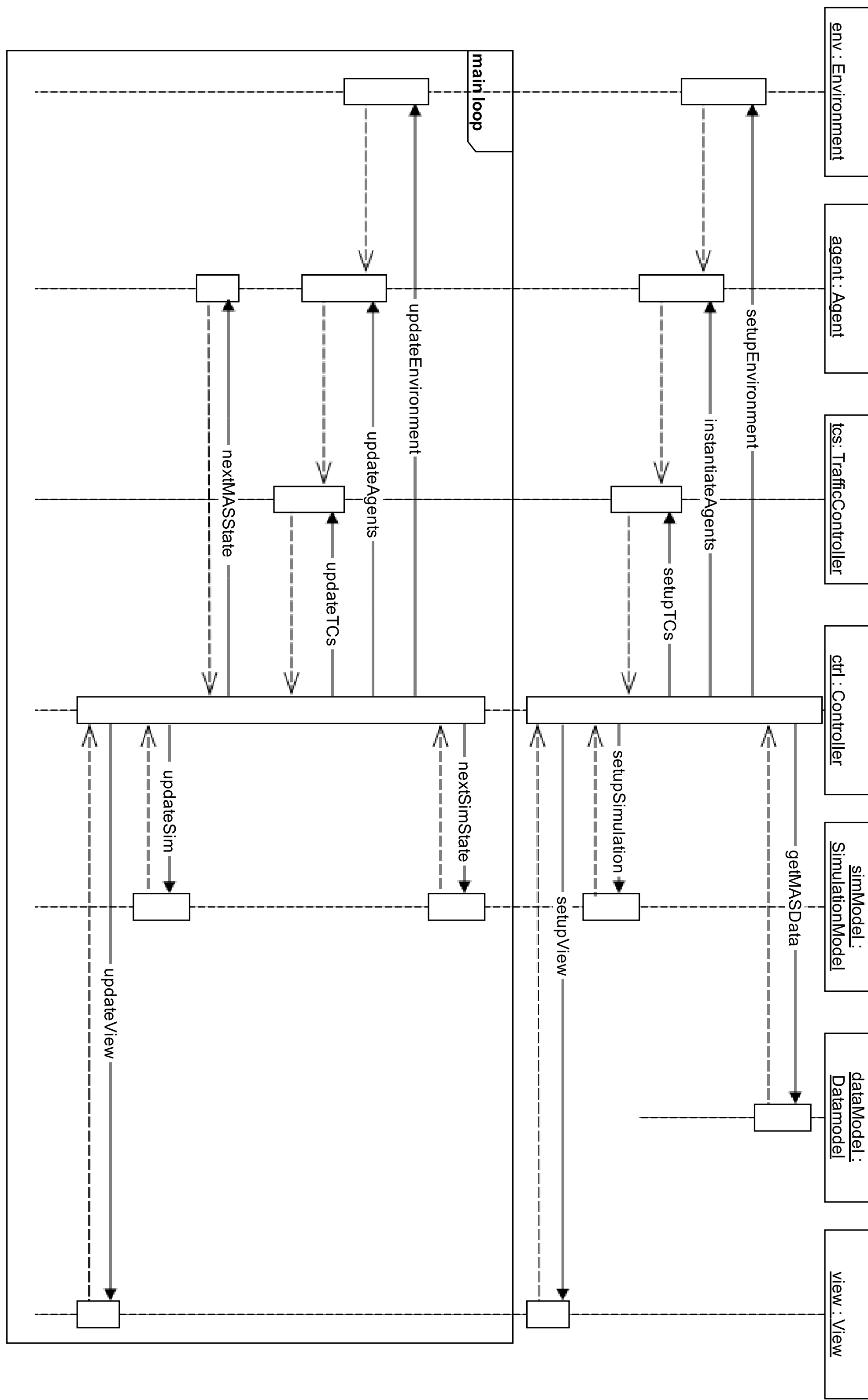

Figure 4.3: UML sequence diagram of the framework control flow.

The next step is to return this information to SUMO, in preparation for its next time step. Finally the view is updated with the new data, which is then again outputted to the console.

4.3.2 Road Network

The road network package, depicted by Figure 4.4, defines the environment. It contains the Node class, which corresponds to cities, junctions and/or destinations. Since we use the same network XML file as SUMO does, a node object simply consists of an `X` and `Y` coordinate. The Road class is defined exactly as specified in Definition 3.2, containing a *from* and *to* node, since all roads are directed. Furthermore, each Road object contains a list of lanes, and a length, which simply is the calculated length between two nodes.

The `Lane` class has a lane index, denoting which part of the road it represents. Finally, the road network contains the `Sensor` class. The `Sensor` class is derived from the `PhysicalObject` class, thus, it has a certain location, that is a position on a lane belonging to a certain edge. A sensor reports every vehicle that travels over it to an associated controller.

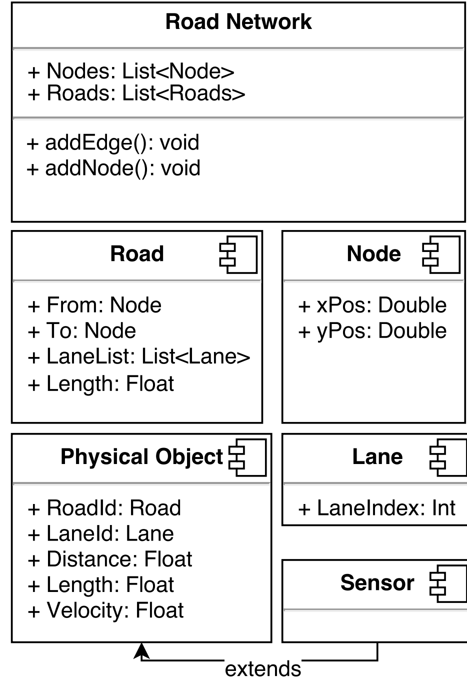

Figure 4.4: UML overview of the road network class.

4.3.3 Norms

In the norm package, depicted by Figure 4.5 the norm scheme and instance are implemented. These are two distinctive but equally important norm artifacts. The first one is the norm scheme. The norm scheme contains

several functions and a sanction to be imposed on the agent if it fails to comply. It is important to note that we only implemented the obligation part of norms, since that was sufficient for our purposes.

The first function is `CheckCondition()`, which returns a boolean and uses as input the most recent `AgentData` as acquired by the controller to check if this norm should currently be effectuated. Secondly, `Obligation()`, which receives data about a certain agent and returns an obligation for the agent. An obligation consists of a `AgentData` objects. For an agent to comply with such an obligation, its velocity and position must be equal to the values specified by the obligation. It is possible to specify the obligations only partially, e.g. only specify the lane an agent has to drive on, but not its speed. The agent is then free to choose its speed, but has to drive on the specified lane to evade the sanction. Finally, the norm scheme contains the deadline, which specifies when becomes inactive.

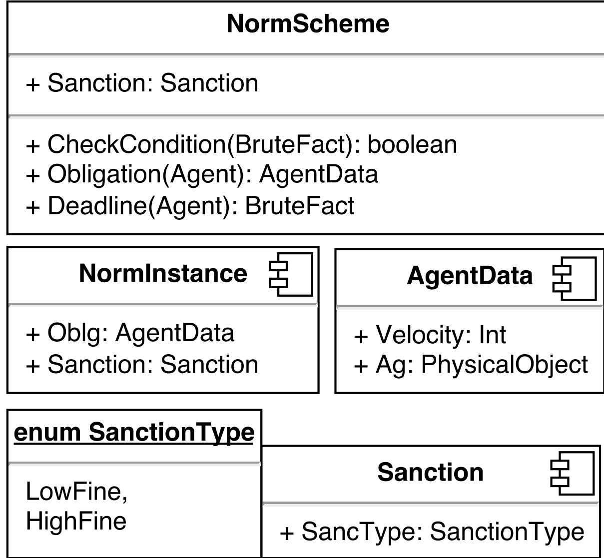

Figure 4.5: UML overview of the norm class.

The deadline is also implemented as a function `Deadline()`, which similarly receives as input an agent and returns a brute fact. If this brute fact will evaluate to be true, the deadline will be met. Finally, the norm scheme contains the sanction to be imposed on the agent if it does not comply with an obligation. A sanction can currently be one of two types, $SanctionType = \{LowFine, HighFine\}$. Note that we kept the monetary amount of the fines abstract. New sanctions that specify a monetary amount can easily be created, but this limited collection is sufficient for the scope this thesis.

Norm instances are norm schemes with agent-specific information attached. It is possible to assign each norm instance of a certain norm scheme with a unique velocity. Thus, each agent will be obliged to drive at this specific velocity and the system can influence individual agents in distinct

ways. The norm instance class has two attributes, the first is the obligation as described above. Secondly, the norm instance contains a sanction, this means that the agent is aware of which sanction will be imposed on him when he fails to comply to either obligation.

4.3.4 Norm-based controllers

Similar to the rest of the classes, the controllers were kept close to the formal definitions as provided in Chapter 3. The controllers are structured in the following way. The controller class contains firstly, a list of subscribed controllers, `SubcribedControllers`. These controllers receive all information about vehicles observed by the function `readSensors()` at every tick. Secondly, a list of norm schemes that will be effectuated by said controller during runtime. These norm schemes are listed in the `org.xml` file which we use to initialize the program. Thirdly, a list of norm instances. These norm instances are generated by the controller and sent to the coupled agents in the function `instantiateNorms()`. Fourthly, every controller keeps track of the most recent brute facts. As we mentioned above, these facts are either read by sensors, which are physical objects, or are received from controllers the controller is subscribed to. The knowledge base is updated every tick with the `updateBeliefs()` function. Finally, the controller class contains a list of sensors associated with the controller. These sensors are part of the Road Network class described in Section 4.3.2.

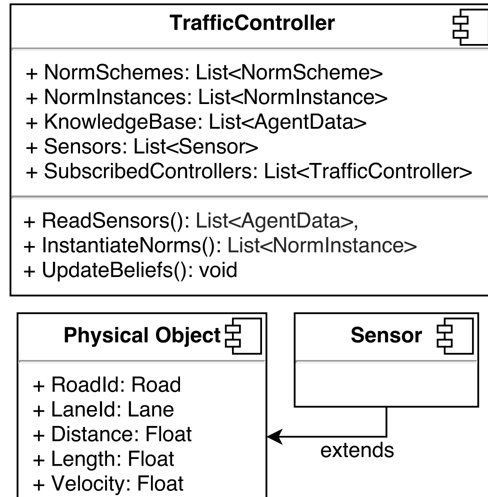

Figure 4.6: UML overview of the controller class.

4.3.5 Agents

The structure of the agent package is depicted in Figure 4.7. It is structured as follows. There is a base class Agent, which contains the utility and the `doAction` function as defined in Definition 3.12. The time grading function and sanction grading function are implemented in agent subclasses. For example, an affluent agent might have a sanction grading function that values monetary sanctions lower than other sanctions, while a poor agent might value these monetary sanctions higher. Different agent types can be defined by selecting a certain `AgentProfile`, which all extend from the `BasicAgent` class. The actions an agent can execute at any time are as specified in Definition 3.9. Each action class also implements the action effect function e and the expected arrival time function f as defined in Chapter 3. In the implementation, these functions are called `getState` and `getTime` respectively. Furthermore, each action also has a function `isRelevant`, which specifies in which states an action is relevant. For example, the action l_{left} is not relevant when the road the vehicle traverses on consist of only one lane. The usage of the `isRelevant` function speeds up the program, since it reduces the number of actions that are considered each deliberation cycle.

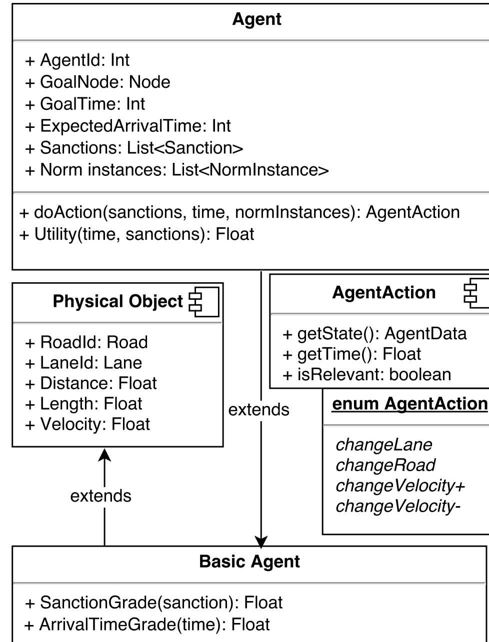

Figure 4.7: UML overview of the agent class.

4.4 Chapter Summary

In this chapter, we discussed the implementation of our model. We talked about the SUMO simulator which we used as a basis to build on. Furthermore, we showed how we divided our implementation into several disjoint

parts using the Model-View-Controller paradigm. Finally, we provided a detailed explanation of how we implemented the various parts of the model, using UML diagrams.

Chapter 5

Experiments

In the previous chapters we have given the formal definition of our framework, as well as a description of how our framework was implemented. The aim of this chapter is to provide some scenarios in which norm based traffic control systems are useful. To this end, we tested the performance of our normative agent based traffic approach using four experiments. The first experiment considers a ramp-merging scenario where the main road consists of a single lane, while on the second, third and fourth experiment the main road has two lanes. In the second experiment the second lane is accessible for all drivers, but in the third experiment the second lane is marked as an “emergency only” lane. Finally, in the fourth experiment, the ramp-merging scenario is used twice in succession in order to demonstrate the use of communication and coordination between decentralized traffic controllers.

Since the goal of this chapter is to present the enabling technologies that our framework provides, we have deliberately exaggerated some aspects of the experiments. This way the effect of norm based systems becomes more apparent. We also report some observed values from the scenarios, to give some indication of the performance of our system when compared to a certain baseline. The contents of this chapter are based on an earlier publication of our work in [6].

The experiments were set up as follows. Each experiment was run for a length of one simulation hour (3600 ticks). The spawn rate shown in the tables of the experiments is defined as the chance of a driver spawning every tick. If there is not enough room to spawn a driver at a certain time, SUMO puts the vehicle on hold and spawns it at the earliest possible time when space is available.

We gathered results by running each experiments in two different scenarios. The first scenario was used to establish a baseline of performance, the second scenario was used to see if an improvement could be made on this baseline. Both scenarios were run one hundred times for every experiment. The values displayed in the tables are the averages over those hundred runs. In the tables, the *throughput* is defined as the number of vehicles leaving the simulation every tick. Similarly, the *average speed* is the average speed over all runs in m/s, and finally the *average gap* is the average distance between two cars in meters. Also defined for each experiment is the *maximum expected throughput*, this is the expected throughput if each vehicle could keep driving its maximum speed throughout the scenario and can be calculated by the following formula: $throughput_{max} = 60p$, where p is the probability of a car entering the simulation on a tick.

5.1 Merge scenario

The experiments we use in this chapter are all based on a ramp merging scenario. We briefly expand on this scenario here, but more details and an in depth treatment on ramp merging is explained by Basker *et al* ([4]). We illustrate our framework by applying it to the common example of merging a ramp (access road) and a main road onto a single traffic stream. The goal in this scenario is to make optimal use of the output capacity of the network whilst not causing unnecessary traffic jams for the secondary traffic stream or compromising safety.

In Figure 5.1, a schematic representation of the aforementioned ramp merging scenario is given. Triangles are vehicles that travel in the direction towards they point. White vehicles are the vehicles that have not yet received their obligation from the control system, while the black vehicles have passed a sensor and thus received a personalized norm instance. In Figure 5.1 the vehicles without a norm instance are vehicles A, B, C and D. On the road there are lane sensors (s_1 to s_5) which can detect the status of vehicles that are residing on them. For the scenario to work correctly, it is necessary that either the sensors are sufficiently long, or the vehicles sufficiently slow, so that no vehicle can pass undetected. The sensors should also be placed at a distance far enough from the merge point m, so that vehicles have enough time to comply to the obligation before the deadline. There are two important points on the road, point m where the two roads merge, and point e where the vehicles exit the scenario. Distances d_A and d_C are agent's A

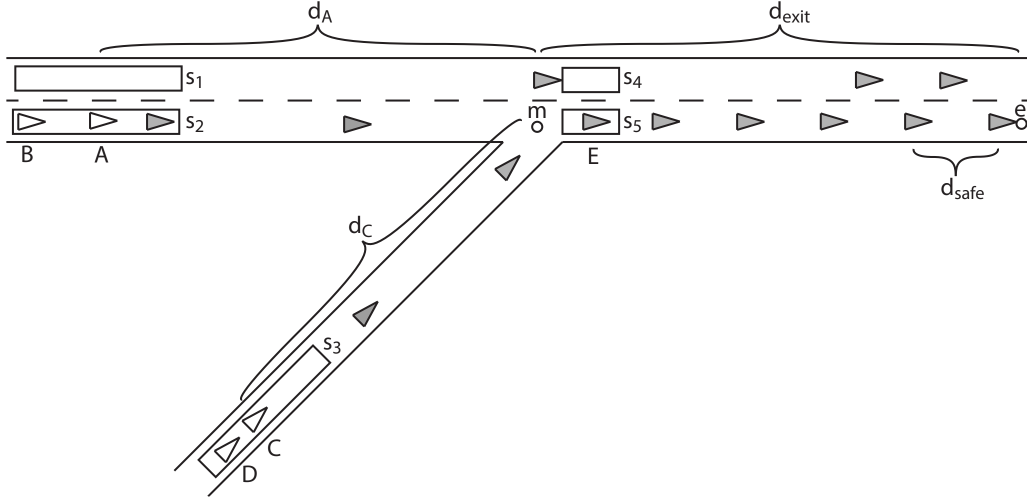

Figure 5.1: The ramp merging scenario

and C's distances to m, and d_{exit} is the distance from the merge point to the exit point, and d_{safe} is the distance between vehicles that is deemed safe, which we call the *minimal gap*. Ideally A and C traverse d_A and d_C such that they arrive at m with a distance d_{safe} and can accelerate to their maximum speed within the distance d_{exit}.

Monitoring happens through interpreting the observations of sensors. In the case of SUMO we use lane detectors that can sense the vehicles that driving along the area they cover. Specifically, each sensor can detect the identity, velocity and position of each vehicle on the sensor's area. Further parameters such as the maximum velocity, acceleration and deceleration capabilities can be assumed within reasonable margins. The traffic control system uses the sensor data to monitor and control the traffic on the road.

5.2 Experiment 1: SUMO and our extension

The first experiment illustrates the distinction in behavior between the default SUMO vehicles and the norm-aware agents implemented in our model. This experiment implements a classic ramp-merging situation, where both the main road and ramp both consist of a single lane. In the scenario with our extension, a single traffic controller observes the vehicles and communicates tailored norm instances to each vehicle. A norm instance in this

experiment consists of just a personalized target velocity, since there is no choice of lanes on the main road. The expected result is that the usage of norms in the scenario with our extension results in a higher average velocity and a better throughput of vehicles since traffic jams will be prevented.

The personalized speed for each vehicle is calculated as follows. First, the sensor readings of the sensors on the main road and the ramp road are combined to create a *merge list*. This is done using the algorithm described in [39]. This merge list contains the desired order in which the vehicles should arrive at the merge point. It is based on the arrival time at the merge point of the vehicles at their current velocity. Next, the personalized speed for each vehicle is calculated. The personalized velocity is calculated by solving a set of equations so that the arrival time of a vehicle at the merge point is at least 2.5 seconds after the car in front of it. This way, traffic safety is ensured, since an adequate distance is maintained between vehicles.

The spawn rate of the vehicle input stream will be slightly higher on the main road to create a realistic traffic situation. In the first scenario of the experiment, the main road has priority over the ramp, comparable to an real life merging situation. In the second scenario, three sensors are placed on the road, one on the main road, one on the ramp and a control sensor on the output road. A single traffic control system observes the vehicles in the simulation and communicates tailored norm instances to each vehicle. In this scenario, the norm scheme used provides a personalized speed obligation to vehicles.

	SUMO agents	Norm-aware agents
Main road Spawn rate	20%	20%
Ramp Spawn rate	15%	15%
Throughput	16,16	21,01
Max throughput	21	21
Throughput %	76,95%	100,05%
Average Speed	3	20,97
Average Gap	13,81	101,82

Table 5.1: The results for the first scenario

As is clear from the results in Table 5.1, there is an increase in both throughput, average speed and the average amount of space between the vehicles. This is the case since in the SUMO scenario a traffic jam instantly

forms on the ramp, because of the relatively high density of cars on the main road. In the scenario with our extension however, the vehicles adjust their speed so that the merge happens smoothly (Figure 5.2). These results confirm our expectation that coordination by a norm-based traffic control system improves traffic flow classic ramp-merging scenarios. Note that the throughput % value exceeds a hundred percent. This is possible because the spawn rate is probability based and thus can exceed the maximum expected throughput.

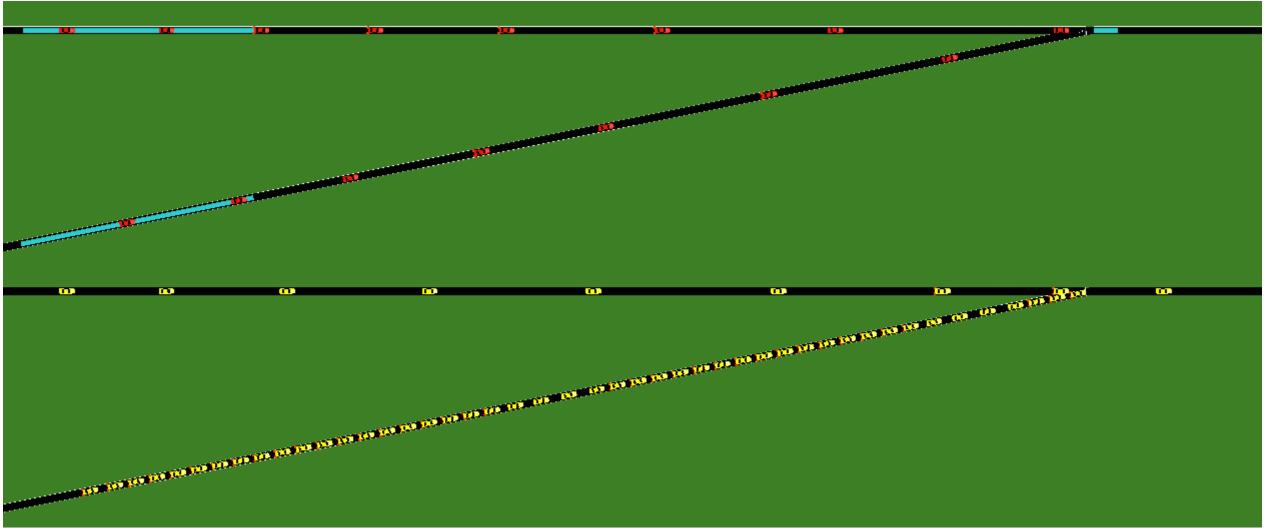

Figure 5.2: Screenshot depicting the difference in performance in the first scenario. The top scenario uses our norm aware agents and merge norm. The bottom scenario uses the default SUMO drivers.

5.3 Experiment 2: Simple norms and Advanced norms

The goal of the second experiment is to compare traffic control systems using simple and advanced norms. In each scenario, the road network is observed by a traffic control system. In the SimpleNorm scenario, the traffic controller employs the same norm as in the first scenario. In the AdvancedNorm scenario, the traffic control system can also the regulation for the vehicles

to change lanes in order to reduce the pressure on the rightmost lane and prevent congestion. The lane change directive will be given to a driver when its calculated velocity on the merge point is below a certain threshold. For this experiment the threshold was set to $\frac{v_{max}}{2}$. Our expectation is that in this multi-lane scenario, the control system with the advanced norm can successfully cope with a higher input stream of vehicles while the traffic controller with the simple norm cannot.

The setup for the SimpleNorm scenario is a copy of the extension scenario in experiment 1, except that in this case the main road has two lanes instead of one, and moreover, the input stream of vehicles of both roads are increased. The AdvancedNorm scenario implements extra sensors on the second lane, but is exactly the same in every other aspect.

As can be observed from the results in Table 5.2, the simple norm cannot handle the increased spawn rate of vehicles in this scenario. The average speed is has diminished severely, as well as the average gap between vehicles. This means congestion is abundant in the SimpleNorm scenario. However, the AdvancedNorm copes very well with the increased input stream of vehicles. In this scenario the throughput approximates the maximum expected throughput closely, which indicates that the vehicles move throughout the simulation without much congestion.

	SimpleNorm	AdvancedNorm
Main road Spawn rate	30%	30%
Ramp Spawn rate	20%	20%
Throughput	20,38	29,91
Max throughput	30	30
Max throughput %	67,93%	99,70%
Average Speed	3,31	14,91
Average Gap	14,2	61,49

Table 5.2: The results for the second scenario

5.4 Experiment 3: Violating norms

The third experiment illustrates that autonomous vehicles are able to reason about norms. Experiment 1 has shown that the vehicles are norm-aware. However, autonomous vehicles in our model also have the capabilities to not

comply with certain obligations if these are of low importance to them. In this experiment the leftmost lane is an emergency lane, reserved for certain traffic in order to help with accidents and other emergencies. Therefore regular vehicles will get sanctioned if caught driving on this lane. Since this lane remains mostly empty, this is a viable option for autonomous vehicles who greatly value a faster arrival time and whose owners are in a financial position which makes them willing to take a fine. We expect that the vehicles of affluent owners will choose to take a sanction and win some time, while the vehicles of poor owners choose not to.

The scenario is the same as with experiment 2. However, the leftmost lane is only open for emergency vehicles. In the Poor Drivers scenario, the input stream consists of drivers who are impatient, but in a substandard financial position. The Affluent Drivers scenario spawns drivers who care about being sanctioned, but are willing to take a fine if by doing so they can arrive closer to their goal arrival time with a significant amount of time.

The results for this experiment are shown in Table 5.3. We can see that with this experiment the difference in throughput, average speed and average gap is much smaller. However, a large distinction in the number of sanctions can be seen. This indicates a difference in behavior between the groups of drivers. On average about 133 affluent drivers decide to drive on the emergency lane in an hour of simulation. This shows a clear difference in behavior from the poor drivers, who never decide to change lanes.

	Poor drivers	Affluent drivers
Main road Spawn rate	20%	20%
Ramp Spawn rate	15%	15%
Throughput	20,48	20,88
Average Speed	12,95	14,42
Average Gap	48,37	69,29
Sanctions	0	133,12

Table 5.3: The results for the third scenario

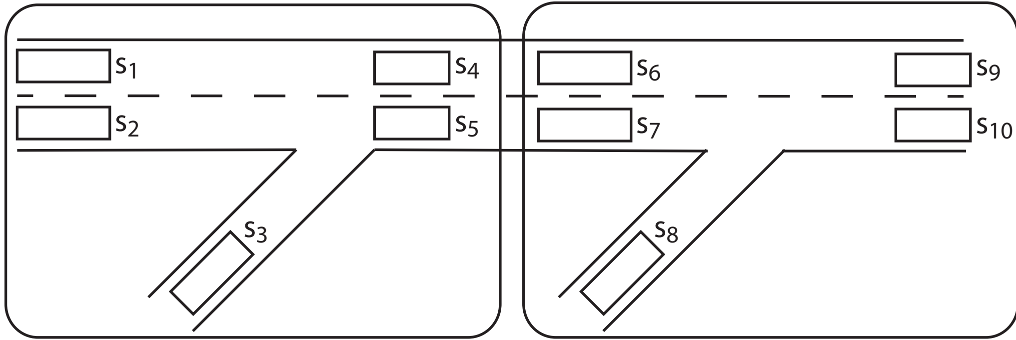

Figure 5.3: Distributed traffic control setting. Rounded boxes indicate local traffic controllers. The left controller is connected to sensors 1 to 5 and the right controller to sensors 6 to 10

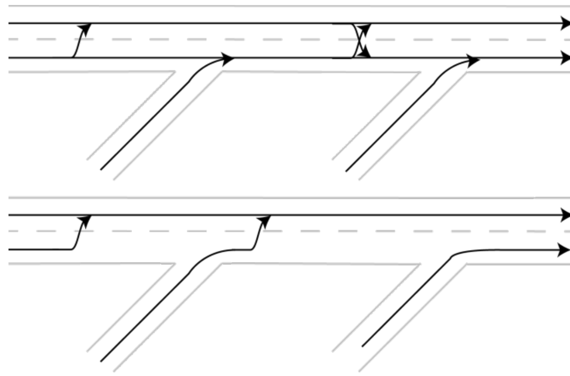

Figure 5.4: Traffic streams (arrows) without coordination. Bottom: traffic streams with coordination.

5.5 Experiment 4: Observation sharing between controllers

In our final experiment, we demonstrate the distributed control features of our framework. More specifically, we demonstrate ability of traffic controllers to share sensor readings, so that they can effectuate their norms more effectively. The scenario in this experiment consists of two merge points in succession (Figure 5.3). At each merge point, a norm is used by a local traffic controller to merge the traffic. For this, each controller uses local road sensors.

If the traffic controllers do not share their observations, the traffic streams flow like the top situation in Figure 5.4. The average speed on the left lane is higher than on the right lane, since no merging occurs there. So the traffic that arrives on the left lane stays there, and some traffic on arriving on the right lane switches to the left lane as well. The ramp traffic streams merge in on the right lane of the main road. After the first merge point, the vehicles can move freely move from left to right and vice versa. However, this behavior can cause congestion at the second merge point when the traffic stream of the second ramp road is too dense.

The solution that we implemented in our observation sharing scenario was to redirect all traffic observed by the left control system to the left lane when the traffic stream on the second ramp rood is too dense. This way, the traffic on the second ramp can continue on the main road unhindered by oncoming traffic. However, by default the left control system relies on only its own sensor readings and cannot detect the traffic stream on the second ramp. Therefore the controller from the right needs to inform the controller on the left about the traffic density on the second ramp.

Hence, the left traffic control system subscribes to the observations of the right controller. Therefore, it receives the sensor data of the sensor s_8 from the right traffic controller. When a high traffic density on the second ramp is detected by the left controller, it issues directives to the vehicles to move to the left lane. Else, it issues the directives used in the merge scenarios discussed earlier. The resulting traffic streams should resemble the streams in the bottom depiction of Figure 5.3. We expect that the coordinating traffic control systems perform better in terms of throughput, average speed and average gap, since less congestion should occur at the second merge point.

The results of experiment 4 are listed in Table 5.4. A small increase in the throughput and a larger increase the average speed and gap in the coordinated control systems scenario compared with uncoordinated control systems scenario can be observed. Thus, giving vehicles on the main road the obligation to change to the left lane quickly after the first merging point appears to prevent the delays as observed in the original scenario. These preliminary results support our hypothesis that observation sharing and communication between traffic control systems are beneficial for traffic regulation.

	No Coordination	Coordination
Main road Spawn rate	25%	25%
Ramp #1 Spawn rate	15%	15%
Ramp #2 Spawn rate	35%	35%
Throughput	43,81	44.74
Average Speed	11.32	14.60
Average Gap	62.47	66.03
Max throughput	45	45
Max throughput %	97,36%	99,36%

Table 5.4: The results for experiment four

5.6 Chapter Summary

In this chapter we discussed various experiments we performed using our framework. First, we presented an experiment that revolved around the notion of norm-aware agents. Secondly, we showed an experiment that demonstrated a more complex norm. Thirdly, we discussed an experiment in which showed that regulations could be ignored . Finally, we presented an experiment in which distributed controllers were employed. In general, the outcomes of these experiments were positive. That is, our framework performed better than the SUMO baseline and simpler versions of our framework.

Chapter 6

Conclusions & Future Research

The aim of this thesis was to answer the question *how can future traffic be modeled and implemented using norm-based controllers within the MAS paradigm, and can this paradigm be used to improve traffic safety and efficiency?*. We divided this question up into three subquestions about the formal traffic model, the implementation of the model and the experiments with it. In this chapter we will review the answers to the subquestions. We will critically discuss our work and give some suggestions for future research.

6.1 Answering the research questions

The first subquestion *how can we model future traffic within the MAS paradigm?* was answered in Chapter 3. We presented our formal model of future traffic, which couples the notions of traffic controller, autonomous vehicle and traffic regulation to that of the multi-agent system concepts of controller, agent and norm introduced in Chapter 2. We specified how agents deliberate about norms issued by controllers, and how these agents could be endowed with different behaviors using agent profiles. Our formalization was deliberately kept abstract, allowing for various implementations.

In Chapter 4 we answered the subquestion *how can we implement the MAS model of future traffic?*. We discussed our implementation and the choices we made during development. Our implementation is written in Java and build on top of the SUMO traffic simulation package. It is constructed using the Model-View-Control design pattern, which allows for easy changing of the domain specific model and the visual representation. In the implementation, one can easily design different scenarios consisting of vari-

ous traffic situations and regulations.

Finally, we presented some experiments using our framework in Chapter 5. The goal of the experiments was to show that the norm based multi agent system approach was useful to model and improve certain traffic situation. We presented four experiments, all based on ramp merge scenarios. Each experiment consisted of two scenarios, where the second scenario included a more complex instantiation of our framework than the first one. In all experiments, the second scenario performed better than the first one in terms of efficiency. Furthermore, we also demonstrated that our framework can be used to improve the safety in traffic by creating a norm which causes agents to keep a safe distance from each other while also increasing traffic efficiency. Thus, the subquestion *can norm-based controllers in conjunction with the MAS paradigm improve on traffic safety and efficiency?* was answered positively.

6.2 Discussion

In this section we discuss our presented framework and take note of some strengths and weaknesses. First of all, we presented positive results of our experiments in Chapter 5. However, we acknowledge that some scenarios were not completely realistic. In the first experiment, the merge scenario did not have an acceleration lane aligned with the main road. Furthermore, in the final experiment the traffic density of the second merge lane was higher than one would expect in the real life.

Nonetheless, we feel that this is not a large issue. The aim of our experiments was not to demonstrate that our model supports extensive realistic scenarios. Our goal was rather to present our extension as an enabling technology for specifying and testing norm based traffic control systems. Furthermore, since our framework allows for a wide range of agent and norm implementations, we believe that more complex scenarios can be modeled by refining the agents and norms.

Second of all, this refinement of agent and norms is possible because of the abstract nature of our framework. We especially kept very few constraints on how norm schemes and instances work. The upside of this is that this gives a lot of freedom to users of our framework. For example, they can construct norms which keep a record of previous violations, allowing norms to adapt to changing environments, or they can implement violation functions in such a way that vehicles aren't punished for violating norms that

they could not possibly obey. The downside is that this freedom might make it more difficult for the users to construct norms at all. By restricting the norms to a small logic or a domain specific language, one can do less but in an easier way.

Third of all, in our framework, sanctions are used in a restricted way. They are only interpreted as being a monetary fine. One could also add support for non-monetary sanctions, such as the revoking of a permit to drive on a certain road. Furthermore, in the literature sanctions are used not only as a way to correct wrong behavior, but also as a way to correct wrong states. Thus, a sanction does not not be a punishment to the violator, but can also be a countermeasure to return from an undesired to a desired state.

Finally, our framework does not allow explicitly for regimentation. However, we are unsure if this is possible in a traffic environment. A vehicle cannot bring about every state that it wishes, but is also dependent on the vehicles around it. It is possible to issue a norm instance with a sanction of infinite euros to all vehicles to ensure that they try their best to comply with that norm. However, this is still no guarantee that the undesirable state can be reached. For example, a car cannot reduce it speed from 120 km/h to 0 km/h in one second, no matter how high the fine.

6.3 Future research

Our framework can be extended in various ways. First of all, one possible extension could be to add support for contrary to duty norms. Our framework could be extended in a number of ways. One can add support for contrary to duty norms ([30]). A contrary to duty norms consist of a hierarchy of norms. An agent should comply with the norms in every layer of the hierarchy. However if it does not comply with the first layer (and thus incurring a sanction), it should at least comply with the second layer or incur an even higher sanction. An example of a contrary to duty norm is the norm "You shouldn't break the speed limit, but if you do, you should drive on the leftmost lane."

Secondly, the distributed controllers could be extended. Currently, our framework only allows for the sharing of sensor data. One could also add the ability of controllers to share sanctions, norms and aggregates of sensor data so that controllers can regulate traffic more effectively.

Third of all, vehicles could also be endowed with the ability to communicate with each other and with the controllers. In this way, they might be able to coordinate their behavior better with other vehicles. Furthermore, they could communicate with controllers do discuss their preferences.

Fourth of all, the framework could be used for another domain. In this thesis, we focused on traffic, but it might also be used for say, the robot soccer domain. It should not proof too difficult to couple this to our framework, since we separated the domain specific parts of our implementation from the more general parts.

Fifth of all, prohibitions could be implemented in our framework. The formal model already allows for prohibitive regulations, but we did not yet implement it.

Finally, a Graphical User Interface (GUI) could be added to the implementation. This GUI could be used for the easy creation of scenarios. It could also be used for allow for on the fly monitoring and editing of norms. For example, with a GUI one could investigate how the vehicle behavior changes when one changes the sanction severity of a norm. Because of our Model View Controller structure of the framework, this can be implemented by only altering the View part of our extension.

Bibliography

[1] Alechina, N., Dastani, M., and Logan, B. Programming norm-aware agents. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 2* (2012), International Foundation for Autonomous Agents and Multiagent Systems, pp. 1057–1064.

[2] Baines, V., and Padget, J. A situational awareness approach to intelligent vehicle agents. In *Modeling Mobility with Open Data*. Springer, 2015, pp. 77–103.

[3] Balke, T., De Vos, M., Padget, J., and Traskas, D. On-line reasoning for institutionally-situated bdi agents. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 3* (2011), International Foundation for Autonomous Agents and Multiagent Systems, pp. 1109–1110.

[4] Baskar, L. D., De Schutter, B., Hellendoorn, J., and Papp, Z. Traffic control and intelligent vehicle highway systems: a survey. *IET Intelligent Transport Systems 5*, 1 (2011), 38–52.

[5] Bates, J., et al. The role of emotion in believable agents. *Communications of the ACM 37*, 7 (1994), 122–125.

[6] Baumfalk, J., Dastani, M., Poot, B., and Testerink, B. Distributed normative traffic control systems as a sumo extension. In *Intermodal Simulation for Intermodal Transport*. Springer, 2016, p. to appear.

[7] Beck, K. *Test-driven development: by example*. Addison-Wesley Professional, 2003.

[8] Behrisch, M., Bieker, L., Erdmann, J., and Krajzewicz, D. Sumo-simulation of urban mobility-an overview. In *SIMUL 2011, The Third International Conference on Advances in System Simulation* (2011), pp. 55–60.

[9] Boella, G., Van Der Torre, L., and Verhagen, H. Introduction to normative multiagent systems. *Computational & Mathematical Organization Theory 12*, 2-3 (2006), 71–79.

[10] Bonabeau, E. Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences 99*, suppl 3 (2002), 7280–7287.

[11] Bora, K. Nissan gets into self-driving mode, says its autonomous cars will be ready by 2020, May 2015. [Online; checked 26-06-2015].

[12] Bratman, M. Intention, plans, and practical reason.

[13] Cohen, P. R., and Levesque, H. J. Intention is choice with commitment. *Artificial intelligence 42*, 2 (1990), 213–261.

[14] Dastani, M., Grossi, D., Meyer, J.-J. C., and Tinnemeier, N. Normative multi-agent programs and their logics. In *Knowledge Representation for Agents and Multi-Agent Systems*. Springer, 2009, pp. 16–31.

[15] Dennett, D. C. *The intentional stance*. MIT press, 1989.

[16] Dignum, F. Autonomous agents with norms. *Artificial Intelligence and Law 7*, 1 (1999), 69–79.

[17] Dignum, V., Vázquez-Salceda, J., and Dignum, F. Omni: Introducing social structure, norms and ontologies into agent organizations. In *Programming multi-agent systems*. Springer, 2005, pp. 181–198.

[18] Doniec, A., Espie, S., Mandiau, R., and Piechowiak, S. Non-normative behaviour in multi-agent system: Some experiments in traffic simulation. In *Proceedings of the IEEE/WIC/ACM international conference on Intelligent Agent Technology* (2006), IEEE Computer Society, pp. 30–36.

[19] Elefteriadou, L. *An introduction to traffic flow theory*. Springer, 2014.

[20] Grossi, D., Aldewereld, H., and Dignum, F. Ubi lex, ibi poena: Designing norm enforcement in e-institutions. In *Coordination, organizations, institutions, and norms in agent systems II*. Springer, 2007, pp. 101–114.

[21] Guizzo, E. How google's self-driving car works. *IEEE Spectrum Online, October 18* (2011).

[22] Jones, A. J., and Sergot, M. On the characterisation of law and computer systems: The normative systems perspective. *Deontic logic in computer science: normative system specification* (1993), 275–307.

[23] Krajzewicz, D., Erdmann, J., Behrisch, M., and Bieker, L. Recent development and applications of SUMO - Simulation of Urban MObility. *International Journal On Advances in Systems and Measurements 5*, 3&4 (December 2012), 128–138.

[24] Krajzewicz, D., Erdmann, J., Behrisch, M., and Bieker, L. Recent development and applications of sumo–simulation of urban mobility. *International Journal On Advances in Systems and Measurements 5*, 3&4 (2012).

[25] Krauss, S., Wagner, P., and Gawron, C. Metastable states in a microscopic model of traffic flow. *Physical Review E 55*, 5 (1997), 5597.

[26] Meneguzzi, F., and Luck, M. Norm-based behaviour modification in bdi agents. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1* (2009), International Foundation for Autonomous Agents and Multiagent Systems, pp. 177–184.

[27] Meyer, J.-J., and Wieringa, R. J. Deontic logic: A concise overview.

[28] Meyer, J.-J. C. Reasoning about emotional agents. *International journal of intelligent systems 21*, 6 (2006), 601–619.

[29] Nilsson, N. J. *Principles of artificial intelligence*. Morgan Kaufmann, 2014.

[30] Prakken, H., and Sergot, M. Contrary-to-duty obligations. *Studia Logica 57*, 1 (1996), 91–115.

[31] Rao, A. S., and Georgeff, M. P. Modeling rational agents within a bdi-architecture. *KR 91* (1991), 473–484.

[32] Searle, J. R. *The construction of social reality*. Simon and Schuster, 1995.

[33] Smith, R. The contract net protocol: Highlevel communication and control in a distributed problem solver, 1980. *IEEE Trans. on Computers, C 29*, 12.

[34] Testerink, B., Dastani, M., and Meyer, J.-J. Norm monitoring through observation sharing. In *Proceedings of the European Conference on Social Intelligence* (2014), pp. 291–304.

[35] Testerink, B., Dastani, M., and Meyer, J.-J. Norms in distributed organizations. In *Coordination, Organizations, Institutions, and Norms in Agent Systems IX*. Springer, 2014, pp. 120–135.

[36] Tinnemeier, N. Organizing agent organizations: syntax and operational semantics of an organization-oriented programming language. *SIKS Dissertation Series 2011*, 02 (2011).

[37] Tinnemeier, N. A., Dastani, M., and Meyer, J.-J. C. Orwell's nightmare for agents, programming multi-agent organisations. In *Programming Multi-Agent Systems*. Springer, 2009, pp. 56–71.

[38] Von Wright, G. H. Deontic logic. *Mind* (1951), 1–15.

[39] Wang, Z., Kulik, L., and Ramamohanarao, K. Proactive traffic merging strategies for sensor-enabled cars. In *Proceedings of the fourth ACM international workshop on Vehicular ad hoc networks* (2007), ACM, pp. 39–48.

[40] Wooldridge, M. *An introduction to multiagent systems*. John Wiley & Sons, 2009.

[41] Wooldridge, M., and Jennings, N. R. Intelligent agents: Theory and practice. *The knowledge engineering review 10*, 02 (1995), 115–152.

Appendix: User Guide

Our framework is open source and available online on Github at
https://github.com/baumfalk/TrafficMAS. It can be compiled from source, or it can be downloaded as a binary version.

About our framework

When implementing the framework we used the test-driven development methodology. ([7]). With test-driven development, one first writes the code that tests the implementation before the implementation is made. If the test passes, the implementation is assumed to be correct. Unit tests are used to quickly detect regressions after adding new features to the code base. If a regression occurred some unit tests that previously passed would now fail, notifying the developers of a bug in the code.

We used Github as the storage and version management software of our implementation. There are several reasons for this choice. First of all, it allows for easy sharing of our work with the rest of the world. The repositories are open source by default, thus everyone can download, run and modify our implementation. Second of all, Github has a state of the art issue, milestone and release system. This system allowed us to focus our development efforts and keep track of our progress. Finally, Github allows for other services to connect with it. We used this to couple a continuous integration service, Travis, to our repository. A continuous integration service is used to continually combine the new code with the existing code base and report errors early on in the process. Every time some code was uploaded to Github, the Travis service ran our entire test suite of unit tests and reported if the integration went well or if some unit tests failed.

How to run it

Our framework can be run as follows. Assuming you use the binary JAR file, a scenario can be run with the following command:

`java -jar TrafficMAS.jar ./scen/ scenario.mas.xml path/to/sumo scenario.sumocfg [seed].`

In this command `scen` is the directory the scenario is located in, `scenario.mas.xml` is the main configuration file for the scenario and `path/to/sumo` denotes the sumo executable to use. The sumo-gui program can also be used. The parameter `scenario.sumocfg` denotes the sumo configuration file used by the scenario. Finally, the parameter `seed` is used to prepare the random number generator, which is used to spawn vehicles in a probabilistic fashion. If no seed is provided, a random one is generated by the system.

Seeing the framework in action

To see the framework in action with a simple merge scenario, follow these steps:

1. Download and install the latest version of SUMO from www.sumo-sim.org. At the time of writing, the latest version is 0.23.

2. Download the zip file SimpleExample.zip from www.github.com/baumfalk/TrafficMAS/releases/tag/SimpleExample

3. Unzip the file and run the following command in the folder you unzipped it in:

   ```
   java -jar TrafficMAS.jar sim/mergeExample/ comparetest.mas.xml
   sumo-gui comparetest.cfg.xml
   ```

You should see the merge scenario we described in Experiment 1 of Chapter 5. On the top of the screen our norm-based framework can be seen, in the bottom the vehicles with the default SUMO behavior are shown.

How to create your own scenario

Our framework also allows for the creation of your own scenarios. A TrafficMAS scenario consists of several XML-files:

- a global configuration file, containing the paths to the other xml files, as well as the simulation duration.

- a configuration file specifying which norms are used. In this file the norms are also parameterized with scenario specific information, such as road names.

- a configuration file which describes the norm based traffic controllers. The file is used to define which controllers there are, which sensors they have access to and to which other controllers they are subscribed.

- a configuration file containing the vehicle profile distributions. This file contains the distributions of the various driver profiles and the traffic density of the different roads.

- various sumo xml files: the xml file containing the nodes, the edges, the sensors and the routes.

Example scenarios and their corresponding XML-files can be found on the Github page, under the `sim` directory.