An Experimental Space for Conducting Driving Behavior Studies Based on a Multiuser Networked 3D Virtual Environment and the Scenario Markup Language

Kugamoorthy GAJANANAN

Department of Informatics,
School of Multidisciplinary Sciences

The Graduate University for Advanced Studies (SOKENDAI)

Doctoral Dissertation

Doctor of Philosophy

September 2013

A dissertation submitted to the Department of Informatics,
School of Multidisciplinary Sciences,
The Graduate University for Advanced Studies(SOKENDAI)
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Informatics

Advisory committee

Professor Helmut Prendinger (SOKENDAI)
Professor Ken Satoh (SOKENDAI)
Associate Professor Tetsunari Inamura
Associate Professor Asanobu Kitamoto (SOKENDAI)
Associate Professor Ryutaro Ichise (SOKENDAI)
Professor Masao Kuwahara (Tohoku University)

Abstract

This thesis describes a new framework for conducting controlled driving behavior studies based on multiuser networked three-dimensional (3D) virtual environments. The framework supports (a) the simulation of multiuser immersive driving, (b) the visualization of interactive surrounding traffic, (c) the specification and creation of reproducible traffic scenarios, and (d) the collection of meaningful driving behavior data. Thus the framework allows traffic engineers to investigate complex traffic situations that depend on the interaction between multiple drivers. Specifically, we use our framework to investigate the 'rubbernecking' phenomenon, which refers to the slowing down of a driver due to an accident on the opposite side of the road, and its effect on the following drivers. The main contribution of the thesis is two-fold. First, we developed the Scenario Markup Language (SML) as a practical tool to specify dynamic traffic situations (e.g. an accident). Second, we designed the SML Framework to simulate interactive ambient vehicles in a multiuser driving simulator and to ensure the reproducibility of particular traffic situations, so that traffic engineers can obtain comparable data and draw valid conclusions. To demonstrate the effectiveness of our framework, we collaborated with traffic engineers to specify the traffic accident scenario in SML and to conduct a study on the rubbernecking phenomenon.

In the Chapter 1, main stream of alternative approaches for driver behavior experiments are introduced, and their relative pros and cons are discussed. Then, the importance of multiuser driving simulators and of simulating realistic traffic scenarios to create predictable experiences for human participants in driver experiments, are discussed. Next, the challenges in simulating realistic traffic scenarios in driving simulators are introduced.

In the Chapter 2, related works are introduced in various areas related to this research. We specifically review the techniques used for simulating ambient traffic in driving simulator environment, scenario authoring

as high level specification, scenario control system implementation, and intermediate mapping representations that maps high level specification to scenario implementation.

The Chapter 3 describes the motivating traffic phenomenon called 'rubbernecking and the tagging structures of the Scenario Markup Language (SML) by walking through an example from our target traffic situation, the specification of an accident using the Scenario Markup Language.

The Chapter 4 covers a description of the SML Framework. Here we show how the framework has been designed on top of our in-house three dimensional (3D) virtual environment technologies. Specifically, we present the approaches utilized for a) simulating multiuser immersive driving in 3D virtual environment, b) simulating ambient traffic in a multiuser driving simulator, and c) implementing the intermediate representation mapping and scenario control scheme. In addition, we provide the details of how research and technical challenges are addressed.

In the Chapter 5, we explain how we demonstrate the effectiveness of our framework, by conducting a multiuser driving behavior experiment using SML Framework. We describe our multiuser driving behavior experiment on the traffic phenomenon called rubbernecking effect. Here, we report on the results of our study from two viewpoints: (a) the reproducibility of the traffic accident situation (i.e. state variables of interest are recreated successfully in 78% of the cases), and (b) the interactive car-following behavior of human subjects embedded in the traffic situation of the virtual environment.

Finally, in Chapter 6, we summarize the research work and conclude the thesis. In addition, we indicate the directions for further research.

Acknowledgements

First of all, I extend my deepest gratitude to my supervisor, Professor Helmut Prendinger, for accepting my request to conduct doctoral research in his laboratory and for providing me an opportunity to work in a unique, highly dynamic research environment during last three years. Prof. Prendinger helped me in numerous ways that includes formulating the research topic, presenting and publishing research outcome at various venues, letting me collaborate with many researchers.

I thank the members of my Ph.D thesis committee, Professor Ken Satoh, Assoc.Professor Tetsunari Inamura, Assoc.Professor Asanobu Kitamoto, and Assoc.Professor Ryutaro Ichise for their critical, yet constructive comments, suggestions and feedback during the meetings I had with them.

Next, I would like to thank my collaborators, in particular, Prof. Edward Chung, Dr. Marc Miska, and Dr. Alfredo Nantes from the Smart Transport Research Center, Queensland University of Technology, in Australia. Prof. Edward Chung and Dr. Miska invited me as a visiting researcher and helped me to conduct a driver behavioral experiment at their center. Moreover, Dr. Miska helped me to gain knowledge about subject area, and contributed towards developing a traffic simulator. Dr. Alfredo Nantes helped me to formulate the hypotheses from traffic engineering,and to analyze the behavioral data to draw valid conclusions on those hypotheses.

Then, I thank Dr. Arturo Nakasone, who helped me design the Scenario Markup Language, which is an integral part of my research. Apart from that, he also helped in general about conducting research, by sharing his own experience.

I also thank all of my lab mates namely Martin Lindner, Edgar Santos, Marconi Madruga, Eurico Doirado, Goncalo Pereira, Sandra Gama, Pedro Cuba, Edern Grey, Joana Silva, Iker Azorin, Kuan-Lin Chiu, Edern Grey and Ricardo Mendes for their valuable contribution in this work. Without their hard work and contribution, certainly none of my thesis work would exist.

Moreover, I thank Dr. Keisuke Okuno, who has been my mentor ever since I joined the PhD program and even now. His in-depth questions and critics about my research work made me to think objectively about the problems and solved them very efficiently.

Further, I thank Dr. Johan Olstam from VTI, the Swedish National Road and Transport Research Institute. I came across his work in my literature review and contacted him for technical questions about his approach. Dr. Olstam helped me by answering my questions, as well sharing his knowledge and experience in traffic scenario simulation.

I thank National Institute of Informatics for providing me the necessary financial, human and technological resources support. I also thank the Graduate University for Advanced Studies (SOKENDAI) for awarding me the Short-Stay Study Abroad Program grant, which helped me to conduct the driving experiment in the Smart Transport Research Center.

Further, I also thank NII secretaries for the their assistance to handle all the administrative and paper work in Japanese language throughout my research work, which made my life easy a foreign student.

Finally, I think my parents, my wife and my friends for their love, encouragement and great support through out my research, which helped me to handle the challenging research situations very well.

To my parents, for their love and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and Background

Traffic engineers study driving behavior by exposing human participants to specific traffic situations, such as an accident or a merging maneuver. The natural responses of drivers during these studies are captured and analyzed to validate relevant hypotheses about how drivers respond to some traffic situations of interest. Large scale behavioral studies are required by traffic engineers to properly support these hypotheses. Examples include traffic phenomena like the rubber necking effect (explained later in Chapter 3), intersection safety, car following in traffic congestion, and so on.

Traffic engineers rarely conduct behavioral studies in the real world, even though it is the most realistic environment. One reason is that, real world is unpredictable with regards to weather and road traffic conditions. Another reason is that the limited control possibilities in the real world can undermine the design of driver behavior experiments with reproducible conditions for all subjects [37]. Test tracks offer a safe environment and the possibility of providing driver subjects equivalent conditions. However, test tracks have drawbacks notably with regard to the variety and complexity of the driving context [37]. Alternatively, they use web-based surveys which, however, restrict the type of behavioral data collected. In recent years, simulators were built to address several aspects of transport, including driving behavior, traffic flow, road safety, and road design (e.g. [19], [37], [60], [46]). For example, Hattrori *et* *al.*[19] successfully used a driving simulator to elicit a driving behavior model from human driving behaviors. Driving simulators support driving in a simulated three dimensional (3D)virtual environment and hence offer a somewhat less realistic environment than the real world. However, one can safely control, vary and reproduce the experimental conditions in in a simulated virtual environment. In addition, a simulated environment has proven to be not only a highly desirable platform for transport behavioral studies but also a powerful analysis tool of the simulation results. The transport behavioral studies conducted in a simulated environment can yield better understanding of traffic phenomenon being studied and can be a a powerful way to gain widespread acceptance of complex strategies tested. Unlike other driving data collection methods, driving simulators show high validity of study results, which is crucial to assess how drivers will react in real-life situations [32].

To conduct transport behavioral experiments in simulated environments, traffic engineers want to create predictable experiences for human participants. To achieve this, they require realistic scenarios and precisely controlled events [22]. Here, traffic engineers author scenarios that unfold in a simulated environment, whereby the scenario script orchestrates the simulation run to create predictable experiences for the participants.

The scenario-based programming is a known tool for authoring and creating realistic scenarios and events in a simulated environment [2]. Here, the challenges are to create believable entity (e.g. a car) behavior and to orchestrate different entities behavior into credible, controlled scenarios [2]. The scenario-based programming mainly deals with handling the interaction between scenario entities. However, the utilization of a simulated environment is often complicated by the knowledge gap that exists between (a) the specification of traffic scenarios by experts, such as traffic engineers; and (b) the technical implementation by software developers. To bridge this gap, three challenges are to be addressed in an integrated manner:

- Design of an authoring language that is practical for traffic engineers to use .
- Design of a practical intermediate representation that maps high level specifications of traffic situations to the scenario implementation.
- Integration of the driving simulator with (i) a general purpose traffic simulator to simulate surrounding vehicles ('ambient traffic') and (ii) a special purpose simulator to create specific incidents, such as a traffic accident.

Willemsen *et al.*[57] proposed an interpreted scripting language and Devillers and Donikian [11] presented a programming style language as authoring languages to orchestrate the events of traffic situations (the first challenge). However, the use of the language of Willemsen *et al.*[57] is tightly linked to a pre-defined road network and, hence, is difficult to reuse outside the framework of the authors. The language of Devillers and Donikian [11], on the other hand, assumes solid programming skills, which may not be possessed by traffic domain experts. In contrast, the design of our scenario language was informed by the domain experts and disentangled from domain specific information (e.g. road network).

Willemsen *et al.*[57] addressed the mapping problem (the second challenge) by employing Hierarchical Concurrent State Machines (HCSMs), whereas Devillers and Donikian [11] used super-step semantics of state-charts. As an alternative to the state machine representations, we propose Behavior trees (BT) [31]. Compared to state machines, BT are scalable, reusable and modular [25].

As for the third challenge, current approaches fall between two extreme cases of control schemes [15]: (i) tightly controlled or scripted vehicles [57], and (ii) fully autonomous vehicle. Approach (i) leads to high reproducibility of scenarios, but requires complex scenario programming [37]. Approach (ii) makes reproducibility of traffic scenarios very hard and is thus rarely used in scenario creation. As an alternative, Olstam *et al.*[37] propose to combine approaches (i) and (ii), and separate phases of autonomous vehicles with phases of controlled vehicles. In the latter phase, add-on behaviors are attached on top of the 'normal' behavioral models used by autonomous vehicles, i.e. dedicated parameters of the behavioral models underlying the 'normal' behavioral updates are manipulated to create scenarios. We will use a similar combined approach, but for increased flexibility, we disentangle the scenario control scheme from the module that simulates the ambient traffic.

All existing proposals in the transport domain, that create scenarios and exceptional events in a simulated environment to conduct behavioral experiments, rely on single-user driving simulation. In these simulations, traffic engineers cannot analyze more complex phenomena, such as the interaction among multiple human drivers and pedestrians. This limitation makes it difficult to collect large scale realistic behavioral data, because in real life, human drivers have to deal with multiple interactions.There are study cases which could benefit from the existence of multiple drivers in driving simulators. One of them is a driver behavior interaction study where traffic engineers really want to investigate how real drivers deal with each other when they perceive a situation, but whose interactions are too dangerous or not allowed to study in the real world.

The introduction of multiple drivers in driving simulator experiments brings in more challenges : a) how to simulate ambient traffic for all subjects who share a common simulation space, b) how to create scenarios using the surrounding traffic which is now shared among multiple drivers simultaneously driving, c) how to design and conduct driver behavior experiments which involves several subjects and investigating interactive driving behavior of multiple drivers in case of exceptional events, and d)how data from such an experiment should be analyzed.

1.2 Aim and Objectives

In this work, we aim to develop an integrated framework which supports :

- Simulation of multiuser immersive driving in a large scale 3D virtual environment.
- Simulation of surrounding traffic in a large scale 3D virtual environment.
- Specification and creation of traffic scenarios (with sufficient reproducibility and controllability)
- Collection of meaningful driving behavior data, and the impact of an exceptional event or scenarios on these behaviors

1.3 Summary of Contributions

To help traffic engineers to author and create scenarios in a simulated 3D environment for the purpose conducting transport behavioral studies, we have to address three challenges:

- Design of an authoring language that is practical for traffic engineers to use.
- Design of a practical intermediate representation that maps high level specifications of traffic situations to the scenario implementation.
- Integration of the driving simulator with (i) a general purpose traffic simulator to simulate surrounding vehicles ('ambient traffic') and (ii) a special purpose simulator to create specific incidents, such as a traffic accident.

Some approaches or ideas addressing above challenges, discussed in this thesis have been explored in other related studies individually but this work made a contribution on the integrated use of them.

Therefore the main contribution of this thesis is the integrated framework which is able to simulate ambient vehicles in a multiuser driving simulator and to enable authoring and creating reproducible traffic scenarios with in the 3D virtual environment. The contributions also include:

- Scenario Markup Language, as a high level specification tool for authoring traffic scenarios
- Flexible Integration of the Microscopic Traffic Simulator and the Multiuser Driving Simulator for the purpose of simulating ambient traffic in a large 3D virtual environment
- Scenario Control Scheme that exploits the above integration, hence allow creating traffic scenarios in the context of a multiuser 3D virtual environment
- Design of a Multiuser Driving Experiment in a simulated 3D environment, analysis of collected multiuser interactive driver behavioral data and extraction of useful results
- The standard to represent the road network in 2D and 3D virtual environment, which is used as a base/input for the Microscopic Traffic Simulator

1.4 Delimitations

The thesis only discusses the following topics to a limited extent.

- Literature survey of traffic simulators
- Literature survey of driving simulators

1.5 Thesis Outline

The thesis is structured as follows. In the Chapter 2, we review techniques used for simulating ambient traffic in driving simulator environment, scenario authoring, intermediate mapping representations, and scenario control system implementation.

Chapter 3 is dedicated to describing the motivating traffic phenomenon called 'rubbernecking and the tagging structures of the Scenario Markup Language (SML) by walking through an example from our target traffic situation, the specification of an accident.

Chapter 4 covers a description of the SML Framework. Here we show how the framework has been designed on top of our in-house three dimensional (3D) virtual environment technologies. Specifically, we present the approaches utilized for a) simulating multiuser immersive driving in 3D virtual environment, b) simulating ambient traffic in a multiuser driving simulator, and c) implementing the intermediate representation mapping and scenario control scheme. In addition, we provide the details of how research and technical challenges are addressed.

In Chapter 5, we describe our multiuser driving behavior experiment on the rubbernecking effect. The analysis of the results will reflect both the perspectives of informatics and traffic engineering.

Finally, in Chapter 6 we will summarize the research work and conclude the thesis. In addition, we indicate the directions for further research.

Chapter 2

Related Work

2.1 Introduction

This chapter reviews the related research work on a) driving simulators used in driver behavior studies, b) the simulation frameworks that generate and simulate ambient or surrounding vehicles in driving simulators, and c) addressing challenges of scenario authoring and implementation. We also discuss the limitations on the related research work in the mentioned areas.

2.2 Driving Simulators

Driving simulators are a great tool to create realistic driver sensations in laboratory environments, and are widely used for driver behavioral experiments on several aspects of transport, including driving behavior, traffic flow, road safety, and road design (e.g. [19], [37], [60], [46]). This is mainly because, driving simulators offer a safe, but easy to control simulated environment in which experimental conditions can be reproduced for all participating drivers.

The driving simulators range from high fidelity ones that uses real vehicle cabins and advanced motion systems with multiple degrees of freedom, that support the drivers visual impression of the vehicles movements to low fidelity ones that simply utilize a simple desktop simulators that are merely a computer monitor with a video game-type steering wheel and pedals. The detailed discussion on the types research driving simulators, with a wide range of capabilities is beyond the scope of this thesis. However, interested readers should refer to [5], [6], [47], and [54] .

Driving simulators a) uses complex or simplified computer models to calculate the physics and movements of the driving simulator driven vehicle in the simulated environment, corresponding to drivers use of the steering wheel and the pedals, b) renders the view of the driver in the virtual environment.

One main limitation in the related research work that utilize driving simulators to investigate travel and driver behavior in case of exceptional events, traffic scenarios, is that almost all of them rely on only a single-user driving simulator. To compensate the non-existence of other user-driven vehicles, a single-user driving simulator is often integrated with a microscopic traffic simulation to generate the surrounding vehicles. In such context, most of the time, only the closest neighborhood of the user-driven vehicle is populated with ambient traffic.

2.3 Simulation of Surrounding Vehicles in Driving Simulators

Since one of the main ideas of driving simulator based experiments is to create a safe environment that resembles driving in real traffic systems as closely as possible, it is important that ambient or surrounding traffic, that behave in a realistic and trustworthy way is required [38]. This is the case for most of the related work, which does not consider multiuser based driving simulators. Therefore, most of the related work use the microscopic simulators for simulating the ambient vehicles. Microscopic traffic simulators, model and simulate traffic and vehicles at the microscopic, or vehicle-by-vehicle, level.

Some of the related approaches for the simulation of the surrounding vehicles use available commercial microscopic traffic simulation tools. For example, Punzo and Ciuffo [42], Ciuffo*et al.* [8], and Gregoire*et al.*[24] used AIMSUN [3], while Jenkins

[21] utilised VISSIM [14] to simulate ambient traffic in driving simulators.

Main limitations in using commercial microscopic traffic simulation tools for simulating ambient traffic is that, the autonomous vehicles behavior can not be controlled flexibly so that they fulfill the the additional demands on the creation of traffic scenarios (Next section discuss how ambient vehicles are controlled to create scenarios in a virtual environment). Most of the time, source code is not available for access and the interface to modify or adapt the behaviors of individual vehicle's behavior or whole traffic system is limited.

For the sake of flexibility of using microscopic traffic simulators to simulate ambient vehicles which can be later controlled for scenario creation, some researchers develop models specialized for the application of simulation of surrounding vehicles in driving simulators. The examples include, the ARCHISIM model [[17], [13]], the NADS model [36], the DRIVERSIM model [58] and the VTI Model [[38], [37]]. Except the work by Olstam and his colleagues, other researchers in this area, has mainly focused on decision making modeling concepts behind the traffic models.

The researchers develop models specialized for the application of simulation of surrounding vehicles in driving simulators, have relied on the concept of area of interest, which is the closest neighborhood of the driving simulator vehicle and it is only within this neighborhood that ambient vehicles have to be simulated [38]. The area of interest moves with the same speed as the driving simulator vehicle and can be interpreted as a moving window, which is centered on the simulator vehicle [38]. The concept of area of interest for simulating ambient traffic in driving simulator was originally proposed in [13] and [7]. The main motivation behind the use of area of interest for generating traffic in driving simulators is that, to avoid simulating vehicles several miles ahead of or behind the simulator vehicle, which is not efficient from a computational point of view [38].

However, simulating ambient traffic for multiuser driving simulators, which share a common simulation environment, is difficult. This is because high computational efforts is required not only to simulate and but also to visualize the ambient traffic, in very large areas and thereby many vehicles, in each of the driving simulators. The moving window technique cannot be directly applied for simulating ambient traffic as in the case for a single-user driving simulator. This is because,in multi user driving simulator environment, generating vehicles within the sight distance of a driving vehicle would make the generated vehicle to pop up in or out front or back of the other simulator vehicle which share the same space. In addition, when running long driving simulator experiments in a large area with multiple users, one-hour experiment at a traffic flow of 1000 vehicles per hour will require that on average, 1000 vehicles per simulation time step have to be updated.

2.4 Addressing Challenges of Scenario Authoring and Implementation

This section reports on work related to the three interrelated challenges of (i) specifying scenarios in a high level language appropriate for non-programmers, (ii) designing an intermediate representation that maps high level specifications to the scenario control implementation, and (iii) implementing scenario control schemes to create specific incidents, such as a traffic accident.

2.4.1 Scenario Authoring Languages

In scenario based programming, the challenges are to create believable entity behavior (animation and simulation) and to orchestrate different entities behavior into credible, controlled scenarios in 3D simulated environments.

First we look at the scripting languages targeted for behavior programming for entities in 3D simulated environments. There has been a family of languages introduced for animation and simulation behaviors of entities in 3D:

- Perlin *et al.*[39] introduced the Improv, which is an action based scripting language to empower computer animators to create behavioral entities which can be controlled by scripts. Using Improv, well-defined actions, which are routines that control an objects degrees of freedom for a period of time, can be created and later invoked in the simulation.

- Conway developed Alice [9], which is programming environment to make 3D animation accessible to novices. Alice provides its own scripting language based on the Python language to control and describe the movements of objects in the environment.

- UnrealEngine [51], which is a game development tool set designed with ease of content creation and programming in mind, provides UnrealScript language [49] as a powerful, built-in programming language that maps naturally onto the needs and nuances of game programming. UnrealScript language defines a distinct notion of state and can be used to build autonomous agents

- Unity3D [50],very similar to UnrealEngine, is a software platform that consists of an integrated tool for design and development of 3D real-time simulation content or games. Unity3D allows developers to write simple behavior scripts in JavaScript, C# or Boo.

It is obvious that, except Alice, all of the above mentioned scripting languages are indented for 3D real-time simulation programmers who work on the low level behavior programming.

Secondly, we review what languages are available for authoring scenarios, especially in traffic domain.

Besides the scripting language of Willemsen *et al.*[57], and the programming style language of Devillers and Donikian [11], other approaches for traffic scenario authoring were developed.

Early work on scenario scripting language by Van Winsum [[52], [53]] propose an approach which is conceptually similar to Improv scripting language [39]; in addition it allows the simulation entities to be commanded to perform certain actions.

Allen *et al.* [45] use a textual description, whereas others utilized a specialized representation called 'tile' [[59], [48]]. The idea of specifying a series of tiles in a configuration file can be sufficient for simple scenarios, but cannot scale well for complex dynamic and non-deterministic scenarios. Another disadvantage with textual and tile based approaches is that the entire scenario (static and dynamic elements) need to be specified beforehand in text or tiles.

In our earlier work, Prendinger *et al.*[[40],[41]] developed XML based markup languages to control complex scenarios involving sequential and parallel activities of animated agents in virtual worlds. These works target the creation of believable dialogue between life-like agents, rather than reproducible traffic situations for behavioral driver studies.

2.4.2 Intermediate Representations that Map High Level Specifications to the Scenario Control Implementations

Researchers used different intermediate representations to bridge the gap between high level specifications and the scenario control implementations. The systems of Kearney *et al.*[22] and Willemsen *et al.*[57] translate scenario instructions into Hierarchical Concurrent State Machines (HCSMs). They used HCSMs for scenario control implementation as well as for modeling ambient vehicle behavior. In his early work on behavior programming or description languages, Donikian [12] proposed hierarchical, parallel, transition systems or states (HPTS) as an approach to capture the behaviors of entities in a interactive simulation. In addition, the HTPS base language can also be used to program scenarios. Later, Devillers and Donikian employed the superstep semantics of statecharts [18] to specify the semantics of scenario instructions [11]. Atir and Harel proposed an approach based on the adaptation of live sequence charts

(LSCs)[2]. However, LSCs do not scale well compared to state machines since the way execution occurs is highly centralized and LSCs are insufficient for complex scenarios. Miguel *et al.*[20] developed a scenario language based on the Grafcet language using Petri Nets (PN), which internally makes use of state machines.

All of the existing approaches use some state machine-based representations, which are difficult to maintain and reuse. Conversely, the Behavior trees (BT) [31] we use are highly modular and scalable.

2.4.3 Control Schemes for Scenario Implementation

Several researchers have proposed methodologies to implement traffic scenarios with surrounding traffic in driving simulators. They fall into the following categories:

- Approaches with tightly controlled vehicles [E.g. [10], [7], and [22]]
- Approaches with autonomous vehicles
- Approaches with directable semi autonomous vehicles [E.g. [57]]
- Approaches that combined or mixed two of the above approaches (i.e. combine autonomous and controlled vehicles in scenario creation or combine autonomous and semi-autonomous vehicles in scenario creation) [E.g. [37]].

2.4.3.1 Controlled Vehicles

The approaches with tightly controlled vehicles offer high controllability of the created traffic situation and reproducibility of the experimental conditions , but they are not scalable to realistic traffic scenarios involving a large number of vehicles [37].

2.4.3.2 Autonomous Vehicles

The autonomous agents and their behavior models are instilled with a basic competence to perform required actions and with the agility to engage in realistic interactions with unpredictable subjects (i.e. users)of the virtual environment [57].

Autonomous agents form the heart of an engaging, interactive virtual environment experience [57]. Realistic autonomous agent behavior contributes to suspension of disbelief on the users's behalf and immerse the user into the virtual environment [26].

The approaches with autonomous vehicles show increased realism of the traffic scenario, but the reproducibility of particular traffic situations is an issue due to the autonomous decisions of the cars [37]. There are virtually no approaches that utilize autonomous vehicles alone for traffic scenario creation. However, autonomous vehicles are used to provide ambient environment to enhance the immersive feeling of real traffic system with in the virtual environment.

2.4.3.3 Directable Semi Autonomous Vehicles

An important issue realized by both driving simulator researchers and autonomous agent researchers is that it is important to orchestrate or direct autonomous behaviors in the environment to create controlled experiences [57]. Related work involved with directable autonomous behaviors originated from the research work in autonomous agent architectures for robotics and animation [[28], [26], [29]]. To convert autonomous agents, into a a directable one that can accept direction, one has to program the agents with public communication interfaces through which they can receive messages or commands from directors (e.g. scenario control). For example, a vehicle agent, which performs a normative autonomous activity in other circumstances, can be commanded to speed up or slow down or to brake abruptly. The directable agents behaviors are truly semi-autonomous since they combine a responsiveness to guidance with their normative autonomous activity [57].

2.4.3.4 Combined or Mixed Approaches

Other researchers proposed to combine autonomous and controlled vehicles in scenario creation [[1], [37], [55], and[56]]. Inspired by the work of Alloyer *et al.*[1] and Wassink *et al.*[[55],[56]], Olstam *et al.*[37] propose an approach to combine the autonomous and semi-autonomous vehicles in scenario creation. In Olstam's work [37], periods with autonomous vehicles are combined with periods with only controlled vehicles. Here the basic idea is to let the surrounding vehicles run in autonomous mode between controlled phases in which predetermined scenarios unfold. In addition, the simulation of the surrounding vehicles should change from the autonomous to the controlled mode to participate in scenario creation.

In Olstam's work [37], the algorithm to bring about a traffic situation is tightly integrated to the simulation time step that calculates behavioral update for all surrounding vehicles based on the ordinary traffic models. The algorithm has to ensure that the surrounding vehicles switch from autonomous to controlled mode to participate in the scenario execution. During the controlled phase, some particular traffic situation is created (e.g. a catching up maneuver, as explained in [37]) by overriding the normative behavior of some vehicles, which actively participate in a scenario, with add-on behaviors. This is achieved by manipulating the parameters of the behavioral models. This approach is sufficient in the case of a single-user driving simulator environment in which only a small number of autonomous vehicles surround a single user driver and some of them actively create a scenario in the closest neighborhood.

In our approach, on the other hand, we decided to restrict the scenario control scheme (that brings about a traffic situation) to orchestrate the behavioral updates of only the vehicles that participate in the scenario. This approach has the advantage that the scenario control scheme for traffic situation creation is disentangled from the simulation time step which calculates behavioral update for other surrounding vehicles that do not participate in the scenario. In this way, our approach is able to create specific traffic situations, while simulating ambient traffic for the multi-user driving simulator, in a shared simulated environment.

2.5 Conclusion

In this chapter, we reviewed related work, that use a) driving simulators for investigating driver behavior in case of exceptional traffic events and scenarios, and b) different approaches to simulate ambient traffic in driving simulators, and that address the challenges of scenario authoring and implementation. We also discussed the limitations in the related approaches in the above mentioned areas.

The research work presented in this thesis relates directly to the discussed body of work, and uses many of the ideas and guidelines for simulating ambient traffic in driving simulators and creating scenarios in this context. However, our work is different from the related work in terms of: a) how we integrate a microscopic traffic simulator with the multiuser driving simulator and b)how we separate scenario authoring and control system from the rest of the simulation.

Our chief goal is to a) make scenario authoring practical for traffic engineers to use, b) simulate ambient traffic in multiuser driving simulators which shared a common, large scale 3D environment, and c) specify and create traffic scenarios with sufficient reproducibility and controllability. In the next chapters, we show how our goal achieved.

Chapter 3

Traffic Scenario Authoring with Scenario Markup Language

3.1 Introduction

SML is a high-level, practical specification language for authoring traffic scenarios. A preliminary version of the basic concepts of the SML was already introduced in Gajananan *et al.*[16]. In this section, we describe the core tagging structures of the SML language, which are used to specify the target traffic scenario.

Note that the usage of the word 'scenario' sometimes refers to both the static structure of the virtual environment or scene (e.g. road network) and the dynamic characteristics of a simulation (e.g. critical events) [15]. We use scenario (script) to refer to entities (vehicles) in the simulation and the orchestration of their behaviors (interactions) to create a specific situation or event [10].

3.2 Motivating Traffic Scenario on Rubbernecking Effect

For this work, we choose a traffic accident scenario as a study case which could benefit from the existence of multiple drivers in driving simulators. As for the accident scenario, we specifically mean the situation in which a following car collides with a leading car from behind. The chosen scenario helps traffic engineers to investigate

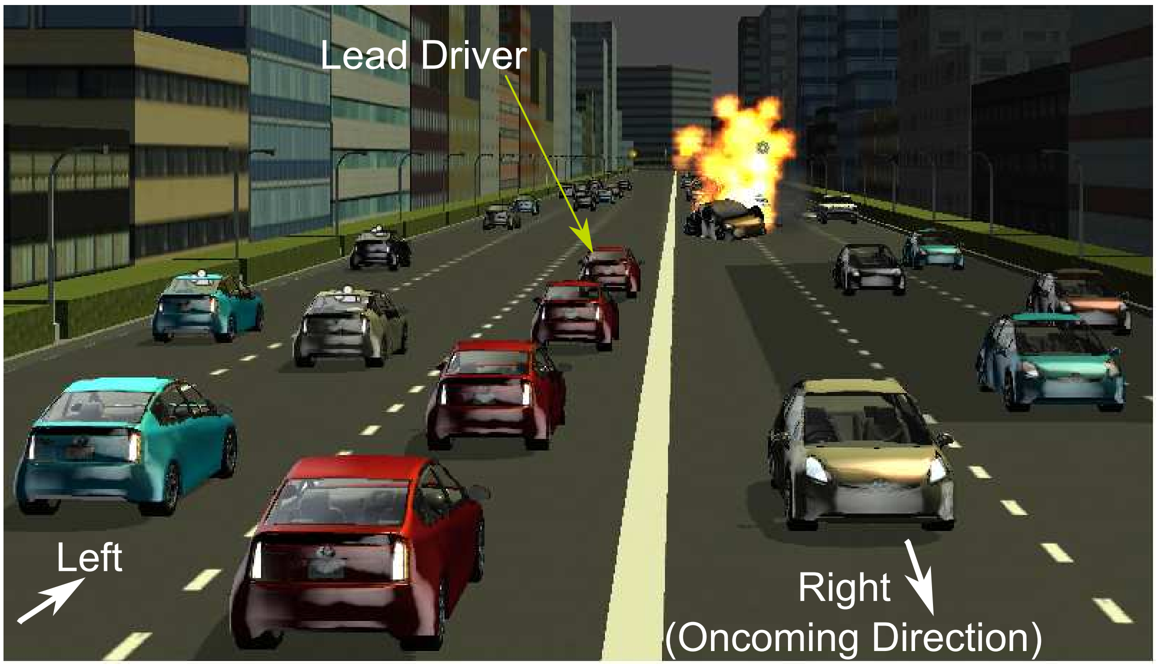


Figure 3.1: Illustration of a rubbernecking scenario.

how real drivers react to each other when they perceive the accident situation (i.e. how drivers change their operational driving behavior at the incident site), but whose interactions are too dangerous or unpractical to be studied in the real world.

This particular scenario is motivated by the traffic phenomenon called rubbernecking; whereby drivers tend to crane their neck in order to get a better view of a nearby car crash. In Fig. 3.1, an illustration of a rubbernecking phenomenon is provided. In this example, an accident has happened on the right side of a road (oncoming direction). The first driver on the left side (reference side), is the one who first perceives the accident and, likely, initiates the formation of traffic jam.

The rubbernecking phenomenon can cause traffic congestion to surge, for drivers may become distracted and drastically change their rate of travel, which in turn may cause traffic congestion on the opposite road side of the road where the accident happened [23].Traffic engineers identified this traffic phenomenon as a leading cause of traffic congestion and secondary accidents. They claim that drivers looking at other vehicle crashes and other roadside traffic incidents cause the rubbernecking effect. We use this traffic accident scenario as a running example to describe the tagging structures of the Scenario Markup Language (see the following section in this Chapter), to explain the technique to create reproducible traffic situations (see Chapter 4) and validate hypotheses about the rubbernecking effect, in the multiuser experiment (see Chapter 6).

3.3 Scenario Markup Language Concepts

We conceptualize a scenario as the script that specifies the behaviors (defined by the interests of a scenario author) of entities, and how those behaviors are orchestrated, by creating, destroying or modifying characteristics of entities and coordinating their actions, to produce experiences for human users immersed in a 3D virtual environment. In this case, an entity refers to a materialized or embodied virtual object (e.g. a car) or to a virtual character represented as a human avatar (a synthetic human user or a bot). Users who are participants of a behavioral study may interact with one or more entities in the virtual world during a simulation run designated for the behavioral study.

A behavioral entity perceives its environment, makes decisions, performs actions and has an interface for communication. Our conceptualization of defining the behavior of entities is similar to the definition above.

3.3.1 SML Elements and Entities

The root element of a scenario script is SML. The tagging structure *`<SML></SML>`* contains all other tagging structures, in particular 'entities' - the entities involved in a scenario, 'user' - the user referenced in a scenario, 'director' - the portion of script that orchestrates certain behaviors of specified entities, and 'events' - generated as an outcome of executing the scenario, 'behaviors' - a set of interesting behaviors for the entities defined above, from a scenario point of view. We assume that entities

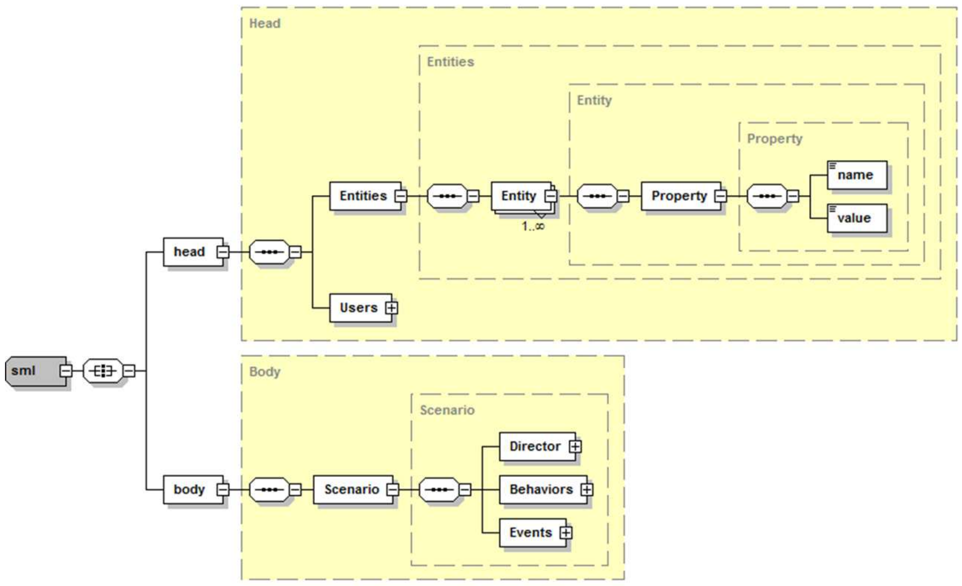


Figure 3.2: The overall structrual diagram of a SML scenario script

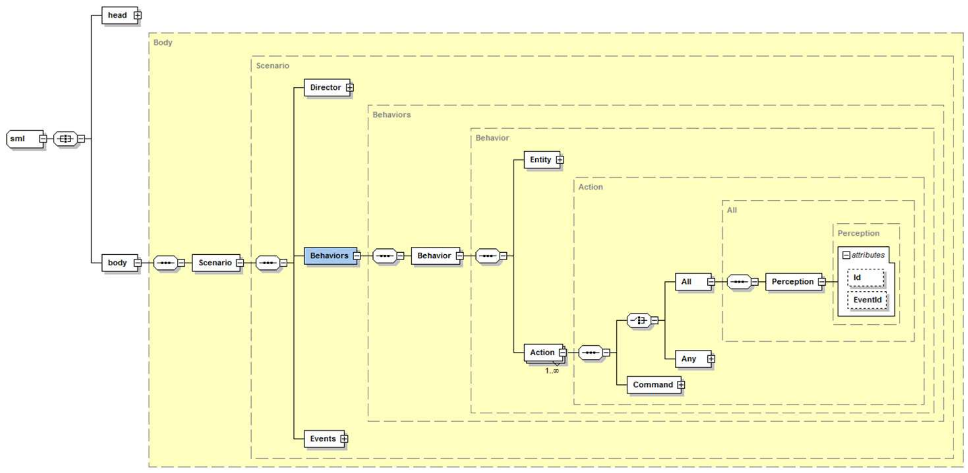


Figure 3.3: The overall structrual diagram of the header part of a SML scenario script

will have their autonomous behaviors implemented, will behave accordingly mostly except the time the behaviors defined in a scenarios are executed. In addition, we implement a set of interface methods to direct or command an entity from a scenario.

For the accident scenario, we define two car entities along with their attributes and properties (see Fig. 3.5). We identify each entity by its id. The ids of entities are assigned once the candidates are chosen from the simulation space (e.g. cars that are selected to create an accident). Each entity has a type. Currently, the *type* attribute supports 'Vehicle', which was sufficient for the scenario used in the experiment (see

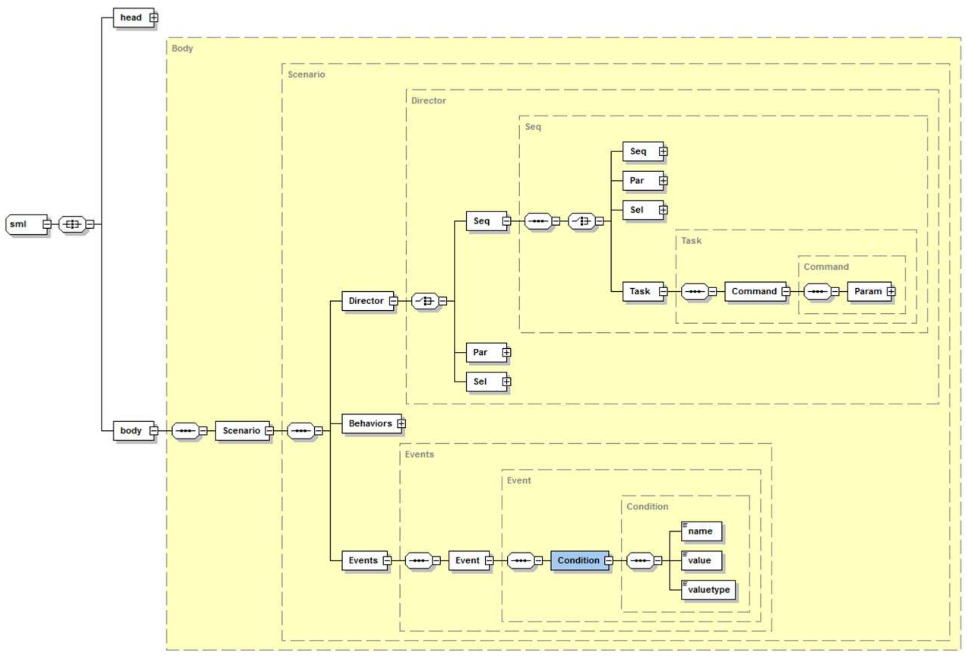


Figure 3.4: The overall structrual diagram of the body part of a SML scenario script

Section V). It can be extended to other types of entities (e.g. pedestrian), once they are implemented in the simulation. To define the properties of an entity, we specify an element called *Property*. Using this tag inside the *Entity* structure, a script author can define the information to initialize an entity for a scenario.

An SML script lets a script author freely define the properties for an entity. However he/she has to ensure that the component responsible for the entity supports the specified properties. In the example script shown in Fig. 3.5, we show some example properties that are currently supported for each type of entities used in our simulation. For a car entity, we define (a) *desiredspeed* that specifies the desired speed, (b) *desiredaccel* that specifies the desired acceleration and, (c) *scenarioroleloc* that specifies what location the entity has to take initially in the scenario execution. Later on, in Section IV, we will discuss how the properties are used.

Our target environment has to handle multiple users simultaneously. We decided that there has to be at least one *User* as the focal point of a scenario, that is, the experimental condition is to be reproduced from the viewpoint of one specified user.

```
<entity name = AI Car A" id=  " type = "Vehicle">
   <property name = "desiredspeed" value = 40m/s"/>
   <property name = desiredaccel" value = 1.6m/s2"/>
   <property name = "scenariorole" value = "Follow_Car"/>
   <property name = scenarioroleloc" value = ROLE_LOC_A"/>
</entity>
<entity name = AI Car B" id=  "  type = "Vehicle" >
   <property name = " desiredspeed " value = 30m/s"/>
   <property name = desiredaccel " value = 1.5m/s2"/>
   <property name = "scenariorole" value = "Lead_Car"/>
   <property name = scenarioroleloc"  value = ROLE_LOC_B"/>
</entity>
```

Figure 3.5: Snippet of an SML script that specifies two car entities.

```
<user name = TestUser01" id=  U01" >
   <property name = "usecarid " value = 01"/>
   <property name = scenariorole" value = referenceuser"/>
</user>
```

Figure 3.6: Snippet of an SML script that specifies the reference user.

This user is identified as shown in Fig. 3.6.

3.3.2 SML Event

One can define a set of events (e.g. an accident) in a scenario specification to create interesting experiences for users.The event definition specifies the conditions that decide when the event is triggered, as shown in Fig. 3.7. The conditions for triggering events can be based on an interaction of a human participant with an entity, or the environment, or of entities with others. The conditions can also be based on the changes in the simulation state, or a timer or the changes that occur in the external applications such as a traffic simulation application.

```
<event>
    <condition>
        <var name =EventId" valueType ="string" value="Accident_Event_01"/>
         <var name ="Collider" valueType ="string" value="CarB"/>
         <var name ="Collidedwith" valueType ="string" value ="CarA"/>
         <var name ="Location" valueType ="vector3" value ="ACC_LOC"/>
   </condition>
</event>
```

Figure 3.7: An example SML script that specifies an accident event

```
<behavior  ref =  "VMS_A"  id = "ManageMessages">
   <action id = "UpdateMessage">
    <perception id = "p3" Eventid = "Accident_Event_01"/>
     <command id = "update">
      <param name = "msg" value = "Reduce Speed  Accident Ahead" />
     </command>
    </action>
</behavior>
```

Figure 3.8: An example SML script that specifies a behavior for an entity.

3.3.3 SML Behavior, Perception with its Container, Action, Command

For each behavior (e.g. ManageMessages), we specify one or more actions (e.g. UpdateMessage etc.) inside each behavior specification as shown in Fig. 3.8. Each action specifies a perception container which has a set of perceptions defined in it. A behavioral entity perceives its environment (e.g. Accident Event) via the perceptions. Each perception is contingent upon an event, which means that a perception is the minimum unit of event handling from the script perspective. Hence the perception is activated when the corresponding event is triggered during a simulation. This may lead to the activation of the perception container the perception belongs to (decision making aspect of an entity). In turn, this leads to the execution of an entitys action that the perception container belongs to. Therfore a perception acts as a pre-condition for an action.

An action definition can have a set of commands defined in it. Therefore, a behavioral entity performs an action by executing the commands specified for that action. A command is the minimum unit of execution from a scenario and entity perspectives. It carries an instruction or a functional call with required parameters for the component that controls the entity. The parameters used with the commands are passed using call by value semantics.

```
<director>
   <seq>
      <task  id="MoveToTargetLocation">
       <command>
         <param name=" entityId value = ""/>
         <param  name="entityName" value = "AI Car A "/>
         <param  name="targetLocation" value ="ACC_LOC"/>
       </command>
      </task>
      <task  id="MoveToTargetLocation">
       <command>
         <param name="entityId" value =""/>
         <param  name="entityName" value = "AI Car B"/>
         <param  name="targetLocation" value ="ACC_LOC"/>
       </command>
      </task>
   </seq>
</director>
```

Figure 3.9: An example SML script that specifies the supervisor element (director) of the accident scenario

3.3.4 SML Director Element

In order to orchestrate the behavior of entities involved in a traffic scenario, we propose the concept of a supervisor, specified as *director* in the script. For the director, we specify task elements which contain information about the actions an entity has to perform. Each task defines a command for actions to be executed by an entity. Therefore each task refers to a single entity and is executed by the director of a scenario. A task execution (i.e. sending commands to entities) is performed via the communication interface available for each entity.

For example, as shown in Fig. 3.9, we define the director for the scenario script that creates the accident situation. In this example, we specify two tasks for the director, whereby each task defines a command to be sent to a different car entity: AI (Artificial Intelligence) Car A and AI Car B.

As shown in Fig. 3.5, the two car entities involved in the scenario are initialized with a property called 'scenarioroleloc' that specifies what role the entity has to take initially in the scenario execution. In this example, AI Car A takes the 'leading role' hence its 'scenarioroleloc' is specified to be in front of that of AI Car B, which takes the 'following role'. When these car entities successfully reach their respective ‘scenarioroleloc’, the director executes its task (‘MoveToTargetLocation’) to create an accident at the specified target location during a simulation run. This operation sends a command to the component that controls the car entity with the parameters specifying the target accident location. The component controlling the car entity tries to execute the command on the respective car entity so that an accident can be created.

A scenario director can orchestrate the behavior of entities by executing corresponding tasks. This can be done by using scenario the temporal elements of director, such as *seq* (sequential), *par* (parallel) or *sel* (selective). With the *seq* element one can specify that tasks follow immediately one after the other. The *par* element defines that tasks should be executed in parallel. The *sel* element defines that tasks should be executed depending on the condition.

These temporal elements enable SML to provide high-level synchronization of the behavior of entities that result in a particular traffic situation. For example, as shown in Fig. 3.9, the task of the director tasks defined in seq temporal element orchestrates the two car entities behaviors by issuing necessary commands, so that an accident, e.g. the following car (AI Car B) hits the leading car (AI Car A), can be created.

Finally, we use a XML schema definition for SML scenario specification, to check the logical coherence of a SML script.

3.4 Conclusion

Here we wish to note that an empirical evaluation of a scripting tool such as SML is notoriously difficult, as related approaches are sparse and setting up test cases for comparison is unpractical. So, as preliminary empirical evidence on SML’s ease-of-use, it was important that our collaborators from the Smart Transport Research Centre at Queensland University of Technology actually used SML to specify the scenario.

Chapter 4

SML Framework - Overall Architecture

4.1 Introduction

In this chapter, we will explain the SML Framework, as illustrated in Fig. 4.1, by describing its main components. The SML Framework is a 3D experimental space for driving behavior studies.The framework is designed on top of the following technologies:

- The DiVE (Distributed Virtual Environment)framework [27] provides networking features for simultaneously connecting and synchronizing multiple users (i.e. a driving simulator) via Unity3D[50] clients.
- The OpenTraffic middleware integrates microscopic traffic simulation, road network, and users [35].
- Driving Simulator Client provides an interface for driving, was developed by tweaking a high-end, physically accurate and realistic car model created by Unity3D. Please see the Fig. 4.2 which shows the hardware of the desktop based driving simulator apparatus.

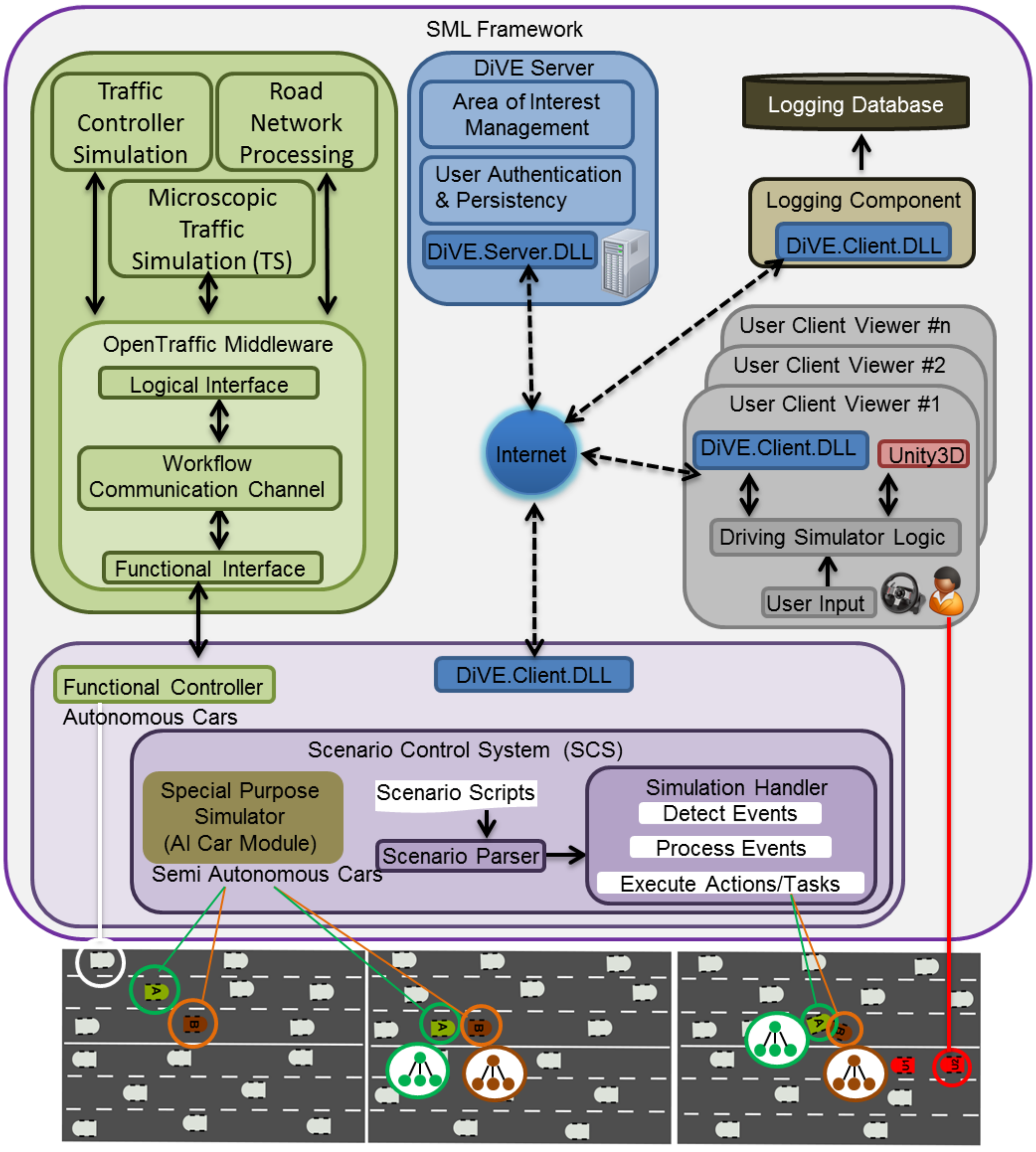


Figure 4.1: SML Framework Design Diagram

4.2 Simulation of Multiuser Immersive Driving in a Large Scale 3D Virtual Environment

4.2.1 Unity3D Virtual Environment

Unity3D [50] is a software platform that consists of an integrated tool for design and development of 3D real-time simulation content (can create and customize content through a programming and scripting interface). Unity3D also provides an engine (physics, graphics and sound) for executing the simulation applications. Additionally,

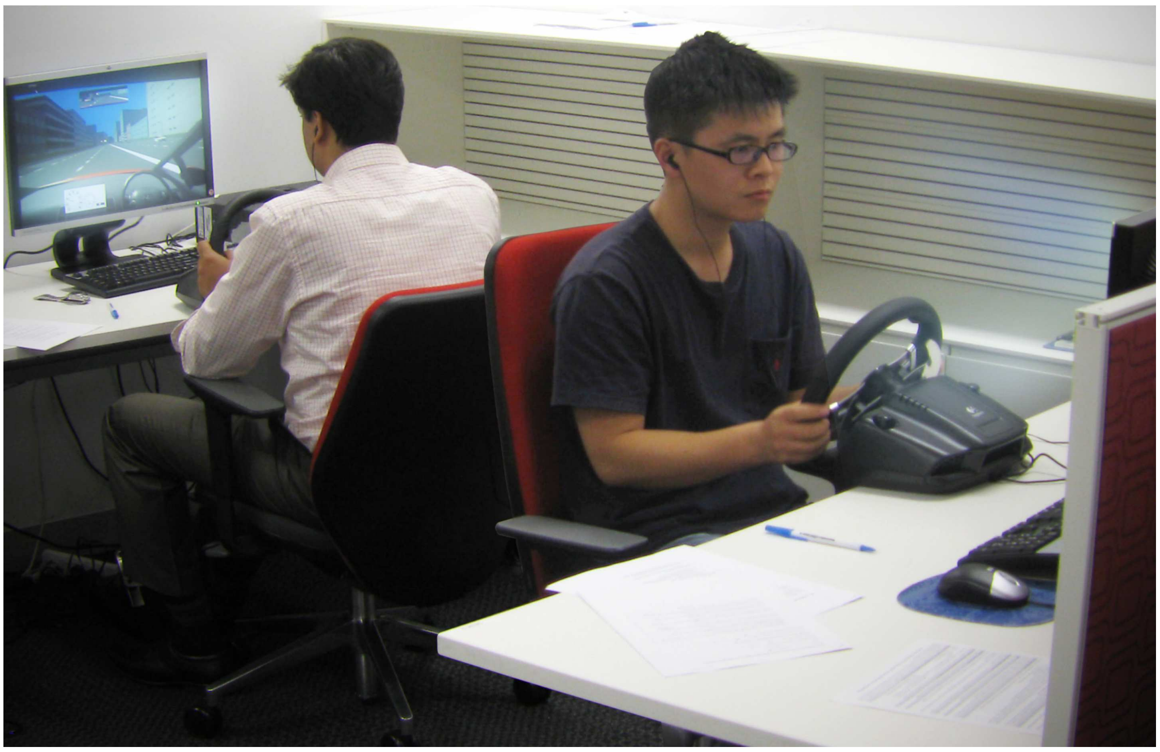


Figure 4.2: The driving simulator apparatus

Unity3D allows for an easy distribution of the driving simulation application as an executable in various platforms. Unity3D based client contains all necessary features, such as rendering, lighting, terrains, audio, physics and input control.

4.2.1.1 3D Model of Virtual Environment

Urban areas are probably one of the most challenging road networks to simulate in a virtual environment compared to the freeways and motorways. Most of the related driving simulator based experiments mainly focus on freeways and motorways because of their simplicity. However, we build a 3D virtual city environment, a model of the Chiyoda-ku urban area in Tokyo, as the virtual driving experiment space where many participants can drive simultaneously using their respective driving simulators.

To build 3D virtual city environment, we use CityEngine [44], which is a 3D Modeling application, specialized in the procedural generation of three dimensional urban environments such as cities and road networks. CityEngine helps us creating the de-

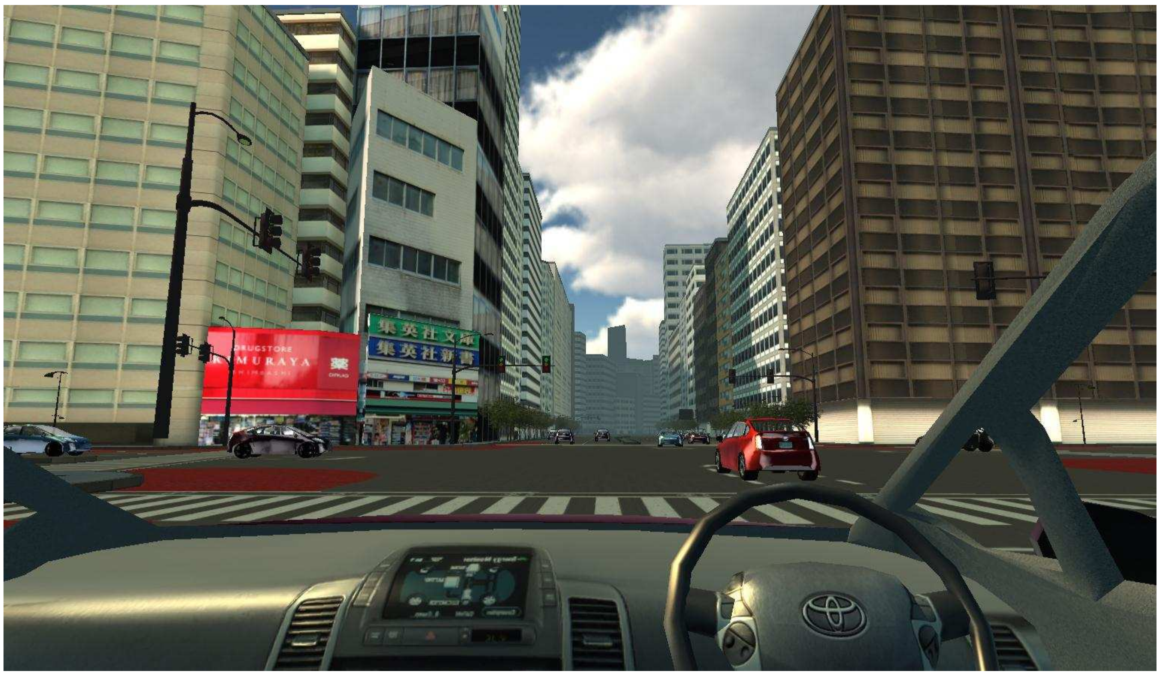


Figure 4.3: A first person or driver view of the 3D virtual environment

tailed large-scale 3D city model, which is then imported in the Unity3D environment for simulation and 3D visualization of the simulation space, including roads, road furniture, and buildings.

However, we needed to optimize the organization of the graphical environment (the meshes and textures), with the help of a graphic designer and later with an automated approach (e.g., scripts to combine meshes and texture).

4.2.2 Distributed Virtual Environments

We have developed DiVE (Distributed Virtual Environments), a new framework for massively multiuser networked 3D virtual worlds [27]. The framework provides the networking features for connecting multiple users to a shared virtual environment. DiVE was developed at the National Institute of Informatics and has already been used in several applications in the transport domain. DiVE currently supports around 100 simultaneous user drivers. DiVE consists of two main components: a client library and a server library.

- The client library provides the functionality for client applications (the driving simulator), e.g. to create and synchronize the car entities within the virtual environment, forward in-application events between individual clients or send remote-control commands to other components of the system.

- The server library receives operations from its clients and sends events back to them. Each client and its respective entities are represented on the server to keep a synchronized state among clients. To maintain that state, the server keeps track of entities positions and forwards changes to affected clients - as determined by clients individual areas of interest. Furthermore the server takes care of authentication and access control, application event propagation, etc.

The rendering of a large number of vehicles in the client computer puts a high demand on the graphics card. To overcome this problem, DiVE supports the concept of an interest area based on visibility, where each entity in the environment can have an interest area of an arbitrary size and the respective client will only receive updates about entities within that reach from the server. This means that the relevant data is delivered only to interested clients, where the client library uses them to update the entities properties of the client applications accordingly.

4.2.3 Multiuser Immersive Driving Simulator Client

The driving simulator client is based on Unity3D. The client viewer serves three functions:

- Immersive driving simulation, i.e. vehicles graphics, physics, and sound, as well as the interface to the input device a driving wheel and pedals,

- 3D visualization of the simulation space, including roads, road furniture, and buildings, and

- 3D visualization of ambient traffic (i.e. surrounding autonomous vehicles).

Fig. 4.1 shows how the multiuser immersive driving simulator fits in the SML Framework (as indicated by Client Viewer #1, #2 and so on.) in Unity3D. Fig. 4.3 illustrates the first person or driver view of the 3D virtual environment, which shows a user, driving a simulator vehicle, interacts with the simulated autonomous vehicles in a 3D virtual city environment. Fig. 4.2 shows the hardware of the desktop based driving simulator apparatus.

The setup of the driving simulator uses steering wheel and pedals set as input devices. Actions of the driver are picked up by the driving simulator logic component, which then renders the view of the driver depending on the setup of the driving station(i.e. simple PC in standalone or web player modes, multi-screen driving simulation setup or portable devices like iPad).

DiVE client component in the driver simulator would receive the updates of the surrounding computer vehicles from the DiVE server and render them in the driving simulator client. DiVE client module would send the updates of driving simulator vehicles information (e.g. positions, velocity, and rotations) to the DiVE server, which will distribute the information to all registered clients to allow for a real multiuser immersive driving experience. This would enable the traffic client (OpenTraffic middleware) to receive the updates of driving simulator vehicles information (see Fig. 4.1) so that autonomous vehicle can recognize user driven vehicles in the multiuser shared 3D virtual environment.

4.3 Integration of Multiuser Immersive Driving Simulation with Microscopic Traffic Simulator

This section explains the simulation of surrounding traffic in a large Scale 3D distributed virtual environment which connects multiple driving simulator clients simultaneously.

A core function of our framework is to integrate our immersive multiuser driving simulator with a microscopic traffic simulator. The role of the microscopic traffic simulator is to create ambient traffic and increase the realism of the driving environment. We opted to use a generic microscopic traffic simulator [33]. To ensure the smooth and flexible integration of different simulators and other functionality, we have developed the OpenTraffic Middleware or OTM [35]. The middleware was developed at the National Institute of Informatics, in collaboration with Queensland University of Technology, for the purpose of supporting this thesis work.

4.3.1 OpenTraffic Middleware

OTM is a common simulation platform that uses various modules to perform microscopic traffic simulation. The middleware supports two types of modules: logical and functional.

- Logical modules include road network processing, traffic simulator, traffic control and traffic evaluation.
- The functional control module supports the visualization of the traffic simulation. Logical and functional modules communicate with the middleware via interfaces. The modules can be developed independently and implement the corresponding OTM interfaces such as logical or functional

Before listing some examples of logical modules, we want to mention the Smart Transport Research Centre (STRC) road network standard, developed at Queensland University of Technology, which is an ontology-based approach to describe a road network in terms of its geographical and logical representation at many levels of details [34]. We use the STRC road network standard to encode the road network of our 3D virtual city environment. The STRC road network standard is also used to specify the required input parameters for the logical modules of OTM.

- Network processing: road network encoding of virtual 3D city, including nodes, links and lanes

- Traffic simulator: e.g. traffic demand
- Traffic control: e.g. traffic signal information, such as location and cycle time

The functional controller is responsible for interfacing with the logical modules of OTM. Its main functions are:

- Initialize the logical modules by providing the necessary inputs
- Tick the logical modules with the time elapsed since last update to receive new frames
- Parse the new frames from OTMs logical modules to visualize ambient traffic
- Inform the positional information of all user-driven vehicles to the logical modules of OTM

The functional controller parses each new frame from OTM and initializes or updates necessary entities (e.g. car entity and traffic control entity) in the DiVE virtual environment. This functionality is supported by the DiVE client library, which in turn sends relevant information of entities to the DiVE server library. Finally, the DiVE server library broadcasts the updates about entities to the clients. In this way, we can simulate ambient traffic with a traffic simulator in the context of a multiuser driving simulator.

In the other direction, at every tick, the functional controller informs the positional information of all user-driven vehicles to the traffic simulator via OTM messages. This enables the vehicles simulated by the traffic simulator to be aware of the presence of user-driven vehicles in the simulation space.

This two-way communication mechanism allows us to maintain synchronization between the information processed by OTMs logical modules (such as the traffic simulator) and the information processed by the functional controller (from the virtual world), and hence all users interacting with driving simulator clients can share the same simulated traffic scenario.

To the best of our knowledge, the integration of a general purpose traffic simulator with a massively multiuser driving simulation is unique among approaches aiming to collect human driving data, which only consider single-user driving simulators.

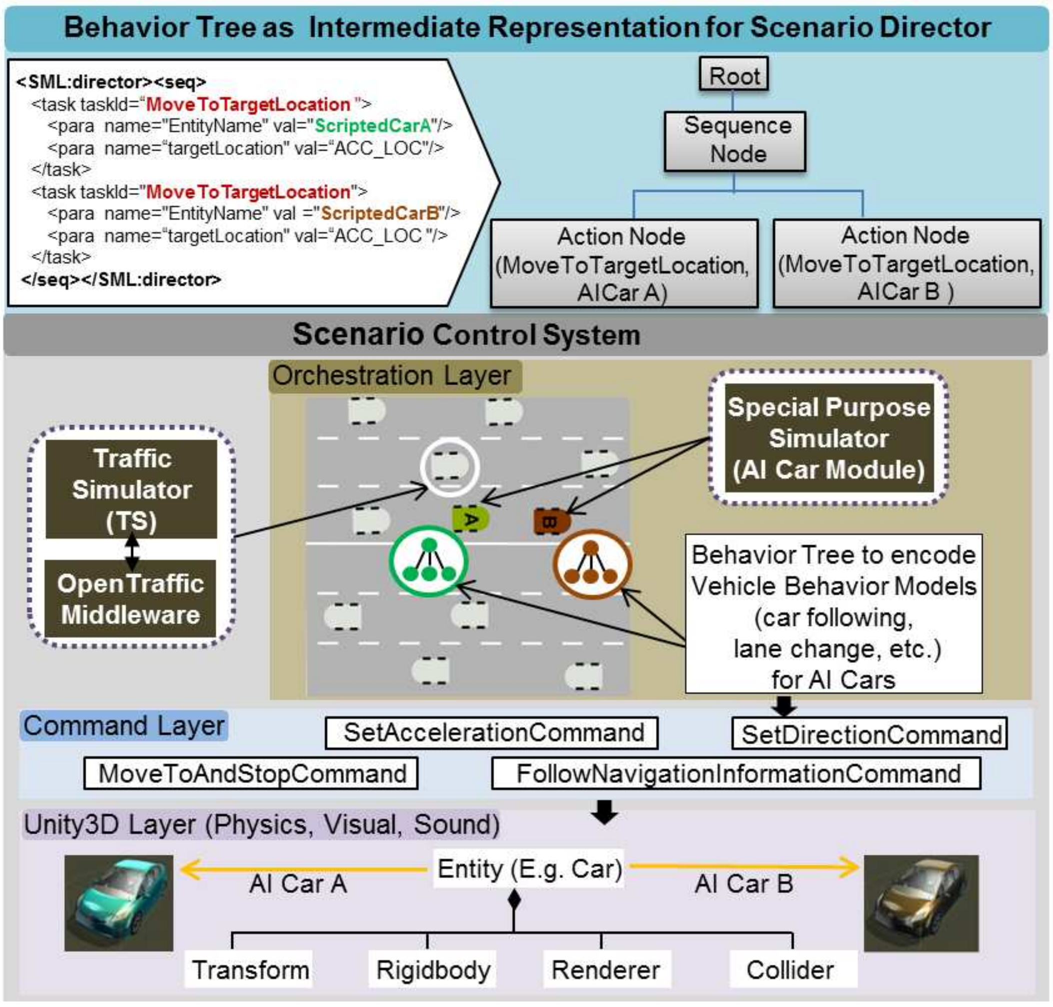
To summarize, the integration was facilitated by two of our in-house technologies: (a) our Distributed Virtual Environments (DiVE) technology supports multiuser driving, and (b) our OpenTraffic Middleware (OTM) allows us having traffic simulation as an independent module.

4.4 Scenario Implementation

Fig. 4.4 shows the conceptual diagram of the the Scenario Markup Language (SML) Framework which consist of three layers: 1) High level specification of scenarios in SML shown in Fig. 3.2 - Fig. 3.9 , 2) BT representation of SML Director and, 3) Scenario Control System (which in turn has three sub layers). The framework accepts SML scripts as input, generates a behavior tree (BT) as the intermediate representation for a scenario specification in SML, and unfolds the scenario in the 3D simulation space.

4.4.1 Behavior Tree as an Intermediate Representation

We use behavior trees [31] as an intermediate representation to handle the execution of the tasks of the 'director' component of a scenario script. For instance, the tasks of the 'director' component of a scenario (see Fig. 3.9) are represented by the behavior tree shown in Fig. 4.4. Tasks and temporal elements (e.g. seq, par, and sel) in the example script shown in Fig. 3.9 will become the main building blocks of a behavior tree shown in Fig. 4.4. The task elements will translate into action (leaf) nodes, which execute the task defined in it. The temporal elements of the 'director' component for


Behavior Tree as Intermediate Representation for Scenario Director
Root
Sequence Node
Action Node (MoveToTargetLocation, AICar A)
Action Node (MoveToTargetLocation, AICar B)
Scenario Control System
Orchestration Layer
Traffic Simulator (TS)
OpenTraffic Middleware
Special Purpose Simulator (AI Car Module)
Behavior Tree to encode Vehicle Behavior Models (car following, lane change, etc.) for AI Cars
Command Layer
SetAccelerationCommand
SetDirectionCommand
MoveToAndStopCommand
FollowNavigationInformationCommand
Unity3D Layer (Physics, Visual, Sound)
AI Car A
Entity (E.g. Car)
AI Car B
Transform
Rigidbody
Renderer
Collider

```
<SML:director><seq>
  <task taskId="MoveToTargetLocation ">
    <para name="EntityName" val="ScriptedCarA"/>
    <para name="targetLocation" val="ACC_LOC"/>
  </task>
  <task taskId="MoveToTargetLocation">
    <para name="EntityName" val ="ScriptedCarB"/>
    <para name="targetLocation" val="ACC_LOC "/>
  </task>
</seq></SML:director>
```

Figure 4.4: A conceptual diagram of the Scenario Markup Language (SML) Framework

a scenario will translate into the inner nodes of the behavior tree. The task elements or other temporal elements, which are embedded into other temporal elements, will be converted into a behavior tree with sub-trees while preserving hierarchy of behaviors. The BT approach fits well here, because director's tasks (mainly concerned with orchestrating the behaviors of entities) are embedded in the temporal elements. Hence they are composed as hierarchies of sub-trees in behavior trees.

4.4.2 Scenario Control System

One way to create specific traffic situations is to reuse the microscopic traffic simulator that populates all surrounding vehicles. In this approach, dedicated parameters of

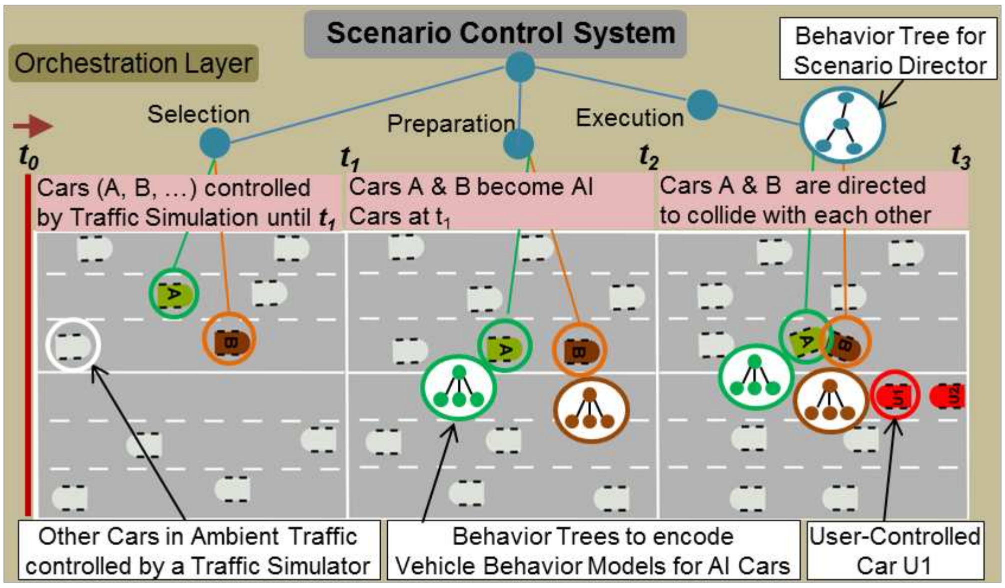

Scenario Control System
Orchestration Layer
Behavior Tree for Scenario Director
Selection
Preparation
Execution
t_0
t_1
t_2
t_3
Cars (A, B, ...) controlled by Traffic Simulation until t_1
Cars A & B become AI Cars at t_1
Cars A & B are directed to collide with each other
A
B
U1
U2
Other Cars in Ambient Traffic controlled by a Traffic Simulator
Behavior Trees to encode Vehicle Behavior Models for AI Cars
User-Controlled Car U1

Figure 4.5: The concept diagram that illustrates three functionality of the Scenario Control System: a) selection, b) preparation, and c) execution.

the driver behavioral models (underlying the traffic simulator) for some surrounding vehicles are manipulated to create traffic situations [37].

In our case, we devise a different method for creating traffic situations (e.g. an accident), because we use a general purpose microscopic traffic simulator [33] to simulate ambient traffic in a multi user networked environment. The traffic simulator does not support the required operations to create the traffic situations.

Instead, we decouple the scenario creation functionality from the traffic simulator that populates ambient traffic. In this way, we entangle the scenario creation from the time step that calculates behavioral update for all surrounding vehicles based on the ordinary traffic models.

Concretely, we introduce the Scenario Control System (SCS), which performs the following operations (see Fig. 4.5):

- The SCS monitors the ambient traffic, which is simulated by the traffic simulator, and analyzes the suitability of the vehicles to create a specific scenario.

- The SCS requests the traffic simulator to release the control of the selected number of candidate vehicles which are chosen based on the suitability criteria.

- The SCS assumes the control of the candidate vehicles and initializes their driving behavior with simplified traffic models and assigns them to a role to take part in a specific scenario, i.e. to reach some location as lead or following vehicle in the example presented in this work.

We refer to these selected computer vehicles, now controlled by SCS, as Artificial Intelligence (AI) Cars. In terms of simulation, these AI Cars are treated as any other vehicles. These AI Cars are directable semi-autonomous vehicles, which performs a normal autonomous activity in other circumstances, except the time when they are are commanded or directed from scenario control.

4.4.2.1 Orchestration Layer - AI Car Module

In the AI Car module, we use behavior trees to represent and implement the microscopic driving models for the AI Cars. Fig. 4.6 shows the conceptual diagram for the behavior tree (NavigationBT) representation of overall driving behavioral model, which has several sub-behavior trees embedded: BasicDrivingBT, MoveStopTarget-LocationBT, and LaneChangeBT.

The BasicDrivingBT sub-behavior tree is expanded with more detail as shown in Fig. 4.7. In the BasicDrivingBT, there are four parallel-running action nodes such as FollowWayPoints, FollowLeader, MaintainSpeed, and StopAtDesiredLocation.

FollowWayPoints refers to the lane tracking behavior represented in traditional traffic models. In our case, this behavior ensures that an AI Car navigates in the simulation space based on a navigation network. For that, we built a navigation network with additional information (e.g. extra way points) on top of the Smart Transport Research Center road (STRC) network standard[34].

FollowLeader, MaintainSpeed, StopAtDesiredLocation represents a typical car following, speed adapting and stopping behaviors, respectively, in traffic flow models.

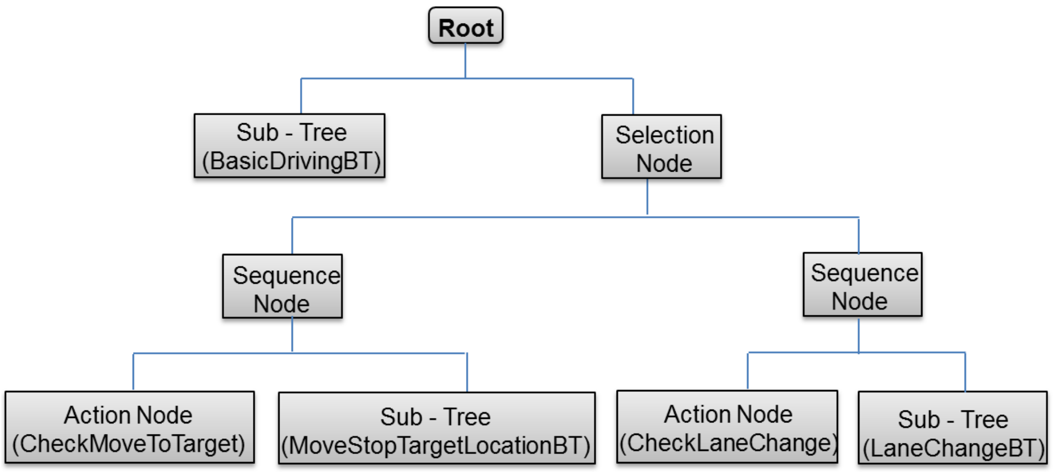

Root
Sub - Tree (BasicDrivingBT)
Selection Node
Sequence Node
Sequence Node
Action Node (CheckMoveToTarget)
Sub - Tree (MoveStopTargetLocationBT)
Action Node (CheckLaneChange)
Sub - Tree (LaneChangeBT)

Figure 4.6: The overall driving behavior (NavigationBT).

The functions of these behaviors are mainly concerned with the computation of the acceleration magnitude of an AI Car for the respective driving behavior such as car following or speed adaptation or stopping behavior.

The advantage of the behavior tree approach is that it provides modularity and flexibility to replace the underlying driving models such as car following, cruising, stopping, lane changing, by new models, if needed.

Further, all the driving behavior models are parameterized, which means that, when a car is initialized with these models, a set of appropriate values can be assigned to the parameters. For example, for the car following behavior, the parameters such as desired-speed, desired-acceleration can be assigned with the values taken from car entities properties as shown in Fig. 3.5.

4.4.2.2 Command Layer

The competing behaviors models such as car following, speed adapting, and stopping behavior (these behaviors runs in parallel) calculate acceleration magnitudes. MakeDrivingDecision (see Fig. 4.7) action node will choose one of acceleration magnitudes to apply to an AI Car.

As proposed by Cremer *et al.* in [10], we use the most conservative rule that choose the minimum acceleration produced by competing behaviors. Then the computed

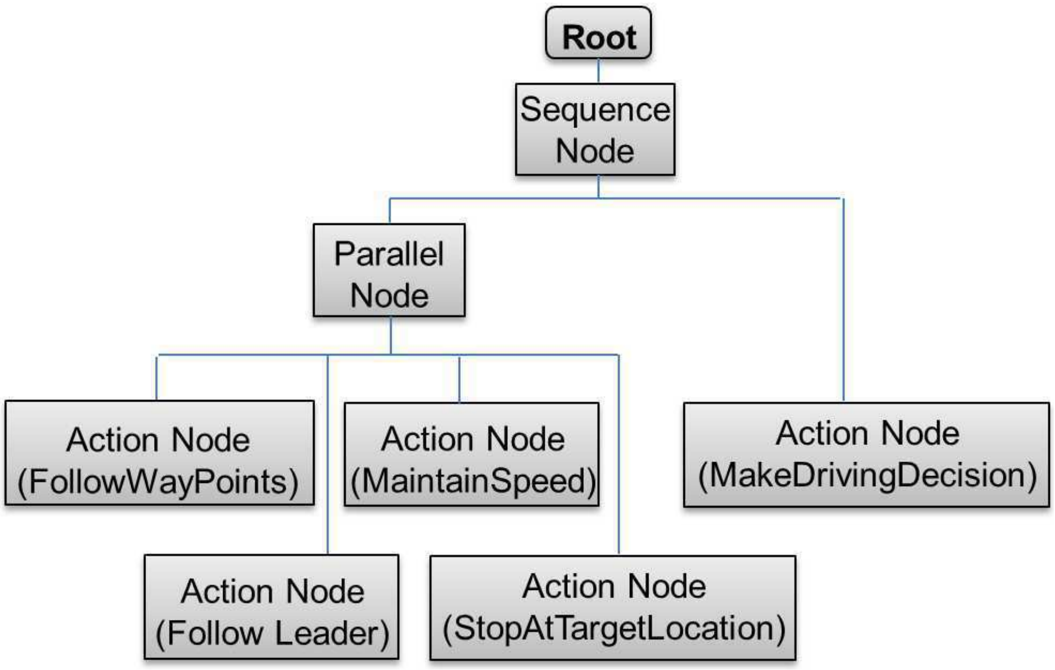

Root
Sequence Node
Parallel Node
Action Node (FollowWayPoints)
Action Node (MaintainSpeed)
Action Node (MakeDrivingDecision)
Action Node (Follow Leader)
Action Node (StopAtTargetLocation)

Figure 4.7: The behavior that represents basic driving behavior (BasicDrivingBT)

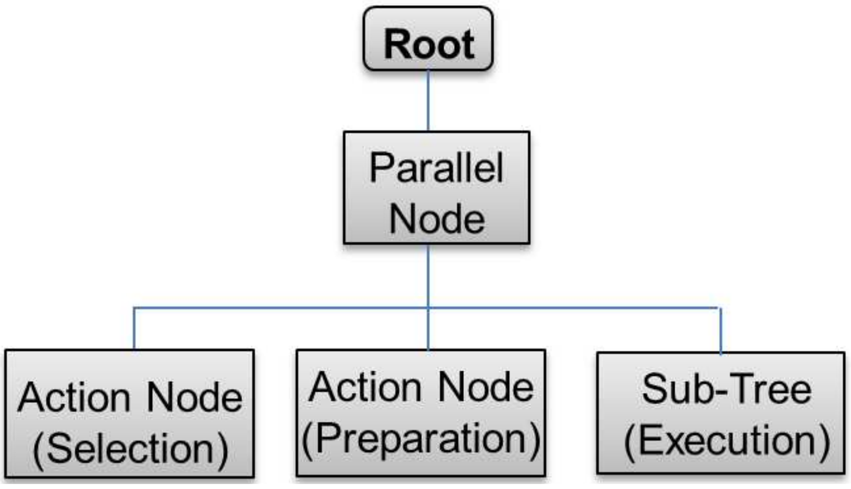


Figure 4.8: The behavior tree that represents the functions of Scenario Control System.

acceleration is sent to the command layer (e.g. SetAccelerationCommand) which in turn directs a steering behavior component that steers an AI Car in the simulation space (Unity3D layer in Fig. 4.4).

Further, the MakeDrivingDecision behavior node decides if an AI Car should: a) move-and-stop at a target location (MoveStopTargetLocationBT), or b) change lane (LaneChangeBT), as shown in the Fig. 4.6. When such a decision is made, execution of overall driving behavior represented by NavigationBT will switch from the branch that represents BasicDrivingBT to the respective branches that correspond to move-and-stop or lane change behaviors.

4.4.2.3 Unity3D Layer

We use Unity3D to simulate the semi autonomous vehicles or AI Cars (physics and graphics) and other autonomous vehicles in our 3D virtual environment, as illustrated in Fig. 4.4.

The Unity3D based environment motivated us to use 'Behave', which is a system for designing, integrating and running behavior logic using behavior trees for simulated characters in Unity3D simulation [4]. The Behave system is used to implement behavior trees based on driving behavior models for AI Cars at orchestration level.

Each behavior tree shown in Fig. 4.6 and Fig. 4.7 is designed using the Behave system. The advantage of using the Behave system is that it seamlessly integrates to our Unity3D based simulation and its runtime environment. The Unity3D runtime environment takes care of executing the driving behavior models represented as behavior trees by triggering its functions of temporal elements (e.g. parallel node) and its leaf nodes (e.g. action nodes of vehicle behavior models).

4.4.3 Incident Creation Functions

The Scenario Control System functions as a runtime environment to execute the scenarios that are specified in the SML script and represented as a behavior tree (see

Fig. 4.4). In the following, we describe the key functions of SCS. Behavior trees are used to implement the functions of SCS. Fig. 4.5 shows the conceptual execution diagram of the behavior tree that represents three functions of SCS: (a) selection, (b) preparation, and (c) execution.

Let us recall the requirement for our example scenario: the two AI Cars (one follows the other) are required to create an accident situation when the following-car collides with the leading-car from rear-end. The accident must happen at a particular location and a certain distance from the oncoming subject vehicle (leading user-driven car) so that lead-user can experience the accident (via visual and sound channels) which may possibly recreate the rubbernecking effect in the simulation space (as illustrated in Fig. 3.1). For this to happen, SCS needs to use its three functions as explained below.

4.4.3.1 Selection

The selection function identifies the required number of computer vehicles, e.g. two cars for creating the mentioned accident scenario within the time interval $t_0 - t_1$ shown in Fig. 4.5.

To find the most suitable vehicles from the currently available pool of oncoming computer vehicles simulated by the traffic simulation, we use the suitability criteria equation proposed by [37].

First, computer vehicles that currently run on the specified part of the road are queried. To evaluate how suitable a computer vehicle is for a scenario, we use the initial state specified for each entity involved in the scenario. Here the initial state refers to the 'scenarioroleloc' and 'desiredspeed' properties for entities involved in the scenario (see Fig. 3.5). The 'scenarioroleloc' property refers to (a) the initial location and (b) desired speed of the referred entity right before the 'director' component of a scenario is executed. Olstam *et al.*[37] specifies the initial position and speed as relative to the user-driven vehicle (e.g. 400 meters ahead and 15% slower). We made a different assumption regarding the initial state specified for entities involved in a scenario by assigning them absolute values for the initial position and speed. This is due to the reason that traffic engineers had the requirement to fix the location of accident at an absolute position instead of relative position from the lead user.

At time t_1, the selection function is completed and has chosen the best available candidate vehicles (AI Cars) to be handed over to the preparation phase. In addition, the selection function initializes the driving behavior models of the AI Cars and their roles in the scenario by using the properties specified in the entity definition, such as desiredspeed, desiredaccel and scenarioroleloc (see Fig. 3.5).

4.4.3.2 Preparation

The purpose of the preparation function is to ensure that the selected AI Cars reach their initial scenario-role positions and speed in time left for preparation (i.e. $t_1 - t_2$ in Fig. 4.5).

To successfully prepare the selected AI Cars to participate in a scenario, the AI Car module computes the acceleration that an AI Car needs to apply in order to reach its initial scenario role. The acceleration value is calculated based on the difference between the required average speed for an AI Car to reach its role and its current speed divided by the time left for preparation (i.e. $t_1 - t_2$ in Fig. 4.5). Here the average travel speed required for an AI Car to reach its initial scenario role is computed based on the leading user-driven car's average travel speed, the distance to the initial scenario role location and time left for preparation. To compute the leading user-driven car's average travel speed, we use the equation proposed by[37]. In using that equation, we set the estimated desired speed of the leading user-driven car to 60 km/h. Such was the speed that all users were instructed to maintain, throughout all experiments. However this assumption caused the problem to accurately reproduce the accident later as the subjects sometime did not drive at 60 km/h.

4.4.3.3 Execution

Once the AI Cars reach their initial scenario-role positions (specified as 'scenariorole-loc') in time, SCS will execute the tasks of the 'director' component of a scenario script (i.e. the action nodes of the behavior tree that represents SML Director as shown in Fig. 4.4) to create an accident at a target location and at a specific distance from the oncoming leading user-driven car (as illustrated in Fig. 5.2). This operation sends the command MoveToTargetLocation to the AI Car module with the parameters specifying the target accident location. The AI Car module tries to execute the command (at the command and Unity3D layers) on the respective AI Car so that an accident can be created.

The execution function of SCS and AI Car module must complete in time t_3 to successfully create a scenario as specified.

4.4.3.4 Time Estimation

The accident situation has to be reproduced from the lead users' point of view (the reference car) at a precise point in time. Therefore the time left for selection, preparation and execution (i.e. the time intervals $t_0 - t_1$, $t_1 - t_2$, and $t_2 - t_3$ in Fig. 4.5) functions of SCS have to be computed precisely. For that, we use the time estimation equation proposed in [37]. The time estimation consists of estimating the user-driven vehicle's average travel speed.

4.5 Collection of Driver Behavior and Traffic Data

Table. 4.1 summarizes the schema of the data stored from user driven vehicles and surrounding ambient traffic (autonomous and semi-autonomous vehicles). The framework logs (1) positions, velocities, and accelerations behavior, and (2) status of steering wheel, acceleration and brake pedals of the user-driven vehicles.

In addition, the headway and the reaction time-lag data can be calculated based on the velocity and location of each car in the simulation. This shows that data on interaction between user-driven vehicles can be obtained in our framework, which is important for validating traffic models. Note that this data is generally not directly observable from common traffic data.

Furthermore, based on the collected data and vehicle specific information (e.g. size), traffic data like lateral positions of leading and following vehicles on the longitudinal gap, vehicle composition in the traffic stream, lateral distribution of vehicles, lateral distribution of vehicles on a road of certain width and lateral gaps and longitudinal gaps can be derived easily. This data is useful for traffic engineers to analyze: (a) certain driver behaviors (a car tries to overtake a truck), (b) how narrow lanes affect the traffic stream and why, and (c) mixed traffic situations.

In addition, the following data can be logged in the framework: high level user-driven vehicle data like number of collisions, speed limit violation, traffic signs missed, pedestrians hit, centerline crossings, road edge excursions, off road accidents, travel time between two designated locations, headway distance between two specific user-driven vehicles, lane deviation, speed deviation, lane excursions, time to collision, fuel usage, and local speed at certain locations of the road network. Traffic engineers are interested in using these data to investigate various aspects of driver behavior like eco-friendliness, aggressiveness, and factors influencing lane selection.

Further, the framework can also log specific events(e.g. accident), which are useful for analysis of the experimental results.

It should be noted that, we would have a limitation regarding how much data we can log because currently the data to be logged need to be transmitted in the network because traffic simulation client, driving simulation clients and logging component are connected via DiVE framework, so that logging component can grab the data from the network.

Another important information is that how the timestamp logged relate to the

Attribute	*Type or Unit*	*Attribute*	*Type or Unit*
Vehicle Id	Integer	Timestamp	HH:MM:SS
Vehicle Type	String	Gas *	Double
Current Position	X:Y:Z (Double)	Gear *	Double
Origin#	X:Y:Z (Double)	RPM *	Double
Destination#	X:Y:Z (Double)	Distance Travelled	Double
Velocity	X:Y:Z (Double)	CO2 Emission	Double
Direction	X:Y:Z (Double)	Indicator Left	Boolean
Acceleration	X:Y:Z (Double)	Indicator Right	Boolean
Steering Wheel Position*	Float(-1 to 1)	Break Light	Boolean
Acceleration Pedal Position*	Float 0 to 1)	Road Link Id	Integer
Brake Pedal Position*	Float(0 to 1)	Lane Number	Integer

Table 4.1: Driver behavioral and microscopic vehicle traffic data schema (* refers to user driven vehicles only, # refers to autonomous and semi-autonomous vehicles only)

simulation time. The difference between two consecutive timestamps should be constant, because of constant sampling rate. However, currently the timestamp relates to the real global time of whole simulation system, and does not relate to the simulation time. The logging component tries to achieve the sampling rate of 100 ms (i.e 10 times per second, which is acceptable in traffic domain) at its best effort. if it grabs too much data to log, then it suffers to maintain its sampling rate constant (this is another reason we have to restrict the data we log). Therefore one can assume that the sampling rate is semi-constant.

4.6 Addressing Technical Challenges

In this section, we discuss the identified challenges and how we addressed them using our framework.

- One is to make the scenario authoring language practical to use. In that, first we expected to full fill traffic engineers requirement in capturing scenario specification. Then, we wanted to make the language to be powerful enough to specify different kind of traffic situations. To address this challenge, we proposed an XML-like language called SML with domain independent language structures to let traffic engineers to author scenarios. For that, we defined appropriate elements to the SML to capture traffic engineers requirement. We made SML as a high level specification language, by abstracting the complexity into scenario control system.

- The second challenge is to increase the scalability of simulation of ambient traffic for multiuser driving simulator to cover large scale 3D virtual environment. To address this challenge, we used a flexibly integrated microscopic traffic simulation, with less effort, on top of our in-house technologies such as DiVE framework and OpenTraffic middleware. That helped us to simulate ambient traffic for multiuser driving simulator in a large scale 3D virtual environment

  In addition, we separated the execution of scenarios from the simulation of ambient traffic to make it more controllable and flexible in terms of of creating and simulating different traffic situation as close as possible to what is specified by traffic engineers. Further, we used the tested algorithms from related work to create controllable and reproducible situations so that the same experimental conditions can be reconstructed (see "Incident Creation Functions" section).

  Our framework can simulate: a)autonomous vehicles as ambient traffic that serve the purpose of enhancing the realism, and b) directable semi autonomous vehicles to create scenarnios with sufficient reproducibility and controllability.

- Third challenge is the accessibility. Our Framework has the potential to be accessed over the (3D) Internet, thanks to our in-house technologies such as as DiVE framework and OpenTraffic middleware and immerisive driving in multiple platform (i.e. simple PC in standalone or web player modes, multi-screen driving simulation setup or portable devices like iPad)

- Fourth challenge is the multiuser immersive 3D environment. Compared to the related work which are limited to creating scenarios from single users point of view, since our framework allows multiple users to participate in driver experiments, there is a potential to create scenarios from multiple users point of view. That would give traffic engineers the in-depth understand of the inter-active behaviors of multiple drivers, i.e. how they react to each other and the behaviors of vehicle platoons in different conditions.

- Fifth challenge is the extensibility. With most of of related work, it is difficult to integrate and test with other microscopic traffic simulators. However, our framework allows flexible integration and testing of different microscopic traffic simulators.

4.7 Conclusion

Some approaches or ideas that were used to simulate ambient traffic in a multiuser driving simulator and implement scenario control system in this thesis, have been explored in other related studies individually but this work made a contribution on the integrated use of them.

The simulation approach described in this section, presented an integrated framework that provide the platform for simulating ambient traffic in a multiuser driving simulator that exist in a 3D virtual environment in which scenarios and situations can be formed around a multiple human subjects. In addition, the framework allows researchers to create experiences for multiple human subjects, with predictability, reproducibility and controllability in real-time, in the 3D virtual environment and capture the natural responses of human subjects which are later analyzed to validate relevant hypotheses about how drivers respond to some traffic situations of interest.

Chapter 5

Multiuser Driving Experiment

5.1 Introduction

Using the technologies presented in this thesis, we conducted an empirical study on the rubbernecking effect. For that purpose, we specified and implemented a scenario where a traffic accident happens on the opposite side of the road from the perspective of the subjects who were driving as a group of four drivers on a single lane.

Fig. 5.1 shows the simulation setup of the 3D virtual environment which include a) the locations where user start and finish driving, b) origin and destination from which traffic simulation cars spawned and sinked, c) location where scenario preparation starts, d)location where scenario execution starts, and e) possible accident location. Fig. 5.1 also illustrates that where a group of fours user driven cars would start driving and follow the route until reaching the finishing location.

Our study investigates the rubbernecking scenario by testing if our approach can reproduce certain important state variables regarding the accident location, such as the distance to the lead user car or the distance between the desired and actual accident location. We also analyze how drivers change their operational driving behavior at the incident site.

Precisely speaking, we investigate a special case of the rubbernecking scenario, where the accident just happened in the driver's vicinity. In the general rubbernecking scenario, the accident has already happened and the driver is approaching the accident

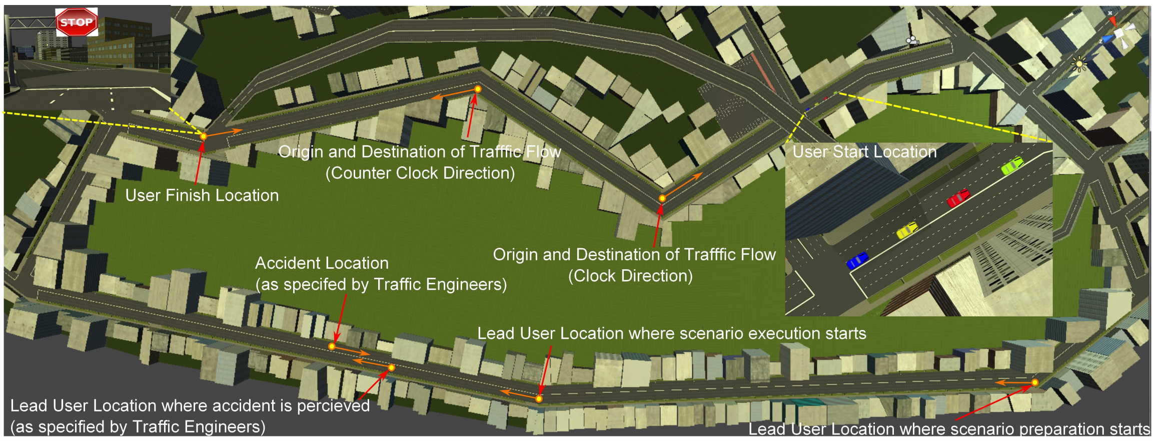

STOP
Origin and Destination of Trafffic Flow
(Counter Clock Direction)
User Start Location
User Finish Location
Accident Location
(as specifed by Traffic Engineers)
Origin and Destination of Trafffic Flow
(Clock Direction)
Lead User Location where scenario execution starts
Lead User Location where accident is percieved
(as specified by Traffic Engineers)
Lead User Location where scenario preparation starts

Figure 5.1: The birdview of 3D virtual environment setup in a 2D diagram.

location[43]. From the literature on the rubbernecking effect[30], we predicted:

1. The headway of subjects' vehicles increases after passing the accident location.
2. The subjects' speed decreases upon the perception of accident.
3. The subjects' speed variance (delta speed) reduces after passing the accident site.

The following sections describe our method and results.

5.2 Method

5.2.1 Subjects

Nine groups with four subjects each participated in the experiment. All subjects were either staff member or research student at the QUT. There were 5 females and 31 males, on average 29 years of age, and 7 years of driving experience. 19 had previous experience with driving simulators. The limited availability of subjects with previous experience with driving simulators led us to include subjects with no experience. However, we provided subjects sufficient time to practice in the driving simulator.

5.2.2 Apparatus

In the experiment, the driving simulation workstation is a DELL Precision m4600 with a 2.2 GHz Intel(R) Core (TM) i7-2720QM and 8 GB RAM, and a nVidia Quadro 1000M graphic processing unit. A Logitech G27 model game wheel was prepared for each subject as an input device to interact with the driving simulator. Subjects were given the Sony Stereo headphone to listen to the sound (e.g. car engine sound and ambient sound) created as part of the simulation. Subjects were seated in front of 21 inch monitors, at a distance of 40cm, and could not see each other's monitor. Subjects used the computers at the Smart Transport Research Center, which is located at QUT.

5.2.3 Experimental Procedure

We conducted the experiment in three days, in February 2012, in multiple sessions. All subjects were provided with an information sheet regarding the experiment they were to undergo and the related consent form to sign. The information sheet stated that the goal of the experiment was to investigate human driving behavior. Then, we introduced subjects to the driving simulation environment and let them drive for 5 minutes to get used to the simulation environment and practice their driving using the game wheel. After that, the official experiment runs started.

The experiment consisted of three conditions:

1. Drive Alone(DA): subjects driving in the environment without other traffic (for familiarization)

2. Drive With Traffic(DwT): subjects driving in the environment containing other traffic, in other lanes

3. 'Drive With Traffic DI' (DwT-DI)(Dynamic Incident): similar to (b); with the difference that the system created an accident scenario while subjects were driving

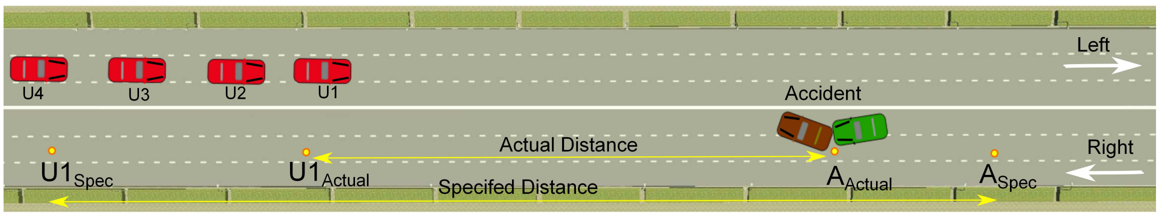


Figure 5.2: The traffic accident scenario concept in a 2D diagram. U1 is lead driver; U2, U3, and U4 are followers.

In each condition, four subjects drove together as a group (platoon) on a single lane, as illustrated by Fig. 12. Users were asked not to engage in any overtaking maneuver. In conditions(1) and (2), Traffic volume in terms of flow was set to 2000 vehicles/h/direction (per two lanes). A five minute rest period was included between trial runs.

5.3 Experimental Results

5.3.1 Informatics View - Accuracy of Accident Creation

In this thesis, we use $U1_{Spec}$ to denote the specified location of the lead user driver, when the accident happens. Similarly, A_{Spec} will indicate the specified location for the accident to happen (see Fig. 5.2).The symbols $U1_{Actual}$ and A_{Actual} denote the locations of lead user car and accident, respectively.

We postulate that our approach can reproduce the accident at a specified distance from the lead user car, that is, between $U1_{Spec}$ and A_{Spec}. Traffic engineers suggested such a distance to be 85m (i.e. identified as 'Specified Distance' in Fig. 5.2), with the tolerance of 20m.

Fig. 5.3 illustrates a line line graph which shows the distances between the location of the user-driven car at the time of the accident (noted as $U1_{Actual}$ in Fig. 5.2) and the location where an accident occurs (noted as A_{Actual} in Fig. 5.2) in each session. The distances shown are the actual distances in meters. The two vertical red lines

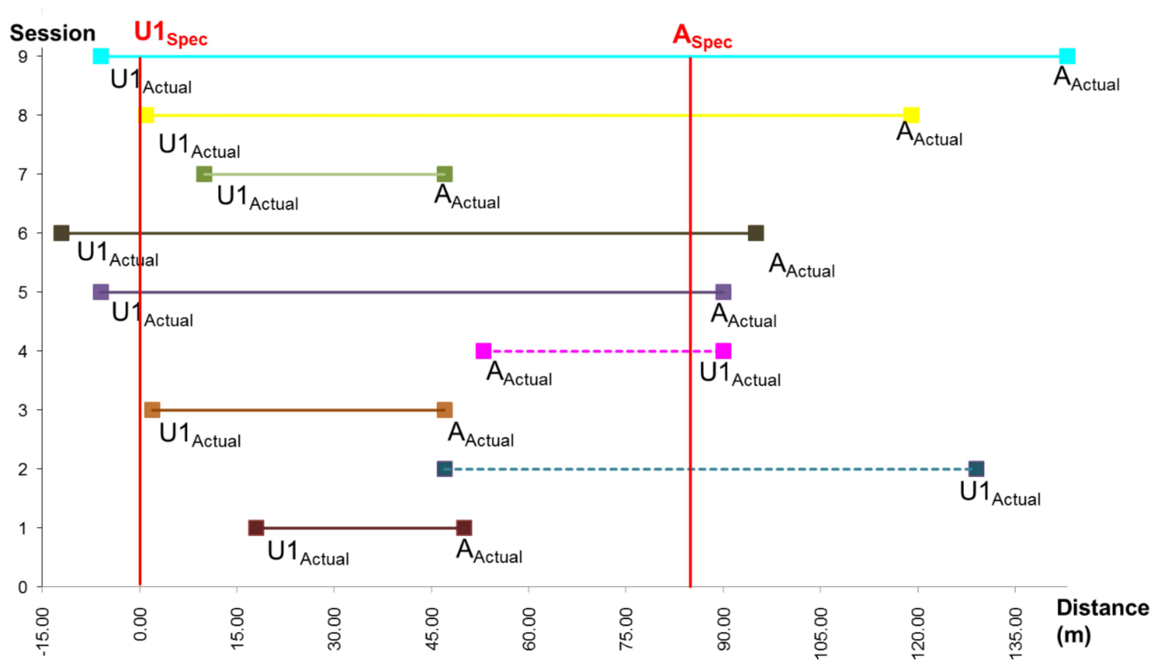


Figure 5.3: The line line graph showing the distances between the actual and specified locations of lead user and accident

compose the 85m window that represents the distance between the specified accident location and the specified location of the lead user-driven car, which is ‘Specified Distance’ in Fig. 5.2. Unsuccessful sessions are indicated by dashed lines. From the Fig. 5.3, we interpret that in seven, out of nine sessions, our approach was able to reproduce the accident scenario successfully. That is, the accident was generated before the first user drive passes the accident location.

In the cases of successful attempts (in Sessions 1, 3, 5, 6, and 7), the actual distance (identified as ‘Actual Distance’ in Fig. 5.2) between lead-user and accident location was close to the specified distance (85m) with the tolerance of 20m . In the remaining two sessions, actual distances were longer than the specified distance by approximately 50m (in Sessions 8 and 9).

As for the two unsuccessful sessions (Sessions 2 and 4 in Fig. 5.3), the system was still able to create the accident close to the specified location. However, the lead-user had already passed the location where the accident was created. The reason for this inaccuracy was that that a user desired speed of 60km/h was assumed, when

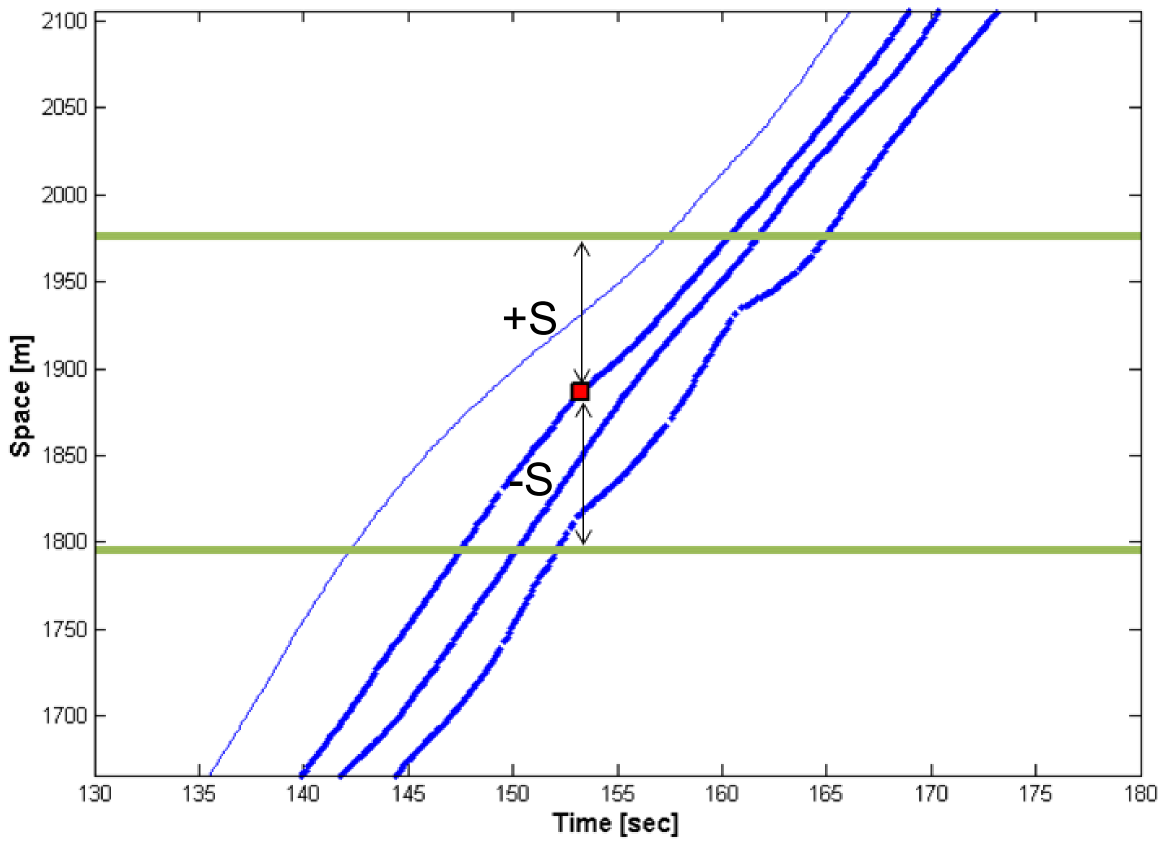


Figure 5.4: The space-time diagram for choosing observation window

calculating the average travel speed (hence time) of user driver as in [37] which was used in preparation and selection phases to decide the suitable AI Cars and to compute exact time for AI Cars to create the accident. However, the leaders in session 2 and 4 were observed to drive at 62 km/h on average, compared to the successful sessions where the leaders were driving 57 km/h on average during the preparation and selection phases.

5.3.2 Traffic Engineering View - Rubbernecking

Before describing the results, we explain the process of data analysis that was required to inform the hypotheses.

5.3.2.1 Data Analysis Process

Fig. 5.4 illustrates the pace-time diagram for choosing observation window. The thinnest line is the leading car; the red square is the point in time and space where the second car reaches the incident location; the green lines are the +/- 90 meter window, spanning the space domain of course, centered at the red square. As shown in Fig. 5.4, we only analyzed the data falling within an observation (space) window. Anything outside this region was assumed to carry irrelevant information, insofar as the rubbernecking phenomenon was concerned. We therefore used space as an independent variable and distance-to-incident as a reference point The space widows had size 2*S (meters), ranging from -S to +S. Here, S denotes the distance from vehicle-to incident location, along the direction of travel. Negative values indicate that the incident location had yet to be reached.

We treated headway, speed and delta speed as dependent variables and obtain them from the data (position and time) falling within the observation window. These data from nine sessions were aggregated and analyzed as a whole. In other words, the data from all nine experiment sessions was clustered into three groups; these being 'Drive Alone', 'Drive With Traffic', 'Drive With Traffic DI'. Then, for each group of data, we performed the following data processing:

1. Cluster the data into n space bins

2. Fit some, parametric, unimodal distribution (e.g. Gaussian or Gamma) that best describes the shape of the binned data

3. Compute the mode (i.e. the maximum of the distribution) and variance for each distribution for each space bin, in order to characterize the behavior of that bin

Step (1) enabled us to study the behavior of drivers at various locations of the road, before approaching the accident location. In step (2), the use of parametric descriptions allowed a concise general description of the data. The disadvantage

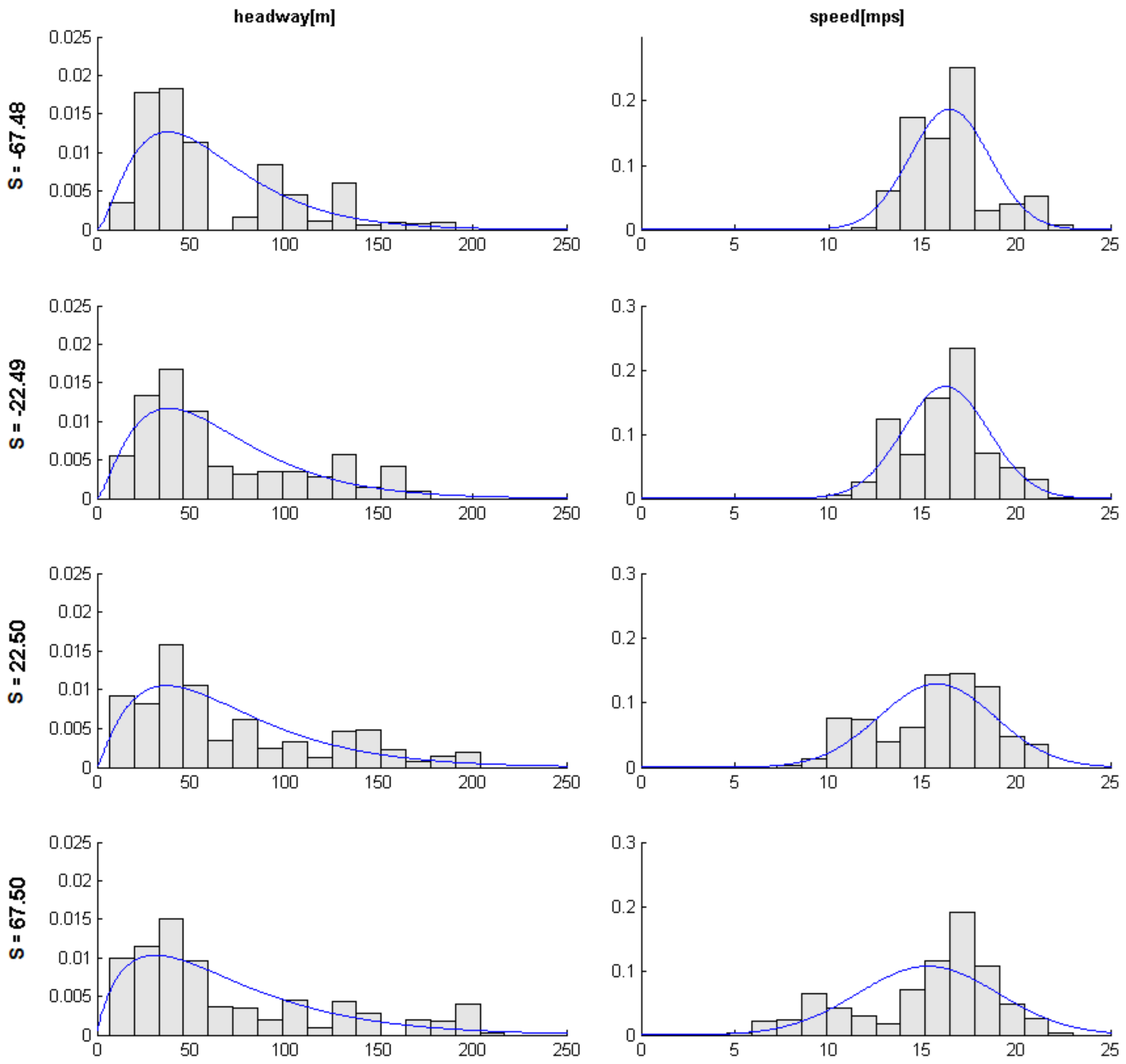


Figure 5.5: The plots for behavioural distributions of headway and speed. The graphs refers to the 'Drive With Traffic' runs from all nine sessions.

of parametric versus non-parametric distributions (e.g. histograms) is that in the former, the shape of the data is assumed to be known. By contrast, non-parametric approaches make no assumption on the dataset, although they are typically less robust to outliers in the case of small datasets and, by nature, cannot generalize[1]. Fig. 5.5 shows the difference between parametric (thin curve) and non-parametric (light grey histogram) distributions. Finally, step (3) was performed in order to determine the representative values for headway, speed and delta speed; and the deviation from the

[1]In Pattern Recognition, the term generalization denotes the capability of a model to make accurate inference upon new data.

overall behavior from all drivers. Representative headways, speeds and delta speeds were given by the mode of the distribution. The standard deviation denoted the variance from the overall trend.

As far as concerns the distributions, choosing the best model was achieved through Maximum Likelihood Estimation (MLE), that is, by choosing the density function whose fitting would produce the highest likelihood. The functions investigated were Gaussian, Gamma and Extreme Value. It turned out that headway was best described by a Gamma function, whereas speed and delta speed were both normal-shaped (Fig. 5.5).

In order to draw the mode and standard deviation (SD) profile for the space window considered, we linearly interpolated mode and SD points from each space bin, as shown in Fig. 5.6 and in Fig. 5.7.

The fourth row of graphs in Fig. 5.6 and Fig. 5.7 represent the difference between the behavior (mode and variance) between accident scenario (i.e. 'Drive With Traffic DI') and baseline (i.e. 'Drive With Traffic'). We now turn to the discussion of our results.

5.3.2.2 Hypotheses and Results

The first hypothesis states that the headway of subjects' vehicles increases after passing the accident location. In our experiment, however, we observed that the mode headway decreased gradually upon the perception of the incident. At the end of the observation window (90m after the incident), the speed was reduced by almost 2mps (7.2kph) and the headway had dropped by 10 meters.

The second hypothesis predicts that subjects' speed decreases upon the perception of the accident. The mode plot graphs in Fig. 5.6 show the mode speed decreased gradually upon the perception of the incident. This result supports the hypothesis of speed reduction as a consequence of the rubbernecking effect. This result is in agreement with what has been observed from real freeway rubbernecking data.

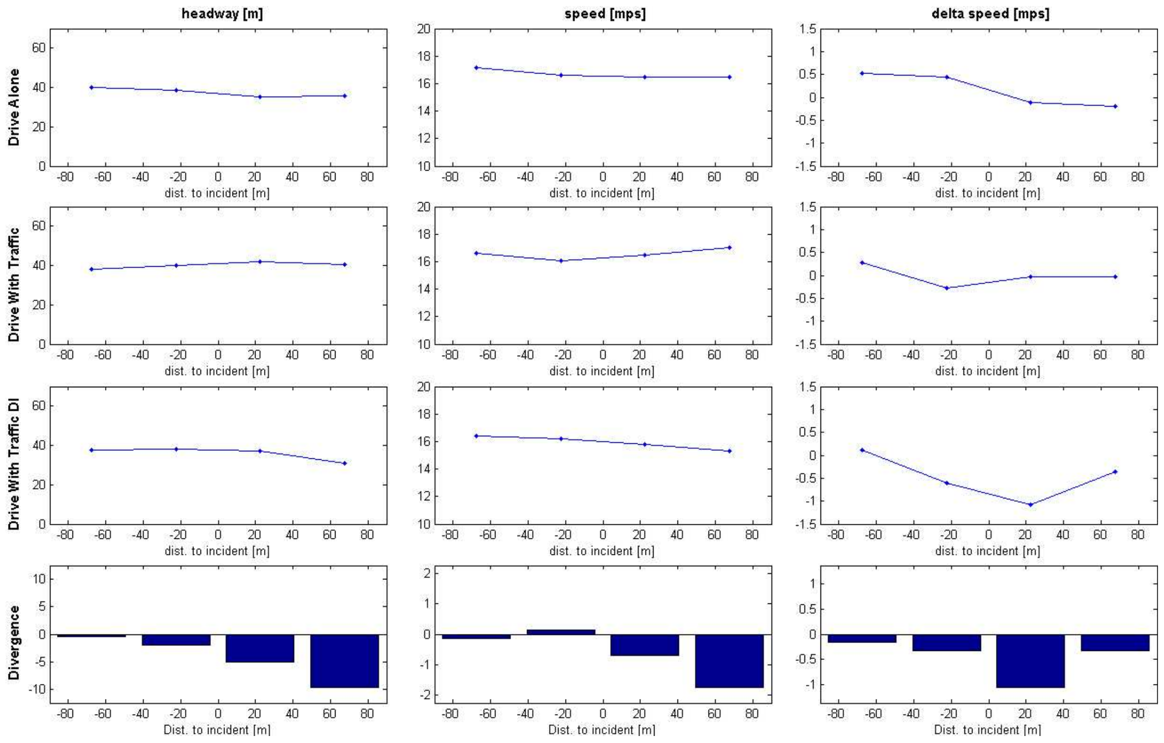


Figure 5.6: The mode plots for headway, speed, and delta speed of overall driving behavior of four drivers in a platoon from all nine sessions.

The third hypothesis claims that the difference between the speed of adjacent vehicles (delta speed) deceases after passing the accident site. Fig. 5.6 shows that at about 20 meters after the incident the platoon underwent a delta-speed reduction of about 1 mps (3.6kph).

Overall, the variance plot graphs in Fig. 5.7 suggest that drivers experienced difficulties in maintaining some desired headway and speed after perceiving the accident. A large, positive divergence value indicates that the headway and speed variance of the accident scenario (i.e. 'Drive With Traffic DI') is bigger than the one of the baseline (i.e. 'Drive With Traffic'). In other words, the sight of the accident made the subjects uncertain about the headway and speed they wanted to maintain.

Further we investigated if the change of headway and speed is statistically significant among the three conditions: 'DA', 'DwT', 'DwT–DI'. For this, we performed One-Way Multivariate Analysis of Variance (MANOVA) tests among three conditions (or groups). The dependent variables were the headway, and speed, of the

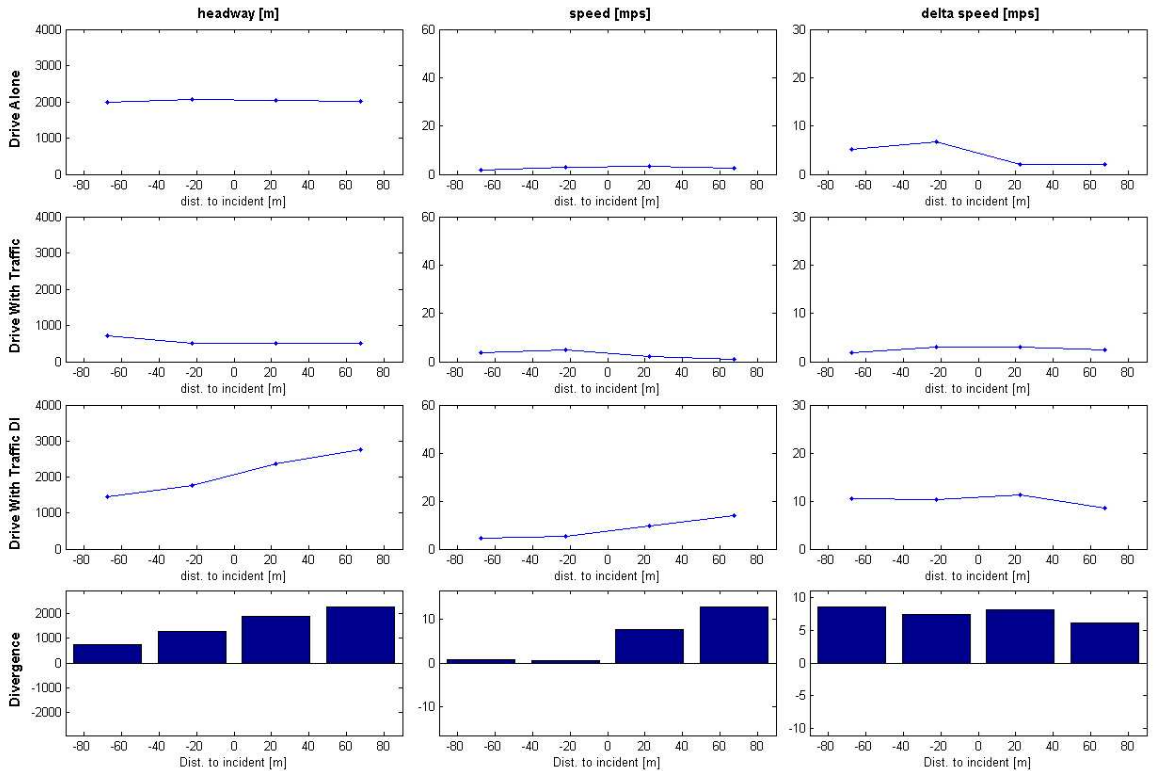


Figure 5.7: The variance plots for headway, speed, and delta speed of overall driving behavior of four drivers in a platoon from nine sessions.

vehicles at the four space bins, around the incident location. The MANOVA tested if the 4D means or groups centroids, actually formed one, two or three different groups, according to the Mahalanobis distance between groups. The MANOVA test on the speed data, enabled us to reject both the one group, and two-group hypotheses ($p_1 < 0.001, p_2 < 0.001$). By contrast, the MANOVA test on the headway data, showed ($p_1 < 0.0001, p_2 = 0.1560$) that the centroid are best described by two groups only. As the Mahalanobis distance between 'DwT–DI' and the other two groups is bigger than the distance between 'DA' and 'DwT', we concluded that 'DA' and 'DwT' are similar, but both statistically different from 'DwT–DI'. On average, 14% of the data points per session were cleansed for which headway values greater than 150m as the study condition (driving as platoon) was not met. Moreover, we concede that randomization of the three conditions helps to avoid any carry over effects. However, in our particular case, we chose not to randomize the three conditions, based on the assumption that having seen the accident would significantly affect users driving in the other condition (DwT).

As for the delta speed, the standard deviation from the accident scenario was always bigger than the one from the baseline, throughout the observation window. This result could be attributed to the small size of the dataset, among other things. Just like acceleration versus velocity and position, the delta speed between vehicles is expected to vary more rapidly than speed and headway. If the dataset is too small and the data has high variance, it may be hard to guess what the characteristic behavior of the target phenomenon is, regardless of the model or distribution that is used.

5.4 Conclusion

Please note that the introduction of multiple drivers in a driving simulator experiment brought us more challenges : a) how to design and conduct driver behavior experiments which involves several subjects and investigating interactive driving behavior of multiple drivers in case of exceptional events, and b) how data from such an experiment should be analyzed. This is because, when adding more subjects in an experiment like the one reported, we tend to lose some statistical power to analyze the data in a meaningful way.

To demonstrate the effectiveness of our framework and to address the challenges mentioned, we collaborated with traffic engineers to design and conduct a driver behavior study focusing on the rubbernecking phenomenon.

In this chapter, we presented a) experimental design of a multiuser driver behavior study in a 3D virtual environment, b) the data analysis process, and c) the experimental results from both the perspectives of informatics and traffic engineering.

Chapter 6

Conclusion

6.1 Summary

In this thesis, we have presented a new framework for conducting controlled driving behavior studies in a multiuser networked 3D virtual environment, which supports (a) the simulation of multiuser immersive driving, (b) the visualization of interactive surrounding traffic, (c) the specification and creation of traffic scenarios, and (d) the collection of meaningful driving behavior data.

Using our framework, we address the gap between the specification of scenarios in a high level language (by experts) and the technical implementation of the specification of scenarios with simulators (by developers). Specifically, we propose (a) the Scenario Markup Language as a high level specification language, (b) a technique to map high level descriptions to the scenario control implementation based on behavior trees, and (c) the Scenario Control System that uses an external traffic simulation, which is integrated with a multiuser driving simulator to implement scenarios.

Our technique to create reproducible traffic situations allows both autonomous and directable semi-autonomous vehicles to co-exist in the simulation environment. This enables us to simulate realistic ambient traffic while being able to create tightly controlled traffic situations. Thus the framework empowers traffic engineers to investigate complex traffic situations that depend on the interaction between multiple drivers. Specifically, we used our framework to investigate the 'rubbernecking' phenomenon.

From an informatics view, our study shows promising results (i.e. accident scenario created successfully 78% in the empirical study) in terms of reconstructing a reproducible traffic accident scenario in a multiuser driving simulation context in multiple experiment sessions. This allows us to compare data and draw valid conclusions regarding how a traffic accident scenario (perceived by drivers as it happens) affects the car-following behavior (headway, speed, delta speed) of four users driving as a platoon in a single lane, at the close proximity of the incident site. We reckon the source of inaccuracy is from the assumption regarding user-driven vehicles desired speed which caused the problem to accurately reproduce the accident in few occasions as the subjects sometimes did not drive at 60 km/h and the time duration for selection and preparation phases of scenario vehicles were short.

Further, we recall that the reproducibility of scenarios (i.e. state variables) in the simulation depends on the precise calculation of timing of (a) starting and completing the selection of suitable candidate vehicles and assigning the scenario roles to the selected vehicles (selection phase), (b) transporting the selected vehicles to the respective scenarios role location (preparation phase), and (c) executing the scenario directors commands, i.e. when to issue the commands to the selected vehicles (execution phase). Further, the time calculation takes in account the lead drivers driver speed as key factor, because scenarios have to be created so that lead user can perceive it in time. Therefore, to improve our framework in its ability to create reproducibility of scenarios, we identified two approaches: (a) a more precise calculation of timing by computing the lead users desired speed as the speed averaged over a window of time, and (b) an increment of the time duration for selection and preparation phases of scenario vehicles.

From a traffic engineering view, the study supports two of the three hypotheses regarding the rubbernecking effect. As predicted, the speed and the delta speed of the subjects decreases upon the perception of accident. However, it does not support the hypothesis that the headway increases. Although there is evidence that the perception of an accident does change the behavior of all drivers, further research will need to be conducted to understand the headway drop phenomena at the accident location. In general, our framework was able to measure the effect of perceiving the accident by a group of drivers, on headway, speed and delta speed variance.

6.2 Limitations

In regarding improvement in the simulation, observations made during the experiments indicate the need for smaller adjustments for ambient vehicles behavior at the intersection as well as curves. Further, we only simulated cars for the microscopic traffic simulation and subjects, participated in the experiment, commented that there needs to be different types of vehicles such as buses, trucks, cycles, and pedestrians as well.

We used a finite automata based microscopic simulation of surrounding traffic (i.e. autonomous vehicles). It increases the realism of the ambient environment in driving simulator. However, it also increases the variation in experimental conditions between multiple sessions, making it difficult to compare results because of the increased statistical variation. To solve this issue, we could replace the current microscopic traffic simulator with the agent-based simulation of ambient vehicles. In this way, we believe that we can avoid variations in experimental conditions between multiple sessions because the focus of agent-based simulation is on modeling the decision making processes and detailed motion control of autonomous vehicles [57].

Driving simulator client lacks certain features that are essential for any vehicles. For example, current implementation of driving simulator interface does not have side mirrors, which led us to restrict the humans drivers not to overtake or change lane during the experiment reported here. Hence, we also needed to restrict the experiment to focus only on the aspect of longitudinal driving behavior (or car following), avoiding any lateral driving behavior which involves lane changing or overtaking. The side mirrors have to implemented, so that we can extend the experiments that can investigate not only the longitudinal driving behavior, but also lateral one.

For the driving simulator experiment conducted, we used a simplified traffic accident scenario to test if our approach can reconstruct the situation. In improving this scenario creation, we would consider relaxing the requirement on the fixed accident location by specifying it relative to the lead user vehicle. In this way, we hope to increase the accuracy of reproducing the accident successfully.

6.3 Implications for Research and Policy

The simulated framework presented in this thesis offers researchers and policy makers a low-cost, virtual experiment station, that allows studying the effect of various exceptional events and traffic scenarios on the travel and driver behavior of multiple human drivers in a safe and controlled environment. Generation and simulation of surrounding traffic for multiuser driving simulators increases the realism of the simulated environment that resembles driving in real traffic systems as closely as possible.

Researchers are enabled to test new strategies,methods, and algorithms to address various transportation issues, while policy makers can analyze the impacts of those prior to application and real-life trials.

Additionally, every experiment with the system provides feedback data to the simulation component for calibration, so to provide an even more realistic environment. Specifically, the framework creates opportunities for developing new traffic simulation models or enhancing existing models. All data (e.g. vehicle trajectory data) concerning the driving simulator vehicle as well as the ambient vehicles, are logged. The collected data can be analyzed to improve or calibrate the the traffic flow models such as car-following, lane-changing, and overtaking behavior by incoperating the additional knowledge gained regarding the human behavior changes due to the impact of an exceptional event or traffic scenarios.

Further, the integration of a multiuser driving simulator and a microscopic traffic simulator, along with scenario control can lead to the development of new methods for validation of existing traffic simulation models.

6.4 Future Work

Future work will address several limitations of our framework. Our current scenario example only uses two car entities to create scenarios; there can be other cases which involve multiple different entities and more interactions among them (e.g. more cars and traffic signals). For example, an accident situation involving many cars at an intersection controlled by a traffic signal can be more difficult in terms of preparing the entities to reproduce the scenario accurately and realistically.

Another example would be to simulate different geographical locations of a city which has complex geometry. Since we have integrated tools for automatic generation of 3D virtual cities along with road infrastructure and furniture, it would be feasible for us to generate more cities and highways easily, rather than have to design them manually, so that we can conduct experiments in different geographical locations to investigate various traffic situations that are influenced by the geometrical structures of the locations.

In addition, currently we proposed an XML like language for scenario authoring. In future, we consider developing a graphical tool for scenario authoring and make authoring more intuitive for non-programmers.

Further, more studies need to be conducted that focus on more complex transport scenarios (e.g. high density of traffic and changes in road geometry will increase the complexity of the preparation of the entities involved in the scenario) to draw more solid conclusions about our simulation framework. Future experiments can also focus on the simulation of different weather and road conditions as part of the test case scenarios, with the involvement of multiple users taking part.

Chapter 7

Publications

Some parts of this thesis have been published in other publications.

Journal Papers as the First Author

1. Kugamoorthy Gajananan, Alfredo Nantes, Marc Miska, Arturo Nakasone, Helmut Prendinger, An Experimental Space for Conducting Driving Behavior Studies based on a Multiuser Networked 3D Virtual Environment and the Scenario Markup Language, IEEE Transactions on Human-Machine Systems, 2013. (In press.) The paper was recommended for publication in the former IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans (2011 Impact Factor: 2.123)

Journal Papers as a Co-Author

1. Helmut Prendinger, Marc Miska, Kugamoorthy Gajananan, and Alfredo Nantes, A Cyber-Physical System Simulator for Risk Free Transport Studies, Journal of Computer-Aided Civil and Infrastructure Engineering, 2013.(Under Submission)

2. Helmut Prendinger, Kugamoorthy Gajananan, Ahmed Bayoumy, Ahmed Fares, Reinaert Molenaar, Daniel Urbano, Hans van Lint and Walid Gomaa Tokyo

Virtual Living Lab: Designing Smart Cities based on the 3D Internet, IEEE Internet Computing, Special Issue on Smart Cities, 2013.(Under 2nd Review)

Refereed Conference Papers as the First Author

The first version of the Scenario Markup Language was originally presented in :

1. Kugamoorthy Gajananan, Arturo Nakasone, Helmut Prendinger, and Marc Miska Scenario markup language for authoring behavioral driver studies in 3D virtual worlds, Proc IEEE Symp on Visual Languages and Human-Centric Computing (VL/HCC'11), Pittsburgh, PA, USA, 2011.9, pp 43-46.

2. Kugamoorthy Gajananan, Arturo Nakasone, Edgar Santos, Helmut Prendinger, and Marc Miska Scenario authoring for driver behavioral data collection in 3D virtual environments, Proc 2011 IEEE Int'l Conf on Virtual Environments, Human-Computer Interfaces and Measurement Systems (VECIMS'11), Ottawa, Ontario, Canada, 2011.9, pp 1 - 5

3. Kugamoorthy Gajananan, Arturo Nakasone, Helmut Prendinger, Eurico Doirado, Pedro Cuba, Marc Miska. Creating Interactive Driver Experiences with the Scenario Markup Language, Proc 8th International Conference on Advances in Computer Entertainment Technolgoy (ACE'11), Lisbon, Portugal, 2011.11, Article No. 41.

4. Kugamoorthy Gajananan, Sra Sontisirikit, Jianyue Zhang, Marc Miska, Edward Chung, Eward Guha, and Helmut Prendinger. A Cooperative ITS study on green light optimization using an integrated traffic, driving, and communication simulator. Proc 36th Australasian Transport Research Forum (ATRF'13), Brisbane, Australia, 2013.10. (Abstract accepted)

Refereed Conference Papers as a Co-Author

The standard to represent the road network in 2D and 3D virtual environment, which is used as a base/input for the Microscopic Traffic Simulator is presented in the following paper. I contributed to the paper by helped Dr. Marc Miska in developing a new road network encoding standard, in terms of researching the related work, finding the limitations in them, identifying the new requirements from the target projects where standard to be used.

1. Marc Miska, Kugamoorthy Gajananan, Eduard Chung, and Helmut Prendinger. A traffic simulation standard based on data marts, Electronic Proc 34th Australasian Transport Research Forum(ATRF'11), Adelaide, South Australia, 2011.9.

Bibliography

[1] Olivier Alloyer, Esmail Bonakdarian, James Cremer, Joseph Kearney, and Peter Willemsen. Embedding scenarios in ambient traffic. In *In Driving Simulation Conference*, pages 75–84, 1997.

[2] Yoram Atir and David Harel. Using lscs for scenario authoring in tactical simulators. In *Proceedings of the 2007 summer computer simulation conference*, SCSC, pages 437–442, San Diego, CA, USA, 2007. Society for Computer Simulation International.

[3] Jaime Barcel and Jordi Casas. Dynamic network simulation with aimsun. In Ryuichi Kitamura and Maso Kuwahara, editors, *Simulation Approaches in Transportation Analysis*, volume 31 of *Operations Research/Computer Science Interfaces Series*, pages 57–98. Springer US, 2005.

[4] Behave, August 2011.

[5] E. Blana and University of Leeds. Institute for Transport Studies. *Driving Simulator Validation Studies: A Literature Review.* ITS working paper. Institute for Transport Studies, University of Leeds, 1996.

[6] E. Blana and University of Leeds. Institute for Transport Studies. *A Survey of Driving Research Simulators Around the World.* Working paper (University of Leeds. Institute for Transport Studies). Institute for Transport Studies, University of Leeds, 1996.

[7] Esmail Bonakdarian, James Cremer, Joseph Kearney, and Pete Willemsen. Generation of ambient traffic for real-time driving simulation. In *In Image Conference*, pages 123–133, 1998.

[8] Torrieri V. Ciuffo B., Punzo V. Integrated environment of driving and traffic simulation. In *Proceedings of Road Safety and Simulation International Conference.*, pages 7–9, November 2007.

[9] M.J. Conway. *Alice: Easy-to-learn 3D Scripting for Novices.* University of Virginia, 1998.

[10] James Cremer, Joseph Kearney, and Yiannis Papelis. Hcsm: a framework for behavior and scenario control in virtual environments. *ACM Trans. Model. Comput. Simul.*, 5(3):242–267, July 1995.

[11] Frédéric Devillers and Stéphane Donikian. A scenario language to orchestrate virtual world evolution. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '03, pages 265–275, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[12] Stéphane Donikian. Hpts: a behaviour modelling language for autonomous agents. In *Proceedings of the fifth international conference on Autonomous agents*, AGENTS '01, pages 401–408, New York, NY, USA, 2001. ACM.

[13] Stephane Espi. Archisim: Multiactor parallel architecture for traffic simulation. In *In Proceedings of the Second World Congress on Intelligent Transport Systems*, page vol. IV, November 1995.

[14] Martin Fellendorf and Peter Vortisch. Microscopic traffic flow simulator vissim. In Jaume Barcel, editor, *Fundamentals of Traffic Simulation*, volume 145 of *International Series in Operations Research & Management Science*, pages 63–93. Springer New York, 2010.

[15] D.L. Fisher, M. Rizzo, J. Caird, and J.D. Lee. *Handbook of Driving Simulation for Engineering, Medicine, and Psychology.* Taylor & Francis, 2010.

[16] K. Gajananan, A. Nakasone, H. Prendinger, and M. Miska. Scenario markup language for authoring behavioral driver studies in 3d virtual worlds. In *Visual Languages and Human-Centric Computing (VL/HCC), 2011 IEEE Symposium on*, pages 43 –46, sept. 2011.

[17] Sameh El Hadouaj and Stephane Espi. *A Generic Road Traffic Simulation Model*, chapter 180, pages 1314–1321.

[18] David Harel. Statecharts: A visual formalism for complex systems. *Sci. Comput. Program.*, 8(3):231–274, June 1987.

[19] H. Hattori, Y. Nakajima, and T. Ishida. Learning from humans: Agent modeling with individual human behaviors. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 41(1):1 –9, jan. 2011.

[20] F. Nunes Ferreira J. Miguel Leito, A. Augusto Sousa. A scripting language for multi-level control of autonomous agents in a driving simulator. In *In Driving Simulation Conference, Paris*, pages 339–351, 1999.

[21] Jacqueline Marie Jenkins. *Modeling the interaction between passenger cars and trucks.* PhD thesis, Texas A&M University, 2004.

[22] Joseph Kearney, Peter Willemsen, Stephane Donikian, and Frederic Devillers. Scenario languages for driving simulation. In *Driving Simulation Conference*, DSC'99, pages 377–393, 1999.

[23] Victor L. Knoop, Serge P. Hoogendoorn, and Henk J. van Zuylen. Capacity Reduction at Incidents: Empirical Data Collected from a Helicopter. *Transportation Research Record*, 2071:19–25, 2008.

[24] Gregoire S. Larue, Inhi Kim, Andry Rakotonirainy, Luis Ferreira, and Narelle L. Haworth. Integrating driving and traffic simulators to study railway level crossing safety interventions : a methodology. In *13th International Conference on Computer System Design and Operation in the Railway and other Transit Systems*, pages 719–732. WIT Press, September 2012.

[25] Chong-U Lim, Robin Baumgarten, and Simon Colton. Evolving behaviour trees for the commercial game defcon. In *Proceedings of the 2010 international conference on Applications of Evolutionary Computation - Volume Part I*, EvoApplicatons'10, pages 100–110, Berlin, Heidelberg, 2010. Springer-Verlag.

[26] A. Bryan Loyall and Joseph Bates. Real-time control of animated broad agents. In *In Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*, 1993.

[27] Marconi Madruga Filho, Helmut Prendinger, Todd Tilma, Martin Lindner, Edgar Santos, and Arturo Nakasone. Practicing eco-safe driving at scale. In *Proceedings of the 2012 ACM annual conference extended abstracts on Human Factors in Computing Systems Extended Abstracts*, CHI EA '12, pages 2147–2152, New York, NY, USA, 2012. ACM.

[28] Pattie Maes. The agent network architecture (ana). *SIGART Bull.*, 2(4):115–120, July 1991.

[29] Pattie Maes. Modeling adaptive autonomous agents. *Artificial Life*, 1:135–162, 1994.

[30] Jonathan P. Masinick and Hualiang (Harry) Teng. An analysis on the impact of rubbernecking on urban freeway traffic. Technical Report UVACTS-15-0-62, Center for Transportation Studies,University of Virginia, 351 McCormick Road, P.O. Box 400742, Charlottesville, VA 22904-4742. USA., August 2004.

[31] I. Millington and J. Funge. *Artificial Intelligence for Games, Second Edition.* Morgan Kaufmann Publishers, 2009.

[32] M. Miska, H. Prendinger, A. Nakasone, and M. Kuwahara. Driving and traveller behavior studies using 3d internet. In *Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference on*, pages 1632 –1637, sept. 2010.

[33] Marc Miska. *Microscopic Online Simulation for Real time Traffic Management.* PhD thesis, The Netherlands TRAIL Research School, 2007.

[34] Marc Miska, Kugamoorthy Gajananan, Edward Chung, and Helmut Prendinger. A traffic simulation standard based on data marts. In Peter Tisato, Lindsay Oxlad, and Michael Taylor, editors, *34th Australasian Transport Research Forum 2011*, pages 1–11, Adelaide, S. Aust., September 2011. PATREC.

[35] Marc Miska, Edgar Santos, Edward Chung, and Helmut Prendinger. Opentraffic - an open source platform for traffic simulation. In Peter Tisato, Lindsay Oxlad, and Michael Taylor, editors, *34th Australasian Transport Research Forum 2011*, Adelaide, S. Aust., September 2011. PATREC.

[36] Y. Papelis O. Ahmad. A comprehensive microscopic autonomous driver model for use in high- fidelity driving simulation environments. In *In Proceedings of 81st Annual Meeting of the Transportation Research Board, Washington, USA*, 2001.

[37] Johan Olstam, Stphane Espi, Selina Mardh, Jonas Jansson, and Jan Lundgren. An algorithm for combining autonomous vehicles and controlled events in driving simulator experiments. *Transportation Research Part C: Emerging Technologies*, 19(6):1185 – 1201, 2011.

[38] Johan Janson Olstam, Jan Lundgren, Mikael Adlers, and Pontus Matstoms. A framework for simulation of surrounding vehicles in driving simulators. *ACM Trans. Model. Comput. Simul.*, 18(3):9:1–9:24, July 2008.

[39] Ken Perlin and Athomas Goldberg. Improv: a system for scripting interactive actors in virtual worlds. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 205–216, New York, NY, USA, 1996. ACM.

[40] H. Prendinger, S. Descamps, and M. Ishizuka. Mpml: A markup language for controlling the behavior of life-like characters. *Journal of Visual Languages and Computing*, 15(2):183–203, 2004.

[41] H. Prendinger, S. Ullrich, A. Nakasone, and M. Ishizuka. Mpml3d: Scripting agents for the 3d internet. *Visualization and Computer Graphics, IEEE Transactions on*, 17(5):655 –668, may 2011.

[42] V. Punzo and B. Ciuffo. Integration of driving and traffic simulation: Issues and first solutions. *Trans. Intell. Transport. Sys.*, 12(2):354–363, June 2011.

[43] Karel Brookhuis Raymond Hoogendoorn, Serge P. Hoogendoorn and Winnie Daamen. Mental workload, longitudinal driving behavior, and adequacy of car-following models for incidents in other driving lane. *Transportation Research Record: Journal of the Transportation Research Board, No. 2188, Transportation Research Board of the National Academies, Washington D.C.*, page 6473, 2010.

[44] J. Russell and R. Cohn. *Cityengine*. Book on Demand, 2012.

[45] T.J. Rosenthal R.W.Allen and G. Park. Scenarios produced by procedural methods for driving research, assessment and training applications. In *In Driving Simulation Conference*, page Paper No. 621, 2003.

[46] Majid Sarvi, Masao Kuwahara, and Avishai Ceder. Freeway ramp merging phenomena in congested traffic using simulation combined with a driving simulator. *Computer-Aided Civil and Infrastructure Engineering*, 19(5):351–363, 2004.

[47] JJ Slob. State-of-the-art driving simulators, a literature survey. *Eindhoven: Technische Universiteit*, 2008.

[48] P. Suresh and R.R. Mourant. A tile manager for deploying scenarios in virtual driving environments. In *In Driving Simulation Conference, North America*, pages 21–29, 2005.

[49] Unrealscript reference, August 2012.

[50] Unity3d, August 2011.

[51] Unrealengine, August 2012.

[52] W. van Winsum. Ssl/nsl specification release 1.2. Trc, University of Groningen, November 1994.

[53] P. van Wolffelaar and W. van Winsum. Traffic modeling and driving simulation - an integrated approach. In *In Driving Simulation Conference*, DSC'95, pages 235–244, 1995.

[54] Ying Wang, Wei Zhang, Su Wu, and Yang Guo. Simulators for driving safety study a literature review. In Randall Shumaker, editor, *Virtual Reality*, volume 4563 of *Lecture Notes in Computer Science*, pages 584–593. Springer Berlin Heidelberg, 2007.

[55] I. H. C. Wassink, E. M. A. G. van Dijk, J. Zwiers, A. Nijholt, J. Kuipers, and A. O. Brugman. Bringing hollywood to the driving school: dynamic scenario generation in simulations and games. In *Proceedings of the First international con-*

ference on Intelligent Technologies for Interactive Entertainment, INTETAIN'05, pages 288–292, Berlin, Heidelberg, 2005. Springer-Verlag.

[56] Ingo Wassink, Betsy van Dijk, Job Zwiers, Anton Nijholt, Jorrit Kuipers, and Arnd Brugman. In the truman show: Generating dynamic scenarios in a driving simulator. *IEEE Intelligent Systems*, 21(5):28–32, September 2006.

[57] Peter Willemsen. *Behavior and Scenario Modeling for Real-time Virtual Environments.* PhD thesis, Department of Computer Science, University of Iowa, 2000.

[58] Steven Wright. *Supporting intelligent traffic in the leeds driving simulator.* PhD thesis, School of Computing, University of Leeds, 2000.

[59] G. Watson Y. Papelis, O. Ahmad. Developing scenarios to determine effects of driver performance: Techniques for authoring and lessons learned. In *In Driving Simulation Conference, North America*, 2003.

[60] K. Yamada and J. K. Kuchar. Preliminary study of behavioral and safety effects of driver dependence on a warning system in a driving simulator. *Trans. Sys. Man Cyber. Part A*, 36(3):602–610, May 2006.

Appendix A

Scenario Markup Language Documentation

SML is a XML-based scripting language to describe a scenario in a 3D virtual simulation environment based on .NET platform

Note that the usage of the word scenario sometimes refers to both the static structure of the virtual environment or scene (e.g. road network) and the dynamic characteristics of a simulation (e.g. critical events) [16]. We use scenario (script) to refer to entities or actors (or computer controlled agents) in the simulation, the behavior of entities, and the orchestration or coordination of their behaviors (interactions) to create a specific situation or event.

Although SML is not constrained to one single virtual world platform by design, the current version only supports to create scenarios using the entities inhabiting in a simulation environment based on .NET platform, e.g., Unity3D (http://unity3d.com/) or Second Life (http://www.secondlife.com) or OpenSimulator (http://opensimulator.org).

With SML, it is easy to construct scenarios, involving different entities in a simulation environment.

In the main text of the thesis, we have utilized Scenario Markup Language in the context of creating a traffic scenario, hence entities are vehicles or traffic lights. However, Scenario Markup Language Framework itself is generic enough to handle other contexts other than traffic.

Note: In this appendix, we provide a detail explanation of the SML Framework as a software which can be used in different simulation contexts. Please note that, SML Framework has been a work-in-progress software, early versions have been reported in the main text of the thesis and there have many changes overtime, we provide here the latest version of the specification and programming reference. As part of it, we introduce SML Execution System, which refers to the reusable software originated from SML Framework which is described in the main thesis.

A.1 Specification of Scenario Markup Language

A.1.1 SML

```
<SML>
...
</SML>
```

Description: The SML-element contains encapsulates the whole scenario script. It must be the root element of every SML script.

Attributes

- None

Legal child elements

- $< Head >$

- $< Body >$

A.1.2 Head

```
<Head>
...
</Head>
```

Description: The Head-element contains a general description of the entities or actors and users. This description includes a specification of the entities which are present in a simulation program and their Property.

Attributes

- None

Legal child elements

- < *Entities* >

- < *User* >

A.1.3 Entities

```
<Entities>
...
</Entities>
```

Description: The Entities-element contains a description for each entity present in a simulation. This description includes a specification of the entities which are present in a simulation program and their Property. The Entity-elements are optional to the Entities-element, however typically at least one entity is present. *Attributes*

- None

Legal child elements

- < *Entity* >

A.1.4 Entity

```
<Entity id = "AI Car D" name="AI Car D" type="vehicle">
 <property name = "scenariorole" valueType = "string" value = "Lead_Car" />
 <property name = "MaximumAcceleration" valueType = "float" value = "5m/s2"/>
</Entity>
```

Description: The Entity-elements build the core of every SML script. Everything in a simulation that acts, reacts or 'just is', is represented by an entity. Entity also is sometimes referred as actor.

Attributes

- id [required]: Since entities have to interact with each other they need an id to be used to address them when assigning commands to entities. These names are case-sensitive and have to be unique throughout the whole SMl script!

- name [required]: If ids of the entities are not proper noun, scenario author can choose to name entities to be recognizable in the SML script.

- type [required]: The type of the entity. ”vehicle” is the current example, while ”‘vehicle”-entities represent vehicle (so-called) agents in simulation program.

Legal child elements

- *< Property >*

A.1.5 Property

```
<property name = "scenariorole" value = "Lead_Car"/>
```

Description: To configure Entities Property-elements are used. For example all vehicle Entities need to know what their scenario role ”scenariorole” property as shown in the example above.

Attributes

- name [required]: The name of the property to set. Every property may only be set once!

- valueType [required]: This refers to the type of the property being set.

- value [required]: This refers to the value of the property being set.

Legal child elements

- None

A.1.6 User

```
<User userid="User1">
...
</user>
```

Description: Our target environment has to handle multiple users simultaneously. We decided that there has to be at least one User as the focal point of a scenario; that is, scenario is to be reproduced from the viewpoint of one specified user.

Attributes

- id [required]: Since users have to interact with each other they need an id to be used to address them.

- name [required]: If ids of the users are not proper noun, scenario author can choose to name users to be recognizable in the SML script.

Legal child elements

- < *Property* >

A.1.7 Body

```
<Body>
...
</Body>
```

Description: The Body-element contains a general description of the dynamics of a scenario. This description includes a specification of the entities'e behaviors, events, and the special element called Director, which contains the script for orchestrating entities.

Attributes

- None

Legal child elements

- < *Scenario* >

- < *Event* >

- < *Behavior* >

A.1.8 Scenario

```
<Scenario>
...
</Scenario>
```

Description: The Scenario-element contains a general description of the dynamics of a scenario, specifically the special element called Director, which contains the script for orchestrating entities.

Attributes

- scenarioId [required] : Since multiple scenarios can be loaded at the same time, and have to interact with each other they need an id to be used to address them.

- scenarioName [required] : If ids of the scenarios are not proper noun, scenario author can choose to name scenarios to be recognizable in the SML script.

- scenarioDescription [optional] : A small description for describing what the scenario is about.

Legal child elements

- *< Director >*

A.1.9 Director

```
<director>
 <seq>
  <task taskId="Accelerate">
   <param  name="actorId" valueType ="string" value ="AI Car A"/>
   <param  name="acceletationMagnitude" valueType ="float" value ="2.5f"/>
  </task>
  <par>
   <task taskId="Accelerate">
     <param  name="actorId" valueType ="string" value ="AI Car B"/>
     <param  name="acceletationMagnitude" valueType ="float" value ="1.5f"/>
   </task>
   <task taskId="ChangeStatus">
    <param  name="actorId" valueType ="string" value ="TrafficLightA"/>
    <param  name="targetStatus" valueType ="float" value ="Red"/>
   </task>
  </par>
 </seq>
</director>
```

Description: The Director-element contains a set of task organized using temporal elements. A scenario director can orchestrate the behavior of entities by executing corresponding tasks. This can be done by using the temporal elements of scenario director, such as seq (sequential), par (parallel) or sel (selective). These temporal elements enable SML to provide high-level synchronization of the behavior of entities that result in a particular situation.

Attributes

- None

Legal child elements

- $< seq >$
- $< sel >$
- $< par >$
- $< task >$

A.1.10 Seq

```
<seq>
...
</seq>
```

Description: The Sequential-element defines a sequence of activities that are performed sequentially one after the other once the Sequential-activity is started. A Sequential-activity finishes when the last of its child-activities has finished.

Attributes

- None

Legal child elements

- $< seq >$
- $< sel >$
- $< par >$
- $< task >$

A.1.11 Par

```
<par>
...
</par>
```

Description: The Parallel-element defines a set of activities that are performed concurrently and started all at the same time when the Parallel-activity is started. A Parallel-activity finishes when all of its child-activities have finished

Attributes

- None

Legal child elements

- $< seq >$
- $< sel >$
- $< par >$
- $< task >$

A.1.12 Sel

```
<sel>
...
</sel>
```

Description: The Selected-element defines a set of activities of which one is randomly chosen and started when the Selected-activity is started. An activity from that set is only chosen once if the Selected-activity is started repeatedly. After all activities were chosen once, they all become available again. A Selected-activity finishes when the chosen child-activity finishes.

Attributes

- None

Legal child elements

- < *seq* >

- < *sel* >

- < *par* >

- < *task* >

A.1.13 Task

```
<task>
...
</task>
```

Description: The Task-element contains a command to control an entity's behavior. For example, as shown in Code listing, the task of the director tasks defined in seq temporal element orchestrates the two car entities and a traffic lights' behaviors by issuing necessary commands, so that a traffic light turns red, while two AI Cars' crossing the red light.

Attributes

- taskId [required] : taskId should be unique across the SML Execution System, which is used to identify a task uniquely.

Legal child elements

- < *param* >

A.1.14 Event

```
 <Event eventid ="UserFarFromAICar_Event0">
    <Conditions>
      <Condition Condexpres ="CameFarFromAICar:UserFarFromAICar_Event0">
        <Variables>
          <Variable name ="User" valueType ="string" value ="User1"/>
          <Variable name ="AICar" valueType ="string" value="ScriptedCarA"/>
          <Variable name ="distance" valueType ="int" value ="50"/>
        </Variables>
      </Condition>
     </Conditions>
   </Event>
```

Description: The Event-element contains a the event definition which specifies the conditions that decide when the event is triggered. Simulation program should retrieve the set of events as shown later the code listing and check for the conditional expression and variables' values and trigger the event.

Attributes

- eventid [required] : eventid should be unique across the SML Execution System, which is used to identify an event uniquely.

Legal child elements

- *< Conditions >*

A.1.15 Conditions

```
<Conditions>
...
</Conditions>
```

Description: The Conditions-element contains a set conditions to be satisfied so that event can be triggered in the simulation program.

Attributes

- None

Legal child elements

- *< Condition >*

A.1.16 Condition

```
<Condition Condexpres ="CameFarFromAICar:UserFarFromAICar_Event0">
...
</Condition>
```

Description: The Condition-element contains an expression, which is validated along with variables the condition contain.

Attributes

- condexpres [required] : condexpres should be unique across the SML Execution System, which is used to identify a condition uniquely.

Legal child elements

- *< Variables >*

A.1.17 Variables

```
<Variables>
...
</Variables>
```

Description: The Variables-element contains a list of variables that should be validated to trigger the event.

Attributes

- None

Legal child elements

- $< Variable >$

A.1.18 Variable

```
<Variable name ="User" valueType ="string" value ="User1"/>
...
</Variable>
```

Description: The Variable-element contains a variable which should be validated against the simulator conditions, to trigger the event.

Attributes

- name [required] : The name of the variable. The name has to be unique and must not conflict with variable names used by entities!

- valueType [required] : The type of the variable value. Must be one of string, float or boolean

- value [required] : The value the variable is initialized with.

Legal child elements

- None

A.1.19 Behavior

```
<behavior behaviorid="ScriptedCarA_Behavior0" refActor ="AI Car A"/>
...
</behavior>
```

Description: The Behavior-element contains a list of actions, referring to an entity, that can be executed due to an triggered event.

Attributes

- behaviorid [required] : behaviorid should be unique across the SML Execution System, which is used to identify a behavior uniquely.

- refActor [required] : The referred actor the behavior belongs to. This should be one of the entities defined in the header part of the SML script.

Legal child elements

- < *Action* >

A.1.20 Action

```
<action actionid="ScriptedCarA_Action0">
...
</action>
```

Description: The Action-element defines an activity which is to be executed by an entity when the Action-element is started. An action definition can have a set of perceptions contained in one of the container ALL or ANY. An action definition can have a set of commands defined in it. Therefore, SML Execution system executes a behavioral entity's an action by executing the commands specified for that action. Since commands instruct entities to do certain things, Action are the activities which actually do something visible. An Action or finishes when its command finishes.

Attributes

- actionid [required] : actionid should be unique across the SML Execution System, which is used to identify an action uniquely.

Legal child elements

- *< ALL >*

- *< ANY >*

- *< Command >*

A.1.21 ALL

```
<ALL>
...
</ALL>
```

Description: The All-element is a perception-container which occurs after all of the enclosed child perceptions have occurred. After the All-element occurred, all enclosed perceptions are reset and the All-perception can occur again.

Attributes

- None

Legal child elements

- *< Perception >*

A.1.22 ANY

```
<ANY>
...
</ANY>
```

Description: The Any-element is a perception-container which occurs when one of the enclosed child perceptions occurs.

Attributes

- None

Legal child elements

- *< Perception >*

A.1.23 Perception

```
<perception activated ="false" id ="UserCloseToAICar_Event0_Perception0"
    event ="UserCloseToAICar_Event0">
```

Description: The Perception-element defines a perception-command. Perceptions are activated when the corresponding event occurs. Each perception is contingent upon an event, which means that a perception is the minimum unit of event handling from the script perspective. Hence the per- ception is activated when the corresponding event is triggered during a simulation. This may lead to the activation of the perception container the perception belongs to.

Attributes

- id : id should be unique across the SML Execution System, which is used to identify a perception uniquely.

- activated : This must be false initially, will be reset by the SML Execution System.

Legal child elements

- None

A.1.24 Command

```
<command commandid="Accelerate">
...
</command>
```

Description: The Command-element contains a command which is the minimum unit of execution from a scenario and entity perspectives. It carries an instruction or a functional call with required parameters for the component that controls the entity. The parameters used with the commands are passed using call by value semantics

Attributes

- commandid [required] : commandid should be unique across the SML Execution System, which is used to identify a command uniquely.

Legal child elements

- < *Param* >

A.1.25 Param

```
<param name ="actor" value ="ScriptedCarA" valueType ="SMLFramework.Core.Actor"/>
```

Description: The Param-element contains a parameter, that need to be passed a long with a command or task.

Attributes

- name [required] : The name of the parameter. The name has to be unique and must not conflict with parameter names used by entities!
- valueType [required] : The type of the parameter value. Must be one of string, float or boolean or any other custom object type
- value [required] : The value the parameter is initialized with.

Legal child elements

- None

A.2 How to use SML Execution System in a simulation program (Programmers Reference)

SML Execution System consists of two main parts:

1. $ScenarioControlSystem$
2. $IScenarioInterface$

A.2.1 Example Usage of SML Execution System

In the SMLFrameworkTest.cs file provided in the SML package can be seen how the SML Execution System functions are called from a C# program. The following is a partial display of the file:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Timers;
using SMLFramework;
using C5;
using Action = SMLFramework.Core.Action;
using Event = SMLFramework.Core.Event;
using Task = SMLFramework.Core.Task;

namespace SMLFramework.Tests
{
  public class SMLFrameworkTest
  {
    private const string Filename = @"InconspicuousCarfollowing.xml";

    private const string Filepath = @"C:\SMLExample\SMLFramework.Tests\";

    private static  ScenarioControlSystem _controlSystem;

    private SMLFrameworkTest smlTest;

    private bool triggered = false;

    static void Main(string[] args)
    {
        smlTest = new SMLFrameworkTest();

      _controlSystem = ScenarioControlSystem.Instance();

      _controlSystem.ParseSmlScenarioScript(Filepath, Filename);

      _controlSystem.InitializeScenariControlSystem();

      smlTest.InitializeActors();

      ...

      _controlSystem.ExecuteScenariDirectorTasks();
    }


    public void InitializeActors()
    {

     var exampleActor1 = new ExampleActor("AI Car A");

     var exampleActor2 = new ExampleActor("AI Car B");

     var exampleActor3 = new ExampleActor("TrafficLightA");

     actorIntefaces = new HashDictionary<string, IScenarioInterface>();

     var smlActors = _controlSystem.RetriveActors();

     foreach (var actor in smlActors)
     {
      if (actor.Key == exampleActor1.ActorId)
      {
        actorIntefaces.Add(exampleActor1.ActorId, exampleActor1);
      }

      if (actor.Key == exampleActor2.ActorId)
      {
        actorIntefaces.Add(exampleActor2.ActorId, exampleActor2);
      }

      if (actor.Key == exampleActor3.ActorId)
      {
        actorIntefaces.Add(exampleActor3.ActorId, exampleActor3);
      }

     }

     _controlSystem.SetScenarioInterface(actorIntefaces);
    }

    public void TestTriggeringEvents()
    {
      if (!triggered)
      {
        foreach (var kvp in _controlSystem.RetriveEvents())
        {
          if (kvp.Value.EventCond.Expression.Equals("Event0"))
          {
            var eventArg = new Event.EventArguments(kvp.Value);
            kvp.Value.TriggerEvent(eventArg);
            triggered = true;
          }
        }
      }
    }

  }

  public class ExampleActor : IScenarioInterface
  {
    public string ActorId { private set; get; }

    public ExampleActor(string actorId)
    {
      ActorId = actorId;
    }

    public void ExecuteAction(Action action)
    {
      Console.WriteLine("Executing action " + action.ActionId);
    }

    public void ExecuteTask(Task task)
    {
      Console.WriteLine("Executing task " + task.TaskId);
    }

   }
}
```

A.2.2 ScenarioControlSystem

ScenarioControlSystem functions as the main entry point to the SML Execution System.

Within *ScenarioControlSystem*, the SML instruction parser and executor are implemented as a DLL. If a user wants to use this DLL in his/her own application, there are some steps that must be performed. First of all, he/she should include an import for SMlFramework as shown in line 6 in above code listing : e.g., $using\ SMLFramework$;

Additionally, this DLL has a simple public interface consisting of the following methods. The simulators that integrates to SML Execution System, can make use of the following methods to interact with it.

1. $Instance()$: This method creates an $ScenarioControlSystem$ object instance and return a reference to the instance back to the simulation program. (See line 30 in the above code listing.)

2. $ParseSmlScenarioScript(string\ filepath, string\ filename)$ This method accepts two parameters : file path where a SML script is located, and the name of the SML script, then it executes the parsing functionality of the SML system. After the successful execution of this step, all the instructions specified in the SML script would be loaded in the memory and are ready to be executed. (See line 32 in the above code listing.)

3. $InitializeScenarioControlSystem()$: This method initialize the execution functionality of the SML System. (See line 34 in the above code listing.)

4. $StopScenarioControlSystem()$: This method stops the execution of the SML System as a whole.

5. $RetriveActors()$: The method return a dictionary of actors initialized from a SML script : $GuardedDictionary < string,\ Actor >$. (See line 55 in the above code listing.) Please note that, what scenario authors specify as entities will be treated as actors with in the SML Execution System, so actors and entities are referring to the same concept. The simulators that uses the SML Execution System, should use this method to retrieve all the actors specified in the scenario script.

6. $SetScenarioInterface$

   $(HashDictionary < string, \ IScenarioInterface > \ \ actorInterfaces)$: After retrieving the list of reference to actors from the SML Execution System, a simulation program must hook them up with the actual actors or entities or computer agents inhibiting inside the simulation program and let the SML Execution System which actors in the scenario script are referring to which actual entities in side the simulation program so that SML Execution System can control the entities as specified in the scenario script.

   First, a simulation program must define actors as shown in the above code listing (line 98 - 117)and each actor class must implement the $IScenarioInterface$. Please see the detail in the section $IScenarioInterface$. Then, $HashDictionary < string, \ IScenarioInterface > \ \ actorInterfaces$ has to be constructed as shown in $InitializeActors()$ method in the above code listing (line 44 - 77) and using $SetScenarioInterface$, simulation program must hook up the actors specified in the script, with the actual actors or entities in the simulation program.

7. $RetriveUsers()$: The method return a dictionary of users initialized from a SML script : $GuardedDictionary < string, \ User >$

8. $RetriveEvents()$: The method return a dictionary of events initialized from a SML script : $GuardedDictionary < string, \ Event >$ (See line 83 in the above code listing.)

9. $TriggerEvent$: This method should be invoked by the simulation program, when the conditional expression and the variables matches, so that SML Execution System can activate perception that may lead the execution of some of the entities' behavior. (See line 89 in the above code listing.)

10. $ExecuteScenarioDirectorTasks()$: A simulation program should invoke this

method so that SML Execution System can execute the tasks specified as part of SML Director Element. SML Execution System would take care in which order the task must be executed, as per the temporal elements specified in the SML Director Element. However, the simulation program must decide, when to invoke *ExecuteScenarioDirectorTasks*() method. For example, section 'Incident Creation Functions' in the main text explains a case where SML Director's task are executed.

A.2.3 IScenarioInterface

IScenarioInterface refers to the interface, which has to be implemented by the actors or entities in the simulation so that SML Execution System can interact with them according to an SML script definition. *IScenarioInterface* basically define the following two methods that need to be implemented by the the respective entities.

When a simulation program define actors as shown in the above code listing (line 98 - 117) and each actor class must implement the *IScenarioInterface*. The following two methods are the ways, SML Execution System can control the actors within the simulation program.

1. *ExecuteAction*(*Action action*) This function must be implemented by the simulation program within the Actor class, which will be invoked by the SML Execution System, when the behavior of the correspond entity defined in the scenario script is executed.

   How the execution of an action should take place is up to the simulation program that defines the actor.

2. *ExecuteTask*(*Task task*) This function must be implemented by the simulation program within the Actor class, which will be invoked by the SML Execution System, when SML directors tasks defined in the scenario script is executed.

How the execution of a task should take place is up to the simulation program that defines the corresponding actor.

Appendix B

Scenario Markup Language Script

Here we present the SML script that was used to create the rubbernecking scenario for the experiment reported in this thesis.

B.1 SML Header Script

```
<sml:head>
  <sml:entities>
      <sml:entity name ="AI Car A" Id ="AI Car A" type = "Vehicle">
        <sml:property name ="BreakingAccelerationMagnitude" valueType ="float" value = "50"></sml:property>
        <sml:property name ="CyclicPath" valueType ="bool" value = "true"></sml:property>
        <sml:property name ="DistanceToBreak" valueType ="float" value = "50"></sml:property>
        <sml:property name ="MaximumAcceleration" valueType ="float" value = "200"></sml:property>
        <sml:property name ="AccelerationMagnitude" valueType ="float" value = "30"></sml:property>
        <sml:property name ="DistanceToChangeSpeed" valueType ="float" value = "50"></sml:property>
        <sml:property name ="MaintainSpeed" valueType ="bool" value = "true"></sml:property>
        <sml:property name ="DesiredSpeed" valueType ="float" value = "40m/s"></sml:property>
        <sml:property name ="DesiredAccel" valueType ="float" value = "1.6m/s2"></sml:property>
        <sml:property name ="ScenarioRole" valueType ="string" value = "Follow_Car"></sml:property>
        <sml:property name ="ScenarioRoleLoc" valueType ="vector3d" value = "ROLE_LOC_A"></sml:property>
      </sml:entity>
      <sml:entity name ="AI Car B" Id ="AI Car B" type = "Vehicle">
        <sml:property name ="BreakingAccelerationMagnitude" valueType ="float" value = "50"></sml:property>
        <sml:property name ="CyclicPath" valueType ="bool" value = "true"></sml:property>
        <sml:property name ="DistanceToBreak" valueType ="float" value = "50"></sml:property>
        <sml:property name ="MaximumAcceleration" valueType ="float" value = "200"></sml:property>
        <sml:property name ="AccelerationMagnitude" valueType ="float" value = "30"></sml:property>
        <sml:property name ="DistanceToChangeSpeed" valueType ="float" value = "50"></sml:property>
        <sml:property name ="MinDistanceToMoveAndStop" valueType ="float" value = "10"></sml:property>
        <sml:property name ="MaintainSpeed" valueType ="bool" value = "true"></sml:property>
        <sml:property name ="DesiredSpeed" valueType ="float" value = "30m/s"></sml:property>
        <sml:property name ="DesiredAccel" valueType ="float" value = "1.5m/s2"></sml:property>
        <sml:property name ="ScenarioRole" valueType ="string" value = "Lead_Car"></sml:property>
        <sml:property name ="ScenarioRoleLoc" valueType ="vector3d" value = "ROLE_LOC_B"></sml:property>
      </sml:entity>
  </sml:entities>
  <sml:users>
      <sml:user name = TestUser01">
        <sml:property name = "usecarid" value = "U1"/>
        <sml:property name = "scenariorole" value = "User"/>
      </user>
  </sml:users>
</sml:head>
```

B.2 SML Body Script

```
<sml:body>
  <sml:scenario scenarioId="CarAccidentScenario" scenarioName="CarAccidentScenario">
    <sml:director>
      <sml:seq>
        <sml:task taskId="MoveToTargetLocation">
            <sml:params>
              <sml:param  name="entityid" value ="AI Car A"/>
              <sml:param  name="entityName" value ="AI Car A"/>
              <sml:param  name="targetLocation" value ="ACC_LOC"/>
            </sml:params>
        </sml:task>
        <sml:task taskId="MoveToTargetLocation">
           <sml:params>
              <sml:param  name="entityid" value ="AI Car B"/>
              <sml:param  name="entityidName" value ="AI Car B"/>
              <sml:param  name="targetLocation" value ="ACC_LOC"/>
            </sml:params>
         </sml:task>
      </sml:seq>
     </sml:director>
      </sml:scenario>
  <sml:event>
    <sml:conditions>
      <sml:condition>
        <sml:variables>
          <sml:variable name ="incidentid" valueType ="string" value="2011-01"/>
          <sml:variable name ="incidenttype" valueType ="string" value="Accident"/>
          <sml:variable name ="severity" valueType ="string" value="High"/>
          <sml:variable name ="collider" valueType ="string" value="ScriptedCarB"/>
          <sml:variable name ="collidedwith" valueType ="string" value ="ScriptedCarA"/>
          <sml:variable name ="location" valueType ="vector3" value ="ACC_LOC"/>
        </sml:variables>
      </sml:condition>
    </sml:conditions>
  </sml:event>
</sml:body>
```

Appendix C

Questionnaire

This is the questionnaire used in the experiment described in Chapter 5.

We are working on a project with the aim of investigating driver behavior in various transport circumstances.

You have been driving the simulator in 15 minutes in a 3D virtual city. The following questions will treat how similar driving in the simulator and driving a car in the real world.

First there will be a couple of questions regarding your background information, then questions related to the driving experience in the simulation environment (the realism in the driving of the vehicle and the realism in other drivers behavior), finally the questions related to the simulation environment.

C.1 Background Information

1. Respondent ID :
2. Gender : Male or Female
3. Date of Birth :
4. Occupation :
5. Car Usage (Hours driven per week, during last one year):

6. Year of license issued :
7. How much experience do you have with computers?
8. How much experience do you have in playing video games?
9. Do you have previous experience in a driving simulator?

C.2 Questions related to the simulation environment

1. Did you experience any traffic situation that felt strange or unrealistic during your ride? In other words did any other driver behave strange or unrealistic at any time?
2. If you answered yes in above question, try to describe the situation/situations Situation : Which Road : Describe the situation: Do you think that this situation could appear in real life? Yes/No
3. If you answered no in question above. In your opinion could all situations during your driving happen in real life?
4. How did you experience the traffic conditions? Was like the normal traffic conditions that you might experience in real world?
5. There was an accident on the course? Did you recognize it?
6. If you have recognized the accident during your ride, please specify the location you saw the accident? what you saw at accident location? what you experience at the accident location?

Appendix D

Participant Information

D.1 Welcome to our Study!

We would kindly ask you to read this page carefully before we start the experiment:

In this study, we want to investigate the interaction (human factor) between multiple users (as drivers) in the driving simulation environment. As a 3D virtual environment, we chose a small part of Tokyo city that mimic the real world to a certain extent.

In order to measure the driver behavior changes in driving simulation environment, we use the following devices:

1. Driving wheel and pedal set
2. Keyboard
3. Desktop computer & Monitor
4. Headphone

The devices are proved to have no effect on your health. Your data will be logged/-stored anonymously. Please wear the headphone during the experiment and refrain from talking.

D.2 Instructions

In the experiment, we will ask you to drive a virtual car, in a platoon of 3 cars, using the driving wheel and pedal set on a selected road in the 3D virtual world environment.

1. Introductory Task: Please practice driving in the virtual environment (alone) for about 5 minutes, up to 50km/h.

2. Main Task: If you are a lead driver - Please drive safely with the speed up to 60km/h.

   If you are a following driver - Please drive at a natural (small but safe) distance from the lead driver you are following, maybe about the length of two cars. - If you get disconnected, you can speed up to 60km/h to catch up with the lead car

For all drivers, please drive the car as normal as possible:

1. all driver stay on one lane (i.e. no lane changing), which is the first lane on left side, precisely, the 3rd from the left.

2. follow traffic rules and manners as in the real world

3. obey the speed limit as instructed above.

Please don't

1. bump into other cars

2. race with other participants

3. exchange cars with other participants

4. drive on the wrong way or route not designated for the experiment (don't leave the main road = don't turn left or right)

5. stop the car and wander around with your avatar

6. stop the car and idle for long

Afterwards the experiment is finished, we will ask you to fill out a questionnaire about your experience with the driving in 3D Virtual World.

Thank you for participating in our study!

Queensland University of Technology
Brisbane Australia

PARTICIPANT INFORMATION FOR QUT RESEARCH PROJECT

– Simulation & Questionnaire –

Multi-user driving simulation experiment

RESEARCH TEAM

Principal Researcher:	Dr Marc Miska, Research Associate, QUT
Associate Researcher:	Kugamoorthy Gajananan, PhD Student, National Institute of Informatics The Graduate University for Advanced Studies, Japan now visiting QUT

DESCRIPTION

This project is being undertaken as part of PhD study for Kugamoorthy Gajananan.

The purpose of this project is to investigate the interaction (human factor) between multiple users (as drivers), using a computer based driving simulation environment.

You are invited to participate in this project because you already are a licensed driver of a motor vehicle and had previous exposure to computer games, which will allow you to easily adapt to the simulation setup.

PARTICIPATION

Your participation in this project is entirely voluntary. If you do agree to participate, you can withdraw from the project at any time without comment or penalty. Any identifiable information already obtained from you will be destroyed. Your decision to participate, or not participate, will in no way impact upon your current or future relationship with QUT or with National Institute of Informatics, The Graduate University for Advanced Studies, Japan.

You will be involved in a computer based driving simulation, during which you will drive a car in a virtual environment, using a steering wheel and pedal set for acceleration and breaking. After the driving experiment, you will be asked to complete an anonymous questionnaire with likert scale answers (i.e., realistic representation – unrealistic representation / slow - fast scale) that will take approximately 15 minutes. Questions will include the driving experience.

1. Rate yourself as a driver in real life
2. Car usage frequency
3. How many years have you been driving

If you agree to participate you do not have to complete any question(s) that you are uncomfortable answering.

EXPECTED BENEFITS

It is expected that this project will not directly benefit you directly if you are not a transport related student/professional. However, it may be of benefit in way that you may get to learn how we study driver behavior in 3D driving simulation environment that may lead us to have a better understanding of the human factors that are involve in driving.

RISKS

There are no risks beyond normal day-to-day living associated with your participation in this project.

PRIVACY AND CONFIDENTIALITY

All comments and responses are anonymous and will be treated confidentially. The names of individual persons are not required in any of the responses.

The project is funded by The Graduate University for Advanced Studies, Japan and they will not have access to the data obtained during the project.

Please note that non-identifiable data collected in this project may be used as comparative data in future projects.

CONSENT TO PARTICIPATE

We would like to ask you to sign a written consent form (enclosed) to confirm your agreement to participate.

QUESTIONS / FURTHER INFORMATION ABOUT THE PROJECT

If have any questions or require any further information please contact one of the research team members below.

Marc Miska – Research Associate (Program Director) | Kugamoorthy Gajananan – Visiting Researcher

Science and Engineering Faculty – Civil Engineering and The Built Environment – Civil Engineering

Phone 3138 2992 | Phone 3138 1462

Email marc.miska@qut.edu.au | Email k-gajananan@nii.ac.jp.

CONCERNS / COMPLAINTS REGARDING THE CONDUCT OF THE PROJECT

QUT is committed to research integrity and the ethical conduct of research projects. However, if you do have any concerns or complaints about the ethical conduct of the project you may contact the QUT Research Ethics Unit on 3138 5123 or email ethicscontact@qut.edu.au. The QUT Research Ethics Unit is not connected with the research project and can facilitate a resolution to your concern in an impartial manner.

Thank you for helping with this research project. Please keep this sheet for your information.

Queensland University of Technology
Brisbane Australia

CONSENT FORM FOR QUT RESEARCH PROJECT
– Simulation & Questionnaire –

Multi-user driving simulation experiment

RESEARCH TEAM CONTACTS

Marc Miska - Research Associate (Program Director)	Kugamoorthy Gajananan – Visiting Researcher

Science and Engineering Faculty – Civil Engineering and The Built Environment – Civil Engineering

Phone 3138 2992	Phone 3138 1462
Email marc.miska@qut.edu.au	Email k-gajananan@nii.ac.jp.

STATEMENT OF CONSENT

By signing below, you are indicating that you:

- Have read and understood the information document regarding this project.
- Have had any questions answered to your satisfaction.
- Understand that if you have any additional questions you can contact the research team.
- Understand that you are free to withdraw at any time, without comment or penalty.
- Understand that you can contact the Research Ethics Unit on 3138 5123 or email ethicscontact@qut.edu.au if you have concerns about the ethical conduct of the project.
- Understand that non-identifiable data collected in this project may be used as comparative data in future projects.
- Agree to participate in the project.

Name ....

Signature ....

Date ....

***Please return this sheet to the investigator*.**

Queensland University of Technology
Brisbane Australia

PARTICIPATE IN RESEARCH

Information for Prospective Participants

The following research activity has been reviewed via QUT arrangements for the conduct of research involving human participation.
If you choose to participate, you will be provided with more detailed participant information, including who you can contact if you have any concerns.

Multi-user driving simulation experiment

Research Team Contacts

Principal Researcher: Dr. Marc Miska, Research Associate, QUT

Associate Researcher: Kugamoorthy Gajananan, PhD Student, National Institute of Informatics, The Graduate University for Advanced Studies, Japan now visiting QUT

What is the purpose of the research?

The purpose of this research is to investigate the interaction (human factor) between multiple users (as drivers) in the driving simulation environment.

Are you looking for people like me?

The research team is looking for research students, staff members and any professionals who have a driver's license, and previous exposure to computer games, due to the nature of the simulation.

What will you ask me to do?

Your participation will involve driving a car in a virtual world using a gamepad. You will be asked to use a headphone and not to talk during the experiment.

Are there any risks for me in taking part?

The research team does not believe there are any risks beyond normal day-to-day living associated with your participation in this research.

It should be noted that if you do agree to participate, you can withdraw from participation at any time during the project without comment or penalty.

Are there any benefits for me in taking part?

It is expected that this project will not benefit you directly if you are not transport related student/professional. However, it may benefit in way that you may get to learn how we study driver behaviour in 3D driving simulation environment that may lead us to have better understanding of the human factors that involve in driving.

Will I be compensated for my time?

There will be no compensation due to limited funds.

I am interested – what should I do next?

If you have any questions or would like to participate in this study, please contact the research team for details of the next step – marc.miska@qut.edu.au, k-gajananan@nii.ac.jp.

You will be provided with further information to ensure that your decision and consent to participate is fully informed.

Thank You!