Creating a Computing Environment in a Driving Simulator to Orchestrate Scenarios with Autonomous Vehicles

Zhitao Xiong

Submitted in accordance with the requirements for the degree of

Doctor of Philosophy

The University of Leeds

Institute for Transport Studies

&

School of Computing

September 2013

The candidate confirms that the work submitted is his/her own, except where work which has formed part of jointly authored publications has been included. The contribution of the candidate and the other authors to this work has been explicitly indicated below. The candidate confirms that appropriate credit has been given within the thesis where reference has been made to the work of others.

The following paper has been used loosely in this thesis:

Zhitao Xiong, Anthony G. Cohn, Oliver Carsten and Hamish Jamson, "*Autonomous local manoeuvre and scenario orchestration based on automated action planning in driving simulation*". In Proceedings of driving simulation conference Europe 2012, pages 233 - 244, Arts Et Métiers Paristech Paris, France, September 2012.

Chapters 5 and 7 include some work in the paper regarding algorithm description (Chapter 5) and experiments/results (Chapter 7). Chapter 7 uses tables, figures from the paper to describe some relevant results.

In this paper, the candidate contributed the algorithm, implementation, experiment and paper drafting, the co-authors, who are also the candidate's PhD supervisors, have contributed the work by suggesting relevant references:

- Oliver Carsten: driver model
- Anthony G. Cohn: automated action planning
- Hamish Jamson: standard scenario/situation library

Co-authors also contributed the paper by giving language-related advice.

This copy has been supplied on the understanding that it is copyright material and that no quotation from the thesis may be published without proper acknowledgement.

©2013 The University of Leeds and Zhitao Xiong

知行合一

Knowledge and action are one inseparable unit - Wang Yangming

Don't Panic
- The Hitchhiker's Guide to the Galaxy

Wake up, Neo...
The Matrix has you...
- The Matrix

Acknowledgements

It is very hard for me to finish this part, because I do not want to miss anyone, or anything.

I would like to thank my supervisors first. Oliver Carsten & Hamish Jamson from Transport Studies and Anthony G. Cohn from Computing have accompanied me from the very beginning of my PhD as my three lovely supervisors. They have evidenced every progress and every mistake I have made. Specially, I would like to thank Hamish, who has spent lots of time to assist me in thesis writing and critical thinking.

I would also thank some friends in ITS. Tony Horrobin has given me suggestions ranging from software test to scenario design. Nick Herbers has managed to proofread my thesis according to my tight schedule. Natasha Merat has assisted me in my VTI trip and helped me in varieties of ways. Daryl Hibberd has given me suggestions in carrying out experiments. Anzir Boodoo, as my nice officemate, has always been informative, helpful and warm-hearted. I also want to thank all the people in Safety & Technology group, as they made my life in Leeds wonderful.

I should thank Johan Olstam, who is a professional and considerate researcher from VTI and Linköping University in Sweden. He has given me inspirations specially within the area of traffic simulation. Our finished and ongoing collaborations have provided some deep thoughts in driving simulation. I should also thank some people from VTI, Jonas Jansson and Jonas Andersson Hultgren have assisted my research when I was in VTI. Maud Göthe-Lundgren kindly invited me to their Christmas lunch and showed me how wonderful a Swedish Christmas could be.

Special thanks go to my three examiners, Yiannis Papelis, Frank Lai and Natasha Chakhlevitch for the valuable discussions throughout my viva. Their comments have provided ideas on how my work should be presented to relevant researchers and practitioners.

Of course, I cannot miss my wife, Jingping Wu, and my parents, XIONG Shuzhong and ZHOU Liqiong. Jingping and my parents have given me not only invaluable moral support during my PhD, but also some financial support in its final stage. Moreover,my parents always supported my decisions without any hesitate. I really appreciate their support in the past 29 years. Also, I should thank my dad, XIONG Shuzhong, for his nice calligraphy of "Zhi Xing He Yi" by WANG Yangming, meaning that "Knowledge and action are one inseparable unit".

At last, I would like to address special thanks to the authors/contributors to some libraries, software or OS: LCM, Boost, OpenroadED, Protégé, MacTex/TexLive, Texmaker, Codeblocks, Xcode, Ubuntu Linux and OS X. They have made my life much easier than I could imagine from the very beginning of my PhD. I also want to thank Macbook pro with retina display, which helped me in VTI's experiments and thesis writing. She is always powerful, unbelievably user-friendly, a good sense of *nix and good looking, really highly recommended for academia.

Thank you everyone/everything that are related to this PhD directly or indirectly.

Zhitao Xiong
July 2013 (Last Modified: November, 2013)

Abstract

A scenario in a driving simulator covers what the human participants experience and what the researchers need: the physical scene, predefined traffic flow, simulated vehicles' interactions with the participants and measurements to be collected.

Current methodologies used to orchestrate scenarios regarding the interactions have the following drawbacks: 1) Action sequences that simulated vehicles should follow in scenarios are specified without the contexts of each Action; 2) programming languages always include platform-dependent details and are not suitable for context modelling and scenario sharing and 3) there is no mechanism to handle scenarios dynamically and deal with failures to deploy a scenario.

To overcome these problems, a concept named Assignment, which represents the task(s) of Virtual Drivers, was first developed to encode the contextual information of proposed Actions for interaction generation, e.g., potential simulated vehicles involved.

The Ontology for Scenario Orchestration (OSO) was then developed to model concepts and their relationships in the domain of scenario orchestration including the concept Assignment. It can also provide a file for machine processing.

An algorithm named NAUSEA (autoNomous locAl manoeUvre and Scenario orchEstration based on automated action plAnning) was finally generated to utilise Assignments recorded in OSO. Encoded in the driver model SAIL (Scenario-Aware drIver modeL), NAUSEA can be used by a Virtual Driver to control simulated vehicles dynamically. Failed interactions, generated by corresponding Assignments, can be

regenerated if necessary. A framework SOAV (Scenario Orchestration with Autonomous simulated Vehicles) was formed to support SAIL/-NAUSEA and orchestrate scenarios with autonomous vehicles.

Three verification experiments were carried out and showed that SOAV was working properly by producing corresponding interactions based on SAIL/NAUSEA and Assignments. They also demonstrated that OSO can provide contextual information in a human-readable and machine processable manner.

The OSO evaluation showed that OSO has several advantages such as being readable, flexible etc., but how it can be presented to varieties of audiences needs further examination.

Table of Contents

List of Figures

List of Tables

Glossary

α, Action "Perform-scenario" or "Free"

In a particular **scenario**, the **Virtual Driver** needs to perform a top **High-Level Action** named α that can be either "Perform-scenario" or "Free". "Perform-scenario" contains four sub-**Actions**, namely, β_0, β_1, β_2 and β_3. "Free" makes the Virtual Driver ignore any **Assignments** and autonomously evolve in the simulated world of a particular **scenario**, in which case the route or the destination will be based on a pre-defined route. Details can be found in Sections 3.3.11 and 5.3.

β_0, Action "Get-to-the-initial-state"

By carrying out **Action** β_0, the **Virtual Driver** drives the **Ego-vehicle/flock** to an initial state (e.g., initial speed, initial target speed etc.). Details can be found in Section 3.3.11.

β_1, Action "Generate-formation"

By carrying out **Action** β_1, the **Virtual Driver** navigates the **Ego-vehicle/flock** to the proposed **Formation Position** in order to perform the corresponding **Assignment-actions**. This Action is handled by the **Regulating layer** in **SAIL**. Details can be found in Section 3.3.11 and 5.3.

β_2, Action "Perform-assignment"

By carrying out **Action** β_2, the **Virtual Driver** finishes the **Assignment-actions** encoded in **Assignments**. This Action is handled by the **Situation Assessment layer** in **SAIL** and **Assignment Assessment procedure** in **NAUSEA**. Details can be found in Section 3.3.11 and 5.3.

β_3, Action "Clean-up"

By carrying out **Action** β_3, the **Virtual Driver** restores the configurations or allowed behaviours of the **Ego-vehicle/flock**. Details can be found in Section 3.3.11.

Action

An Action is what the **Virtual Driver** can do to change the simulated world of a particular **scenario**. It has a release time and a deadline. The former specifies when the Action is executed; the latter specifies when the Action is finished. Actions are divided into two categories: **High-Level Action** and **Low-Level Action**. Details can be found in Sections 3.3.5, 4.3.10 and 5.3.

Action Checker

The Action Checker procedure in **NAUSEA** or the Action Checker layer in **SAIL** checks whether or not an **Assignment-action** has executed, succeeded or failed. Details can be found in Section 5.3.2.5.

Action Executer layer, Action Execution procedure

The Action Execution procedure in **NAUSEA** or the Action Executer layer in **SAIL** checks if the release time of the pending **Assignment-action** is consistent with the **refined metric constraints**. If the answer is yes, this **Action** will be executed by sending out relevant **Smith Orders** stored in the definition of the Assignment-action. Details can be found in Section 5.3.2.5.

Action layer

The Action layer in **SAIL** is in charge of broadcasting **Smith Orders**, containing what should be done to change the simulated world of a particular **scenario**, including the simulated vehicles' states and road conditions. Details can be found in Section 5.2.4.

Actor

Actor refers to the **Ego-vehicle/flock** that is being controlled by the **Virtual Driver** for proposed **Assignments** in order to generate interactions. This term was inspired by Olstam *et al.* (2011).

Agent Smith

Agent Smith is a character in the film trilogy "**The Matrix**". He is one of the AI programs in "The Matrix", which are responsible for maintenance purposes and thus can terminate any human avatars or programs that may bring instability. He finally becomes a virus that gets the power "*to take control over the simulated body of any human wired into the Matrix*" and "*to communicate with each other instantaneously and perceive what other humans wired into the Matrix do via a type of shared consciousness*" (Wikipedia, 2011).

Assignment

An Assignment is a task that a **Virtual Driver** needs to carry out in order to generate required interactions with the **participant**. It provides relevant contextual information to the Virtual Driver: the **Formation Position**, **Monitor(s)**, **Success Condition(s)**, **Failure Condition(s)**, **Assignment-action(s)** and the measurement from the interaction generated by this Assignment. Details can be found in Sections 3.3.10 and 4.3.11.

Assignment-action

An Assignment-action is the **Action** encoded in a particular Assignment to generate a specific interaction, e.g., request to "set the desired speed of the simulated vehicle no.1 as 30 *mph*" or request to "place cones in the road segment whose id is 'r3.2' ". Details can be found in Sections 3.3.5, 3.3.10 and 4.3.11.

Assignment Assessment procedure

The Assignment Assessment procedure in **NAUSEA**, running in the **Situation Assessment layer** in **SAIL**, is responsible for handling **Assignments**. It consists of three sub-procedures: **Assignment Checker**, **Action Execution** and **Action Checker** procedures. Details can be found in Section 5.3.2.5.

Assignment Checker layer, Assignment Checker procedure

The Assignment Checker procedure in **NAUSEA**, running in the **Situation Assessment layer** in **SAIL**, is responsible for triggering any **Assignments** based on **precedence constraints** or **Monitor(s)**. Details can be found in Section 5.3.2.5.

(The) Cognition layer

The Cognition layer in **SAIL** is responsible for maintaining **Memory** and making decisions. It consists of two sub-layers, one is the **Memory layer**, another is the **Decision Making layer**. Details can be found in Section 5.2.

Decision Making layer

Decision Making layer in **SAIL** is used to navigate the Ego-vehicle/flock to a proposed position safely and carry out **Assignments** to generate interactions for **scenarios** in time. It is handled by the algorithm **NAUSEA**. Details can be found in Section 5.2.

Driving Experience layer

Driving Experience layer in **SAIL** contains **Recipes** and makes the **Virtual Driver** know how to perform **High-Level Actions**. Details can be found in Sections 5.2.

Ego-flock

The **Flock** controlled by the **Virtual Driver** is termed an "Ego-flock". Details can be found in Section 3.3.4.

Ego-vehicle

The simulated vehicle controlled by the **Virtual Driver** is termed an "Ego-vehicle". Details can be found in Section 3.3.4.

Failure Condition

This is the condition used to decide whether or not an **Assignment** has failed during its execution. It is a type of **Trigger**. Details can be found in Sections 3.3.6 and 4.3.9.

Flock

A Flock refers to a platoon of simulated vehicles. Details can be found in Section 3.3.4.

Formation Position

Formation Position is a set of pre-defined relative local positions around the **participant's vehicle**, where the **Virtual Driver** usually navigates the **Ego-vehicle/flock** to in order to carry out **Assignments**. Details can be found in Section 3.3.7.

High-Level Action

A High-Level Action is an **Action** that cannot be accomplished in a single way, in one sequence or by one entity. A **Recipe** is needed to specify how to perform a particular High-Level Action. Details can be found in Sections 3.3.5, 4.3.10 and 5.3.

(The) Individual Feature layer

The Individual Feature layer in **SAIL** maintains two sets of Individual Features: **Driving Experience** and **Motivation**. The former contains **Recipes** for **High-Level Actions** and the latter contains **Assignments** that need to be carried out. Details can be found in Section 5.2.2.

Initialization procedure

Initialization procedure in **NAUSEA** is used by the **Virtual Driver** to parse an **SDF** and store relevant information into the **Memory**. An initial **(The) General Plan** will then be built. Details can be found in Section 5.3.

Low-Level Action

A Low-Level Action is an **Action** that can be accomplished in a single way, in one sequence and by one entity. Details can be found in Sections 3.3.5, 4.3.10 and 5.3.

Memory, (The) Memory layer

The Memory layer in **SAIL** contains two sets of Memory: **Individual**

Feature and **World Model**. The former contains **Recipes** for **High-Level Actions** and **Assignments** for a particular **scenario**; the latter contains all the contexts for driving, e.g., road network, **Memory** History, etc. Details can be found in Section 5.2.

metric constraints

Metric constraints specify the differences between two time instants, which can represent the start/finish times of some **Actions**, e.g., "start time of Action β - finish time of Action $\alpha \leqslant 100$ (seconds)" Details can be found in Sections 3.3.11 and 5.3.2.1.

Monitor

This specifies a condition to indicate whether or not the **Assignment-action(s)** in a particular **Assignment** should be executed. When multiple Monitors are present, the Assignment-action will be carried out if all the Monitors become true at the same time. Details can be found in Sections 3.3.6 and 4.3.9.

(The) Motivation layer

The Motivation layer in **SAIL** contains the **Assignments** that will be used to generate interactions in a particular **scenario**. Details can be found in Section 5.2.

NAUSEA

NAUSEA (autoNomous locAl manoeUvre and Scenario orchEstration based on automated action plAnning) is an algorithm based on HTN (Hierarchical Task Network). It is encoded in the **Decision Making layer** of **SAIL**. NAUSEA can be used to recruit simulated vehicles dynamically and prepare for interactions actively. Failed interactions, generated by corresponding **Assignments**, can be regenerated by NAUSEA with its replanning capability. Details can be found in Section 5.3.

Neo

Neo is the principal character in "**The Matrix**". In this film trilogy, Neo is regarded as "The One" who is believed to be able to free humans from being "batteries". Details can be found in Section 1.1.

Neighbourhood

Neighbourhood refers to 12 positions around the **participant's vehicle**, containing the information of simulated vehicles that are of interest to the **Virtual Driver** and will be used for decision making. Details can be found in Sections 4.3.7 and 5.2.1.

OSO, Ontology for Scenario Orchestration

OSO is short for Ontology for Scenario Orchestration (OSO). It is built upon varieties of concepts and their relationships in the domain of scenario orchestration in driving simulation in order to represent **scenarios** in a programming language-independent and logic-based manner. It covers concepts in driving simulation ranging from physical objects such as roads or junctions to virtual ones such as **Assignments** or **Actions**. Details can be found in Chapter 4.

participant, participant's vehicle

Participant refers to the human being who involves in a driving simulator study and drives the simulator vehicle. Participant's vehicle refers to the simulator vehicle.

Perception layer

Perception in **SAIL** involves two procedures, one is to sense the outside world by receiving the raw data; another is to interpret the raw data and maintain the **World Model**. Details can be found in Section 5.2.1.

Plan Evaluation procedure

Plan Evaluation procedure in **NAUSEA** or Plan Evaluation layer in **SAIL** is used to generate the **refined metric constraints** based on the **metric constraints** specified in a particular **scenario** definition. It will indicate the allowed ranges of some **Actions**' release times and deadlines. If the refined metric constraints can not be generated, the metric constraints or

the General Plan will be regarded as "inconsistent". Details can be found in Section 5.3.2.1.

precedence constraints

Precedence constraints specify the relationships between two time intervals, two **Actions** or two **Assignments**, e.g., "Assignment A_i before Assignment A_j $(i \neq j)\}$" represents that the execution of Assignment A_j starts after the execution of A_i finishes because A_i expires or **Success Conditions** of A_i are satisfied. Details can be found in Sections 3.3.11 and 5.3.2.1.

Recipe

A Recipe specifies how to perform a **High-Level Action** by providing a set of **Low-Level Actions** or **Actions** that do not need to be decomposed any more. Details can be found in Sections 3.3.5 and 3.3.11.

refined metric constraints

Refined metric constraints refers to a set of **metric constraints** obtained from the original metric constraints specified in **SDF**. They are generated by the **Plan Evaluation procedure** in **NAUSEA** or **Plan Evaluation layer** in **SAIL** to indicate the required range of the release times/deadlines of some **Actions**. Details can be found in Section 5.3.2.1.

(The) Regulating procedure

The Regulating procedure in **NAUSEA** or Regulating layer in **SAIL** uses any relevant driving behaviours, e.g., overtaking, to drive safely and satisfy the requirements from **Assignments** regarding the **Formation Position**. Details can be found in Section 5.3.2.4.

(The) Role Matching procedure

The Role Matching procedure in **NAUSEA** or Role Matching layer in **SAIL** determines which simulated vehicles the **Virtual Driver** should choose as the **Ego-vehicle/flock** to generate proposed interactions defined in **Assignments**. Details can be found in Section 5.3.2.2.

SAIL

SAIL (Scenario-Aware drIver modeL) is a driver model developed to adopt and test **NAUSEA** in order to provide NAUSEA with data for decision making and interfaces for executing **Actions**. Details can be found in Section 5.2.

Scenario

A scenario is a pre-defined environment that experimenters need a **participant** to experience in a driving simulator. It includes the physical scene, pre-defined traffic flow, simulated vehicles' interactions with the **participant's vehicle** and measurements that need to be collected. Details can be found in Section 4.3.1.

Scenario Observer

Scenario Observer is a module in **SOAV** to record the data packages transferred in SOAV. It can also visualise the number and frequencies of each data package. Details can be found in Section 3.5.1.2.4.

SDF, Scenario Definition File

The Scenario Definition File (SDF) is an XML file that records the **OSO** along with specific requirements of a particular **scenario**. **Assignments** and road network are the main information specified in SDF. Details can be found in Section 3.5.1.1.

Situation

Situation refers to some common interactions that **participants** can be exposed to, which involve more than one simulated vehicle. Details can be found in Section 3.3.8.

(The) Situation Assessment layer

The Situation Assessment layer in **SAIL** manages **the General Plan**, **Assignments** and **Role Matching procedure**, so it contains the **Assignment Assessment**, **Plan Evaluation** and Role Matching procedures in **NAUSEA**. Details can be found in Section 5.3.

Smith

Smith is the implemented **Virtual Driver**. He is the implementation of

SAIL/NAUSEA and named after **Agent Smith**. Details can be found in Section 3.5.1.2.1.

Smith Order

A Smith Order is the order sent from **Smith** to **the Sim/SMM**. Smith Orders contain instructions that the **Ego-vehicle/flock** should follow or SMM should carry out, e.g., request to "set the desired speed of the simulated vehicle no.1 as 30 *mph*". Details can be found in Section 3.5.1.2.3.

SMM

SMM is short for Scenario Management Module, which is used to interpret **Smith Orders** and execute them in **the Sim**. It has been developed as a module within the Sim. Details can be found in Section 3.5.1.2.3.

SOAV

SOAV is a framework used to create a computing environment for driving simulation in order to orchestrate **scenarios** with autonomous simulated vehicles. It consists of several components, which include **OSO (Ontology for Scenario Orchestration)**, **Smith**, **the Sim/SMM** and a **Scenario Observer**. Details can be found in Section 3.5.

speed adaptation

Speed adaptation behaviour in **SAIL** or speed adaptation procedure in **NAUSEA** is mainly used to 1) make **the Virtual Driver** obey the speed limit; 2) maintain a realistic speed trajectory when performing a turning movement or 3) adopt a speed or an acceleration rate to prepare for **Assignments**. Details can be found in Section 5.3.2.4.

subject

Subject refers to the **participant** or **participant's vehicle**. It is being used in the **Ontology for Scenario Orchestration (OSO)**. Details can be found in Section 4.3.4.

Success Condition

This is the condition used to decide whether or not an **Assignment** has succeeded during its execution. It is a type of **Trigger**. Details can be found in Sections 3.3.6 and 4.3.9.

(The) Targeting procedure

The Targeting procedure in **NAUSEA** or Targeting layer in **SAIL** adopts a pre-defined route to tell the **Virtual Driver** where to navigate the **Ego-vehicle/flock**. Details can be found in Section 5.3.2.3.

temporal constraints

Temporal constraints contain the **precedence constraints** and **metric constraints**. Details can be found in Sections 3.3.11 and 5.3.2.1.

The General Plan

The General Plan is an **Action** plan with **temporal constraints** to guide the **Virtual Driver**'s behaviours in **scenarios** by specifying execution orders of Actions and corresponding ranges of their release times/deadlines. Details can be found in Sections 3.3.11 and 5.3.2.1.

"The Matrix" Metaphor

"The Matrix" Metaphor refers to a comparison between some components in **SOAV**, especially **Smith**, to some characters and relevant philosophy in the film trilogy "**The Matrix**". This metaphor or comparison has been mainly taken to design the **Virtual Driver** Smith in the computing environment of driving simulation. The names "The Matrix" and "**Neo**" have been taken to call **the Sim** and **SMM** respectively. Details can be found in Sections 1.3 and 3.5.1.2.

The Sim, The Sim/SMM

The Sim refers to the simulation software for driving simulators and is a part of **SOAV**. It contains **SMM** and the vehicle dynamics and rendering facility for the simulation. Details can be found in Section 3.5.

Trigger

A Trigger is a condition used to execute some **Action(s)**: when it becomes true, the Action(s) will be executed. There are three types of Triggers:

Monitor, **Success Condition** and **Failure Condition**. Details can be found in Sections 3.3.6 and 4.3.9.

Virtual Driver

A Virtual Driver indicates an intelligent controller that can be used to make driving decisions or carry out pre-scheduled **Actions** based on **Assignments**. Details can be found in Section 3.3.1 and Chapter 5.

World Model layer

The World Model layer in **SAIL** maintains driving context such as the road network and relevant interfaces for modifying or querying the driving context. Details can be found in Section 5.2.3.

Chapter 1

Introduction

This is your last chance. After this, there is no turning back. You take the blue pill - the story ends, you wake up in your bed and believe whatever you want to believe. You take the red pill - you stay in Wonderland and I show you how deep the rabbit-hole goes.
- The Matrix

1.1 "The Matrix" Fantasy

What if you just found out that everything in your "real" life is actually fake?

You are "living" in a simulated world and you do not know if you are just a program or a real human being that is connected to this simulated world, could this be the worst moment in your life? You may then start to ask what is "real" and seek the truth. I am not sure if you would face this situation in your life but if you were Neo, you would have experienced the whole situation already.

Neo is the principal character in "The Matrix". In this film trilogy[1], machines created a simulated world called "The Matrix" and managed to get energy from humans who were kept in pods and implanted in the virtual reality "The Matrix". Neo is regarded as "The One" who is believed to be able to free humans from being "batteries", so he knows what "The Matrix" is. Moreover, not all the avatars in "The Matrix" are controlled by humans. Some of them are AI programs

[1]The Matrix (1999), The Matrix Reloaded (2003), The Matrix Revolutions (2003)

responsible for maintenance purposes and can terminate any human avatars or programs that may bring instability. Agent Smith is the most famous AI program in "The Matrix", not because he is a Neo hunter, but because he finally becomes a virus that gets the power "*to take control over the simulated body of any human wired into the Matrix*" and "*to communicate with each other instantaneously and perceive what other humans wired into the Matrix do via a type of shared consciousness*" (Wikipedia, 2011). Hence, Agent Smith is able to put Neo into some intentionally designed and dangerous situations. Neo always fights back to destroy these setups because nobody wants to be terminated, especially the principal character. However, the machines are also being regarded as human protectors, because by doing so, human beings can be kept in a safer environment than the reality: chaos after nuclear war. These machines can be regarded as human-users or human-protectors, but remember, in either case, human beings are under control.

Although the story of "The Matrix" is fantastic, some ideas behind this story may be helpful: we can try to put someone under control without being noticed by adopting vivid scenes and sophisticated intelligent agents that can make autonomous decisions. In driving simulation, researchers are attempting to create realistic and natural virtual worlds, which can be regarded as the driving world version of "The Matrix". However, this version is highly controlled to generate scenarios so that consistent stimuli can be provided to different driving simulator drivers or participants. Sophisticated intelligent vehicles therefore may not be needed. Moreover, no killing is allowed in this driving world version.

1.2 The Scenario Reality in Driving Simulation

By imitating driving activities of the real world, driving simulators can:

1) have repeatable and consistent scenarios;

2) have a substantially risk-free environment;

3) provide some hazardous situations not easily performed in the real world.

A scenario is then the key to provide a pre-defined environment that experimenters need a participant to experience. It includes the physical scene, pre-defined traffic flow, simulated vehicles’ interactions with the participant’s vehicle[1] and measurements that need to be collected. Choreography of scenarios, therefore, plays an important role in driving simulation. In general, the interactions in scenarios should be repeatable and the human participants should be able to make similar decisions in driving simulators as they would in the real world.

Certainly, the algorithm developed in this thesis should be able to handle every aspect of a scenario, such as physical scene modification, traffic flow manipulation, data collection and interactions with participant’s vehicle, however, how to generate proposed interactions has been the only concern of this research so “scenario”in this thesis will be used to describe the interactions between simulated vehicles and the participant’s vehicle.

Generally speaking, there are two basic requirements regarding scenarios: 1) the simulated vehicles should behave in a realistic manner and 2) the scenarios should be repeatable at some suitable level of abstraction.

Researchers always put a focus on making scenarios repeatable, so when it comes to scenarios, simulated vehicles will be controlled by pre-defined instructions that have been crafted by humans beforehand. As a result, failures can sometimes happen if some unexpected situation occurs, e.g., the participant changes lane suddenly and thus misses the simulated vehicle that has already been prepared for such interactions.

1.3 “The Matrix” Metaphor and Research Aims

The methodology and concepts involved in this thesis have been motivated and therefore influenced by the hope of creating “The Matrix” in driving simulation. The steps taken in this thesis are too small to be considered as fantastic as “The Matrix” , however, these steps can answer two fundamental questions in driving simulation:

[1]In this research, only one participant or participant’s vehicle (simulator) is included in a particular scenario.

1) How can we create scenarios with autonomous simulated vehicles so that scenarios can be realistic and repeatable? (Scenario Orchestration for our "The Matrix");

2) How can we describe our knowledge in driving simulation? (Knowledge Base for our "The Matrix") and,

Generally speaking, "The Matrix" metaphor has been mainly applied to design an intelligent virtual driver in the computing environment of driving simulation. The virtual driver is treated as "Agent Smith" and (autonomous) simulated vehicles that the participant could see in the simulation are treated as "simulated humans" in "The Matrix". Virtual drivers should therefore have the ability of "Agent Smith", e.g., 1) progress in the virtual environment autonomously in order to finish some tasks by controlling any entities in driving simulation dynamically and 2) plan actions intelligently with the capability of replanning if any specific task has failed. As an AI program similar to "Agent Smith", a virtual driver should be able to understand the scenario instructions and act according to those instructions with rich, appropriate and scenario-directed behaviours.

This research aims to systematically analyse the issue of scenario orchestration in driving simulation with two purposes:

1) to develop an algorithm for orchestrating scenarios with autonomous vehicles intelligently, so that pre-defined interactions can be generated according to dynamic driving context, with the possibility of restoring scenarios from failures. This is the main focus of this research;

2) to identify essential concepts in driving simulation in order to build a Knowledge Base for scenario orchestration in driving simulation;

1.4 Outline of the Thesis

The structure of this thesis is as follows:

- Chapter 2 includes a review of existing techniques of scenario orchestration in driving simulation and related areas. It serves the purpose of introducing relevant literature that has guided and inspired this research;

- Chapter 3 first includes a description of the scope of this research. followed by a detailed introduction of the essential concepts underpinning this research and a driver model prototype, leading to a general description of the solution developed in this research - a framework called SOAV (Scenario Orchestration with Autonomous simulated Vehicles);

- Chapter 4 presents an Ontology for Scenario Orchestration (OSO) in detail in order to show how scenario requirements and driving context have been modelled in this research;

- Chapter 5 presents the decision making algorithm - NAUSEA (autoNomous locAl manoeUvre and Scenario orchEstration based on automated action plAnning) and its user: the Scenario-Aware drIver modeL (SAIL). SAIL is developed based on the driver model prototype introduced in Chapter 3;

- Chapter 6 presents the implementation details regarding the virtual driver equipped with SAIL/NAUSEA and the platforms based on the simulation software of UoLDS[1] and VTI[2]'s driving simulators;

- Chapters 7 to 11.3.3 present three verification experiments and corresponding results that were used to verify the system;

- Chapter 10 presents a summary of what OSO can provide and an evaluation regarding its advantages and disadvantages;

- Chapter 11 concludes this thesis with a summary and a discussion regarding potential future enhancement of the framework and tools that have been developed.

[1]University of Leeds Driving Simulator
[2]VTI: The Swedish National Road and Transport Research Institute

Chapter 2

Related Work

What is the Matrix?
The Matrix is a computer-generated dreamworld built to keep us under control...
- The Matrix

2.1 Introduction

There are two requirements when it comes to scenarios in driving simulation: realism and repeatability.

Realism indicates the requirement of adopting sufficient driving behaviours. This term is motivated by the work from Papelis & Ahmad (2001), which states that "*In our experience with research studies in high-fidelity simulators, users generally focus their evaluation of the model realism towards the richness of the behaviors, not their fidelity.*" Hence, in this research, trajectories of each simulated vehicle are not covered, e.g., lane-changing trajectory. On the other hand, repeatability means that the essential conditions, road properties and the patterns of traffic, in a scenario should be repeatable in each trial(Fisher *et al.*, 2010b). However, the balance between realism and repeatability can be hard to define because when scenarios are running, unexpected movement of vehicles, including the participant's, may interrupt some pre-defined interactions. As a result, behaviours of each simulated vehicle are always limited in order to guarantee repeatability.

Generally speaking, scenario orchestration in driving simulation involves the control of several or even all of the simulated vehicles in the virtual environment and tries to interrupt their decision making process if they are autonomous. It is the coordination of behaviours of different simulated vehicles with internal and external constraints. The internal constraints need some behaviours to be exhibited in a pre-defined location within a period or at a specific time. The external constraints need those simulated vehicles to consider the environment geometry such as road curvature or road facilities. There are four kinds of behavioural coordination (Devillers & Donikian, 2003) regarding agents, which are some general autonomous decision makers:

1) Rules: start a script or a scenario when a specific situation happens;

2) Ambient: the agents that are evolving autonomously will respond to the reactions from the main agent, which can be the participant in driving simulation. The reactions from the main agent will be studied;

3) Goal-oriented: define the goals that the agents need to achieve;

4) Sequence of actions: define a set of actions and corresponding time schedule.

Ambient and goal-oriented coordination are not common in the area of driving simulation as they do not guarantee repeatability. In order to deal with the other two cases, a script can be used to specify what the agents should do and when they should do it. However, this may lead to some unexpected situations when there is an unpredictable human involved. For instance, a participant in driving simulation may cause the scenario to fail by some unexpected behaviours such as a sudden lane change.

In order to find a solution that can combine realism with repeatability in scenario orchestration in using autonomous simulated vehicles and deal with failures caused by unpredictable human participants, the scenario orchestration problem in driving simulation is first described in order to introduce some existing methodologies in describing scenarios and corresponding implementations using simulated vehicles. This helps find a solution that can combine realism with repeatability and deal with failures caused by unpredictable human participants.

Some limitations of those methodologies are then identified. Two areas are thereafter covered in order to find relevant solutions for the limitations: Automated Planning and Knowledge Bases. The former is covered in order to find an appropriate mechanism for scenario orchestration, while the latter is covered in order to find solutions regarding scenario descriptions.

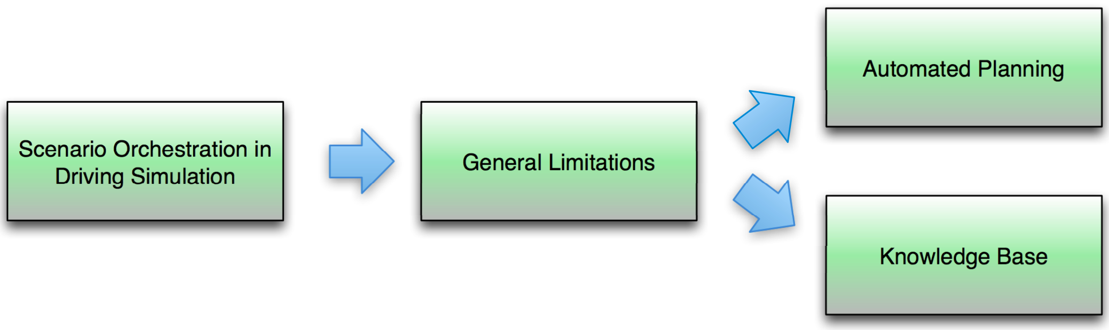


Figure 2.1: The Path of Literature Review

2.2 Scenario Orchestration in Driving Simulation

2.2.1 Methodologies

In driving simulation, orchestration of scenarios has been a focus of research since the mid 1990s regarding how scenarios can be described and how to use those descriptions (e.g., Cremer *et al.* 1995). Less effort has been put into this area in recent years since existing methodologies seem adequate enough for applications. Generally speaking, these methodologies share the same idea: have humans describe every aspect of the scenario, and then author the scenario to relevant simulated vehicles. This process can define the rules or sequence of actions that the simulated vehicles should follow when the system is offline or online.

In the rest of this section, the methodologies used in scenario orchestration will first be introduced with a focus on:

1) the complexity of available actions that can be used to direct simulated vehicles' behaviours;

2) underlying mechanisms that are used to specify when those actions should be executed and,

3) how to direct or change simulated vehicles' behaviours, i.e., how to design those simulated vehicles with appropriate interfaces for controlling.

Wolffelaar *et al.* (1999) proposed a Scenario Specification Language (SSL). With this language, all the simulated vehicles that will appear in the simulation and their corresponding actions can be defined. SSL can also be used to describe intentions of the simulated vehicles and other scenario actions, but no details were given. However, assumptions can be made that intentions may refer to the potential actions, while other scenario actions may refer to any non-driving actions, e.g., "start the simulation".

SSL describes scenarios with a number of scenario blocks and global variables. Each scenario block contains a pair of start and end conditions for activating or deactivating the corresponding scenario block. When a scenario block is activated, three sub-blocks may take effect. The Do-block specifies what should be done as long as the scenario block is still active; the Action-blocks specify some sub-scenario blocks that can also have the start and end conditions. Participant-blocks are used to define and create any number of simulated vehicles in the simulation, whose behaviours can be altered during scenarios based on conditions and actions.

SSL has managed to include some major components in generating the interactions in a scenario: 1) what should be done (actions) and 2) when the actions should be executed and finished (conditions including the start and end conditions). It therefore covers the essential part in scenario orchestration, i.e., an event-driven mechanism that tells the simulated vehicles to change their behaviour when particular events occur.

In general, SSL achieved its purpose and showed some advantages, especially in the standardization of conditions based on logical expressions and the hierarchical organisation of scenario blocks. However, Wolffelaar *et al.* (1999) did not mention how the actions being used in SSL are defined - are those actions basic such as "adopt an acceleration rate" or complex such as "follow a vehicle"? Moreover, the interfaces of simulated vehicles for action execution, i.e., behaviour modification, and underlying architecture of them were not mentioned.

In addition, as every simulated vehicle is exclusively defined and initialized in SSL, two issues may arise: 1) when simulated vehicles are large in number, describing scenarios can be time-consuming and 2) the adaptivity is in doubt as the simulation cannot put up with unexpected interrupts from the human participant.

Leitao *et al.* (1999) proposed a scripting language based on the Grafcet graphical language that describes the working sequence of the simulated vehicles in the driving simulator. Grafcet contains three elements: step, transitions and connections. A step indicates a stable state of the simulation and can be associated with actions, which indicate what should be done when a step is active. A step that is active initially is termed "initial step". A Macro step is also mentioned without details but the assumption can be made that a macro step could have sub-steps.

Transitions between steps can be specified as conditions, each of which has a Boolean function of sensors as listed in Table 2.1 and some input states such as "simulated vehicle A is overtaking simulated vehicle B". In addition, "Break(A)" could be a typo in Leitao *et al.* (1999) and should have been "Brake(A)".

Connections specify links between two or several steps. For instance, "OR Junction" indicates that when any preceding steps are active and corresponding conditions are true, the transition can then be undertaken.

The complete set of elements and graphical representation can be found in Figure 2.3.

Differently from Wolffelaar *et al.* (1999), Leitao *et al.* (1999) introduced not only Grafcet that is used to describe scenarios but also the corresponding architecture of simulated vehicles and the details of available actions that can reflect the architecture. The simulated vehicles have been designed with a three-layer driver model (Figure 2.2), which is derived from Michon (1985).

Table 2.1: Virtual Sensors for Transition Conditions (Leitao *et al.*, 1999)

Sensor	**Description**
Pos(A)	Absolute position of vehicle A
LPos(A)	Longitudinal position of vehicle A
OPos(A)	Offset (lateral) position of vehicle A
Dist(A,B)	$\|Pos(A) - Pos(B)\|$
LDist(A,B)	LPos(A)-LPos(B)
ODist(A,B)	OPos(A)-OPos(B)
Speed(A)	Velocity of vehicle A
Wheel(A)	Position of A's steering wheel
Break(A)	Position of A's brake pedal

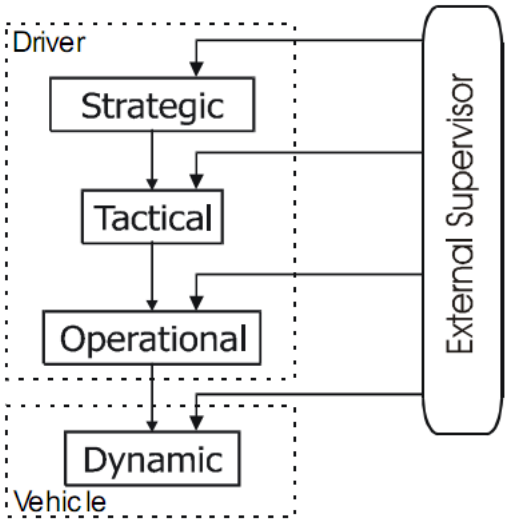


Figure 2.2: Behaviour Model of an Autonomous Vehicle in Leitao *et al.* (1999)

The Strategic layer deals with long-term goals. It plans the path for the simulated vehicle and indicates the Tactical layer of which path to follow.

The Tactical layer deals with medium-term goals. It makes decisions on how a simulated vehicles should follow a path. For instance, the Tactical layer may make decision to overtake if a slower simulated vehicle is ahead. Decisions are

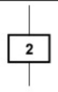
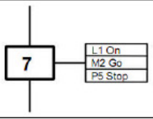
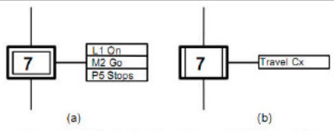
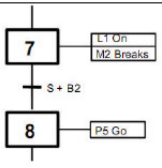
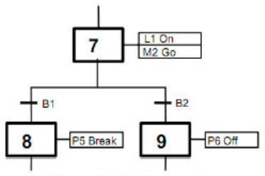
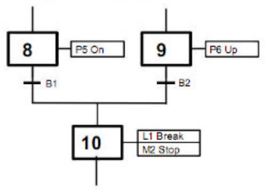
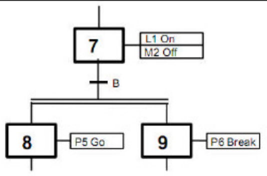
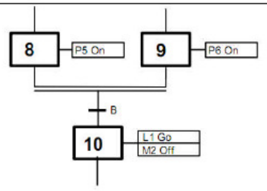
Set	Representation in Grafcet	Functions	Descriptions
Steps and Actions		A single Grafcet Step	A step represents a single stable situation and each step may be active or not
		Actions related with a step	When a step is active, the associated actions should be taken
	 Figure 4. (a) Initial active step; (b) Macro step	a) Initial active step b) Macro step	The initial active step is specified using double lines and macro step is represented using double lines and can be described in a separate diagram
Transition Conditions		Condition for the transition	A condition (S+B2) between two steps will determine the transition. It is usually written as a Boolean function of sensors and inputs state
Connections		OR Separation	When the step 7 is active, at least one condition(B1 or B2) is true, the transition will be made depending on the first condition that becomes true
		OR Junction	The transition will be made when both the preceding steps (8 and 9) are true and the conditions (B1 and B2) are both true too.
		AND Separation	Then the preceding step (7) is active and the condition (B) is true, all the destination steps (8 and 9) will be active at the same time
		AND Junction	The transition is done only when the preceding steps (8 and 9) are active and the condition (B) is true

Figure 2.3: Summary of the sets, functions and representations of the Grafcet language adopted by Leitao *et al.* (1999)

then sent to the Operational layer.

The Operational layer deals with short-term goals. It takes immediate measures based on decisions sent from the Tactical layer and makes decisions on, e.g., steering angle, which could be requested by the lane-changing decision from the Tactical layer.

As a result, based on this three-layer architecture, actions available in Gracfet reflect this three-layer architecture, so they can be used to 1) affect directly the velocity or acceleration of a simulated vehicle in the Operational layer or 2) change the driver's trip plan in the Strategic layer. These actions can change the following parameters:

- Goal velocity
- Destination
- Maximum acceleration
- Position of the steering wheel
- Position of pedals
- Power of engine
- Velocity
- Acceleration
- Position and orientation

Moreover, in order to simplify the process of describing a scenario based on Grafcet, a Grafcet editor was developed as illustrated in Figure 2.4 and provides the users with the Grafcet elements, e.g., step blocks.

Compared to SSL, Grafcet also defines action sequences based on steps along with actions and event-driven mechanisms based on transitions or connections. How to indicate which simulated vehicles will be involved is not specified in Leitao *et al.* (1999), so when there are more and more simulated vehicles involved in a scenario, the design process is likely to be time-consuming. However, the

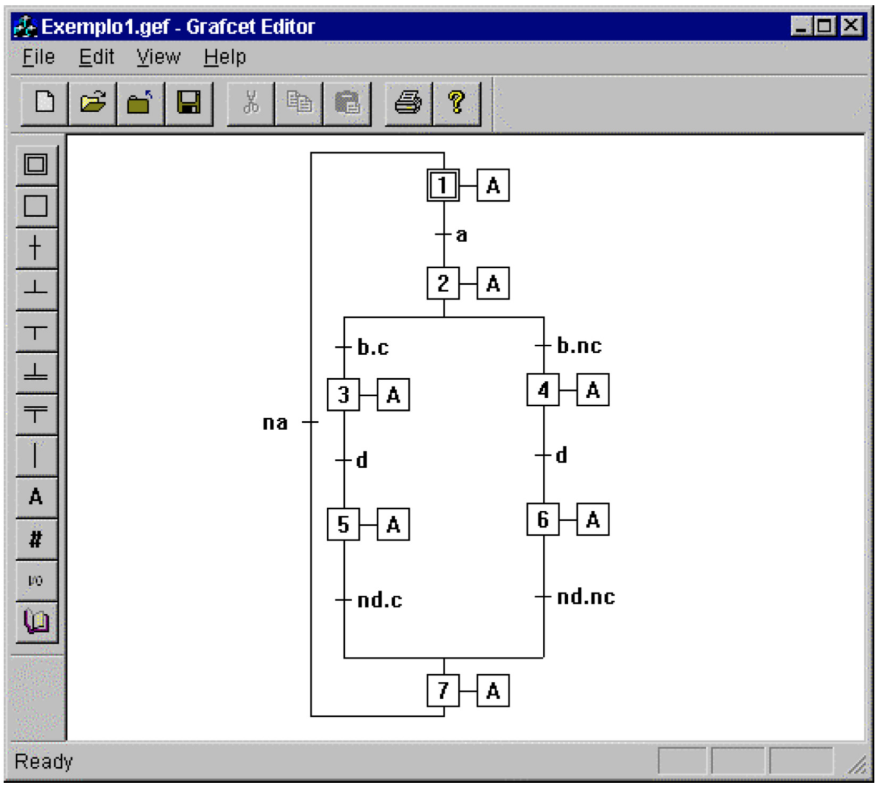


Figure 2.4: Grafect Editor ((Leitao *et al.*, 1999))

scenario orchestration mechanism in Leitao *et al.* (1999), based on Grafcet, has two advantages: 1) controlling of the simulated vehicle can be based on complex actions, i.e., some actions that need a decision making process such as going to a destination, and 2) a GUI assists the scenario orchestration process.

Similar to SSL and Grafcet mentioned above, Devillers & Donikian (2003) also developed a language to specify the action sequence of simulated vehicles in driving simulation. However, a notable difference is that those vehicles are state machine-based.

A state machine (also known as a finite state automata) treats the behaviour of the system as a combination of several states, which have their own action outputs. In a simpler version of state machine-based approach, only one state will be active at a time and one state transfers to another by evaluating the transition conditions. It is easy to design and debug state machines as they can be represented with graphical flows as shown in Figure 2.5(a). However, because the flow is unidirectional, it is very hard to use state machines to model complex systems, whose behaviours are concurrent, interdependent or goal-driven (Wright, 2000), and the number of states will grow exponentially with the number of parameters when the states have to consider many factors, which will lead to a state explosion problem. By contrast, hierarchical and concurrent state machines (e.g., Cremer *et al.* 1995) can have several state machines run at a time.

As illustrated in Figure 2.5(b), several concurrent state machines can run at each step, e.g., state machine for speed control and for steering. A conflict-resolution mechanism can be developed to address the problem of state conflict. A function can be used to evaluate the outputs from different states and generate a final output according to a resolution criteria. As an example, final acceleration rates could be chosen according to the "safe driving principle" from "Speed Control" and "Steering" state machines from Figure 2.5(b), so a lower acceleration rate could be selected.

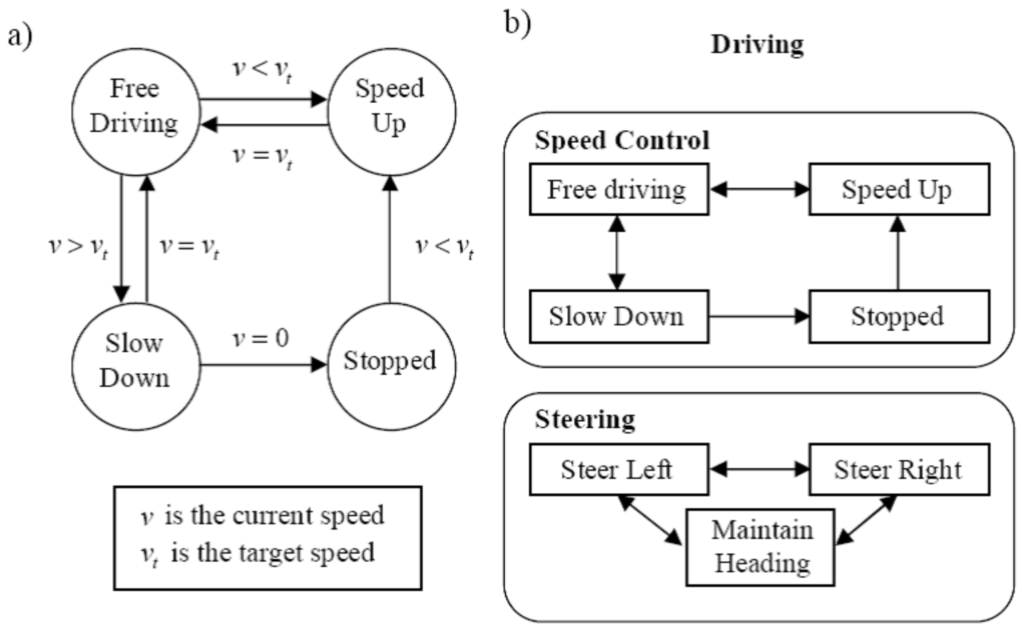


Figure 2.5: State Machine models of driving behaviour (Wright, 2000)

Based on state machines, Devillers & Donikian (2003) used HPTS (Hierarchical Parallel Transition Systems) to design simulated vehicles. HPTS has been designed as a blackbox that has input/output data and control parameters. Specifically, an HPTS has (Donikian, 2001):

- a set of sub-state machines that can nest other parallel state machines;
- an activity function that determines the status of a state by considering the input, control parameters and local variables;
- input and output. These are the input and output signals and the output indicates the proposed action generated by the state machines;
- local variables that are defined within one state machine. These can be initialized on every activation or remain unchanged between activations;
- control parameters that are used to change the behaviour of an entity modelled by state machines. They can be internal or external decisions, e.g., notify some sub-state machines that they have been started;
- an integration function that determines the final output by considering outputs from several concurrent state machines. It is used to unify the output of a state machine.

By using the control parameters, autonomous driving decisions can be overridden and scenarios can be orchestrated by specifying the sequence of sub-scenarios that contain actions, which is similar to the method of Leitao *et al.* (1999), who also developed action sequences to control simulated vehicles. However, Leitao *et al.* (1999) did not adopt temporal concepts while in Devillers & Donikian (2003), delays and durations are explicitly expressed and handled by HPTS.

Cremer *et al.* (1995) also developed simulated vehicles based on state machines - Hierarchical Concurrent State Machines (HCSMs). Apart from state machines that are designed for autonomous driving (e.g., speed control), control panels are also developed to be the standardized communication interface for scenario coordination by receiving external instructions. They contain *Dial* and *Button*. The former is used in order to execute some actions, e.g., "turn right", while the latter is set in order to provide some reference values for actions. Furthermore, Beacons are used to send messages to nearby simulated vehicles, which is a flexible way of coordinating their behaviours, e.g., a Beacon can be placed on the road and instruct a simulated vehicle just passing to decelerate. By using Beacons, simulated vehicles that will be used for some interactions can be recruited dynamically.

Willemsen (2000) proposed SDL - Scenario Description Language, an interpreted language that is translated to HCSMs and executed with the whole simulation. In SDL, there are two essential components: activities specify commands that should be executed in the simulation, e.g., find a location and create a pedestrian at that location; monitors on the other hand specify the time and frequency of carrying out activities.

In Papelis *et al.* (2001), details regarding the scenario orchestration with HCSMs are given, hence, strategies to implement activities of simulated vehicles and monitors are implied by this paper:

1. What are available simulated vehicles that can be controlled in the simulation?

   There are three types: DDOs, DDDOs and ADOs. DDOs are Deterministic Dynamic Objects. Each DDO's path has been pre-scripted and its velocity is also pre-scripted along the path. DDDOs are Dependent DDOs. Each DDDO's path is also pre-scripted but it will change its velocity based on other entities' movements. ADOs are Autonomous Dynamic Objects that are equipped with HCSM. Each ADO is equipped with some parallel state machines that are able to perform the following behaviours respectively: Following, Lane Tracking, Speed Control, Lane Change and Intersection Navigation. Apart from the behaviours, control panels are also provided in ADOs to control their behaviours with external actions and corresponding reference values for those actions.

2. What actions used for activities are available for a single vehicle?

   With the control panels, the following actions are available: change lane to the left, change lane to the right, turn to the leftmost road at the next junction, turn to the rightmost road at the next junction, generate sound and activate light(s). The following reference values are also available: target speed to be achieved, forced velocity to be achieved as quickly as possible, sound to be generated and light(s) to be activated.

3. When should these actions be activated and how monitors can be used?

   Three virtual entities are involved in coordinating the behaviours of entities in simulation: trigger, traffic light manager and traffic manager. Triggers relate conditions to actions, i.e., monitors with activities in Willemsen (2000). They are used to fire some specific actions that are non-driving actions or actions based on the basic actions for each ADO. Conditions that can be used to fire those actions are predicates based on a timer, a specific position/region, time to arrival at a target point or a traffic light status.

The HCSM-based scenario orchestration method has established a system that covers available objects for scenarios, available actions for single objects and a standardized event-driven mechanism with a list of triggers. It also includes a GUI named ISAT (Interactive Scenario Authoring Tool) to assist the scenario orchestration process as illustrated in Figure 2.6.

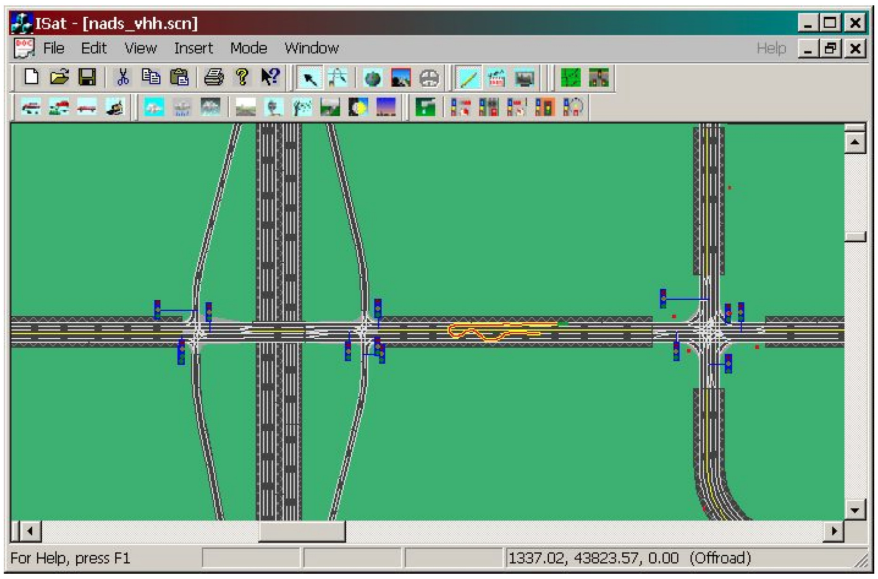


Figure 2.6: ISAT (Interactive Scenario Authoring Tool) (Papelis *et al.*, 2001)

The HCSM-based approach still focuses on orchestrating action sequences that could be interrupted by unexpected reactions from the participants.

2.2.2 Summary of Methodologies

Generally speaking, three questions need to be answered in order to orchestrate a particular scenario:

1) which simulated vehicles will be involved in the scenario?

2) how will those simulated vehicles be prepared?

3) how can the scenario be produced?

Question one is known as the actor management problem. Simulated vehicles can be pre-defined and fully described beforehand (e.g., Wolffelaar *et al.* (1999)). However, some existing methodologies can recruit simulated vehicles for interactions in real-time. For instance, Kearney *et al.* (1999) developed a HTPS-based scenario orchestration language that does not need to specify the simulated vehicles to be involved in an interaction beforehand, which implies that recruitment can be done online. In Olstam *et al.* (2011), an actor management algorithm was developed by considering the average speeds of simulated vehicles needed to reach the proposed location for interactions. A simulated vehicle with an inconspicuous speed trajectory can be assigned to interact with the participant.

Question two is known as the actor preparation problem and is related to the "realism" part of the scenario, as in this phase simulated vehicles can have some autonomy to prepare themselves. In general, researchers always assume that simulated vehicles will show up in the right place at the right time. In Olstam *et al.* (2011), some work was done regarding how to prepare simulated vehicles in an inconspicuous manner by adopting a specific speed trajectory towards the required location for interactions. Moreover, the trajectory is calculated by considering the estimated speed of the simulator driver, which is estimated by considering road conditions (e.g., road width), average speed of the traffic flow, speed limit and a parameter Q indicating the rotation of the speed distribution curve, for instance, "*$Q < 1$ imply that a vehicle with a high basic desired speed reduces its speed more than vehicles with a lower basic desired speed*" (Olstam *et al.*, 2008).

Question three is called the scenario execution problem and is related to three aspects: what are available actions, how to trigger an action and how to schedule those actions. In previous work (e.g., Papelis *et al.* (2001)), the event-driven mechanism was widely used and can be regarded as pre-conditions for particular actions: conditions must be true before these actions are applied. Due to the adoption of autonomous simulated vehicles, actions can be high-level or low level: researchers can change a simulated vehicle's destination (Leitao *et al.*, 1999) or just its action speed (Papelis *et al.*, 2001). Moreover, the scheduling of such actions is often crafted by humans in order to specify the execution order of actions or sub-scenarios (e.g., Leitao *et al.* (1999)).

None of these questions have covered scenario failures, which may force the simulated vehicles to take part in the failed interaction again. However, this is questionable not only because retries may not be allowed by scenario design, but also because it may be impossible technologically due to the lack of replanning capability or dynamic environment reconstruction, e.g., creation of new road segments.

2.2.3 General Limitations

The methodologies used for scenario orchestration are pre-defined, which means that the simulator users have to predict what will happen in the simulation and then coordinate the simulated vehicles accordingly. Some scenario development issues may arise (Papelis *et al.*, 2003), such as the following examples:

- Pre-Run Issues:
  - the experimenter describes interaction outcomes without corresponding context and,
  - predict the wrong reactions from participants.
- Run-Time Issues:
  - participants do not want to be engaged in some interactions and,

– some interactions never happen due to design or system issues.

Pre-run issues suggest that an efficient communication mechanism is lacking. Experimenters may fail to foresee the extra actions from a participant as they focus on outcomes only; the experimenters do not have a general picture of the capacities of the scenario orchestration mechanism and the simulation software. This communication problem can make the scenario orchestration process time-consuming and make the scenario liable to run-time issues, so there needs to be some way to share knowledge such that both experimenters and programmers can have some pre-knowledge, which includes, but is not limited to, the capacities of the driving simulator software, the potential pre-conditions and the effects of particular interactions.

Run-time issues invariably cause failures, e.g., a trigger does not fire or participants do not want to be engaged. Those failures are hard to avoid, but there is a possibility that they can be fixed dynamically without help from humans by automatically re-orchestrating the scenarios, if permitted.

In order to deal with the four issues mentioned above, both run-time and pre-run ones, some literature has been covered in order to answer the following two questions: 1) how to make decisions based on proposed interactions and deal with failures and 2) how to represent knowledge based on a scenario description scheme that can concentrate on essential contextual knowledge of scenario orchestration and scenario sharing in the future.

2.3 Automated Planning

In this section, some algorithms used in driving simulation and Autonomous Land Vehicles to design autonomous vehicles are covered. Some domain-independent algorithms that have inspired this research and provided the way of combining existing autonomous planning in driving simulation with scenario instructions are also examined.

2.3.1 Domain-Dependent Planning

Domain-dependent planning is used in dynamic environments where the states of the environment cannot be completely known/predicted. Generally speaking, domain-dependent planning is always used to plan missions (Ghallab *et al.*, 2004), e.g., path planning or overtaking. In this section, domain-dependent planning in two related domains will be introduced: driving simulation and Autonomous Land Vehicle.

2.3.1.1 Autonomous Vehicles in Driving Simulation

In order to present participants with a natural driving context, simulated vehicles are being designed with their own decision-making facilities. Researchers who focus on generating realistic traffic flows have attempted to adopt microsimulation models into driving simulation so that those simulated vehicles, or more precisely, vehicle-driver units, can make driving decisions according to the corresponding driving context. For instance, the simulated vehicles in Olstam *et al.* (2011) used the HDM/IDM car-following model (Treiber *et al.*, 2000, 2006).

The following part in this section will talk about the main modelling approaches in driving simulation in detail, which are used these days to adopt relevant algorithms, e.g., the microsimulation models mentioned above. It should be noted that those approaches are not tied to only one platform; they have been combined together in many platforms to balance their strengths and weaknesses. State machine-based approaches have been covered in Section 2.2.1, so the other two modelling approaches will be included in this section: Rule-based and Eco-resolution principle.

2.3.1.1.1 Rule-based

The rule-based approach is a way of making expert experiences or knowledge into some rules: if (condition) then (action). Hence, when it comes to the process of searching an action, the rule-based system will explore a set of rules and test the condition of each one. When the condition is true, the corresponding rule will be marked as true and the action will be triggered, e.g. in order to model the free driving behaviour, the following three rules can be developed: (example source: (Olstam, 2009))

- If (speed <desired speed) then (increase speed)
- If (speed >desired speed) then (decrease speed)
- If (new speed limit) then (change desired speed)

In order to adopt the property of "unpredictability", a value of certainty that is the likelihood of the action happening in some situations, can be incorporated into rule-based systems (e.g. Wright, 2000). To resolve the conflict, i.e. when choosing a speed from different rules, rule-based approaches always use the following three methods:

- A resolution criteria, e.g. the requirement of safe driving will lead to a lower speed choice;
- Priority assignment, e.g. if the acceleration rate generated by car-following rules has a higher priority than the one generated by lane-changing rules, the final result concerning acceleration rate will be the one from car-following rules and,
- The value of certainty that can be obtained by assessing the weighted average of the outcomes of different rules.

As a simple, flexible, even intelligent method, rule-based approaches are widely used these days (see Al-Shihabi & Mourant, 2003; Salvucci *et al.*, 2001; Wright, 2000 for examples). However, their disadvantage is apparent: rule-based systems have the problem of rule-explosion, which involves a situation in which the rules grow exponentially when new parameters or processes are imported. This makes this approach less useful in some complex situations.

2.3.1.1.2 Eco-resolution Principle

The Eco-resolution principle model was developed by the French research institute INRETS in the architecture named ARCHISIM, which has been used in driving simulation (El Hadouaj & Espié, 2002; Espié *et al.*, 2007). The driver's behaviour is based on some principles, which state that the driver always attempts to minimize his/her interactions with the environment. As an example, when a simulated vehicle driven by an eco-resolution-based driver is following a preceding simulated vehicle, three elements can decide the behaviour of the driver in this situation (Espié *et al.*, 2007),

- The possible interaction with other drivers or objects that are represented by a parameter "Interaction". This element is evaluated as a Boolean parameter (True or False);
- The duration of the interaction that is represented by a parameter "duration" with its degree (long or short).This is described by the time when the interaction could disappear and,
- The possibility of suppressing the interaction, which is represented by a parameter "suppression possibility" with degree/level.

The lane-changing behaviour can be encoded by using the eco-resolution principle as follows (Espié *et al.*, 2007):

Interaction (True) + duration (Long) + Suppression possibility (High) => interaction suppression (change lane)

Interaction (True) + duration (Short)+ Suppression possibility (High) => Short term adaptation (stay in the current lane)

Interaction (True) + duration (Long) + Suppression possibility (Low) => Long term adaptation (keep following)

That is to say, e.g. according to the first rule, the three elements are evaluated one by one,

- the interaction is possible (True);
- the duration is long because the preceding vehicle will keep its speed without changing lane;

- the suppression possibility is high because if the vehicle changes lane, the interaction with the present preceding vehicle will cease.

Therefore, changing lane will be chosen. During this process, the principle of "less interaction" is applied.

This modelling approach is a special case of the Rule-based approach with a dedicated set of rules based on psychological findings.

2.3.1.1.3 Enhancement from Traffic Simulation

Modelling realistic simulated vehicles is also a concern in traffic simulation, which can be adopted by driving simulation. For example, Wright (2000) used a probability function to assign virtual drivers with personalities, which have been used in a framework named DRIVERSIM in driving simulation. More recently, Lacroix *et al.* (2009) used a norm model to generate the profile of virtual drivers' personalities by adopting different parameter values, e.g. desired speed. Fuzzy logic is the extension of standard Boolean operators to fuzzy sets. It leads to the notion of a fuzzy rule, e.g., if (speed is high) then (use higher speed). The condition and actions are all based on fuzzy sets, which are different from the classic rules in rule-based approaches. However, this "fuzzy rules" approach is a good way of modelling human reasoning and decision-making processes, so fuzzy techniques can be complementary to other methods. Wright *et al.* (2002) used this method to model the choice generation process combined with rule-based approaches. Some researchers have adopted a priority-based approach to solve the intersection conflict problem by making the simulated vehicles negotiate their priorities in advance (Champion *et al.*, 2001; Doniec *et al.*, 2006, 2008).

2.3.1.2 Autonomous Land Vehicle

Planning algorithms for driving are also being examined in the area of Autonomous Land Vehicle (ALV), which puts a focus on path planning, mission planning and navigation. Path planning algorithms can be found in a wide body of literature. In Miller *et al.* (2009), the Team Cornell's Skynet, competing in the 2007 DARPA urban challenge, used the A* algorithm(Cormen *et al.*, 2001) to plan their ALV's path. Mission planning is used to generate a realistic trajectory for the vehicle locally from one point to another, an example algorithm can be found in Miller *et al.* (2009). Finally, navigation is mainly used for obstacle avoidance and some classic algorithms, e.g., potential fields, can be used (e.g., Khatib (1986)).

Algorithms in ALV can provide some insight for the design of simulated vehicles in driving simulation to provide some level of autonomy for scenario orchestration:

- Path planning can be used to enhance the route planning ability of vehicles in driving simulation. However, due to the simplicity of road networks and the fact that participants are often driving a fixed route, path planning in driving simulation may not be a priority;
- Mission planning can be adopted to generate different sets of rules in driving simulation to represent different contexts when making decisions, e.g., when crossing a junction, vehicles can have different rules and driving styles.
- Navigation algorithms can be enhanced in driving simulation by considering similar algorithms in ALV, e.g., obstacle avoidance algorithms based on potential fields.

Simulated vehicles in driving simulation concentrate on making the correct decision. However, ALV concentrates not only on making the correct decision but also on generating feasible trajectories. Moreover, the environment in which the simulated vehicles are evolving is virtual and predictable and is thus simpler than that which ALVs are in.

2.3.2 Domain-Independent Planning

Domain-independent planning, unlike its domain-dependent counterpart, concentrates on general models. By applying search-based algorithms, a plan of actions can be developed.

Discussions regarding the use of domain-independent planning algorithms in real-world problems have been covered by many researchers (e.g., Musliner *et al.*

(1995); Wilkins *et al.* (2001)). Generally speaking, real-world problems require *"numerical reasoning, concurrent actions, context-dependent effects, interaction with users, execution monitoring, replanning, and scalability"* (Wilkins *et al.*, 2001).

Because of its capability to handle the features above, HTN (Hierarchical Task Network) has been demonstrated as a real-world problem solver (Wilkins *et al.*, 2001). It specifies the tasks that a decision maker or an agent should carry out instead of specifying the goal state that the agent needs to drive the system towards.

As elaborated in Ghallab *et al.* (2004), a Task Network contains a set of tasks and a set of constraints:

1. A *task* is an action along with its executor - an agent or a set of agents.
2. *precedence constraints*: expressions used to specify the relationships between two time intervals. For instance, "task one *before* task two" means that task one should be carried out before task two starts;
3. *before-constraints*: expressions used to specify the conditions that should be true in the state before a task is carried out;
4. *after-constraints*: expressions used to specify the conditions that should be true in the state after a task has been carried out;
5. *between-constraints*: expressions used to specify the conditions that should be true in the state between two tasks.

An HTN method is a set of basic tasks used to perform a complex task, which cannot be executed in one way, in one sequence or by one agent. It contains subtasks and related constraints. Therefore, an HTN problem contains the initial state, the initial task network, a set of basic actions and a set of methods. By finding a set of basic actions that can decompose all complex tasks in the initial task network into basic ones based on the set of methods, an HTN problem can be solved and a plan can be generated. This plan will be a set of basic tasks with relevant constraints, e.g., precedence constraints and before-constraints. Hence, by having the final task network based on decomposition, an agent or a set of agents can be notified of what should be done (actions/tasks) and when these tasks should be carried out (before-constraints and precedence-constraints).

Many HTN-based planners have been developed from as early as the 1970s: NOAH (Sacerdoti, 1975), NONLIN (Tate, 1977), SIPE-2 (Wilkins, 1991), O-Plan (Currie & Tate, 1991), O-Plan2 (Tate *et al.*, 1994), UMCP (Erol *et al.*, 1994) and SHOP2 (Nau *et al.*, 2003). They have been used to tackle real-world problems, e.g., in Drabble *et al.* (1997), O-Plan was used to support spacecraft assembly, integration and verification. However, planner users need to work out a description of the domain of interest and a complete list of available tasks, methods, before-constraints, etc. in order to indicate what to plan on and how to plan. This process can be time-consuming.

In Morisset & Ghallab (2008), HTN was used to build robots' skills, which can be further decomposed into low-level actions available for the robot. However, this HTN-planner for skills lacks a temporal reasoning algorithm, although it does demonstrate the possibility of using HTN to guide robots' task selection in real-world problems. Hadad *et al.* (2003) developed an easy-to-use temporal reasoning algorithm to deal with cooperative planning and scheduling. A plan of actions containing temporal information and parallel actions can be generated. A scheduling process is then used to make sure that the maximum lateness of a plan S is less than or equal to zero. The lateness of a plan is calculated as $max_i\{f_{\beta_i} - d_{\beta_i}\}$, where f_{β_i} refers to the finish time of the execution of action $\beta_i \in S$ and d_{β_i} refers to the proposed deadline of action β_i.

Hadad *et al.* (2003) divided actions into two categories: basic and complex. A recipe is set of basic actions that can be carried out to finish a complex action. Agents need to finish a complex action called α, whose recipe can be as illustrated in Figure 2.7, indicating a set of basic actions that are needed to finish action α. T_1 represents action β_1 carried out by agent no.1; T_2 represents action β_2 carried out by agent no.2 and T_3 represents action β_3 carried out by agent no.1.

There are also some temporal constraints. For instance, some example constraints from Hadad *et al.* (2003) can be used, such as:

- Precedence Constraints:

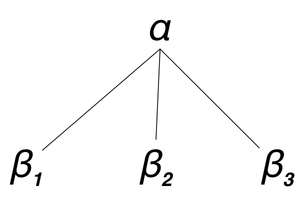


Figure 2.7: Recipe Tree of Top Action α

  - β_1 before β_2;
  - β_1 before β_3;

- Metric Constraints:
  - 0 minute $\leqslant$ start time of T_3 - finish time of T_1 $\leqslant$ 60 minutes;

  That is,

- $0 \leqslant s_{\beta_3} - f_{\beta_1} \leqslant 60$.

A temporal constraint graph Gr_α can be established as illustrated in Figure 2.8(a) by using all the metric constraints from Hadad *et al.* (2003). $s_{\alpha plan}$ represents the instant when the planning begins.

Then temporal reasoning on Gr_α can be based on the Floyd Warshall algorithm (Cormen *et al.*, 2001), which will generate a new graph from Figure 2.8(a) to Figure 2.8(b). If the plan is inconsistent (minimum distance between the same node in the graph is smaller than zero), a backtracking algorithm will be invoked to delete newly added recipes and build another plan based on new ones. The scheduling component will try to schedule those actions based on the lateness and notify the planning algorithm if the plan has failed because of a lateness greater than zero. Because the planning algorithm sends out actions to scheduling components before the whole plan finishes, the planning component will not need to regenerate the whole plan if backtracking is needed to cope with failures.

Although the algorithm from Hadad *et al.* (2003) lacks pre-conditions or before-constraints, which are useful in driving simulation to adopt event-driven coordination, the temporal reasoning procedure with dynamic action planning

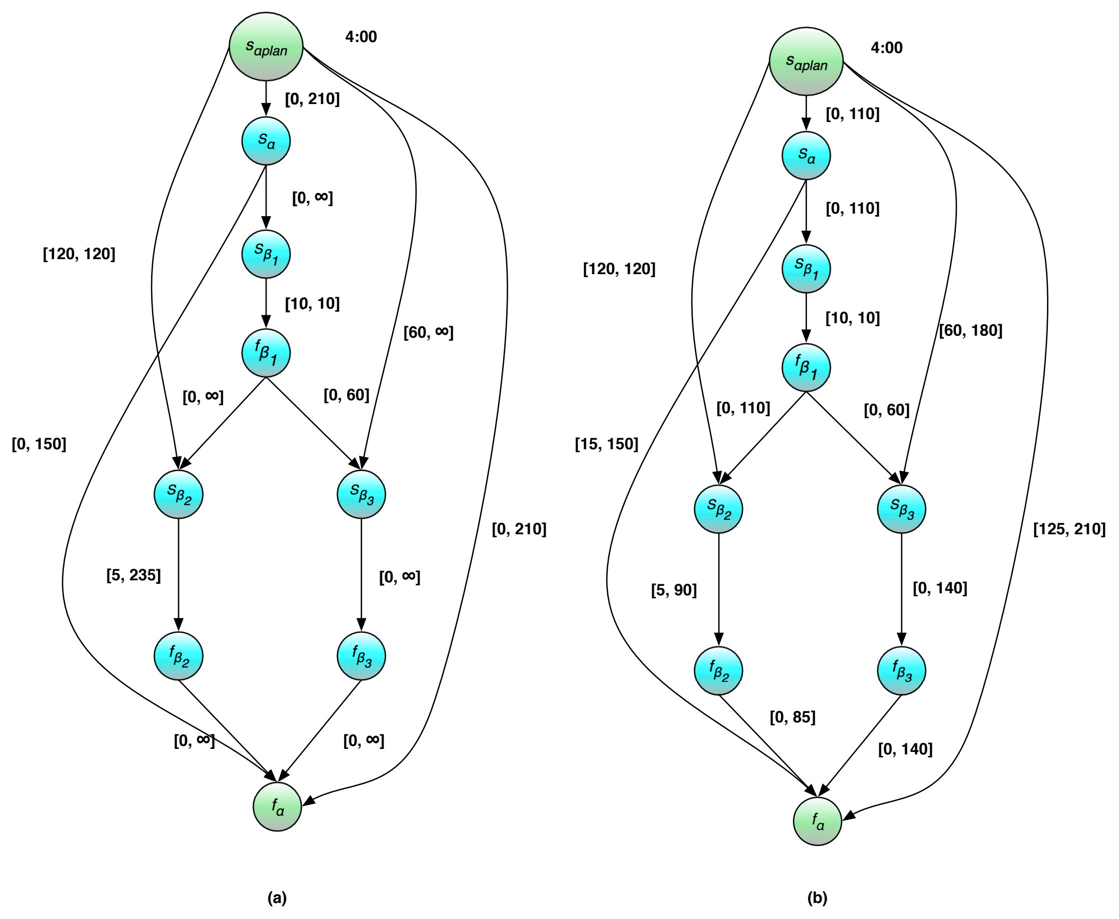


Figure 2.8: Temporal Constraint Graph Gr_{α} from (Hadad *et al.*, 2003)

generation is still inspiring, as it interleaves action planning with real-time scheduling in an easy-to-use manner.

2.4 Knowledge Bases

Human-readable Knowledge Bases exist everywhere: journal articles, conference papers, books, etc., so when it comes to building a Knowledge Base, a combination of human-readable knowledge is a good start. In Fisher *et al.* (2010a), some background knowledge regarding driving simulation has been given. A Wikipedia of Driving (Driving Wiki, 2010a) was also developed to share some thoughts in driving simulation ranging from standard scenarios to standard measures, e.g., mean speed. However, as some informal ways of summarising or sharing knowledge, both have the following shortcomings:

- No uniform definitions of terms: terms have been defined for specific application and no standard has been worked out, e.g., the definition of "scenario"(Fisher *et al.*, 2010b);
- No extraction of essential knowledge: knowledge is expressed in plain language so an additional process is needed to extract relevant information from those descriptions, especially when it comes to scenario sharing among different groups;
- No mechanism of machine processing: knowledge is human-readable but not machine-processable.

Knowledge in driving simulation covers a variety of areas such as road safety and simulation technology, so extracting relevant information in driving simulation is hard. None of the representation methodologies mentioned in Section 2.2.1 are suitable to encode a human-readable knowledge base, because they are programming language-based, thus not suitable for some people who have no programming background.

When any knowledge in driving simulation is used, e.g., different research organisations are working on the same project, the communication will be based on several discussions and recognised knowledge sharing schemes, e.g., tables, figures and so on. In addition, it is impossible for a machine to process such information in a simple fashion.

By contrast, different machine-readable programming languages for scenario orchestration have been developed since 1990s, e.g., SSL (Wolffelaar *et al.*, 1999), but they have the same defect: when they specify actions for simulated vehicles, they always include driving context with what to do, how to do and when to do it in one file. In this case, a machine-readable Knowledge Base has much platform-dependent information. It is not easy to read such a Knowledge Base and extract relevant scenario information. Hence, a programming language-independent way could be used in this circumstance.

Ontologies have been used for a variety of purposes (Corcho *et al.*, 2003), e.g., natural language processing (Dahlgren, 1995). They can be used to describe concepts and their relationships in a domain of interest. There are three major components in an ontology: classes, properties and individuals.

Classes specify the concepts in a domain such as *Vehicle* or *Pedestrian*. Narrowed-down concepts can be described by subclasses, e.g., *Car* is a subclass of *Vehicle*.

Individuals specify the specialised instances of classes. For instance, an individual of the class *Vehicle* can be *vehicle_XXXXXX*, which is a vehicle with a license plate number "XXXXXX".

Properties specify the characteristics of classes, therefore their individuals should be endowed with the same features. For instance, property *hasColour* can be used to indicate the colour of a vehicle, which is an individual of the class *Vehicle*. There are four types of properties (Noy & Mcguinness, 2001): 1) Intrinsic properties, e.g., *hasMaxSpeed* of a vehicle; 2) extrinsic properties, e.g., *hasVehicleModel* of a vehicle; 3) parts, e.g., *hasLane* of a road and 4) Relationships between two individuals from two classes, which are not necessarily the same. Therefore, a property can have two types of values: individuals or value types. An object property specifies the relationship between two individuals and a data property specifies the relationship between an individual and a data type, which can be a string, Boolean, float, etc. For instance, *hasMaxSpeed* is a data property that specifies the relationship between a vehicle and a float value; *hasLane* is an object property that specifies the relationship between a road and a lane.

Properties also have a domain and a range. The domain of a property indicates the class that the property describes. The range of a property indicates the allowed classes that the related individuals can be instantiated from. For instance, the domain of property *hasLane* can be the class *Road*, while the range of *hasLane* can be the class *Lane*.

Property cardinality is proposed to describe how many values a property could have, so a property can be used more than once to describe a class, e.g., *hasLane* can be used more than once to specify the lanes that a specific road possesses.

As a way of describing concepts and their relationships in a domain of interest in a programming language-independent and logic-based manner, ontologies have been used to:

- identify common knowledge in a domain (knowledge representation);
- share common knowledge in a domain (information exchange) and,
- reason about or analyse domain knowledge (knowledge reasoning).

As a result, ontologies have been used to model the driving context for two reasons: 1) representing driving context for information exchange between vehicles (Knowledge Representation) and 2) assisting decision making by providing spatio-temporal reasoning (Knowledge Reasoning). Fuchs *et al.* (2008) elaborated a model for scene representation for DAS (Driver Assistance Systems) and included not only spatio-temporal representation, but also some additional information such as traffic rules. However, this ontology is still in its early phase as "*The ontology was developed with the main intention of providing a framework for description of traffic scenarios...*" (Fuchs *et al.*, 2008). In addition, ontologies can also be used to assist path planning as demonstrated in Provine *et al.* (2004) to assist obstacle reasoning. Recently, Hamilton *et al.* (2013) used ontologies to assist multi-vehicle collaboration in route determination, in order to deal with intersection negotiation and safe-headway maintenance.

2.5 Summary

As discussed in Section 2.2.1, repeatability is needed, so orchestrating scenarios with human-crated, pre-scheduled action sequence is acceptable and should not be a problem, however, if a failure happens, there is no way to restore the situation as those simulated vehicles are not aware of what they should do. They are not autonomously navigating in the world because they need to generate some interactions.

Simulator users tend to specify every aspect of the scenario and assume that all the simulated vehicles can be prepared by pre-defined instructions and always appear in the right place, or their trajectories after creation can be easily manipulated. Hence, not many researchers have attempted to answer the question of "if we really want realistic scenarios, what do we really need?" The answer should not be autonomous simulated vehicles that can be controlled when some conditions are true, but the ones that know their tasks and finish the scenario autonomously by carrying out those tasks. Although the algorithms mentioned in Section 2.3.1 are useful in modelling the decision making process of a virtual driver that can be embedded into the simulated vehicles, they still lack the mechanism of being intelligent enough to "understand" interactions and commit to them. As a result, this research attempts to find a way to let simulated vehicles "know" what to do, so that they can prepare for and finish what they should do.

Moreover, scenario implementation is associated with specific platforms, which makes the sharing of scenarios difficult, as scenario descriptions need to be extracted and reprogrammed in order to reuse them in a different simulator. Furthermore, as the simulated vehicles need to know what is "known" to them, a Knowledge Base is needed.

A language-independent Knowledge Base is desired with a focus on the driving context and the instructions simulated vehicles should follow during a scenario, i.e. relevant knowledge for scenario-directed driving. This Knowledge Base can be a start to make the scenario knowledge reusable among different simulation software regardless of any specific platforms.

This research will explore the application of HTN to utilise a pre-scheduled plan for interactions. Simulated vehicles' behaviours can, therefore, be regulated to generate required interactions with the capability of retrying those interactions if failures occur. This research will also examine the application of ontologies in scenario orchestration to provide a standard scenario description, containing contextual information useful for decision making based on proposed interactions.

Chapter 3 will introduce a framework developed to deal with the scenario orchestration limitations discussed in this chapter, and introduce the pre-scheduled plan. More details will then follow regarding some major components of the framework. Verifications will follow those details.

Chapter 3

Framework Description - SOAV

I imagine that right now, you're feeling a bit like Alice. Hmm? Tumbling down the rabbit hole?
- The Matrix

3.1 Introduction

This research attempts to follow a path that will lead to a standardised Knowledge Base based on an ontology, an algorithm based on HTN (Hierarchical Task Network), a driver model equipped with this algorithm and finally some other modules that support the driver model, e.g., a module that interprets orders from the driver model. The path is illustrated in Figure 3.1.

This research was planned in four phases, the initial Literature Review phase has been covered in the previous chapter. The subsequent Solution Prototyping phase identifies some concepts that can shape the design of the algorithm and a driver model that can be used to support this algorithm for testing. The Solution Generation phase utilises the concepts and the driver model to develop a decision making algorithm along with some other supporting modules. The final Verification Phase uses some experiments to verify the solution. As a result, this plan should lead not only to an algorithm, but also to a framework for scenario orchestration in driving simulation. In this Chapter, the following contents will be introduced sequentially:

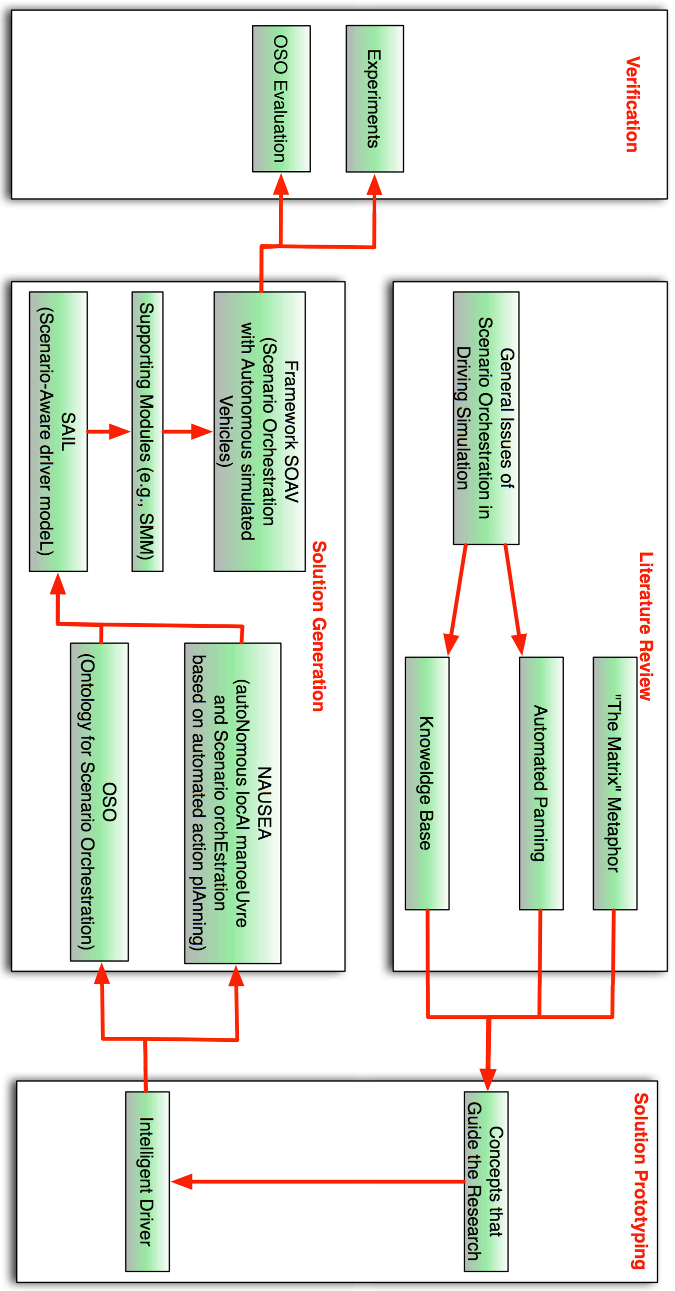

Verification
Experiments
OSO Evaluation
Literature Review
General Issues of Scenario Orchestration in Driving Simulation
"The Matrix" Metaphor
Automated Panning
Knoweldge Base
Solution Generation
Framework SOAV (Scenario Orchestration with Autonomous simulated Vehicles)
Supporting Modules (e.g., SMM)
SAIL (Scenario-Aware driver modeL)
NAUSEA (autoNomous locAl manoeUvre and Scenario orchEstration based on automated action plAnning)
OSO (Ontology for Scenario Orchestration)
Solution Prototyping
Concepts that Guide the Research
Intelligent Driver

Figure 3.1: The Path of Research

1) The scope and objectives of the research will be elaborated;

2) Some concepts that are the basis of this research are introduced, e.g., the concept of "Assignment" has shaped the methodology in terms of how to design the decision making algorithm in the intelligent virtual driver and how to describe the scenario requirement to the virtual driver;

3) The design of an intelligent virtual driver is then included. An intelligent virtual driver is needed to provide a decision making algorithm with relevant information, e.g., how to perceive information, how to interpret this information, how to change the state of the virtual world;

4) The framework SOAV (Scenario Orchestration with Autonomous simulated Vehicles) is finally introduced. It includes four major components: an Ontology for Scenario Orchestration (OSO), virtual driver(s) equipped with SAIL/-NAUSEA and supporting modules named Scenario Management Module (SMM) and Scenario Observer. The virtual driver in SOAV contains the essential algorithm of this research - NAUSEA (autoNomous locAl manoeUvre and Scenario orchEstration based on automated action plAnning). In order to facilitate NAUSEA, a driver model SAIL (Scenario-Aware drIver modeL) was designed to adapt and support NAUSEA. When scenarios are running, a virtual driver equipped with SAIL/NAUSEA can drive one vehicle in the simulation. As the data source for the virtual driver, OSO is used to describe scenarios formally with scenario requirements and relevant driving context. SMM is the interface between the virtual driver and the simulation software and is used to interpret orders from the virtual driver, allowing the driving simulation software to execute those interpreted orders. Scenario Observer is used to record the data transferred within SOAV. SMM and Scenario Observer will be introduced in this chapter, while OSO will be introduced in Chapter 4 and SAIL/NAUSEA will be introduced in Chapter 5.

3.2 Scope and Objectives of the Research

Scenario orchestration is a difficult problem. For realism, simulated vehicles should be able to show rich behaviours according to specific circumstances; for repeatability, simulated vehicles should be controlled and predictable so that every single run of the scenario can produce similar situations for participants. Failures can occur if those simulated vehicles show inappropriate behaviours and make the measurements invalid, or the participant reacts in an unpredictable manner and destroyed the initial design of the scenario.

The causes of failures can range from implementation, such as memory leak of simulation software due to program bug, to scenario design, such as the triggers used to fire an interaction never execute, so it can be impossible to list all causes of failures and thus argue what types of failures can be fixed in this research. However, if an assumption can be made that the interactions proposed by the scenario designers are appropriate for specific measurements, e.g., an interaction caused by a breaking-down leader vehicle ahead of the participant's vehicle is appropriate for collecting measurement of driver reaction time, the outcomes of failures can be:

1) interruption of the whole scenario, in which case the program bugs should be examined and fixed;

2) the triggers are fired, but no measurements are valid. For instance, the participants react in another way that is not anticipated by scenario designers;

3) the triggers are not fired, and no measurements are taken. For instance, the participants do not want to be engaged or react in another way.

This research is concerned about the last two outcomes and in either case, a mechanism is needed to predict a failure state after human participants exposed unexpected reactions. This state becomes true when a failure happens and corresponding mechanism will be needed to re-create the interaction after the failure. This failure state has been adopted as a piece of contextual information. In this case, there is no need to specify all possible responses from human participants and causes of failures.

This research starts from here and put a focus on not only authoring scenarios before hand, but also generating scenarios, particularly the interactions, in an automated manner with the possibility of recovering scenarios from failures by considering relevant contextual information of an interaction.

Based on the research focus, three assumptions have been made:

1) the simulated vehicles are realistic enough for the verification of the framework developed in this research, i.e., they have already been equipped with sufficient tactical behaviours such as overtaking;

2) the traffic flows manipulated for each scenario will not be affected by SOAV and are sufficient for desired ambient traffic flow rate, vehicle type distribution and interactions and,

3) interaction required for measurements are independent, so recruiting simulated vehicles for each interaction with the participant's vehicle can be performed by considering only the successive interaction.

This research is also not concerned with answering every single question that could arise in orchestrating scenarios:

1) Which vehicles should interact with the participant? (actor management problem);

2) How should those vehicles go to their proposed position? (actor preparation problem) and,

3) How should those vehicles be instructed what to do when interacting with participants? (scenario execution problem).

Although the actor management, actor preparation and scenario execution procedures have been included in the algorithm developed in this research, the former two modules will not be covered thoroughly. This research will concentrate on finding a basis for representing contextual information for interactions and automated planning based on those information in scenario orchestration. It will focus on controlling one simulated vehicle in a single interaction. In general, based on the two aims specified in Chapter 1, the specific objectives of this research are:

1) to develop a mechanism that can 1) provide information of the potential participant's reactions that could cause failures and 2) guide the scenario orchestration based on the contextual information for interactions in order to answer the following questions:

    - which simulated vehicle should be involved in the coming interaction?
    - where should the simulated vehicles be driven to?
    - what should be done in order to generate the interactions?
    - when the interactions should be generated, succeed or failed?
    - what measurements should be collected?

2) to develop a programming language-independent and logic-based scenario representation mechanism, which can be used to encode the main contextual information. This mechanism should be human-readable and machine-processable;

3) to recruit simulated vehicles dynamically to generate interactions with the capability of re-generating any failed interactions in a particular scenario, which is regarded as the ability of replanning;

In order to achieve these objectives, this research first identified some concepts as specified in next Section.

3.3 Main Concepts

3.3.1 Virtual Driver

A Virtual Driver in this research indicates an intelligent controller that can be used to make driving decisions or carry out pre-scheduled actions based on scenario requirements.

Driving Decisions indicate long- or short-term decisions regarding strategic, tactical or operational actions that should be undertaken to drive safely and satisfy scenario requirements. The actions that this research will use are: overtaking, speed adaptation and lane-changing;

Pre-scheduled Actions indicate some actions required to change the states of objects in driving simulation in order to generate some interactions, e.g., request to “set the desired speed of the simulated vehicle no.1 as 30 *mph*” or request to “place cones in the road segment whose id is ‘r3.2’ ”;

That is, this Virtual Driver should be able to interfere with the simulated vehicles’ autonomous decision-making process and change the states of some objects in driving simulation, rather than the simulated vehicles themselves.

3.3.2 Time

Time can be included in terms of Instant (specific time point) and Interval (period between two Instants), which are all based on the time in the simulation. It should be provided in order to:

- evaluate the action plan for scenario orchestration to provide information regarding the proposed start or finish times of some actions;
- evaluate the suitability of particular simulated vehicles at the beginning of any scenario and decide which one should be controlled by the Virtual Driver and,
- evaluate if the action has been executed by the Virtual Driver in time.

No matter what questions can be answered by adopting temporal concepts, time should be used as a reference in driving simulation to regulate simulated vehicles’ behaviours.

3.3.3 Monitor System

Monitors are used to supervise the state variables in simulation, e.g., vehicles' speeds and positions. A Monitor System can be developed based on the work from Willemsen (2000), which identified four kinds of Monitors as illustrated in Figure 3.2.

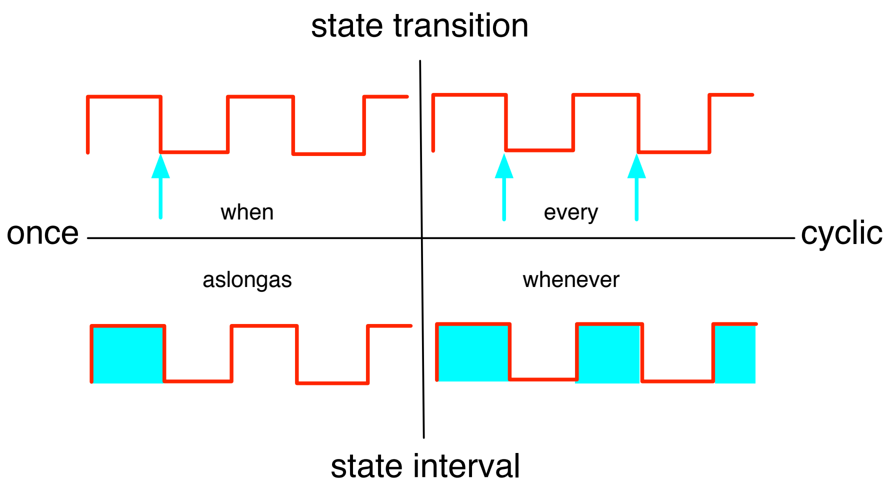


Figure 3.2: Categories of Monitor, reillustrated from Willemsen (2000)

When The Monitor will be activated once if its condition is satisfied. The condition is a state transition.

Every The Monitor will be activated repeatedly if its condition is satisfied. The condition is a state transition.

Aslongas The Monitor will be activated during the time when its condition is satisfied;

Whenever The Monitor will be activated repeatedly if its conditions is satisfied.

As a result, the Monitors can supervise state transitions happening at any Instant or state status during any Interval. The four keywords representing the four types of Monitors will be called Monitor operators in this research.

3.3.4 Flock and Ego-Vehicle/Flock

A Flock refers to a platoon of simulated vehicles. Its leader's states, e.g., position, speed, etc., will be used to represent the Flock's states in this research. A scenario can have several Flocks.

The simulated vehicle or Flock the Virtual Driver is controlling is termed the "Ego-vehicle" or the "Ego-flock" respectively. The leader of the Ego-flock is driven by a Virtual Driver, who will manage that Flock with appropriate orders. Other members of the Ego-flock will simply adopt the same orders from the Virtual Driver.

3.3.5 Action

An Action is what a Virtual Driver can do to change the state of the simulated world. Actions are divided into two categories: High-Level and Low-Level.

A Low-Level Action is defined as an Action that can only be performed in one way, one sequence and by one entity. Examples of Low-Level Actions are: "Set Steering Angle", "Set Speed", "Set Acceleration Rate", "Set Action Lane", "Create an Object" (e.g., a vehicle or a road segment), "Set Action Status", etc.

High-Level Actions need Recipes (Hadad *et al.*, 2003), each of which is a set of Low-Level Actions that specify how to perform a High-Level Action. For instance, "Block" is a High-Level Action that involves three simulated vehicles that pass the participant's vehicle to prevent it from overtaking its leader vehicle.

Each Action contains some information that can be used by a Virtual Driver: 1) the name/id of the Ego-vehicle/flock; 2) the deadline of the Action; 3) the Duration of the Action; and 4) the release time of the Action. Moreover, each Action has a type and an Action profile, e.g., "change desired speed to 10 mph" is an Action whose type is "Low-Level" and "Adapt-Speed". It has an Action profile of setting "Desired Speed" with a value of "10.0" (mph).

3.3.6 Trigger

Monitors mentioned in Section 3.3.3 can be used to establish Triggers: Preconditions, Success Conditions and Failure Conditions.

Because of the importance of Pre-conditions that can make a Virtual Driver perform particular Actions when some conditions are satisfied, the word Monitor has been used to specifically refer to Pre-conditions. Success Conditions and Failure Conditions, on the other hand, specify Triggers that can indicate whether or not an Action has succeed and failed respectively. All three kinds of Triggers should include the following information:

- if the Trigger should dictate a state or an event (state transition).
- which object the Virtual Driver should supervise, e.g., a simulated vehicle, urban area, or a Flock;
- which state variable of the object should be dictated and compared to some threshold value;
- what threshold value should be compared to and,
- how to dictate the identified state variable: *when*, *whenever*, *aslongas*, *every*, see Figure 3.2 on Page 44 for more details.

Moreover, as there are some common events in simulation, e.g., a vehicle enters or leaves a zone, Triggers can also include some Trigger type, which can be: "Cross (a pre-defined line)", "Enter (a pre-defined zone)", "Exit (a pre-defined zone)" or "Timer" (Willemsen, 2000).

3.3.7 Formation Position

Inspired by El Hadouaj *et al.* (2001), the concept of Formation Position (Figure 3.3) has been proposed to specify the spatial goal of a Virtual Driver. For instance, if the Virtual Driver's spatial goal is "Follower", then he/she will try to drive the Ego-vehicle/flock to the "Follower" position as specified in Figure 3.3. Because Formation Position is a set of pre-defined relative local positions around the simulator driver/participant's vehicle, no assumption has been made regarding the absolute spatial relationships between them. However, according to specific applications, values can be assigned in order to indicate how far those positions should be. For instance, 200 metres can be used to indicate the optimal distance between the "Leader" position to the Participant's vehicle's position.

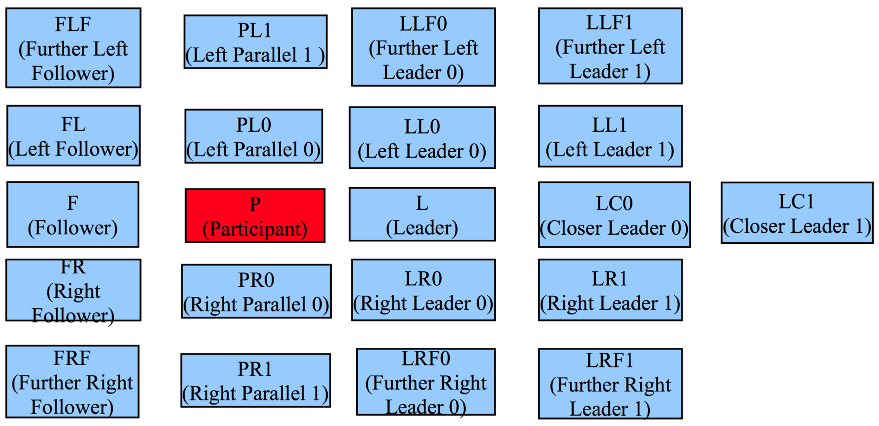


Figure 3.3: Formation Position

3.3.8 Situation

Situation refers to some common interactions that a participant will be exposed to, which involve more than one simulated vehicle, e.g. a "Blocking" Situation involves one simulated vehicle as the leader of the participant's vehicle and another three ones as a Flock that prevents the participant from overtaking the leader.

3.3.9 Role Matching

This is a process used to decide which simulated vehicle a Virtual Driver should control. It is determined by required vehicle type, Formation Position and the required speed. A successful Role Matching should identify any simulated vehicles that 1) are the required vehicle type, e.g., Volvo S40; 2) are at or near the required Formation Position and 3) have sufficient maximum speed that can reach the required Formation Position in time.

3.3.10 Assignment

An Assignment is the task that a Virtual Driver needs to carry out. Every scenario contains at least one Assignment. An Assignment includes three kinds of information: general information, Triggers and Assignment-actions.

Firstly, a Virtual Driver should know the following facts:

- the state of an Assignment, which can be:

  Initial indicates that the Triggers in the Assignment should not be monitored at present;

  Pending indicates that the Triggers in the Assignment are being dictated or should be dictated from the next decision loop;

  Failure indicates that this Assignment is failed because of the failure of its Assignment-action(s) or the Failure Conditions are true;

  Success indicates that this Assignment is now finished and has succeeded;

- any requirements regarding the Ego-vehicle/flock, e.g., required vehicle type;
- the spatial goal of the Virtual Driver where the Ego-vehicle/flock should be driven to before the execution of some Assignment. This spatial goal is based on the Formation Position in Figure 3.3;
- how many times the Virtual Driver can try to finish this Assignment;
- the pre-defined Ego-vehicle/flock for the Assignment. If it is not specified, the Virtual Driver will look for candidates dynamically.

Secondly, a Virtual Driver should know any Triggers that include 1) the Monitor that is used to activate Assignment-actions and 2) the Success and Failure Conditions that indicate the accomplishment or abortion of its parent Assignment.

Finally, a Virtual Driver should know what Actions to undertake in an Assignment, these are detailed in the Assignment-actions. Assignment-actions can be driving actions, e.g., change-lane, or non-driving Actions, e.g., “create new vehicle”, “collect driver’s reaction time (braking)”, “create a road segment”, “activate traffic flow one”, “activate Assignment two” or “activate Situation one”, etc. Assignment-actions can be triggered and executed once unless a failure has been identified. Repetition of Assignment-actions is possible by adopting an indicator stating how many times the Assignment-action can be repeated, but in this research, this is ignored.

3.3.11 General Plan

A Virtual Driver should have some plan to guide its behaviours so a concept of the General Plan is introduced. It is a temporal constraint graph Gr_α (e.g., Figure 2.8 on Page 31) and is built from several pre-defined Actions and Assignments. It contains metric and precedence constraints. For instance, "Action α *before* Action β" is a precedence constraint; " start time of β - finish time of $\alpha \leqslant 100$ (seconds)" is a metric constraint. There are two types of metric constraints: duration and delay. Delay is the period between two Instants that belong to different Intervals, while duration is the period between two Instants that are within the same Interval.

In every scenario, each Virtual Driver needs to finish a top High-Level Action α that can be either *Perform-scenario* or *Free*. *Perform-scenario* has only one Recipe that contains four sub-Actions, namely, β_0, β_1, β_2 and β_3 as shown in Figure 3.4. *Free* makes the Virtual Driver ignore any Assignments and autonomously navigate the world, in which case the route or the destination will be based on a pre-defined route. Recipes are not needed for *Free*.

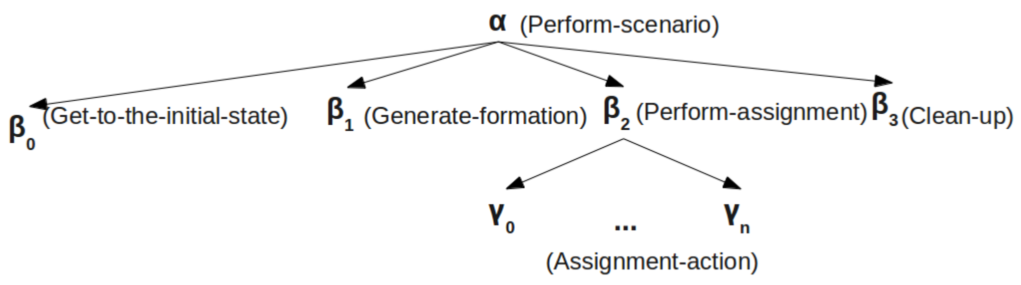


Figure 3.4: Action Recipe for the Virutal Driver (*Perform-sceanrio*)

β_0 (Get-to-the-initial-state) adopts the initial state (e.g., initial speed, initial target speed etc.). β_1 (Generate-formation) means that a Virtual Driver should navigate the Ego-vehicle/flock to the proposed Formation Position in order to perform the corresponding Assignment-actions. Because the recipe of β_1 will change according to the dynamic environment, this Action will not be further divided into sub-Actions, but will be monitored throughout the scenario in order to make sure that the Ego-vehicle/flock can get to the Formation Position in time.

β_2 means Perform-assignment Action, and can be further divided into several Assignment-actions, which are represented as γ_0 through γ_n (n is the number of Assignment-actions a Virtual Driver needs to carry out). Each Assignment-action (γ_0 through γ_n) is contained in a corresponding Assignment. β_3 (Clean-up) should be specified by experimenters as an Assignment-action in most circumstances; however, it can be an autonomous Action by changing it to the top-Action of *Free*. In this research, the definition of "Recipe" has been expanded as it also has some High-Level Actions that will not be further refined, i.e., β_0, β_1 and β_3. Moreover, all Assignment-actions have been regarded as Low-Level Actions in his research.

In addition, the General Plan can be a centralised plan for one Virtual Driver or can be a distributed plan for several Virtual Drivers. In the latter case, the Assignment-actions in the General Plan can be based on High-Level Actions involving different simulated vehicles, each of which will maintain their own General Plan based on the High-Level Action passed to them. This research will deal with one General Plan for one Virtual Driver only and the Virtual Driver will control Ego-vehicle/flock in a centralised manner.

3.3.12 Scenario

A scenario is a pre-defined environment and situation(s) that experimenters need a participant to experience. It includes the physical scene, pre-defined traffic flow, simulated vehicles' interactions with the only one participant's vehicle and measurements that need to be collected.

The interactions in a scenario are generated by the Virtual Driver, who changes the state of the environment by controlling simulated vehicles or modifying the physical scene, e.g., place cones on the road. In the former case, the Virtual Driver controls the Ego-vehicle/flock to produce interactions with the participant's vehicle and in the latter case, the Virtual Driver will request corresponding facilities to modify the physical scene.

A scenario can have several Assignments performed by the Virtual Driver, so a scenario can have different interactions, aiming at providing individual measurements in one trial or run. For instance, a particular scenario can have two interactions, the participant should first be blocked by a lorry at some position for 10 seconds, after which, the participant should experience a five minutes' free drive. The second interaction should happen right after the free drive: a lead car should break down and force the participant to brake.

3.4 Intelligent Driver

3.4.1 Design Goal

In order to develop an architecture that can adopt and then test the decision making algorithm for this research, some principles have been adapted from Albus (1999), who mentions four axioms of intelligent machines:

Axiom 1 *"The functional elements of an intelligent system are behaviour generation, sensory perception, world modelling, and value judgement."*

Axiom 2 *"The functional elements of an intelligent system are supported by a knowledge database that stores a priori and dynamic information about the world in the form of state variables, symbolic entities, symbolic events, rules and equations, structural and dynamic models, task knowledge, signals, images, and maps."*

Axiom 3 *"The functional elements and knowledge database can be implemented by a set of computational modules that are interconnected to form nodes in a control system architecture."*

Axiom 4 *"The complexity inherent in intelligent systems can be managed through hierarchical layering."*

Hence, as an intelligent machine who can make decisions based on dynamic driving context and Assignments, the intelligent virtual driver in this research should satisfy the axioms mentioned above according to some specific needs from the area of driving simulation as specified below:

3.4.1.1 Axiom 1

Behaviour Generation A decision making algorithm based on HTN (Hierarchical Task Network) can be developed to generate and use the General Plan for the Virtual Driver;

Sensory Perception Information regarding the dynamic driving context can be modelled. The intelligent machine, which is the Virtual Driver, should be able to interpret the information and make decisions;

World Modelling A world model can be maintained in this Virtual Driver so that relevant knowledge can be updated/maintained, queried and be used for decision making.

Value Judgement Because the Value Judgement element can not only evaluate the cost, risk and benefits of Actions and the General Plan, but also the reliability of information received, this procedure is of interest to driving simulation. However, this aspect can be ignored because 1) evaluation of the cost, risk and benefits of Actions and the General Plan for behavioural conflict is not the focus of this research and 2) information received by the Virtual Driver is assumed to be reliable.

3.4.1.2 Axiom 2

Knowledge Base A Knowledge Base can be developed to provide the information regarding Assignments and related driving contexts in driving simulation, especially for scenario orchestration purposes.

3.4.1.3 Axiom 3

Interconnected Modules In order to adopt findings from several areas, e.g., psychology, computer science, this Virtual Driver can be designed in a modular manner with the possibility of extension.

3.4.1.4 Axiom 4

Hierarchical Layering This Virtual Driver can have hierarchical layers to manage the complexity in driving and related Assignments, e.g., forcing the Ego-vehicle to decelerate in order to interact with the participant's vehicle.

3.4.2 Driver Model

A PDA(Perception-Decision-Action) architecture from Lacroix *et al.* (2009) has been considered first to be the basis of a virtual driver as illustrated in Figure 3.5. This PDA architecture also contains Michon's model (Michon, 1985) for decision making, which can be used to build the hierarchical layers for decision making as required in Axiom 4.

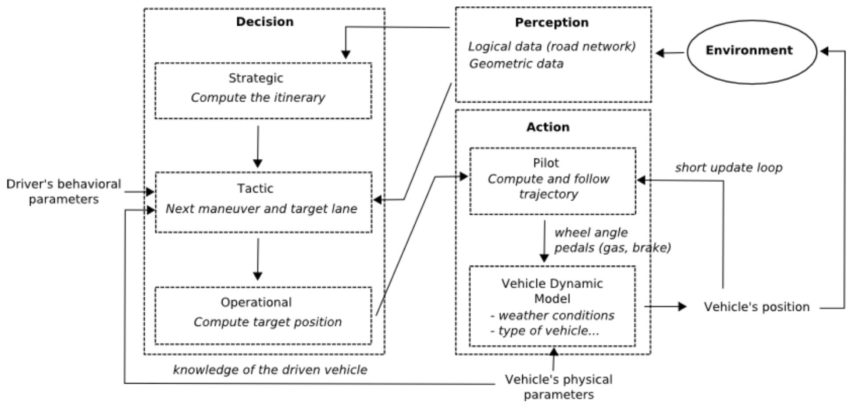


Figure 3.5: PDA (Perception Decision Action) Architecture in Lacroix *et al.* (2009)

Perception can be adopted to host the element of Sensory Perception. Decision can be adopted to host the element of Behaviour Generation. Action can be adopted to execute the decisions made for driving or related tasks. However, as PDA architecture does not have an explicit module for World Modelling, this PDA architecture can be further developed with a Cognition module that is inspired by the Cognition ability of a human driver, as illustrated in Figure 3.6.

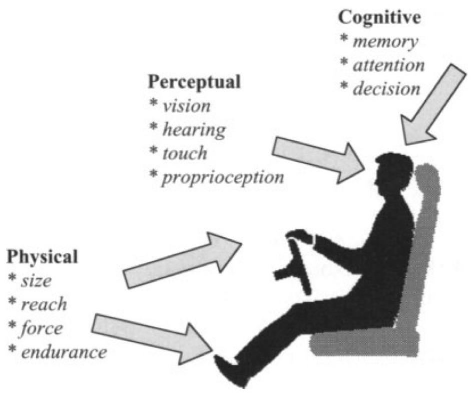


Figure 3.6: Driver Abilities(Peters & Nilsson, 2007a)

As a result, the Decision layer has been modified to become a Cognition layer, which can not only have a Memory layer for World Modelling, but also a Decision Making layer for handling driving Actions and Assignments.

The driver model, therefore, to be used in this research is illustrated in Figure 3.7. It can be used to design an intelligent Dirtual Driver. In Figure 3.7, relationships between the driver model and the elements mentioned in this section are also indicated. It should be noted that the Knowledge Base is a separate module.

3.5 Framework Description

SOAV is a framework used to create a computing environment for driving simulation in order to orchestrate scenarios with autonomous simulated vehicles. It was formed in order to support and test a decision making algorithm, which needed a driver model to provide it with data for decision making and module for decision execution. This driver model was used to develop a Virtual Driver. Some other facilities were also implemented in order to control the simulated vehicles in the simulation according to the decisions sent from the Virtual Driver.

In the following sections, SOAV will be introduced and more details regarding some of its components will be followed in successive chapters.

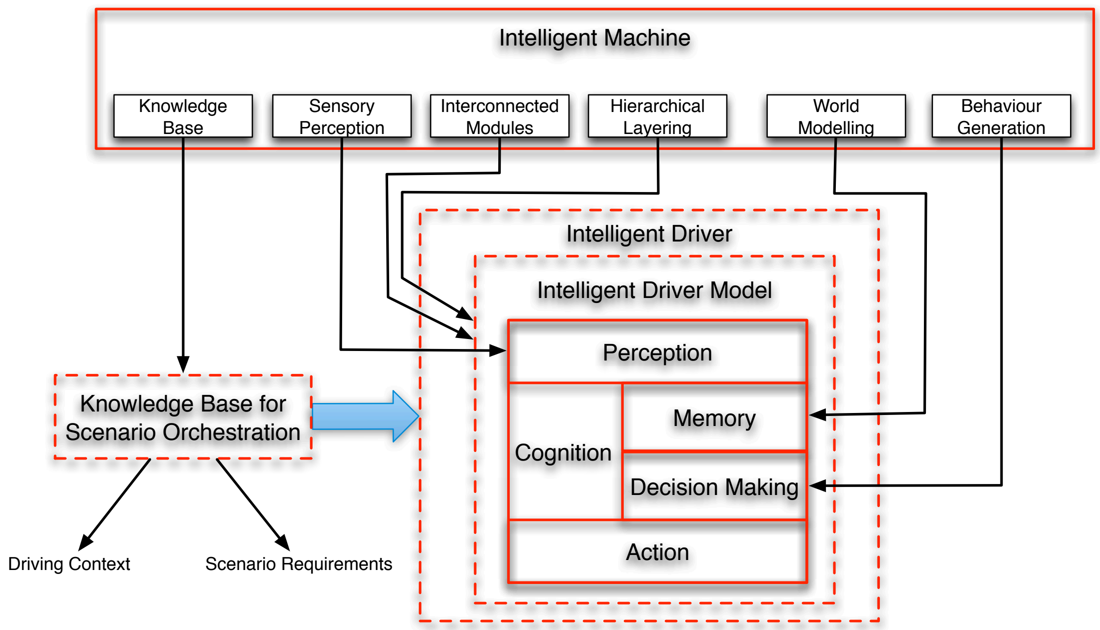
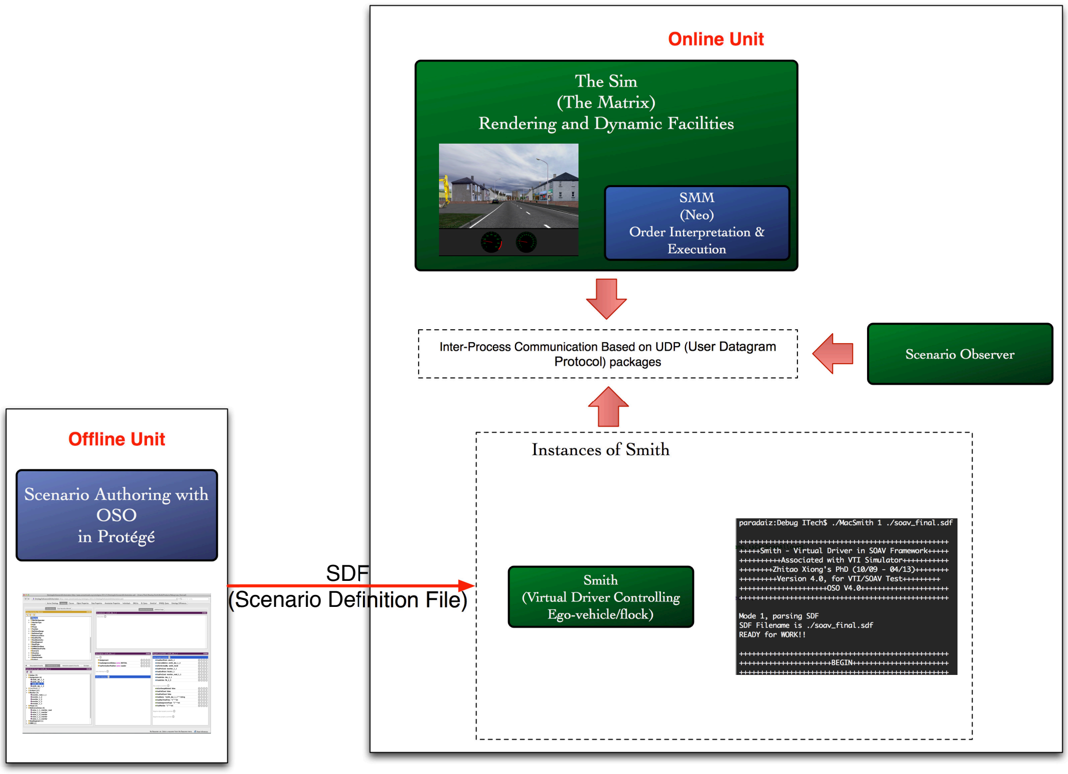


Figure 3.7: the Proposed Intelligent Driver

3.5.1 Components of SOAV

SOAV is generally divided into two parts: Offline and Online (Figure 3.8).


Online Unit
The Sim
(The Matrix)
Rendering and Dynamic Facilities
SMM
(Neo)
Order Interpretation & Execution
Inter-Process Communication Based on UDP (User Datagram Protocol) packages
Scenario Observer
Instances of Smith
Smith
(Virtual Driver Controlling Ego-vehicle/flock)
Offline Unit
Scenario Authoring with OSO in Protégé
SDF
(Scenario Definition File)

```
paradaiz:Debug ITech$ ./MacSmith 1 ./soav_final.sdf

++++++++++++++++++++++++++++++++++++++++++++++
+++++Smith - Virtual Driver in SOAV Framework+++++
++++++++++Associated with VTI Simulator++++++++++
++++++++Zhitao Xiong's PhD (10/09 - 04/13)++++++++
+++++++++Version 4.0, for VTI/SOAV Test+++++++++
+++++++++++++++++++++OSO V4.0+++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++

Mode 1, parsing SDF
SDF Filename is ./soav_final.sdf
READY for WORK!!

++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++BEGIN++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++
```

Figure 3.8: Architecture of SOAV with Offline and Online Units

3.5.1.1 Offline Component

The Offline Component is in charge of describing scenarios using OSO (Ontology for Scenario Orchestration) based on Protégé (Protégé, 2012), which outputs an SDF (Scenario Definition File) with the RDF/OWL syntax recorded in an XML file[1]. As how OSO is recorded does not affect the work in this thesis, RDF, OWL and XML will not be introduced. Details of those languages can be found in Hitzler *et al.* (2011).

[1]RDF is short for Resource Description Framework; OWL stands for Web Ontology Language; XML stands for Extensible Markup Language.

Protégé is an ontology editor and knowledge acquisition system developed by Stanford University and the University of Manchester. It is free, Open Source, widely used and well supported, so Protégé has been chosen as the editor for OSO and thus the scenario orchestration tool. Moreover, Protégé also provides reasoners, e.g, HermiT[1], to check a particular ontology's subsumption: if one class can be a subclass of another class or its consistency: if there are any classes that cannot have any individuals.

OSO is used to model driving simulation, including the logical road network and Assignments. It specifies concepts and their relationships in the driving simulation. Smith can therefore have a Knowledge Base for scenario orchestration, which can also make the scenarios standardised, easier to share and maintain.

OSO is described in more detail in Chapter 5.

3.5.1.2 Online Component

The Online component contains three major modules and one optional module. The four modules are described as below:

3.5.1.2.1 Virtual Driver (Smith)

Smith is the Virtual Driver in SOAV. He is equipped with a driver model named SAIL (Scenario-Aware drIver modeL), which uses an algorithm named NAUSEA (autoNomous locAl manoeUvre and Scenario orchEstration based on automated action plAnning) to make decisions according to Assignments. A running Smith is paired to a simulated vehicle in the Sim and committed to Assignments throughout the scenario. Smith's name has been borrowed from "The Matrix" along with the related philosophy, so Smith is a male and he or his will be used when necessary.

3.5.1.2.2 The Sim (The Matrix)

The Sim contains the vehicle dynamics and rendering facility for the simulation. It updates simulated vehicles' states every frame according to each one's autonomous driving behaviours and related parameters, e.g., desired speed. It

[1]http://protegewiki.stanford.edu/wiki/HermiT

also updates the participant's vehicle's states according to the driving instructions from the participant. It then update the graphics accordingly to display the driving conditions. Moreover, the Sim needs to broadcast information of all vehicles every frame and has been enhanced with a module named SMM described below to receive and execute orders from Smith.

3.5.1.2.3 SMM (Neo)

SMM is short for Scenario Management Module, which is used to interpret orders from Smith and execute them in the Sim. It has been developed as a module within the Sim. Hence, when SMM receives orders from Smith, which are termed Smith Orders, the Sim/SMM will activate the relevant process(es) to handle those orders. The simulated vehicles' behaviours or road conditions (traffic flow or objects on the roads such as cones) will then be updated along with the graphics. A Smith Order can include goal lane to change to, target speed, etc.

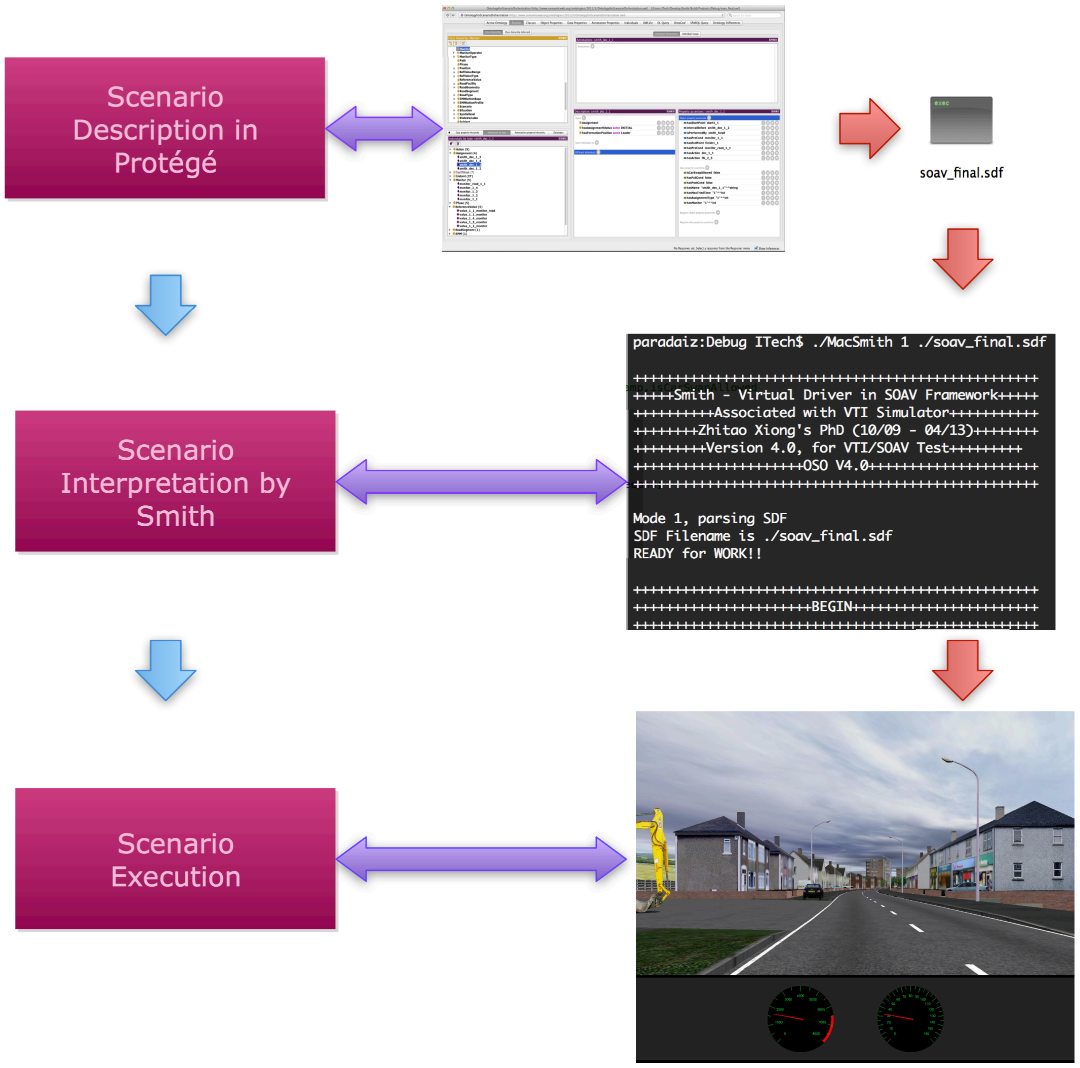
3.5.1.2.4 Scenario Observer

This module is a "listening" module that receives all the data transferred within SOAV. It can visualize and indicate what is happening within SOAV. However, as a GUI is not one of the research goals, this module simply records the data with a visualization of the frequency and number of data packages.

3.5.2 Framework Workflow

As illustrated in Figure 3.9, the scenario description is first generated with instantiated individuals based on classes and properties in OSO. Protégé will then export those individuals along with classes and properties to an SDF (Scenario Definition File).

Smith will then parse the SDF and store the information, which is the initial World Model for the whole simulation. Role Matching will take place if scenario Assignments are found in the World Model. After the Ego-vehicle/flock is found and ready, Smith will then try to finish the scenario according to Assignments. In SOAV, Entity Creation/Destruction is handled by SMM and can be requested by


Scenario Description in Protégé
soav_final.sdf
Scenario Interpretation by Smith
Scenario Execution

```
paradaiz:Debug ITech$ ./MacSmith 1 ./soav_final.sdf

++++++++++++++++++++++++++++++++++++++++++++++++++
+++++Smith - Virtual Driver in SOAV Framework+++++
++++++++++Associated with VTI Simulator+++++++++++
++++++++Zhitao Xiong's PhD (10/09 - 04/13)++++++++
+++++++++Version 4.0, for VTI/SOAV Test+++++++++
+++++++++++++++++++++OSO V4.0+++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++

Mode 1, parsing SDF
SDF Filename is ./soav_final.sdf
READY for WORK!!

++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++BEGIN+++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++
```

Figure 3.9: Workflow of SOAV (Example with UoLDS's simulation software)

Smith. Smith monitors and triggers any Assignments to generate corresponding interactions on his own. This is achieved by sending "Smith Orders" containing relevant Assignment-actions, e.g., the reference lane to track or target speed to achieve, to the Sim. Actions sent in "Smith Orders" are regarded as Low-Level Actions in this research.

3.6 Summary

In this chapter, a framework for scenario orchestration SOAV (Scenario Orchestration with Autonomous simulated Vehicles) was introduced along with its design process.

"Solution Generation Phase" in this research has provided the essential ideas in designing the components in SOAV: OSO (Ontology for Scenario Orchestration) and a Virtual Driver equipped with a driver model named SAIL (Scenario-Aware drIver modeL) and a decision making algorithm NAUSEA that is based on HTN.

In the beginning of this phase, in order to find solutions for the issues identified in literature review, some essential concepts in scenario orchestration have been identified in order to guide the design of potential algorithms. By identifying the essential concept "Assignment" and "the General Plan", an algorithm based on HTN was implied.

Then, a driver model is desirable to support the algorithm by providing data for decision making and interfaces for executing decisions or Assignments. As it is not the concern of this research, instead of reviewing relevant literature regarding driver models, this research used the four axioms from Albus (1999) to guide the design of a suitable driver model that can be adopted by a Virtual Driver who can intelligently generate interactions with the participant's vehicle. According to the components identified by applying the four axioms, a driver model prototype was generated.

Finally, a framework SOAV was introduced. NAUSEA is the algorithm based on HTN and SAIL is the driver model that supports NAUSEA. SMM in the Sim is the communication interface between the Sim and the Virtual Driver Smith who is equipped with SAIL/NAUSEA. A Scenario Observer was also included to record the communication between Smith and the Sim/SMM.

In Chapter 4 and 5, OSO and SAIL/NAUSEA will be elaborated. Details regarding implementation of the Sim/SMM and SAIL/NAUSEA, which is Smith, will be provided in Chapter 6.

Chapter 4

Ontology for Scenario Orchestration - OSO

Is that...?
The Matrix? Yeah.
(The monitors are packed with bizarre codes and equations.)
- The Matrix

4.1 Introduction

An ontology, leading to a shareable Knowledge Base, is built upon varieties of concepts and their relationships in a domain of interest. It is therefore intuitive to regard an ontology as a formal, standard and normative way of representing knowledge. However, just like a sword, an ontology is two-sided because of its applications: being descriptive or being normative.

In this research, Ontology for Scenario Orchestration (OSO) has been designed to be as descriptive as possible to include common concepts that have been identified within the area of driving simulation with the purpose of representing scenarios (See Section 3.3). It covers concepts in driving simulation ranging from physical objects such as roads or junctions to virtual ones such as Assignments or Actions. OSO should be able to provide:

- driving context representation, which is the description of a logical road network and other knowledge regarding physical objects and their states, e.g., simulated vehicles and their vehicle model;

- task representation, which specifies Assignments that Smith should do. An Assignment contains Assignment-action, Trigger(s) and other related information, e.g., Ego-vehicle requirement;
- Action library, indicating what High-Level and Low-Level Actions are available. Recipes are provided to guarantee repeatability by forcing Smith follow the same Action trajectory when facing the same context. They are always used to defined how to perform the "Perform-assignment", which is β_2 as specified in Section 3.3.11 on Page 49;
- Monitor System representation, indicating what should be dictated so that particular Assignment-actions can be triggered, finished or failed when conditions in Triggers are satisfied and,
- temporal representation, specifying temporal relationships between some entities;

OSO should provide information regarding not only context that is essential for decision-making but also Assignments that Smith should be aware of to generate required interactions. It has been developed with Protégé and recorded in an XML file with RDF/OWL syntax.

It should be noted that OSO is an abstract model for scenario orchestration and no assumption has been made regarding what tool should be used to develop OSO and what language should be used to record OSO, i.e., what format should be used to output SDF. As a result, details regarding Protégé or the XML file (SDF) will not be described in this thesis and thus OSO will be shown in directed graphs instead of snapshots of Protégé. Information regarding Protégé and RDF/OWL can be found in Horridge *et al.* (2009) and Hitzler *et al.* (2011) respectively.

In OSO, a class represents a concept in driving simulation, e.g., class *Vehicle* represents all the simulated vehicles in the simulation. A class can have subclasses that represent more detailed concepts than its superclass, e.g., class *LGV* represents Light Goods Vehicles. Individuals, on the other hand, are instantiated classes, representing specific objects in a scenario, e.g., simulated vehicle no.1 can be an individual of *Vehicle* with white colour. When Smith is parsing the SDF (Scenario Definition File), he will look for individuals and instantiate them by mapping each individual with corresponding data in the World Model.

As mentioned in Chapter 2, properties have two types, object properties specify relationships between two individuals and data properties specify relationships between an individual and a data type, which can be a string, Boolean, float, etc. In OSO, properties have been solely treated as restrictions that a class should obey. For instance, an Assignment may have an Action as its Assignment-action. In this case, instead of describing the domain of object property *hasAction* is *Assignment*, OSO specifies that *Assignment* is a subclass of an anonymous class that has the property *hasAction* to indicate its associated Action. There are two common restrictions in OSO:

1) cardinality restriction. For instance, an individual of class A is a subclass of an anonymous class. This anonymous class relates to **minimum**, **maximum** or **exactly** n ($n = 0, 1, 2...$) individuals of class B through an object property, so A relates to **minimum**, **maximum** or **exactly** n ($n = 0, 1, 2...$) individuals of B through an object property.

2) existential restriction. When an object property is used to specify the relationship between two individuals, one of them can be anonymous by stating that 1) an individual of class A is a subclass of an anonymous class and 2) this anonymous class relates to **some** individual of class C. Hence, A relates to **some** individual of class C. This individual of C is, therefore, anonymous.

   An Existential restriction is also useful when a class can be described by either classes, e.g., an individual of class A relates to **some** individual of class D or class E. Moreover, the subclasses of related class(s) can be used to further refine the relationship. For instance, if class E is a subclass of C and an individual of class A relates to **some** individual of class C, then it is possible to refine this relationship to "an individual of class A relates to **some** individual of class E" if class A and E can be related.

By adopting restrictions, classes can be described with details in Protégé as illustrated in Figure 4.1.

From this chapter on, classes, properties and individuals are represented with italic fonts and directed graphs are also used from time to time to illustrate classes, related properties & individuals. In graphs, classes are represented by ellipses; individuals are represented by rhombuses; data properties are represented by dotted arrows while object properties are represented by arrows. Figure 4.2 shows how classes, object properties, data properties and individuals are represented in this thesis along with how restrictions are reflected based on some examples in previous paragraphs.

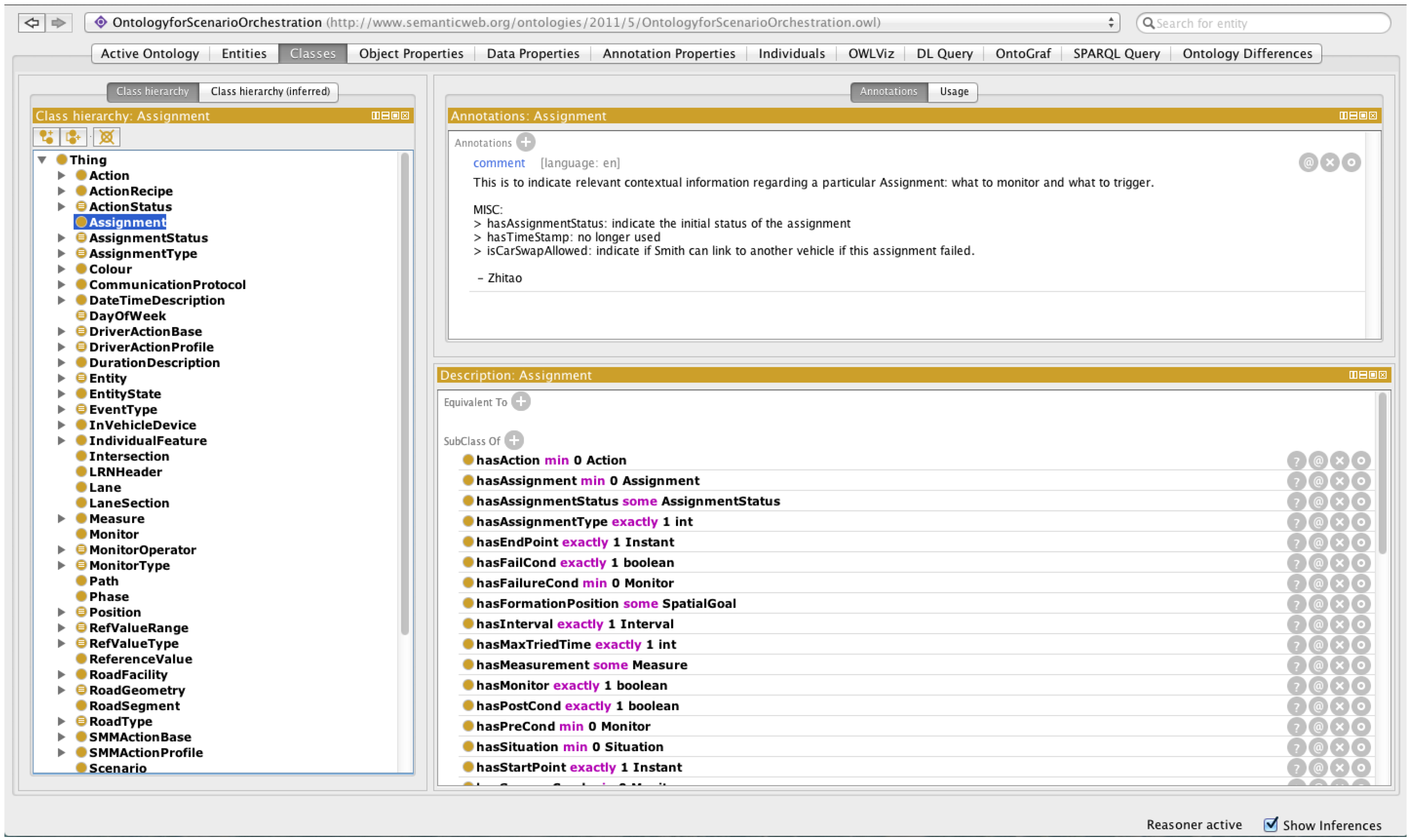
As illustrated in Figure 4.2, different from cardinality restrictions that will be represented in directed graphs by two connected individuals without the specification of cardinality, existential restrictions will be represented by connecting the named individual with a class that the related anonymous individual belongs to. It should be noted that data properties are related to a maximum of a data type in OSO.

In this chapter, the resulting OSO will be presented. Data properties will only be mentioned when they are essential in describing a class and thus influence its individuals. For instance, data property named *hasSpeed* specifies the maximum speed of a simulated vehicle instantiated from *Vehicle* and because of the importance of this value in making decisions, this property will be mentioned when describing relevant classes. Directed graphs will not include data properties. Moreover, how to orchestrate scenarios with OSO will be covered with experiments and be presented in Appendix C.

4.2 Naming Convention

When OSO is being developed or used to represent scenarios, some conventions have been complied with:

- classes representing Actions should be represented with imperative form verbs, e.g., *Overtake*;



OntologyforScenarioOrchestration (http://www.semanticweb.org/ontologies/2011/5/OntologyforScenarioOrchestration.owl)

Class hierarchy: Assignment

- Thing
  - Action
  - ActionRecipe
  - ActionStatus
  - Assignment
  - AssignmentStatus
  - AssignmentType
  - Colour
  - CommunicationProtocol
  - DateTimeDescription
  - DayOfWeek
  - DriverActionBase
  - DriverActionProfile
  - DurationDescription
  - Entity
  - EntityState
  - EventType
  - InVehicleDevice
  - IndividualFeature
  - Intersection
  - LRNHeader
  - Lane
  - LaneSection
  - Measure
  - Monitor
  - MonitorOperator
  - MonitorType
  - Path
  - Phase
  - Position
  - RefValueRange
  - RefValueType
  - ReferenceValue
  - RoadFacility
  - RoadGeometry
  - RoadSegment
  - RoadType
  - SMMActionBase
  - SMMActionProfile
  - Scenario

Annotations: Assignment

Annotations

comment [language: en]

This is to indicate relevant contextual information regarding a particular Assignment: what to monitor and what to trigger.

MISC:
> hasAssignmentStatus: indicate the initial status of the assignment
> hasTimeStamp: no longer used
> isCarSwapAllowed: indicate if Smith can link to another vehicle if this assignment failed.

- Zhitao

Description: Assignment

Equivalent To

SubClass Of

- hasAction min 0 Action
- hasAssignment min 0 Assignment
- hasAssignmentStatus some AssignmentStatus
- hasAssignmentType exactly 1 int
- hasEndPoint exactly 1 Instant
- hasFailCond exactly 1 boolean
- hasFailureCond min 0 Monitor
- hasFormationPosition some SpatialGoal
- hasInterval exactly 1 Interval
- hasMaxTriedTime exactly 1 int
- hasMeasurement some Measure
- hasMonitor exactly 1 boolean
- hasPostCond exactly 1 boolean
- hasPreCond min 0 Monitor
- hasSituation min 0 Situation
- hasStartPoint exactly 1 Instant

Figure 4.1: OSO with Protégé

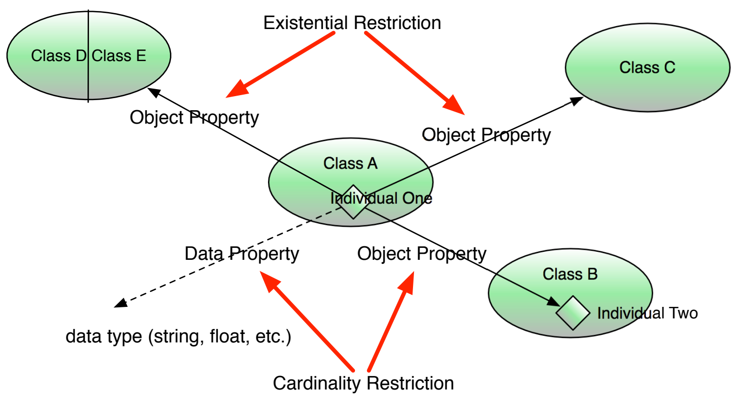


Figure 4.2: Representation of Classes, Properties and Individuals

- every single word in class name should start with a capital letter including the first word, e.g., *VehicleState*;
- no spaces or other marks are allowed to join class or property names, e.g., *VehicleController*, *isPerformedBy*;
- classes are presented in singular forms, e.g., *VirtualDriver*, *Instant*;
- abbreviations should be avoided unless the class/property names are made of more than one word and the meaning of the abbreviation is clear, e.g., *LRN* means Logical Road Network;
- the unit for data properties are SI based, especially regarding length and time, which have the units of metre (m) and second (s) respectively;
- names of properties, including data and object properties, are suggested to start with "has" or "is", and should be followed by words that are started with capital letters. The names should also have specific meanings related to the properties, e.g., *isPerformedBy*;
- individuals are suggested to be presented by words or abbreviations with lower case letters and numbers, e.g., an individual of *VirtualDriver* can be presented by *vd1* and another one can be presented by *vd2*;

- for convenience and readability, individuals of *Assignment* can be named with the following information: performer, keyword of the Assignment-action(s) and order in the scenario. The corresponding Triggers, i.e., Monitors, Success Conditions and Failure Conditions, can be named with the keyword "monitor/post/failure", order in the scenario and keyword of the state variable in this Trigger. Reference values in Triggers can be named with the keyword "value", order in the scenario, a keyword of "monitor/-post/failure" and keyword of the state variable in this Trigger.

  For instance, in Experiment two, there will be an Assignment named *smith_overtake*_1_3, and one of its Monitors is *monitor*_1_3_*hw*, whose reference value is *value_monitor*_1_3_*hw*. *smith* is the performer; *overtake* is the keyword of Assignment-action; *hw* is the keyword of the state variable, referring to time headway; *monitor* and *value* are two keywords.

  It should be noted that if some state variable has been frequently used in Triggers, e.g., "*ttc*" that means time-to-collision, the keyword for the state variable can be ignored in the name of a Trigger and its corresponding reference value. Moreover, another number could be included in those individual names to indicate order and sub-order if more than one Assignment has been generated in order to handle the same interaction, e.g., an Assignment named *smith_overtake*_1_3 would imply that two Assignments have been developed to assist *smith_overtake*_1_3 before its Assignment-action executes, e.g., an Assignment can be used to create the simulated vehicle needed for *smith_overtake*_1_3 and another to clear lanes for the coming Assignment.

4.3 Layout of Ontology for Scenario Orchestration (OSO)

4.3.1 General Description

OSO is developed by expanding the "Scenario" concept, which covers what the human participants experience and what the researchers require: pre-defined virtual scene, simulated vehicles' interactions with human participants, ambient traffic flow without interactions, and measurements that should be collected. "Scenario" has been modelled as a class named *Scenario*, which is related to three sets of classes:

1) Scene: this set contains classes that are being used to represent the physical scene in the simulation. At present, the following classes have been identified: *Entity*, *VehicleModel*, *Intersection*, *RoadSegment*, *RoadType*, *RoadFacility*, *RoadGeometry*, *LaneSection*, *Lane*, *Path*;

2) Measurement: this set contains just one class *Measure* that specifies available measures in the driving simulator. It is adapted from the standard measures list from the Wikipedia of Driving (Driving Wiki, 2010b);

3) Assignment: this set contains the essential class *Assignment* and other classes that are involved in describing an Assignment. The following major classes are included: *Assignment*, *Monitor*, *Action*, *Situation*, *Reference*, *ReferenceValue*, etc.

In addition, there are also some classes that are used to describe the general information of a scenario, e.g., class *CommunicationProtocol* is used to represent the network protocol being used within SOAV. However, due to their minor roles in describing Assignments, they will only be mentioned if necessary in this thesis.

The following sections will focus on the class *Assignment*, and the related classes that are necessary to support it. Classes that are needed to describe other classes will be introduced first. As a result, the following major classes will be introduced one by one from next section: *VehicleModel*, *Measure*, *Entity*, *TemporalEntity*, *MetricConstraints*, *StateVariable*, *ReferenceValue*, *Monitor*, *Action*, and *Assignment*. Classes named *RoadSegment*, *Intersection* and *SimLimitation* that are not related to *Assignment* are presented at the end. They are classes used to describe the road segments, the intersections and limitations of the Sim respectively. The dependencies between those classes are illustrated in Figure 4.3.

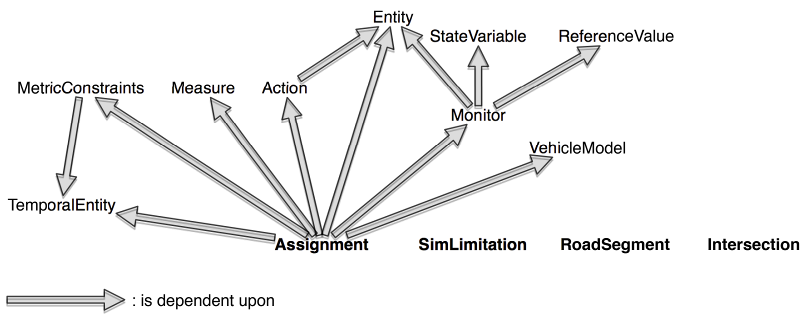


Figure 4.3: Dependences Between Major Classes

4.3.2 $VehicleModel$

The vehicle models available are derived from COBA (2006) and some major models are illustrated in Figure 4.4.

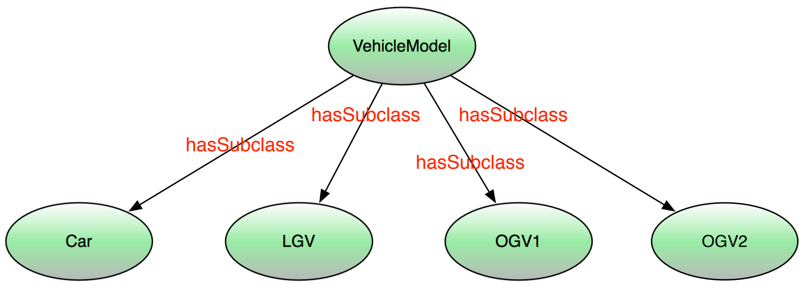


Figure 4.4: Class $VehicleModel$

According to COBA (2006), LGV represents Light Goods Vehicles that are up to 3.5 tonnes gross vehicle weight; $OGV1$ represents Other Goods Vehicles that are over 3.5 tonnes gross vehicle weight with two or three axles while $OGV2$ represents Other Goods Vehicles that are over 3.5 tonnes gross vehicle weight with four or more axles. Lorries or trucks belong to LGV, $OGV1$ or $OGV2$ in this research.

4.3.3 *Measure*

By adopting the standard measurements from Driving Wiki (2010b), OSO has modelled a class named *Measure* and several sub-classes that indicate the subcategories of measurements, e.g., *DriverBehavior*, *PerformanceMeasure*. However, because of the lack of detailed description regarding each measurement, those measurements have been included for demonstration purposes in order to indicate that different Assignments can provide similar measurements.

4.3.4 *Entity*

An Entity in a scenario represents a physical or virtual object, e.g., a traffic light controller, a virtual pedestrian, a simulated vehicle, an SMM (Scenario Management Module), etc. Its major subclasses and related properties are as follows:

VirtualDriver, a subclass of *VirtualHuman*, represents Smith. *VirtualPedestrian*, a subclass of *VritualHuman*, represents the virtual pedestrians evolving in the simulation.

FlockVehicle represents Flocks. *SingleVehicle* represents one simulated vehicle that can be controlled by Smith, some properties have been included to describe a single simulated vehicle, e.g., *hasMaxAccRate* indicates the maximum positive acceleration rate of the Ego-vehicle. Among all the properties, the most useful one is the object property named *hasVehicleModel*, which indicates the vehicle model of the required Ego-vehicle for a particular Assignment. *hasVehicleModel* relates an individual of *SingleVehicle* to some individual of *VehicleModel*.

VirtualObject has three subclasses: *VirtualCircle*, *VirtualLine* and *VirtualZone*, indicating some pre-defined areas that can be used as thresholds in Triggers, e.g., entering some virtual zone or crossing some virtual circle can trigger a particular Assignment.

Subject represents the participant. This class indicates what kind of information would be expected from a participant, including his/her proposed route and travel speed. Hence, the following properties have been identified:

- *hasRoute*: a data property indicating a route that the participant could follow;
- *hasDesiredSpeed*: a data property indicating the instructed desired speed the participant should obey.

In Figure 4.5, major classes regarding *Entity* are illustrated.

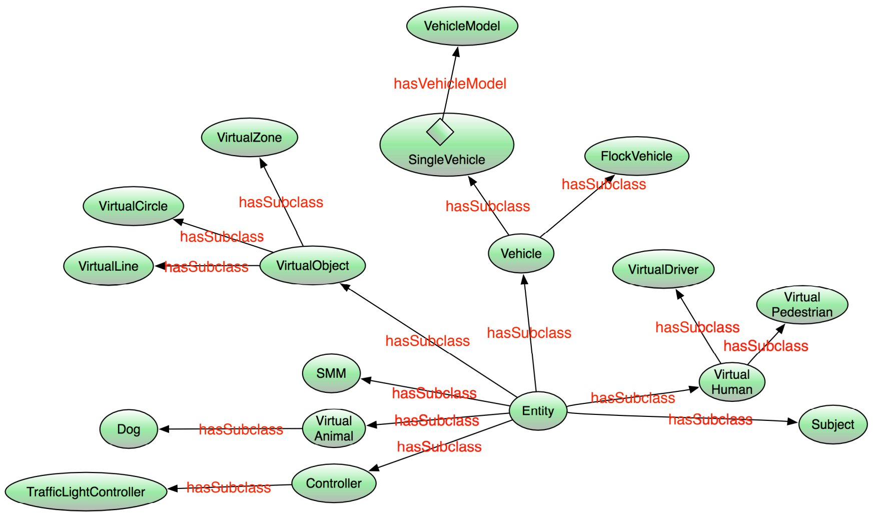


Figure 4.5: Class *Entity* and Related Classes

4.3.5 *TemporalEntity*

Concepts and related properties regarding time have been adapted from the Time ontology (Hobbs & Pan, 2004). The class of *TemporalEntity* therefore has two subclasses: *Instant* and *Interval*. *Instant* represents specific moments or time points, whereas *Interval*, on the other hand, represents periods between two Instants.

In addition, relationships between two Intervals are also adopted from the Time Ontology as illustrated in Figure 4.6, which is derived from Allen's logic (Allen, 1981). In Figure 4.6, *a* and *b* represent two individuals of class *ProperInterval*, which is a subclass of *Interval*. Those relationships are also being used by

Assignments to indicate precedence constraints between two Assignments and can be further used to imply the relationships between two Assignment-actions. The Domain and Range of those properties in Figure 4.6, therefore, are not specified.

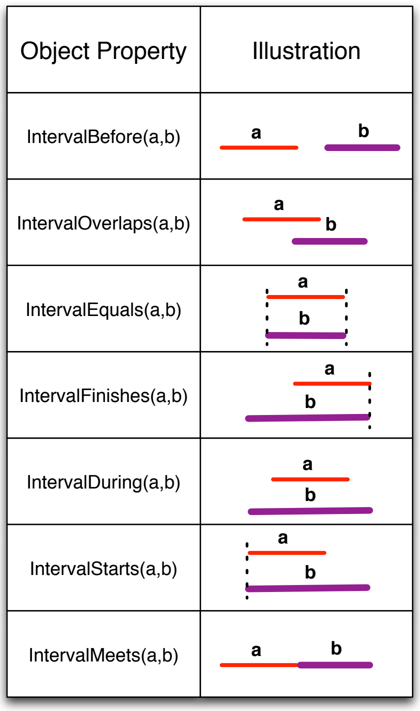


Figure 4.6: The Possible Relationships Between Two Intervals (Allen, 1981)

TemporalEntity is summarised in Figure 4.7 along with relevant object properties.

4.3.6 *MetricConstraints*

Metric constraints have been modelled in a class called *MetricConstraints*, which relates to *Instant* with two object properties: *hasMinuend* and *hasSubtrahend*. Moreover, it also relates to float numbers with two data properties: *hasMetricMax* and *hasMetricMin* as illustrated in Figure 4.8(a).

If there is a metric constraint as "starting time of Action β should be more than 0 seconds but less than 20 seconds after the finishing time of α", then this

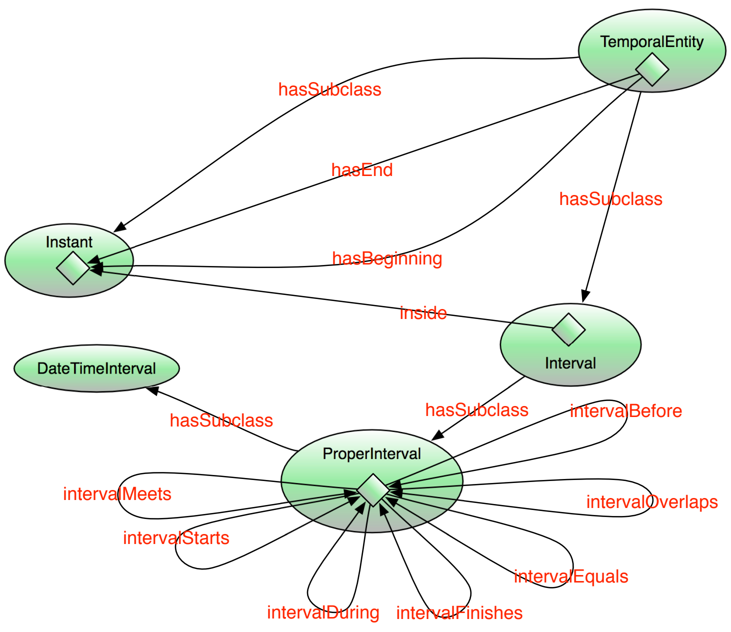


Figure 4.7: Class *TemporalEntity* and Related Classes

metric constraint can be specified with the following inequality:

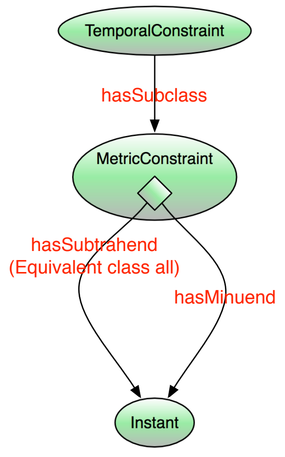
$$0 \leqslant s_\beta - f_\alpha \leqslant 20 \tag{4.1}$$

and as illustrated in Figure 4.8(b), this constraint can be specified as an individual of *MetricConstraint* named *mc1* and the two Instants s_β, f_α as two individuals of *Instant* named *sb* and fa.

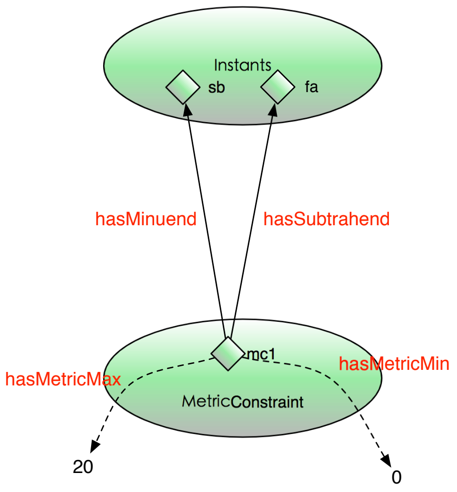
4.3.7 $StateVariable$

State variables are used to describe the states of a reference object in a scenario. It contains the following major subclasses:

- $SV\_OvertakingIntention$: overtaking intention of the simulated vehicle;
- SV_Speed: current speed of the vehicle (participant's vehicle or simulated vehicle) in m/s (metres per second);



(a)



(b)

Figure 4.8: Class *MetricConstraints* and Usage. (a) Class *MetricConstraints* and Related Classes; (b) Specification of Metric Constraint

- *SV_JourneyTime*: time already spent on the journey in seconds, counted from the creation of the vehicle;
- *SV_SpaceHW*: distance headway of the vehicle in metres, see Figure 4.9.

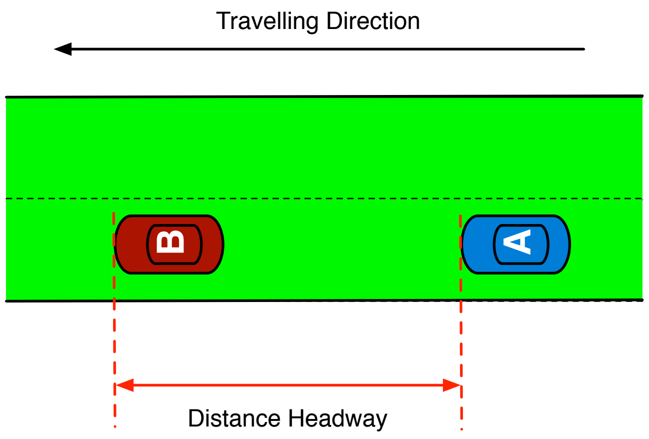


Figure 4.9: Distance Headway of Vehicle A

- *SV_Offset*: lane offset of the vehicle in metres, see Figure 4.10;

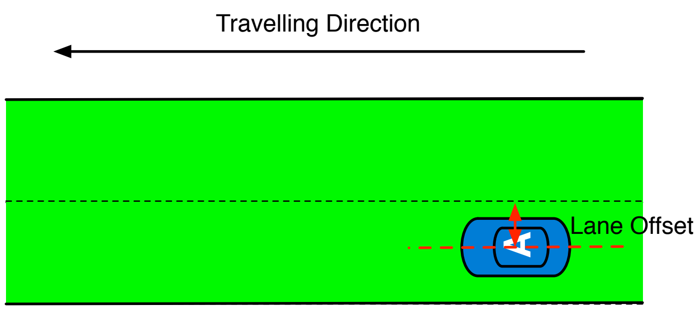


Figure 4.10: Lane Offset of Vehicle A

- *SV_TTC*: time to collision in seconds with the lead object, which can be a vehicle or a road facility, e.g., a traffic light; See Figure 4.11 for an example involving two simulated vehicles, in which case time to collision t_{ttc} can be calculated as:

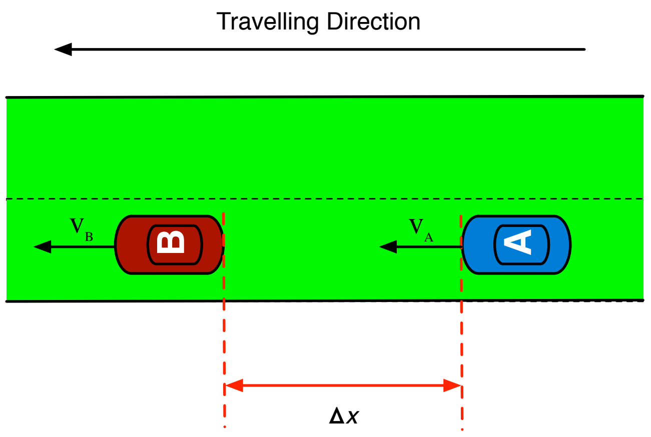


Figure 4.11: Time-To-Collision of Vehicle A

$$t_{ttc} = \frac{\Delta x - \Delta s}{v_A - v_B}(v_A - v_B > 0) \tag{4.2}$$

where,

- v_A = vehicle A's current speed, m/s;
- v_B = vehicle B's current speed, m/s;

  - Δx = the distance between the font of vehicle A to the back of vehicle B in metres;
  - Δs = the pre-defined safe distance in metres. Zero can be assigned.

- *SV_Distance*: travelled distance of the vehicle in metres along the present road segment; it starts from the beginning of the travelling road segment;
- *SV_TimeHW*: time headway of the vehicle in seconds, as illustrated in Figure 4.12 and calculated in Equation 4.3:

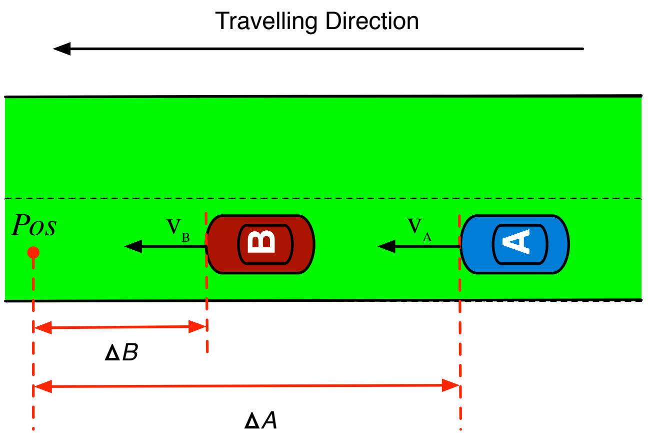


Figure 4.12: Time Headway of Vehicle A

$$t_{hw} = \frac{\Delta A}{v_A} - \frac{\Delta B}{v_B} \tag{4.3}$$

where,

  - v_A = vehicle A's current speed, m/s;
  - v_B = vehicle B's current speed, m/s;
  - ΔA = the distance between vehicle A to the fixed position *Pos* in metres;
  - ΔB = the distance between vehicle B to the fixed position *Pos* in metres.

In addition, Pos can be set within vehicle B, e.g., in the back of vehicle B (i.e., $\Delta B = 0$).

- $SV\_AssignmentStatus$: the status of the Assignment, which can be Initial, Pending, Failure and Success as specified in Section 3.3.10. Those statuses have been represented in a class named $AssignmentStatus$;

- $SV\_Neighbourhood$: the information of simulated vehicles around the participant's vehicle that are of interest to Smith the most in this research and will be used for decision-making. It contains 12 positions as illustrated in Figure 4.13. 12 sub-classes of $SV\_Neighbourhood$ have been modelled.

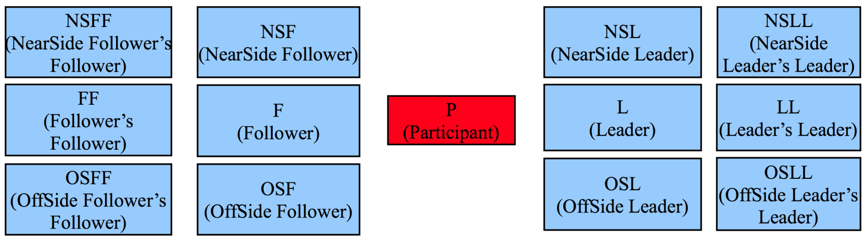


Figure 4.13: Neighbourhood around Participant's Vehicle

- $SV\_JourneyDistance$:distance already travelled in metres, it counts from the creation of the vehicle;

- SV_RdID: the ID of the present road segment;

- SV_Lane: the ID of the present travelling lane, which is represented as an integer in the Sim.

The concept of $StateVariable$ can be expanded in order to adopt more state variables.

SV_TimeHW and SV_TTC in OSO have sub-classes and pre-defined reference objects. SV_TimeHW means the time headway between the reference object and the participant's vehicle; SV_TimeHW_L means the time headway between the reference vehicle and its leader and SV_TimeHW_NSL means the time headway between the reference vehicle and its nearside leader (left). Moreover, SV_TTC means the time-to-collision between the reference object and the participant's vehicle; SV_TTC_L means the time-to-collision between the reference vehicle and its leader.

$StateVariable$ and its subclasses are illustrated in Figure 4.14.

4.3.8 *ReferenceValue*

$ReferenceValue$ represents a threshold that some state variables of an entity should be compared to and how this comparison should be performed, e.g., is bigger or smaller. Its properties include:

- *hasRefValueType* indicates the type of the threshold that will be compared to. The type has been modelled in a class *RefValueType*, which can be $LINE$ (reference line), $POLY$ (reference zone), *ObjectReference* (reference object), $STATUS$ (reference status - Action or Assignment) or $NUMBER$ (reference number). Hence, it relates an individual of *ReferenceValue* to some individual of *RefValueType*;
- *hasRefLine* specifies the threshold based on a reference line, a maximum of one individual of $VirtualLine$ can be related;
- *hasRefZone* specifies the threshold based on a virtual zone, a maximum of one individual of $VirtualZone$ can be related;
- *hasRefObject* specifies the threshold based on an entity, a maximum of one individual of *Entity* can be related;
- $hasAssignmentStatus$ specifies the threshold that is an Assignment status, some individual of $AssignmentStatus$ should be related;
- $hasGeneralValue$ specifies a general threshold value to the threshold, only one can be specified;
- *hasRefValueRange* indicates how to compare the threshold with the value of a state variable from a particular entity. Some individual of the following classes can be related: $BEQUAL(\geqslant)$, $BIGGER(>)$, $EQUAL(=)$,

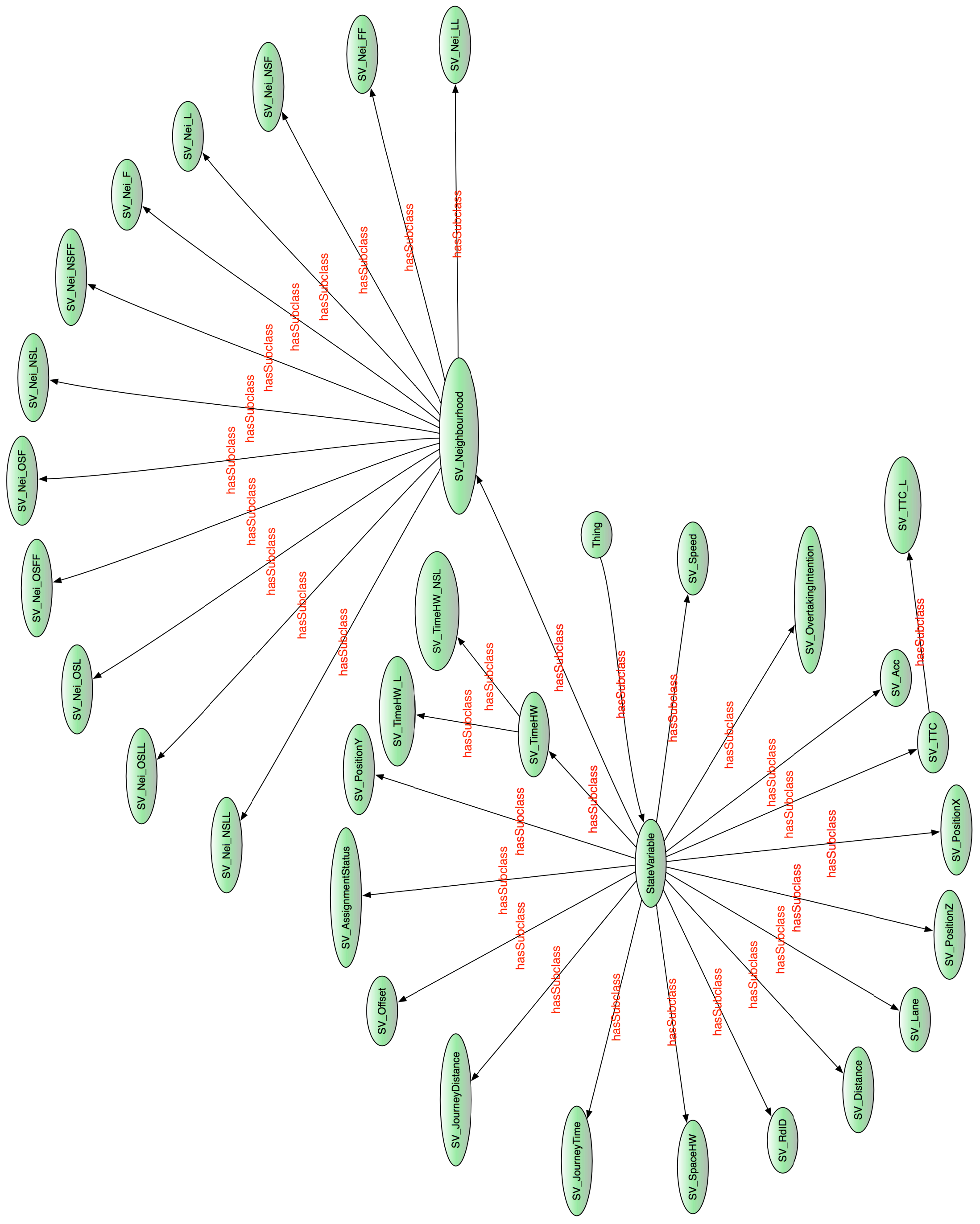


Figure 4.14: Class *StateVariable* and Related Classes

$NEQUAL(\neq)$, $SEQUAL(\leqslant)$ or $SMALLER(<)$. When it comes to an event, they can be also used to specify the state transition, e.g., the transition of some entity's state variable's value from less than a threshold to greater than that threshold can be expressed by $BEQUAL(\geqslant)$, which can be ambiguous because $BEQUAL$, which means greater than or equals to, cannot express state transition. However, by stating if the comparison is based on a state transition (*Event*) or a state interval (*State*), Smith can recognise it without misunderstanding. This flag will be introduced in the next section as *ReferenceValue* does not contain this piece of information.

ReferenceValue and related properties along with associated classes are illustrated in Figure 4.15.

4.3.9 $Monitor$

A Monitor is a facility to keep watch on state variables in the simulation and becomes true when its conditions are satisfied. A state variable of an object and a threshold are needed to specify a condition, so several classes have been modelled in OSO especially for the Monitor: $Monitor$, *EventType*, $MonitorOperator$, $MonitorType$ and *ReferenceValue*.

In general, in order to identify a Monitor, specifications are needed for: 1) what should be monitored by using *hasStateVariable*; 2) what entities in the simulation should be monitored by using *hasReference*; 3) how often should the state variables of the entities be monitored by using *hasEventType, hasMonitorOperator, hasMonitorType* and 4) how the Monitor can be satisfied by using *hasRefValue*. Properties that are used to describe $Monitor$ are as follows:

- *hasReference* specifies the reference entity that the Monitor should dictate, so it relates an individual of $Monitor$ to exactly one individual of *Entity*;

- $hasStateVariable$ specifies the state variables of the entity that the Monitor should dictate, so it relates an individual of $Monitor$ to some individual of $StateVariable$.

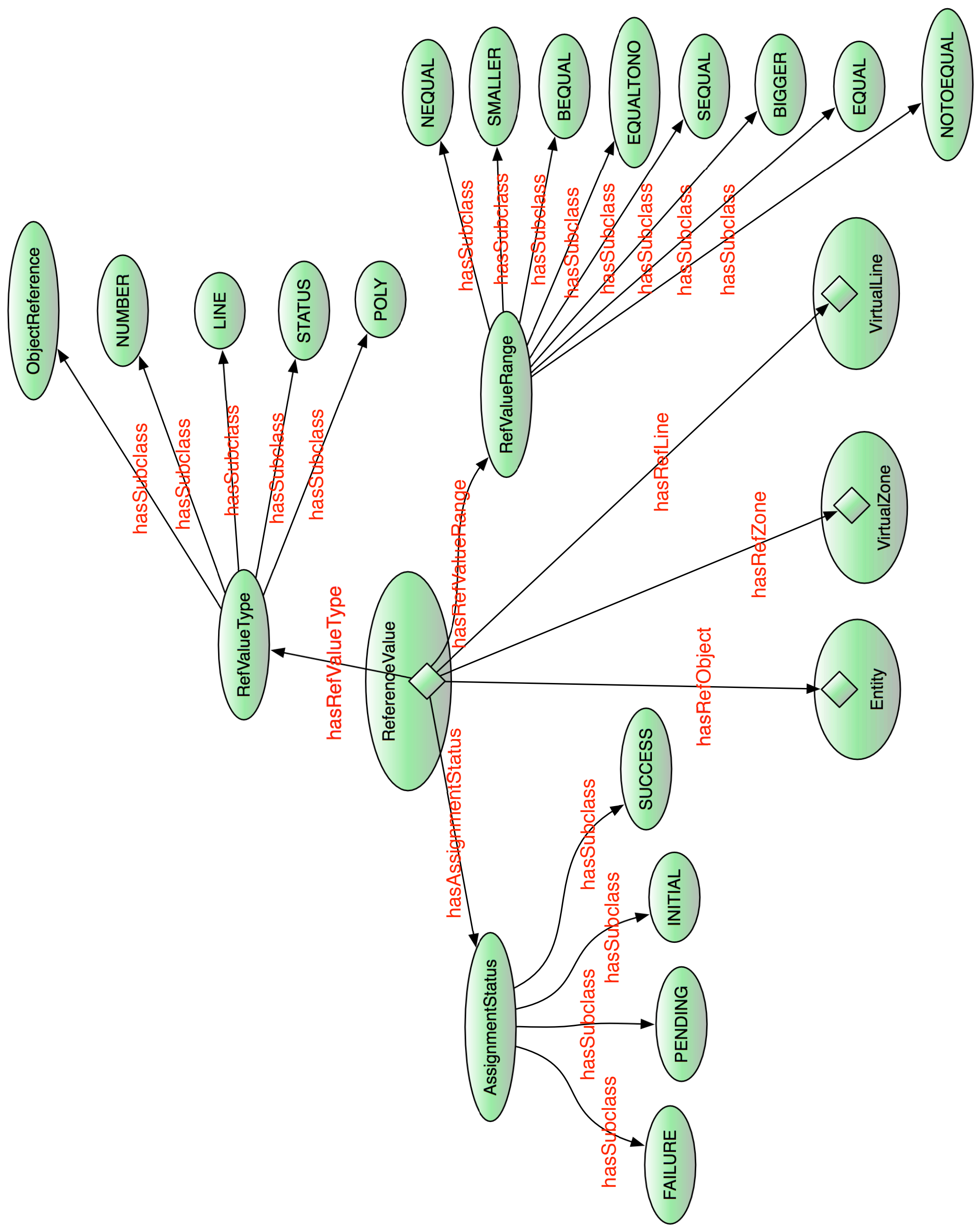


Figure 4.15: Class *ReferenceValue* and Related Classes

- *hasMonitorType* specifies the type of the Monitor, indicating what the Monitor should detect: state intervals(*State*) or state transitions (*Event*); Hence, it related an individual of *Monitor* to some individual of *MonitorType*, which has subclasses of *State* or *Event*;

- *hasEventType* specifies the event type of the Monitor if it is an event. It relates an individual of *Monitor* to some individual of *EventType*, which indicates some common events:

  – *ET_Threshold* indicates that this is just a common state transition and a threshold value should be used;
  – *ET_Cross* indicates that the threshold is a reference line and the condition will become true when the line is crossed by the monitored object;
  – *ET_Enter* and *ET_Exit* indicate that the threshold is a reference zone and the condition will become true when the monitored object enters or exits that area and,
  – *ET_Timer* indicates the event is timer-based, so the Monitor should dictate the timer and compare it with a threshold value.

- *hasMonitorOperator* specifies the operator of the Monitor, so it relates an individual of *Monitor* with some individual of *MonitorOperator*, which specifies the frequency of the Monitor by adopting the four operators elaborated in Section 3.3.3: *WHEN*, *WHENEVER*, *EVERY* and *ASLONGAS*;

- *hasRefValue* specifies the reference value of the Monitor so it relates an individual of *Monitor* with exactly one individual of *ReferenceValue*, which specifies the threshold.

Multiple individuals of *Monitor* for an Assignment in this research will imply that all the conditions in those individuals, which can be Monitors/Success Conditions/Failure Conditions, must be satisfied in order to trigger/finish/fail the

Assignment. Disjunctive individuals of *Monitor* are not currently supported, as they were not required at this stage.

Related classes regarding *Monitor* are summarised in Figure 4.16 with corresponding object properties.

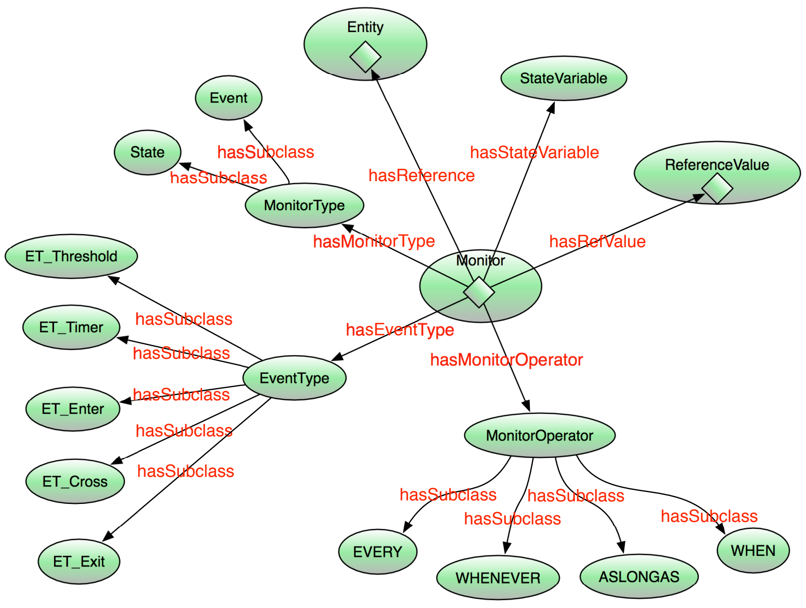


Figure 4.16: Class *Monitor* and Related Classes

4.3.10 *Action*

Action can change the state of the simulation and represents mainly the Actions that can be used by Smith, especially driving Actions. Its available properties include:

- *hasActionType* indicates the type of the Action by notifying Smith if a recipe is needed and which kind of recipe should be found, e.g., an Action with Action type of *Block* will let Smith find the recipe of preventing the participant from overtaking. It relates an individual of *Action* to some individual of *DriverActionBase* or *SMMActionBase*. The former indicates the types of Action that can be performed by Smith and the latter indicates the ones by SMM;

- *hasActionProfile* indicates the flag of what changes should be made, e.g., if it relates an individual of *Action* to some individual of *DesiredAccRate*, Smith will order the Ego-vehicle/flock to adopt a new acceleration rate that is specified by *hasActionAcc*. It relates an individual of *Action* to some individual of *DriverActionProfile* or *SMMActionProfile*. The former indicates the Action profiles used by Smith and the latter indicates the ones by SMM;

- *hasActionCreate* indicates how many simulated vehicles should be created to become the Ego-vehicle/flock. All the simulated vehicles created should comply with the requirements in the Assignment, e.g., required vehicle model;

- *hasOvertakingIntention* indicates the proposed overtaking intention of the Ego-vehicle. The value is a Boolean;

- *hasActionDeadline* indicates the proposed deadline of the Action. The value is a float;

- *hasActionDuration* indicates the proposed duration of the Action. The value is a float;

- *hasActionRelease* indicates the proposed release time of the Action. The value is a float;

- *hasActionSpeed* indicates the speed that the Ego-vehicle/flock should adopt. The value is a float;

- *hasActionAcc* indicates the acceleration rate that the Ego-vehicle/flock should adopt. The value is a float;

- *hasActionLane* indicates the lane that the Ego-vehicle/flock should change into or stay in. The value is a string, e.g., “offside” refers to the right adjacent lane of the Ego-vehicle/flock;

Related classes of *Action* is summarised in Figure 4.17 along with relevant object properties:

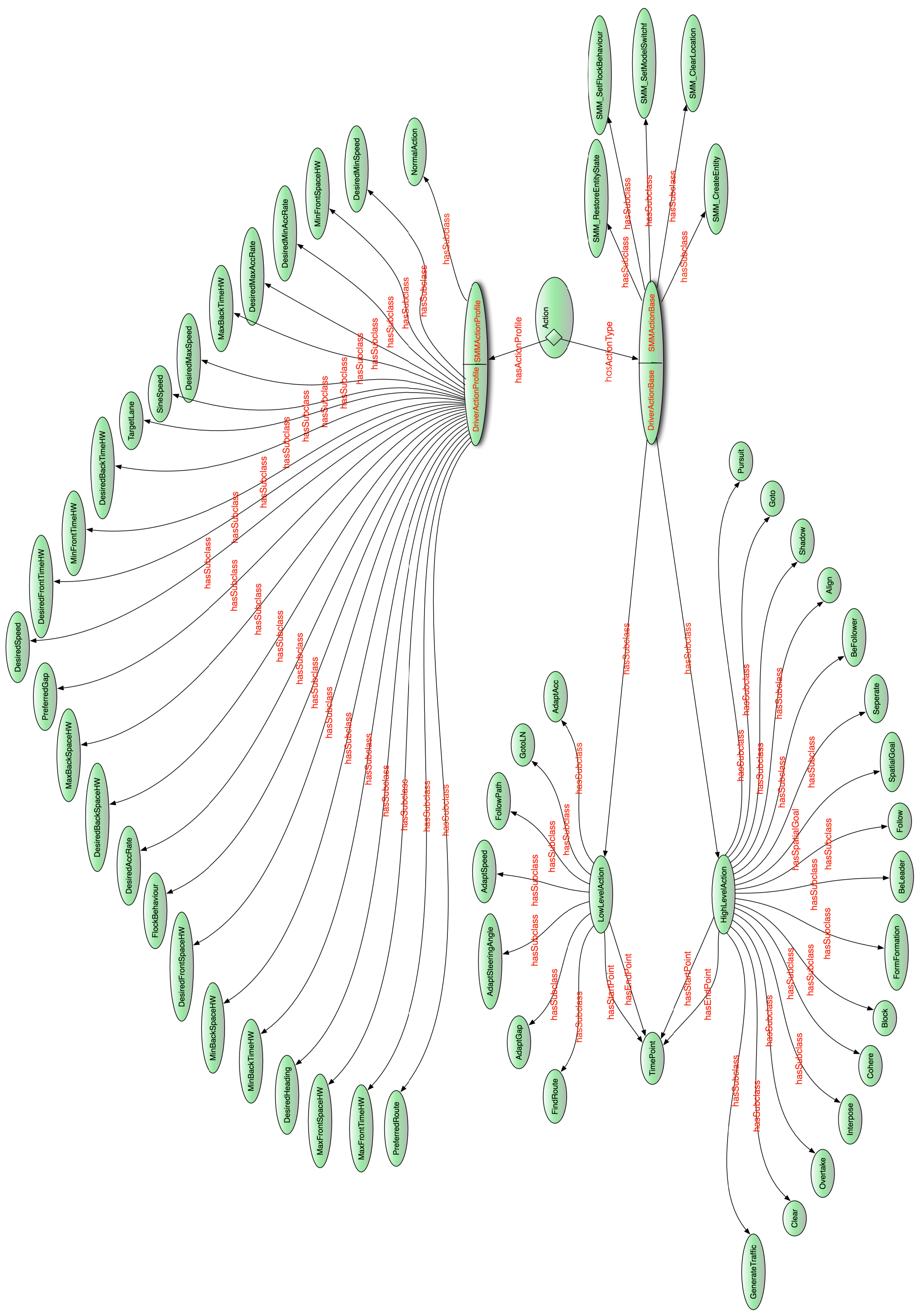


Figure 4.17: Class *Action* and Related Classes

4.3.11 *Assignment*

Assignment is the most important class in OSO, it specifies the tasks of Smith. *Assignment* and its object properties have been illustrated in Figure 4.18:

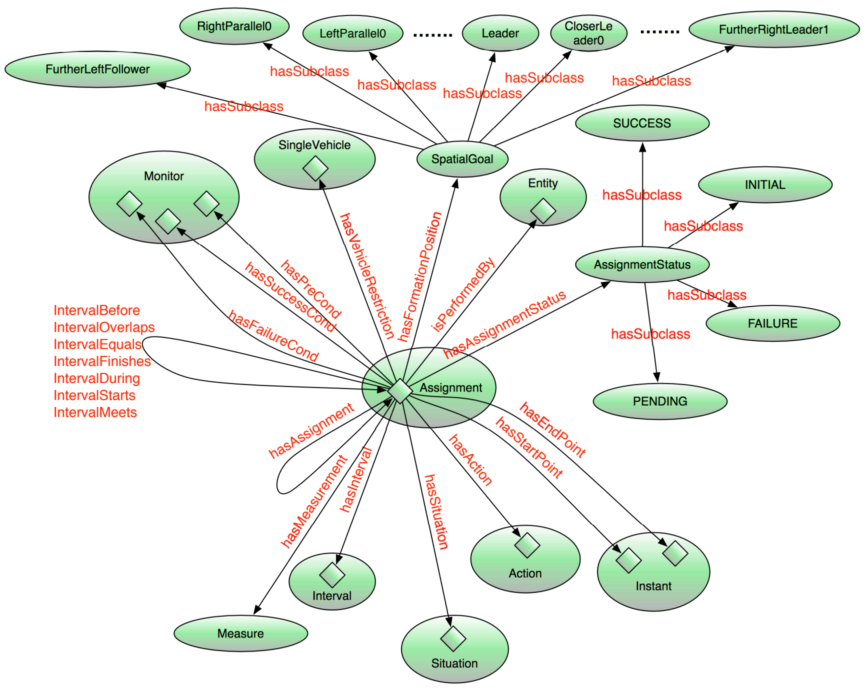


Figure 4.18: Class *Asssignment* and Related Classes

Assignment can provide the following information:

- What to Monitor: properties in this category specify the information related to Triggers, including *hasFailureCond*, *hasPreCond*, *hasSuccessCond*, etc.;

- What to Trigger: properties in this category specify what should be executed when the Monitor becomes true. It includes the following properties: *hasAction*, *hasAssignment*, *hasSituation*;

- Additional Information: properties in this category specify some general information of the Assignment, including *hasAssignmentStatus*, *hasAssignment-*

Type, *hasStartPoint*, *hasEndPoint*, *hasFormationPosition*, *hasMaxTriedTime*, *hasVehicleRestriction*, *hasInterval*, *isCarSwapAllowed*, *isPerformedBy*, *hasMeasurement* and some other properties that specify the temporal relationships between Assignments as illustrated in Section 4.3.5.

In the rest of this section, those properties will be introduced one by one except the properties used for temporal relationships that have already been covered in Section 4.3.5:

- *hasMonitor* relates an individual of *Assignment* to a Boolean value, indicating if this Assignment has a Monitor to trigger its proposed Action/Situation/Assignment;
- *hasPreCond* relates an individual of *Assignment* to a minimum of zero individuals of *Monitor*, indicating the Monitors used to trigger Action/Situation/Assignment;
- *hasPostCond* relates an individual of *Assignment* to a Boolean value, indicating if this Assignment has Success Conditions that can mark the Assignment as *SUCCESS* if they become true;
- *hasSuccessCond* relates an individual of *Assignment* to a minimum of zero individuals of *Monitor*, indicating the Success Conditions that are used to specify if the Assignment has succeeded. Moreover, the Assignment can be succeeded because the Action/Situation/Assignment duration has expired;
- *hasFailCond* relates an individual of *Assignment* to a Boolean value, indicating if this Assignment has Failure Conditions that can mark the Assignment as *FAILURE* if the conditions become true;
- *hasFailureCond* relates an individual of *Assignment* to a minimum of zero individuals of *Monitor*, indicating the Monitor that will be used as Failure Conditions. If they become true, the Assignment will be marked as *FAILURE*. Those conditions can also be used by simulator users to indicate or acknowledge what reactions the participant may have during an Assignment;

- *hasAssignmentType* indicates the type of an Assignment as a data property:
  - Value 1: An Assignment that will trigger Actions;
  - Value 2: An Assignment that will change the state of Assignments;
  - Value 3: An Assignment that will trigger Actions and Assignments;
  - Value 4: An Assignment that will trigger Situations;
  - Value 5: An Assignment that will trigger Actions and Situations;
  - Value 6: An Assignment that will trigger Assignments and Situations;
  - Value 7: An Assignment that will trigger Actions, Assignments and Situations;
- *hasAction* relates an individual of *Assignment* to a minimum of zero individuals of *Action*. It indicates what Action should be triggered;
- *hasAssignment* specifies which Assignment should be marked as $PENDING$ if the Monitor(s) is satisfied. The triggered Assignment is called a child-Assignment while the one that triggers the child-Assignment is called parent-Assignment. In this case, the child-Assignment will start to dictate the simulation to see if its Monitor(s) has been satisfied. An individual of *Assignment* can be related to a minimum of zero individuals of *Assignment*;
- *hasSituation* relates an individual of *Assignment* to a minimum of zero individuals of *Situation*, indicating what Situation(s) should be triggered because of precedence constraints or based on Monitor(s);
- *hasAssignmentStatus* indicates the status of a specific Assignment: $FAILURE$, $INITIAL$, $PENDING$ or $SUCCESS$, which are the subclasses of *AssignmentStatus*, so it relates an individual of *Assignment* to some individual of *AssignmentStatus*.
- *hasStartPoint* relates an individual of *Assignment* to exactly one individual of *Instant*, indicating the proposed start time of an Assignment;

- *hasEndPoint* relates an individual of *Assignment* to exactly one individual of *Instant*, indicating the proposed finish time of an Assignment;
- *hasFormationPosition* relates an individual of *Assignment* to some individual of *SpatialGoal*, which specifies where the Ego-vehicle/flock should be driven to, as illustrated in Figure 3.3.7 on Page 47;
- *hasMaxTriedTime* relates an individual of *Assignment* to an integer, indicating how many times this Assignment can be retried;
- *hasVehicleRestriction* relates an individual of *Assignment* to a maximum of one individual of *SingleVehicle*, indicating if this Assignment has some requirement regarding the Ego-vehicle;
- *hasInterval* relates an individual of *Assignment* to exactly one individual of *Interval* in order to represent the period of the Assignment;
- *isCarSwapAllowed* relates an individual of *Assignment* to a Boolean number, indicating if Smith is allowed to change his Ego-vehicle/flock if the present one cannot finish the Assignment any more;
- *isPerformedBy* relates an individual of *Assignment* to exactly one individual of *Entity*, indicating the pre-defined executer of the Assignment, which is always Smith, so *VirtualDriver* is always related;
- *hasMeasurement* relates an individual of *Assignment* to some individual of *Measure*, indicating the measurements that can be or should be collected. In this research, this property is included to demonstrate that OSO is able to indicate that some Assignments can provide these measures.

4.3.12 *RoadSegment*

RoadSegments are used frequently in this research to represent the road network for any particular scenario. Smith uses these individuals to acknowledge the logic of the road network in order to, e.g., get the travelled distance if there are more than one road segments are present. Road segments are illustrated in Figure 4.19.

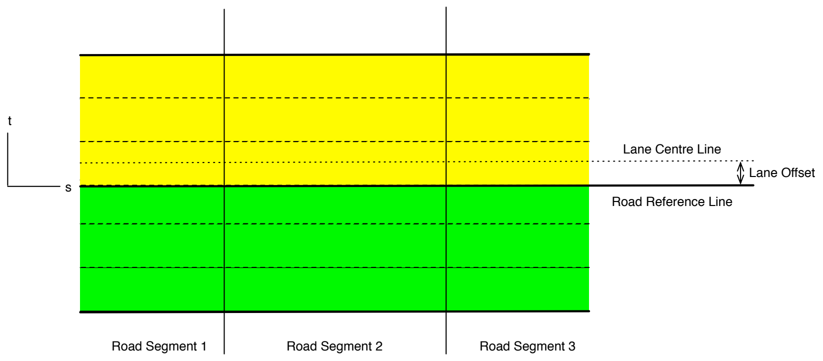


Figure 4.19: Illustration of Road Segment Along a Reference Line

Every individual of *RoadSegment*, i.e., every road segment, has the following properties that are used by Smith:

- *hasPredecessor*: an object property that relates one road segment to some road segments or junctions, indicating its predecessors if any;
- *hasSuccessor*: an object property that relates one road segment to some road segments or junctions, indicating its successors if any;
- *hasLength*: a data property that indicates the length of the road segment;
- *hasSpeedLimit*: a data property that indicates the speed limit of a road segment.

Properties above that are used to describe *RoadSegment* are illustrated in Figure 4.20 along with relevant classes.

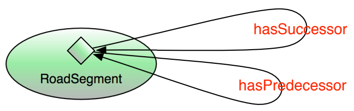


Figure 4.20: Class *RoadSegment* and Related Classes

Other properties, such as lane number, lane offset, etc. have not been modelled along with *RoadSegment* in OSO as they are not the same in different platforms and do not need to be fed to Smith beforehand. However, Smith has relevant data structures and decision-making processes to handle such platform-dependent information.

4.3.13 *Intersection*

The class *Insection* represents road junctions where road segments meet. Three properties are used to describe an intersection:

- *hasPredecessor*: an object property that relates an intersection to some road segments indicating the intersection's predecessors if any;
- *hasSuccessor*: an object property that relates an intersection to some road segments, indicating the intersection's successors if any;
- *hasSpeedLimit*: a data property that indicates the speed limit within an intersection.

4.3.14 *SimLimitation*

SimLimitation is a class that represents limitations of the Sim. A sub-class of *SimLimitation* named *IgnoredObject* has been created to indicate all entities that are invisible to both the simulated vehicles and Smith, which can be further refined to indicate any specific ones, e.g., a class named *Cone* can be used to indicate that cones are not visible so simulated vehicles cannot perceive their existence.

4.4 Summary

By representing concepts and their relationships in the domain of scenario orchestration in driving simulation, OSO has been developed to describe the essential knowledge in scenarios:

1) the proposed context as well as requirements for generating interactions between simulated vehicles and the participant's vehicle;

2) the road network;

3) the proposed simulated vehicles needed for particular interactions;

4) the position that they should be driven to before the interactions;

5) the Actions that should be performed in order to generate the interactions;

6) the conditions that should trigger/fail/finish those Actions and,

7) proposed measurements that can be collected from those interactions.

Scenario requirements regarding interactions have been encoded into Assignments that are represented by class *Assignment* in OSO.

As a result, OSO should be able to not only model driving context along with Assignments for generating required interactions, but also represent knowledge in a programming language-independent and logic-based manner. It should be human-readable and machine-processable.

As OSO is not the major concern in this research and time allocated for OSO development is limited, the evaluation of OSO will be focused on testing its expressiveness by describing scenarios used in this research. Moreover, in order to guide future research, it can also be evaluated against some general criteria to identify both its advantages and disadvantages.

In the next chapter, the driver model SAIL (Scenario Aware drIver modeL) will be introduced along with the major concern of this research - NAUSEA, which is the decision-making algorithm used by SAIL. SAIL/NAUSEA needs OSO as a data source. How Assignments for experiments are described using OSO will be presented in Appendix C. OSO evaluation will be included in Chapter 10.

Chapter 5

The Driver Model SAIL/NAUSEA

[Neo sees a black cat walk by them, and then a similar black cat walks by them just like the first one]
Neo: *Whoa. Déjà vu.*
Neo: *What is it?*
Trinity: *A Déjà vu is usually a glitch in the Matrix. It happens when they change something.*
-The Matrix

5.1 Introduction

NAUSEA (autoNomous locAl manoeUvre and Scenario orchEstration based on automated action plAnning) can be used by a Virtual Driver in driving simulation to generate interactions according to the driving contexts and scenario requirements encoded in Assignments. However, when it comes to application, an algorithm alone cannot be used and thus verified as it requires data for decision making and interfaces for executing decisions or Assignment-actions, which are used to control the Ego-vehicle/flock.

In order to support this algorithm, e.g., providing driving context and scenario Assignments, a driver model named SAIL (Scenario-Aware drIver modeL) has been developed. NAUSEA will be responsible for the decision making-related modules in SAIL, so SAIL/NAUSEA will be used to indicate a driver model SAIL equipped with NAUSEA.

A Virtual Driver who is equipped with SAIL/NAUSEA is endowed with abilities of temporal reasoning, Role Matching, scenario execution and scenario re-planning if necessary by maintaining the General Plan for the whole scenario.

In this chapter, the focus will be placed on NAUSEA but SAIL will be introduced first. It should be noted that NAUSEA can be used to handle driving and non-driving Actions, but this research primarily focuses on the coordination of (autonomous) simulated vehicles' behaviours, so NAUSEA has been mainly used to handle driving Actions.

5.2 SAIL

SAIL was proposed in order to provide data for decision making and interfaces for executing decisions or Assignments, so it has been directly developed based on some existing architectures. In this section, the driver model developed in Chapter 3 will be described in more detail, especially regarding the Decision Making layer.

The ECOM model (Extended Control Model) (Engström & Hollnagel, 2007) has four layers and has inspired the design of the Decision Making layer.

Targeting is used to set goals for driving such as the destination. Monitoring is used to set objectives and plans for Actions and can be used to monitor the condition of vehicles. Regulating is used to drive safely and concentrate on the relative positions between a vehicle and other objects in the traffic, so decisions regarding lane choice, overtaking or obstacle avoidance can be made. Tracking is used to execute the decisions from Regulating and attempting to, e.g., maintain a speed or a safe distance. Because each layer requires a different level of effort from the driver, ECOM also includes some indications regarding the effort required by that layer: anticipatory or compensatory.

Assignments can be regarded as a part of Situation Awareness, which Gugerty (2011) defines as "*a type of knowledge*" and includes factors such as fuel level, destination etc. However, because of limitations in driving simulation, e.g., limited

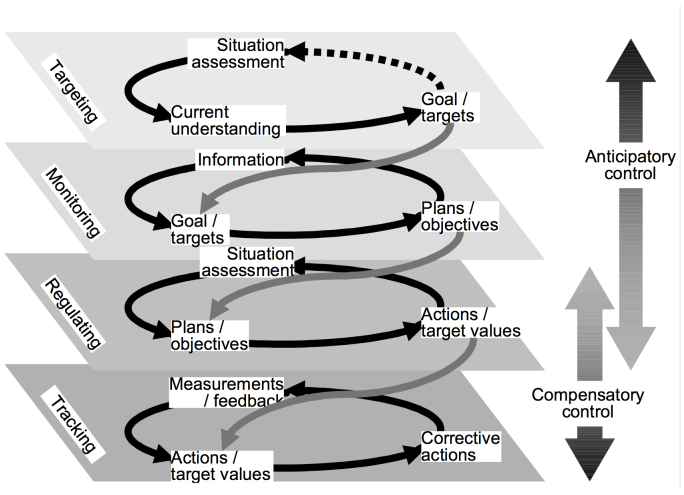


Figure 5.1: The ECOM Architecture from Engström & Hollnagel (2007)

factors in Situation Awareness such as fuel level, Situation Assessment in Regulating has been especially designed as a separate layer to replace the Monitoring layer. Hence, Assignments will be handled by Situation Assessment, which is also used to set objectives and plans for Actions. Although this Virtual Driver will be used to control the simulated vehicles in driving simulation, the new Situation Assessment layer is not designed to monitor the condition of the simulated vehicle, as the condition is assumed to be malfunction-free. In addition, the Tracking layer has been ignored as the decisions from Regulating will be directly sent to the simulated vehicles.

As a result, the Decision Making layer in the new driver model SAIL includes three layers: Targeting, Regulating and Situation Assessment as illustrated in Figure 5.2.

Perception is used to sense the outside environment and make necessary interpretations. A Virtual Driver should possess the following sensing abilities: vision, hearing, touch and proprioception (Peters & Nilsson, 2007b). In this research, the information received by Smith concentrates on visual information, so his sensing abilities include vision only, e.g., the leader vehicle's speed and position.

Cognition has two sub-layers: Memory and Decision Making. There are two kinds of Memory: Individual Feature and World Model. Individual Feature con-

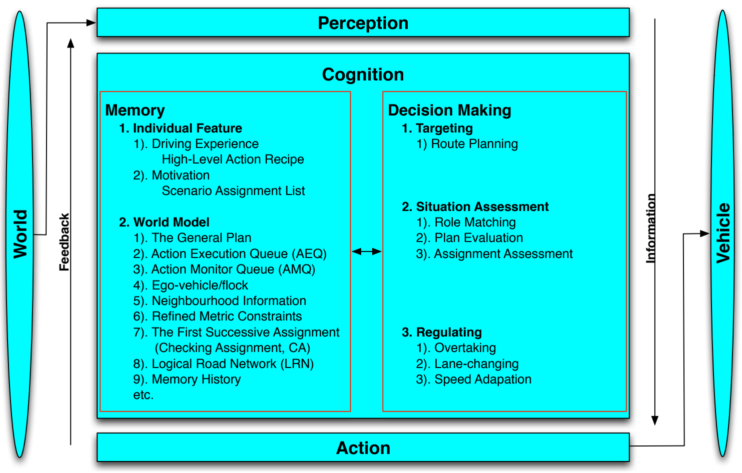


Figure 5.2: Scenario-Aware drIver modeL (SAIL)

tains exactly 14 structures derived from Dewar (2002), e.g. Experience, Personality, Emotions. However, in this research, Smith has adopted two sets of Individual Features only: Driving Experience and Motivation. Driving Experience contains Recipes and lets Smith know how to perform High-Level Actions, e.g., the "Block" Action will let Smith find three simulated vehicles to prevent the participant from overtaking. Motivation includes Assignments from SDF (Scenario Definition File). The World Model includes all the contexts for driving, e.g., logical road network, previous Memory of the World Model (Memory History), etc.

The Decision Making layer has three sub-layers, namely Targeting, Situation Assessment and Regulating. Targeting is used to plan the route for Smith and set goals for the Situation Assessment. The Situation Assessment is used to supervise the Assignments and route. Regulating uses any relevant tactical driving behaviours to drive safely and satisfy the goals of the Situation Assessor. This decision making layer uses NAUSEA to maintain and carry out the General Plan, which is the plan Smith needs to follow in order to finish the whole scenario.

The Action layer is a network communication module for publishing orders to corresponding Ego-vehicle/flock. It sends out Smith Orders, which are instructions that the Ego-vehicle/flock should follow or SMM should carry out. This layer can also notify every layer whether or not an Action has been sent. Smith Orders can come from the Situation Assessment or the Regulating layers.

In the sections below, the following layers or sub-layers will be introduced with more details: Perception, Individual Feature, World Model and Action. The decision making algorithm NAUSEA that is used for the Decision Making layer will be elaborated in Section 5.3.

5.2.1 Perception

Perception in SAIL involves two procedures: one is to sense the outside world by receiving raw data; the other is to interpret the raw data and maintain the World Model. Raw data mainly consists of information regarding all vehicles, including the participant's vehicle. Table 5.1 includes the information of one UDP package broadcast by the Sim, which is used as the raw data for Smith.

Table 5.1: Package Format

Data	**Unit**	**Type**	**Function**
Time Stamp	second(s)	Float	the simulation time in Sim, which starts from zero seconds
Car Count	N/A	Integer	the number of vehicles in Sim including the participant's vehicle
Vehicle Information	N/A	Array	every vehicle's real-time information, the size of this data should be equal to the Car Count

As shown in Table 5.1, Vehicle Information is an one dimension array whose size equals to the Car Count. Each element of the array contains one vehicle's information, which complies with the format shown in Table 5.2.

In a single simulation time frame, the Sim will pack the data with the following procedure:

- Car Count is first obtained;
- when the Sim is updating each vehicle's state in the simulation, e.g., position, the corresponding element in the Vehicle Information will be updated

Table 5.2: Data Format of Vehicle Information

Data	**Unit**	**Type**	**Function**
Vehicle's ID	N/A	Integer	ID of the vehicle, which is unique in the simulation
Flock's ID	N/A	Integer	ID of the vehicle's Flock, this indicates which Flock the vehicle is in
Vehicle Type	N/A	Integer	type of the vehicle, e.g., 1 means lorry/truck
Road ID	N/A	String	ID of the road segment on which the vehicle is travelling on, e.g., $r3.1$ indicates the road segment no. 3.1
Road Distance	metre(m)	Float	distance travelled along the present road segment, this value should be bigger than 0, but smaller than the length of the road segment
Road Offset	metre(m)	Float	vehicle's position offset to the centre line of the road segment, see Chapter 4 for details
Speed	m/s	Float	speed of the vehicle, which is assumed to be non-negative in this research, i.e., no reverse
Acceleration Rate	m/s^2	Float	acceleration (positive) or deceleration (negative) rate of the vehicle
Overtaking Intention	N/A	Integer	status of the overtaking behaviour, 0 means no overtaking is planned, 1 means overtaking has finished and 2 means overtaking is being undertaken
Leader's ID	N/A	Integer	ID of the vehicle's leader
Follower's ID	N/A	Integer	ID of the vehicle's follower
Offside Leader's ID	N/A	Integer	ID of the nearest vehicle ahead in the right adjacent lane based on left driving rule
Offside Follower's ID	N/A	Integer	ID of the nearest vehicle behind in the right adjacent lane based on left driving rule
Nearside Leader's ID	N/A	Integer	ID of the nearest vehicle ahead in the left adjacent lane based on left driving rule
Nearside Follower's ID	N/A	Integer	ID of the nearest vehicle behind on the left adjacent lane based on left driving rule

and each vehicle will have a unique element position in the Array. Hence, when the ID of the vehicle has been identified, its state in the Sim can be obtained and,

- after the update of the Vehicle Information, the time stamp of the Sim is retrieved and assigned to "Time Stamp".

Then, the package will be sent to the network and Smith interprets these data, updating three kinds of memories in the World Model accordingly:

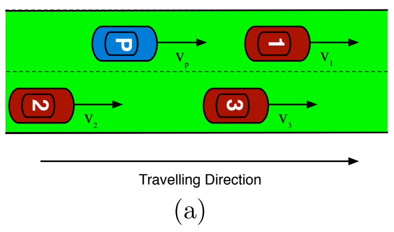
- Participant's Information: the participant's driving status will be updated, e.g., vehicle's position, speed;

- Neighbourhood Information: after Smith has received the participant's information, the relationships between the participant's vehicle and other simulated vehicles will be constructed.

  OSO (The Ontology for Scenario Orchestration) used the complete version of Formation Position. However, a simpler version of Formation Position is adopted in Smith's World Model and represented in OSO as state variables. Figure 5.3(b) is an illustration of the simpler version. This version of Neighbourhood includes all the 12 positions that are of most interest to Smith.

  As illustrated in Figure 5.3(a), Smith is able to build a Neighbourhood around the participant's vehicle to represent the formation around the participant. For instance, from the package, the participant's leader can be obtained by examining the record of the first element (at position 0, as its ID is 0) of Vehicle Information. Leader is vehicle no. 1, so this vehicle's information can be found in the element position of 1 in the array Vehicle Information. As a result, the corresponding information regarding vehicle no. 1 will be written into the Neighbourhood and occupy the "L" element of the Neighbourhood (see Figure 5.3(b));

- Memory History: After Smith has updated the Memory according to the package sent from the Sim and updates from other layers, e.g., updates regarding potential Smith Orders, containing the proposed Actions that



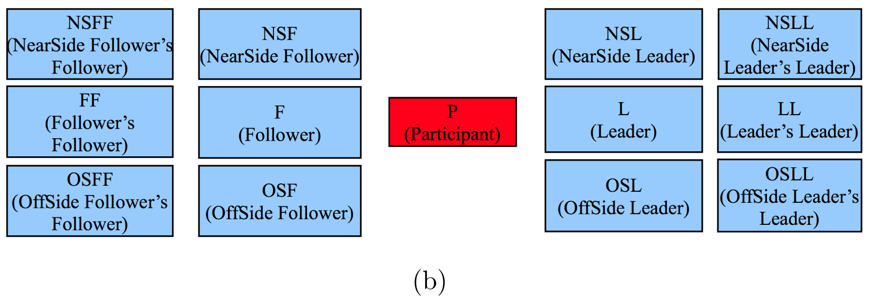


Figure 5.3: Neighbourhood Generation Example

should be executed by corresponding Ego-vehicle/flock or SMM, Smith will then store the Memory into Memory History, which will be used by Triggers to detect state transition/events.

5.2.2 Individual Features

The most common High-Level Actions for Smith are the Actions of β_1 (Generate-formation) and β_2 (Perform-assignment). However, not all the High-Level Actions have Recipes. Smith contains Recipes of 1) how to perform some High-Level Actions for Assignment, e.g., "Block", by specifying recipes in the Driving Experience of Memory or 2) how to perform β_2 by using pre-defined Assignment-actions stored in Assignments.

Motivation stores all the Assignments that Smith needs to finish during the Action of β_2 (Perform-assignment). They cannot be changed during simulation, but interfaces can be provided to modify/delete/add Assignments online.

5.2.3 World Model

The World Model provides Smith with the current driving context and relevant interfaces for modifications or queries: the General Plan, the status of Assignments, the status of Smith and all other driving contexts, e.g., route. It has two main functions:

- Storing Driving Context and Assignment Status: the World Model maintains a model of the driving activity, which refers to the General Plan (including Assignment status), self information (Smith), participant's vehicle's information, Ego-vehicle/flock's information, Neighbourhood information and the information regarding the physical environment. It also keeps a history of the previous World Model for ten simulation time frames. The Driving Context can also include driving regulations and traffic rules, however, they have been ignored in this research due to their absence in most scenarios.

- Modifying World Model and Answering Queries from other layers:

  – The World Model can be modified by other layers in order to keep the information uptodate in every decision loop, e.g., the modification of Smith Orders or Assignment Status;

  – Logical Road Network (LRN) Queries provide answers regarding relationships between different road facilities, e.g., the successor of the present road segment;

  – Plan Queries provide answers regarding the General Plan, e.g., its consistency, specific metric constraints between two Assignments/Actions, precedence constraints between two Assignments;

  – Assignment Queries provide answers about online information regarding Trigger status, running status, Assignment-action status, proposed deadline of Action, etc. and,

  – Vehicle Queries provide answers regarding any simulated vehicles that are of interest to Smith: Ego-vehicle/flock and the ones in the Neighbourhood.

The OpenDRIVE format (Dupuis, 2011) has been used to design the data structure of World Model regarding road networks, although it has not been fully described in OSO. The OpenDRIVE format is designed to describe road networks regarding logic relationships and features such as lanes, signs, geometry property etc. It is organised in nodes and thus can be extended with user data (Dupuis, 2011). As it is designed to be a standard in describing the road network's logic and providing a way of exchanging relevant data between different platforms with the capability of extension (OpenDRIVE, 2013), this format has been adopted by Smith to provide a logical memory of the road network.

5.2.4 Action

The Action layer is in charge of broadcasting Smith Orders. As traffic flow is not the concern of this research, traffic flow creation and maintenance will not be covered and is always handled by existing modules in the Sim and Smith will just use that traffic flow to orchestrate scenarios.

One Smith Order can contain the following major Action indicators:

- Action Vehicle: the ID of the Ego-vehicle that the order should be executed by;
- Action Flock: the ID of the Ego-flock that the order should be executed by;
- Time Stamp: the time when the order if published;
- Action Lane: the lane that the Ego-vehicle/flock should go to;
- Action Speed: the speed that the Ego-vehicle/flock should travel with;
- Target Speed: the speed that the Ego-vehicle/flock should try to regulate to. This Action indicator has been used with Action Speed interchangeably as changing the speed of Ego-vehicle/flock directly is not used in this research;
- Acceleration: the acceleration rate that the Ego-vehicle/flock should use;
- Lane-changing permit: this is used to indicate if lane-changing is permitted;

- Overtaking Intention: this is to let the Ego-vehicle overtake by setting its desired speed to a higher value than its leader. This is actually the substitution for Target Speed. This order has been left for further completeness as Smith should also be able to change the overtaking behaviour of Ego-vehicle;
- Traffic Flow: traffic flow indicator with corresponding parameters: high, low, medium or some other user-defined parameters. Traffic flow generation is not covered in this research so no parameters are needed to regulate traffic flows generated, e.g., traffic flow rate.

Moreover, there are also some indicators for SMM Actions, e.g., indicator of restoring simulated vehicles' original states or putting cones on the road. These indicators are not standardized, so they will be mentioned when necessary.

5.3 NAUSEA

As a Virtual Driver in driving simulation, Smith's task is to orchestrate scenarios according to the description recorded in SDF (Scenario Definition File). He has to interact with the participant according to Assignments. During this process, NAUSEA is used to answer the following questions raised by Smith based on the information from Assignments:

- which vehicle/flock should I control? ("Match Role")
- How should I drive my vehicle/flock there? ("Prepare Actor")
- what do I need to do for the Assignments and can I finish them in time? ("Finish Assignment")
- if something went wrong, e.g., if the Assignment failed to be triggered, what should I do? ("Replanning")

As a result, Smith needs to carry out the following sub-tasks in a scenario:

- Initialize: Smith builds a World Model based on SDF and acknowledges what he should do in the scenario;

- Match Role[1]: after initialization, Smith will build the General Plan and find an Ego-vehicle/flock to control according to the first successive Assignment(s)[2]. A subclass of *Action* named *MatchRole* was created in OSO to generate a pre-defined Assignment to indicate when or where the Match Role process should be invoked if necessary;

- Prepare Actor: after the Ego-vehicle/flock has been identified, Smith will then drive the Ego-vehicle/flock to the proposed Formation Position and prepare for the oncoming Assignments; this process will be followed by β_2, which is the "Finish Assignments" specified below;

- Finish Assignments: after Smith has formed the required formation around the participant's vehicle, he will then carry out and finish the Assignment according to Triggers, precedence constraints and metric constraints;

- Clean Up: after the Assignments, Smith will clean up the scenario by restoring the Ego-vehicle/flock with previous parameters, e.g., previous desired speed. This process is always handled by a pre-defined Assignment because by doing so, restored values can be controlled according to specific needs;

- Replanning: if there is any failure after Initialization, Smith will invoke "Match Role" and find another Ego-vehicle/flock to continue the experiment (Smith may still find the same one), so Smith will go back to the "Prepare Actor" phase.

In order to guide the execution of tasks mentioned above, Smith needs to establish the General Plan Gr_α and evaluate it according to the temporal constraints by using a Plan Evaluation procedure, which is used to see if the metric constraints are consistent. If they are consistent, Smith will then carry out the tasks above and perform relevant Assignment-actions according to any temporal constraints or Triggers in Assignments.

[1]Role Matching in this research has put a focus on local optimization by finding an optimized actor for a single Assignment. Some discussions have been provided in Chapter 11 in order to indicate the possibility of adopting Role Matching based on all successive Assignments for global optimization.

[2]In this research, two parallel Assignments imply that one of them should be a Flock-related or a traffic flow-related Assignment.

5.3.1 Definitions and Notations

In order to present NAUSEA, especially the Plan Evaluation procedure in it, some definitions are given first, although relevant concepts have been covered in Chapter 3. Those definitions are derived from Hadad *et al.* (2003), but have been simplified and tailored to the special needs of this research.

Definition 1. *Let $A = \{A_1, A_2, ..., A_n\}$ be a set of Assignments that Smith needs to carry out in a scenario and let $\theta_A^{prec} = \{(i, j)|A_i < A_j; i \neq j\}$ be a set of precedence constraints associated with A. Each precedence constraint $\{A_i < A_j(i \neq j)\}$ represents that the execution of Assignment A_j starts after the execution of A_i finishes because A_i expires or the Success Conditions of A_i are satisfied.*

Definition 2. *Let $a = \{a_1, a_2, ..., a_m\}$ be a set of Actions that Smith needs to carry out and let $\theta_a^{prec} = \{(i, j)|a_i < a_j; i \neq j\}$ be a set of precedence constraints associated with a. Each precedence constraint $\{a_i < a_j(i \neq j)\}$ represents that the execution of Action a_j starts after the execution of a_i finishes. An Action can be finished by duration-expiration or satisfaction of Success Conditions from its parent Assignment. If no precedence constraints of two Actions can be found, then the two Actions will be scheduled in parallel. If there are no Monitors associated with either Action and they share the same Durations, then they will be started and finished at the same time.*

Definition 3. *If $A = \{A_1, A_2, ..., A_n\}$ is a set of Assignments that Smith needs to carry out in a scenario, then the set of Actions that Smith needs to perform in a scenario, which is $a = \{a_1, a_2, ..., a_m\}$, are made up of pre-defined Actions and Assignment-actions extracted from set A. Let a_{A_i} and a_{A_j} $(i \neq j)$ be two subsets of a, representing all the Assignment-actions needed for the Assignment A_i and A_j respectively. Let $a_p \in a_{A_i}$ and $a_q \in a_{A_j}$ where $p \neq q$. Hence,*

- *$m > n$;*
- *if $A_i < A_j$, then $a_p < a_q$.*

Definition 4. *Let A_i be an Assignment that Smith needs to carry out in a scenario and the set $\{\gamma_0, ..., \gamma_n\}$ be the Assignment-actions in A_i. If $s_{\gamma_j} \leqslant s_{\gamma_i}$ and $f_{\gamma_k} \geqslant f_{\gamma_i}$ ($i \neq j, k$ and $0 \leqslant i, j, k \leqslant n$), then $s_{A_i} = s_{\gamma_j}$ and $f_{A_i} = f_{\gamma_k}$, where s_{γ_j},*

s_{γ_i} and s_{A_i} represent the start time of Assignment-action γ_j, γ_i and Assignment A_i respectively; f_{γ_k}, f_{γ_i} and f_{A_i} represent the finish time of Assignment-action γ_k, γ_i and Assignment A_i respectively.

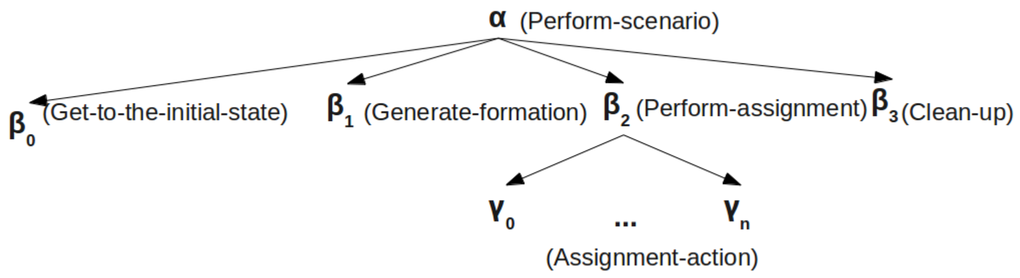


Figure 5.4: Action Recipe for Smith (*Perform-scenario*)

Definition 5. *Given the recipe shown in Figure 5.4, let $\{\alpha, \beta_0, \beta_1, \beta_2, \beta_3\}$ be a set of pre-defined Actions that Smith needs to perform in a scenario where α is the top-Action named "Perform-scenario" and β_0 is "Get-to-the-initial-state", β_1 is "Generate-formation", β_2 is "Perform-assignment", β_3 is "Clean-up". Let $\{\gamma_1, \gamma_2, ..., \gamma_m\}$ be a set of Assignment-actions extracted from the Assignment set $\{A_1, A_2, ..., A_n\}$ ($n \leqslant m$). Hence,*

- *let $\{\alpha, \beta_0, \beta_1, \beta_2, \gamma_0, \gamma_1, ..., \gamma_m, \beta_3\}$ be the set of Actions that Smith derives from pre-defined Actions and Assignment-actions from Assignments;*

- *let $V_\alpha = \{s_{start}, s_\alpha, s_{\beta_0}, s_{\beta_1}, s_{\beta_2}, s_{\gamma_0}, s_{\gamma_1}, ..., s_{\gamma_m}, s_{\beta_3}, f_\alpha, f_{\beta_0}, f_{\beta_1}, f_{\beta_2}, f_{\gamma_0}, f_{\gamma_1}, ..., f_{\gamma_m}, f_{\beta_3}\}$ be a set of Instants, where s_{start} represents the start of the simulation, and Instants starting with s and f represent the start and finish times of each Action respectively. Specially, f_α represents the finish of the scenario, which is not necessarily the finish of the simulation as the simulation can be terminated by external operators such as experimenters. $|V_\alpha|$ represents the number of Instants in V_α;*

- *let $\theta_A^{prec} = \{(i, j)|A_i < A_j; i \neq j\}$ be a set of precedence constraints regarding Assignments and $\theta_{A_i}^{prec} = \{(o, q)|\gamma_o < \gamma_q; o \neq q\}$ be a set of precedence constraints of Assignment-Actions in each Assignment A_i. Let θ_α^{prec} be a general* *set of precedence constraints derived from θ_A^{prec} and set $\{\theta_{A_0}^{prec}, \theta_{A_1}^{prec}, ..., \theta_{A_n}^{prec}\}$, specifying the precedence constraints regarding the set $\{\gamma_1, \gamma_2, ..., \gamma_m\}$;*

- *let $\theta_\alpha^{metric} = \{u_i, v_j \ (1 \leqslant i, j \leqslant |V_\alpha|) \mid a_{i,j} \leqslant (u_i - v_j) \leqslant b_{i,j}\}$ be a set of metric constraints that are associated with $V_\alpha = \{s_{start}, s_\alpha, s_{\beta_0}, s_{\beta_1}, s_{\beta_2}, s_{\gamma_0}, s_{\gamma_1}, ..., s_{\gamma_m}, s_{\beta_3}, f_\alpha, f_{\beta_0}, f_{\beta_1}, f_{\beta_2}, f_{\gamma_0}, f_{\gamma_1}, ..., f_{\gamma_m}, f_{\beta_3}\}$; u_i, v_j are two different Instants from V_α , while $a_{i,j}, b_{i,j}$ are two numbers. θ_α^{metrc} represent the differences between two Instants.*

Then, the General Plan Gr_α is a temporal constraint graph (V_α, E_α) where E_α is a set of edges connecting vertices from V_α. Gr_α therefore contains:

- *the vertex set $V_\alpha = \{s_{start}, s_\alpha, s_{\beta_0}, s_{\beta_1}, s_{\beta_2}, s_{\beta_3}, s_{\gamma_0}, s_{\gamma_1}, ..., s_{\gamma_m}, f_\alpha, f_{\beta_0}, f_{\beta_1}, f_{\beta_2}, f_{\gamma_0}, f_{\gamma_1}, ..., f_{\gamma_m}, f_{\beta_3}\}$;*
- *the edge set E_α that specifies:*
  - *there is a set of duration edges $E_\alpha^{duration} = \{(s_\alpha, f_\alpha), (s_{\beta_0}, f_{\beta_0}), (s_{\beta_1}, f_{\beta_1}), (s_{\beta_2}, f_{\beta_2}), (s_{\gamma_0}, f_{\gamma_0}), ..., (s_{\gamma_m}, f_{\gamma_m}), (s_{\beta_3}, f_{\beta_3}),\} \subseteq E_\alpha$ and each edge is either labelled by the defined duration of that Action or ∞;*
  - *there is a set of delay edges $E_\alpha^{delay} = \{(u_1, v_1), (u_2, v_2), ..., (u_n, v_n)\} \subseteq E_\alpha$ where u_i is the start/finish time of some Action and v_i is the start/finish time of another Action.*
  - *the delay edges $(u_i, v_j) \in E_\alpha^{delay}$ are labelled by weights that reflect corresponding metric constraints:*

$$weight(u_i, v_j) = \begin{cases} [a_{i,j}, b_{i,j}], & \text{if } u_i, v_j \in \theta_\alpha^{metric} \\ \infty, & \text{if } u_i, v_j \notin \theta_\alpha^{metric} \end{cases} \tag{5.1}$$

5.3.2 Algorithm Description

Basically, NAUSEA operates as a procedural process. Figure 5.3 provides an overall summary of the mechanism of NAUSEA in SAIL.

The rest of this section will elaborate the procedures in NAUSEA as shown in Figure 5.3 with more details. Generally speaking, NAUSEA has been designed with several sub-procedures, which have been summarised in Algorithm 1 on Page 110:

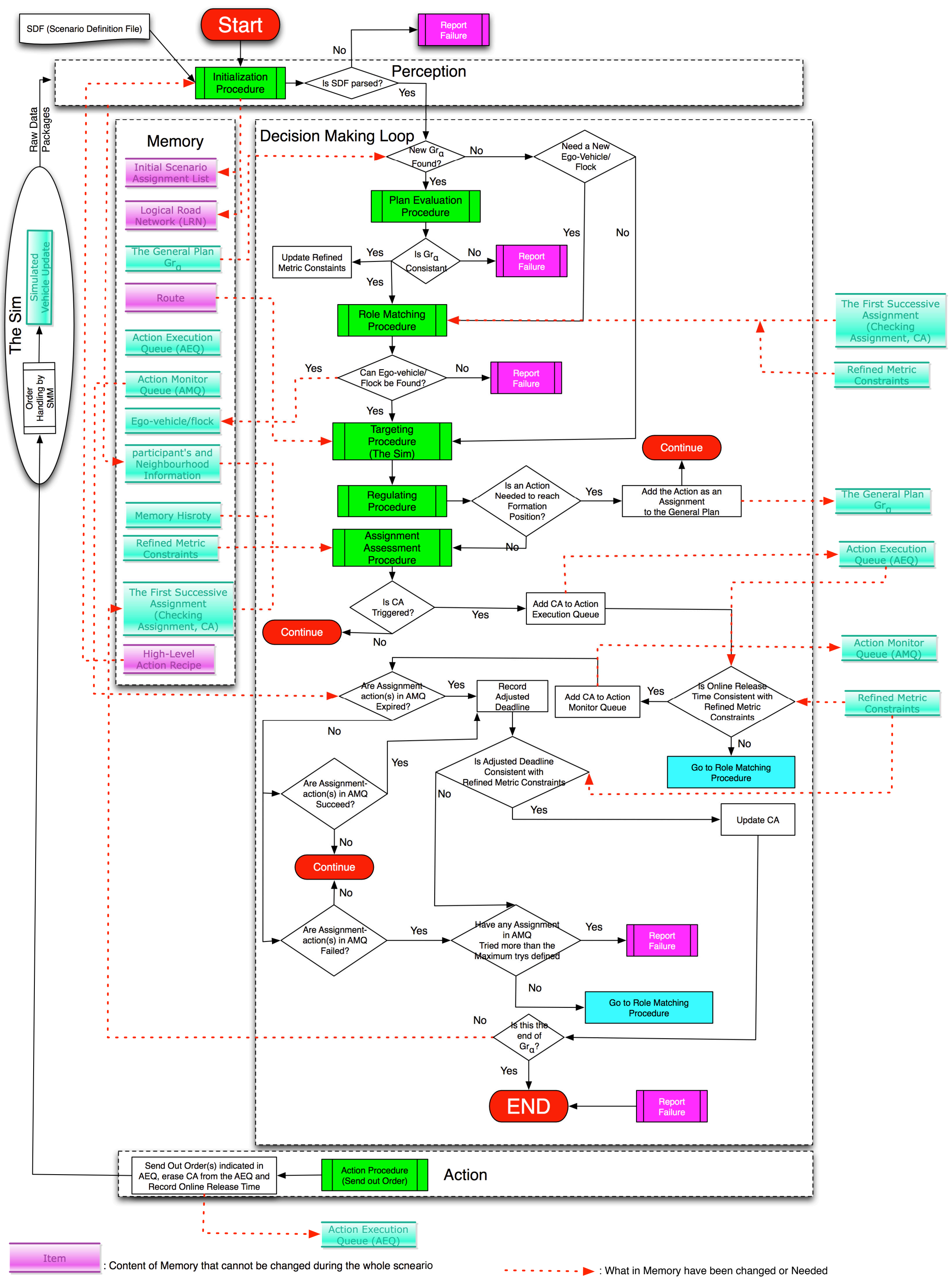

SDF (Scenario Definition File)
Start
Report Failure
No
Initialization Procedure
Is SDF parsed?
Perception
Yes
Raw Data Packages
Memory
Initial Scenario Assignment List
Logical Road Network (LRN)
The General Plan Gr_α
Route
Action Execution Queue (AEQ)
Action Monitor Queue (AMQ)
Ego-vehicle/flock
participant's and Neighbourhood Information
Memory Hisroty
Refined Metric Constraints
The First Successive Assignment (Checking Assignment, CA)
High-Level Action Recipe
The Sim
Simulated Vehicle Update
Order Handling by SMM
Decision Making Loop
New Gr_α Found?
No
Need a New Ego-Vehicle/ Flock
Yes
Plan Evaluation Procedure
Yes
No
Update Refined Metric Constaints
Yes
Is Gr_α Consistant
No
Report Failure
Yes
Role Matching Procedure
The First Successive Assignment (Checking Assignment, CA)
Refined Metric Constraints
Yes
Can Ego-vehicle/ Flock be Found?
No
Report Failure
Yes
Targeting Procedure (The Sim)
Continue
Regulating Procedure
Is an Action Needed to reach Formation Position?
Yes
Add the Action as an Assignment to the General Plan
The General Plan Gr_α
Assignment Assessment Procedure
No
Action Execution Queue (AEQ)
Is CA Triggered?
Yes
Add CA to Action Execution Queue
Continue
No
Action Monitor Queue (AMQ)
Are Assignment action(s) in AMQ Expired?
Yes
Record Adjusted Deadline
Add CA to Action Monitor Queue
Yes
Is Online Release Time Consistent with Refined Metric Constraints
Refined Metric Constraints
No
No
Go to Role Matching Procedure
Yes
Are Assignment-action(s) in AMQ Succeed?
Is Adjusted Deadline Consistent with Refined Metric Constraints
No
Yes
Update CA
No
Continue
No
Are Assignment-action(s) in AMQ Failed?
Yes
Have any Assignment in AMQ Tried more than the Maximum trys defined
Yes
Report Failure
No
Go to Role Matching Procedure
No
Is this the end of Gr_α?
Yes
END
Report Failure
Send Out Order(s) indicated in AEQ, erase CA from the AEQ and Record Online Release Time
Action Procedure (Send out Order)
Action
Action Execution Queue (AEQ)
Item
: Content of Memory that cannot be changed during the whole scneario
: What in Memory have been changed or Needed

Figure 5.5: The Mechanism of SAIL/NAUSEA

1) Initialization procedure: Lines 3 to 12;

2) Plan Evaluation procedure: Lines 14 to 22;

3) Role Matching procedure: Lines 23 to 34;

4) Targeting procedure: Lines 35 to 41;

5) Regulating procedure: Lines 42 to 45;

6) Assignment Assessment procedure: Lines 46 to 55;

7) Action procedure: Lines 56 to 59;

8) Failure Broadcast procedure: Line 61.

Each sub-procedure will be elaborated in more detail from the next section. The Role Matching and Assignment Assessment procedures will be introduced with pseudo codes as well.

5.3.2.1 Plan Evaluation

The Plan Evaluation procedure takes the original General Plan Gr_α generated in the Initialization procedure as input and see if the metric constraints in Gr_α are consistent by generating a set of refined metric constraints from the original ones in Gr_α. The metric constraints are considered to be inconsistent if the minimum difference between an individual Instant (e.g., the start time of an Assignment-action) and itself is smaller than zero. The Floyd-Warshall algorithm (Cormen *et al.*, 2001) has been used to evaluate the General Plan Gr_α and an example has been given to demonstrate how the plan evaluation procedure works in the algorithm and what the refined metric constraints mean to Smith.

IntervalBefore has been implied as the only precedence constraint for building the General Plan. However, *IntervalFinishes* has been included in some examples to indicate that the two Actions should be finished at the same time. This is used to handle ambient traffic flow generation. *IntervalFinishes* is also used to demonstrate that the plan evaluation procedure can handle other constraints as well by building corresponding nodes and edges in the General Plan. As illustrated in Figure 5.6(a), γ_4 and γ_2 will be finished at the same time.

Algorithm 1 Mechanism of NAUSEA

Require: SDF (Scenario Definition File) for Initialization and Memory in SAIL for real-time decision making.

Output: Smith Orders, i.e., Assignment-actions or Actions required by Regulating Procedure, i.e., speed/acceleration rate modifications or Overtaking/Lane-changing intentions/permissions.

```
1. $Initialized, SDF\_Parsed, Plan\_Evaluation\_Invoked, New\_Plan\_is\_Found, End\_of\_Plan, New\_Actor\_is\_Needed \leftarrow$ false
2. $Order\_To\_Be\_Sent \leftarrow \emptyset$
3. $SDF\_Parsed \leftarrow$ parse SDF
4. if $SDF\_Parsed$ then
5.     write parsed information into Memory
6.     $Gr_\alpha \leftarrow$ build initial General Plan based on Memory
7.     $SDF\_Parsed \leftarrow$ false
8.     $Initialized, Plan\_Evaluation\_Invoked, New\_Plan\_is\_Found, New\_Actor\_is\_Needed \leftarrow$ true
9.     $CA \leftarrow$ the first successive Assignment(s) from $Gr_\alpha$
10. else
11.     go to Line 61
12. end if
13. while $\{Initialized$ and not $End\_of\_Plan\}$ do
14.     if $Plan\_Evaluation\_Invoked$ or $New\_Plan\_is\_Found$ then
15.         $Plan\_is\_Consistent \leftarrow$ check consistency of $Gr_\alpha$
16.         if $Plan\_is\_Consistent$ then
17.             store refined metric constraints into Memory
18.             $New\_Actor\_is\_Needed \leftarrow$ true
19.         else
20.             go to Line 61
21.         end if
22.     end if
23.     if $New\_Actor\_is\_Needed$ then
24.         $Found\_Actor \leftarrow$ Perform $Role\_Matching()$
25.         if $Found\_Actor \neq \emptyset$ then
26.             pass the new Actor ID $Found\_Actor$ to Memory and set it as Ego-vehicle/flock ID
27.             $New\_Actor\_is\_Needed \leftarrow$ false
28.             $Plan\_Evaluation\_Invoked \leftarrow$ true
29.         else
30.             go to Line 61
31.         end if
32.     else
33.         continue
34.     end if
35.     $New\_Route\_is\_Found \leftarrow$ get route based on A* algorithm or pre-defined route in Memory
36.     if $New\_Route\_is\_Found$ then
37.         write new route to Memory if it is from A* algorithm
38.         continue
39.     else
40.         $New\_Actor\_is\_Needed \leftarrow$ true
41.     end if
42.     if new Action is needed to navigate to the required Formation Position then
43.         $Gr_\alpha \leftarrow$ add the new Action in the recipe of $\beta_1$
44.         $Plan\_Evaluation\_Invoked \leftarrow$ true
45.     end if
46.     $[Order\_To\_Be\_Sent, Assignment\_Failure] \leftarrow$ perform $Assignment\_Assessment\_Procedure()$
47.     if $\{CA = \emptyset\}$ then
48.         $End\_of\_Plan \leftarrow$ true
49.         $Assignment\_Failure \leftarrow$ false
50.     end if
51.     if $Assignment\_Failure$ then
52.         $New\_Actor\_is\_Needed \leftarrow$ true
53.     else
54.         continue
55.     end if
56.     if $\{Order\_To\_Be\_Sent \neq \emptyset$ and not $End\_of\_Plan\}$ then
57.         Send out Orders in $Order\_To\_Be\_Sent$
58.         $Order\_To\_Be\_Sent \leftarrow \emptyset$
59.     end if
60. end while
61. Broadcast “Failure” or if permitted, request a new simulated vehicle according to the vehicle restriction in the first successsive
    Assignment
```

Example 1. *Let us assume that in some scenario, Smith needs to carry out five Assignments: A_0, A_1, A_2, A_3 and A_4. Their Assignment-actions are γ_0 to γ_4 respectively. They are some general Actions and also contain relevant information, e.g., Triggers.*

The precedence constraints are:

A_0 *IntervalBefore* A_1;

A_0 *IntervalBefore* A_4;

A_1 *IntervalBefore* A_2;

A_2 *IntervalBefore* A_3;

A_2 *IntervalFinishes* A_4;

The metric constraints estimated can be:

$$117.6 \leqslant the\ start\ time\ of A_1 - the\ start\ time\ of \alpha \leqslant 255.0 \tag{5.2}$$
$$614.5 \leqslant the\ finish\ time\ of \beta_2 - the\ start\ time\ of A_1 \leqslant 1527.8 \tag{5.3}$$
$$414.37 \leqslant the\ start\ time\ of A_2 - the\ start\ time\ of A_1 \leqslant 1051.12 \tag{5.4}$$
$$105.78 \leqslant the\ start\ time\ of A_3 - the\ start\ time\ of A_2 \leqslant 1015.13 \tag{5.5}$$

which are:

$$117.6 \leqslant s_{\gamma_1} - s_\alpha \leqslant 255.0 \tag{5.6}$$
$$614.5 \leqslant f_{\beta_2} - s_{\gamma_1} \leqslant 1527.8 \tag{5.7}$$
$$414.37 \leqslant s_{\gamma_2} - s_{\gamma_1} \leqslant 1051.12 \tag{5.8}$$
$$105.78 \leqslant s_{\gamma_3} - s_{\gamma_2} \leqslant 1015.13 \tag{5.9}$$

Where:

s_α *represents the start of planning;*

s_{γ_1}, s_{γ_2} *and* s_{γ_3} *represent the start time of Assignment-action* γ_1, γ_2 *and* γ_3 *respectively;*

f_{β_2} *represents the finish time of pre-defined Action* β_2*, which is "Perform-assignment".*

As Smith starts planning from the very beginning, s_{start} has the same meaning as s_α. When building the Plan, Smith ignores α, β_0 and β_3. Moreover, the duration of the Assignment-action γ_1 in A_1 should be 70 seconds. The duration of γ_2 in A_2 should be 65 seconds. Those durations can be different from the ones specified in the Assignments to make sure that Smith has enough time to dictate the Assignments and will not trigger Failure accidentally.

As a result, by using the constraints, both delays and durations, pre-defined and user-specified, the General Plan Gr_α can be generated as illustrated in Figure 5.6(a). Smith will evaluate this graph by generating a bidirectional graph whose vertexes are the ones in 5.6(a), while edge weights are:

1) *specified metric constrains, e.g., weight of edge [$s_{\gamma3}$, $s_{\gamma2}$] is 1015.13, while weight of edge [s_{γ_2}, $s_{\gamma3}$] is -105.78 according to the constraint of* $105.78 \leqslant s_{\gamma3} - s_{\gamma2} \leqslant 1015.13$;

2) *INFINITE if the weight of edges are not specified.*

The Floyd Warshall algorithm, which is used to find the minimum distance between any two vertexes in the resulted bidirectional graph, will then be used to generate a set of refined metric constraints. Details of applying Floyd Warshall algorithm in temporal constraints graph can be found in Dechter et al. (1991).

The evaluation result is shown in Figure 5.6(b) with the refined metric constraints that are of interest to Smith. When performing Assignments, Smith will ignore unspecified metric constraints with value INFINITE, which are not illustrated in Figure 5.6.

From the results obtained, for instance, the start time of A_1 should be less than 255 seconds after the start of α. Hence, when carrying out A_1, the proposed release time of the Assignment-action γ_1 should be less than 255 seconds and the proposed deadline of γ_1 should be less than 325 seconds.

By applying the Floyd Warshall algorithm and related temporal information, Smith is able to know what to do and if the plan has been carried out in time. The refined metric constraints shown in Figure 5.6(b) should be consistent throughout the scenario.

Plan Evaluation can be invoked for three reasons:

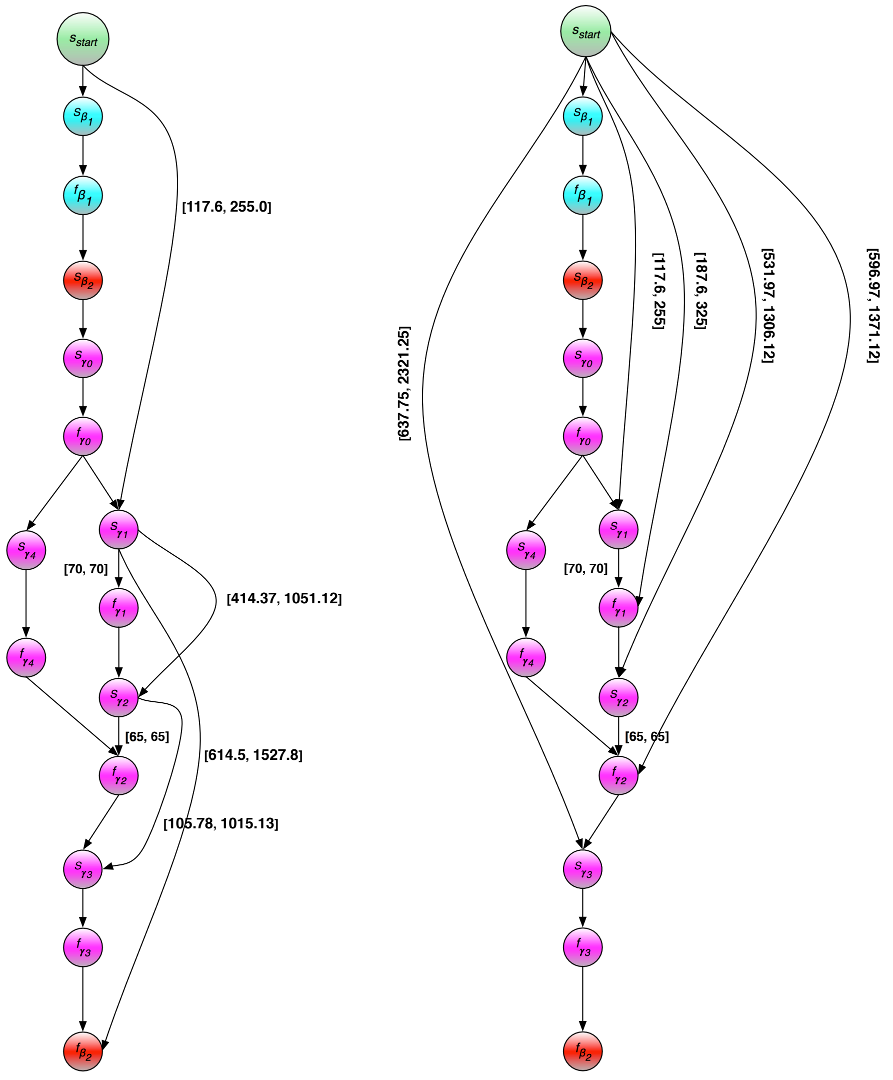


Figure 5.6: The General Plan for Example 1

1) Smith is initializing his World Model. In this case, the Plan Evaluation will generate the refined metric constraints that will be used throughout the scenario to indicate the proposed release time or deadline range of Assignment-actions;

2) Smith needs to add lane-changing Actions into the General Plan according to the Regulating layer. In this case, as the Regulating layer will not add

new constraints except for the lane-changing duration, which has been set to five seconds, the Plan Evaluation is always successful and the refined metric constraints should not be changed;

3) A new Role Matching has performed. In this case, an extra Assignment can be added to assist this new Ego-vehicle, e.g., adopt new speed, however, in this research, Plan Evaluation after Role Matching is always successful as the refined metric constraints should not be changed and no extra Actions will be added. Smith will stick to the pre-defined plan for Assignments.

In general, what Plan Evaluation should do is to evaluate all the constraints to make them consistent with each other, so the refined metric constants can be different from their original counterparts, i.e., initial metric constraints from SDF (Scenario Definition File), but of course, they should have intersections.

5.3.2.2 Role Matching

The Role Matching procedure takes the Assignment-action that is being checked as input and generates an Ego-vehicle/flock ID as output, indicating which Ego-vehicle/flock Smith should control.

In general, when Smith is trying to perform an Assignment, Role Matching will be invoked throughout the Assignment until an Ego-vehicle/flock has been assigned. Meanwhile, it can be invoked just once if changing Ego-vehicle/flock is forbidden. Generally speaking, Role Matching has three steps: the Matching of Formation Position, the Matching of vehicle model, and the Matching of Gr_α. Algorithm 2 shows the mechanism of Role Matching in NAUSEA. Step one and two, which are Matching of Formation Position and Matching of vehicle model respectively, are covered in Lines 10 to 25; step three, which is Matching of the General Plan, is covered in Lines 26 to 37.

When Smith needs to find a simulated vehicle to perform an Assignment, he will attempt to find one near the proposed Formation Position, which is reflected by the Neighbourhood information in Memory. For instance, if an Assignment needs a leader of participant's vehicle, the "Formation Position" of that Assignment will be specified as "Leader" in SDF (according to Figure 5.7). Smith will then try to find a simulated vehicle that is in position "L" in the Neighbourhood.

Algorithm 2 Mechanism of Role Matching in NAUSEA (*function* $Role\_Matching()$)

Require: the pre-defined Formation Position array FP, the Assignment that is being monitored CA in Memory, i.e., the first successive Assignment(s) and the refined metric constraints

Output: $Found\_Actor$ that represents the ID of Ego-vehicle/flock found.

```
1.  according to Figure 5.7, $FP = \begin{matrix} FLF & PL1 & LLF0 & LLF1 & 0 \\ FL & PL0 & LL0 & LL1 & 0 \\ F & P & L & LC0 & LC1 \\ FR & PR1 & LR0 & LR1 & 0 \\ FRF & PR1 & LRF0 & LRF1 & 0 \end{matrix}$
2.  $Found\_Actor \leftarrow \emptyset$
3.  $Actor\_Can\_Be\_Found \leftarrow$ **true**
4.  $[m, n] \leftarrow$ array position of required Formation Position from $CA$
    {e.g., if "$L$" is required, then [m, n] will be assigned with [2, 2] according to $FP$}
5.  $Found\_Final\_Position \leftarrow$ **false**
    {If Role Matching has searched all possible positions, i.e., the required position, the left-side and right-side positions.}
6.  **if** the simulated vehicle in the proposed Formation Position is not the one found in Role Matching **then**
7.      $Found\_Actor = \emptyset$
8.  **end if**
9.  **while** $Found\_Actor = \emptyset$ **do**
10.     **if** vehicle at $[m, n]$ has the required vehicle model **and** it has not been searched before **then**
11.         $Found\_Actor \leftarrow$ vehicle ID at position $[m, n]$ according to the Neighbourhood in Memory
12.     **end if**
13.     **if** $m - 1 \geqslant 0$ **and** $Found\_Actor = \emptyset$ **then**
14.         **if** vehicle at $[m-1, n]$ has the required vehicle model **and** it has not been searched before **then**
15.             $Found\_Actor \leftarrow$ vehicle ID at position $[m-1, n]$ according to the the Neighbourhood in Memory
16.             **if** $m = 4$ **then**
17.                 $Found\_Final\_Position \leftarrow$ **true**
18.             **end if**
19.         **end if**
            {if failed in finding vehicles on the left, try vehicles on the right.}
20.     **else if** $m + 1 \leqslant 4$ **and** $Found\_Actor = \emptyset$ **then**
21.         **if** vehicle at $[m+1, n]$ has the required vehicle model **and** it has not been searched before **then**
22.             $Found\_Actor \leftarrow$ vehicle ID at position $[m+1, n]$ according to the Neighbourhood in Memory
23.             $Found\_Final\_Position \leftarrow$ **true**
24.         **end if**
25.     **end if**
26.     **if** {$Found\_Actor \neq \emptyset$ **and** a position is presented in Monitors} **then**
27.         $t_p \leftarrow$ the time needed to reach the required position according to current speed
28.         **if** $t_p >$ the refined temporal constraints according to $Gr_\alpha$ **then**
29.             $t_p \leftarrow$ the time needed to reach the required position according to speed limit
30.             **if** $t_p >$ the refined temporal constraints according to $Gr_\alpha$ **then**
31.                 mark $Found\_Actor$ as searched vehicle in the Neighbourhood in Memory
32.                 $Found\_Actor \leftarrow \emptyset$
33.             **end if**
34.         **else**
35.             **return** $Found\_Actor$
36.         **end if**
37.     **end if**
38.     **if** $Found\_Actor = \emptyset$ **and** $Found\_Final\_Position$ **then**
39.         **return** 0
40.     **end if**
41. **end while**
```

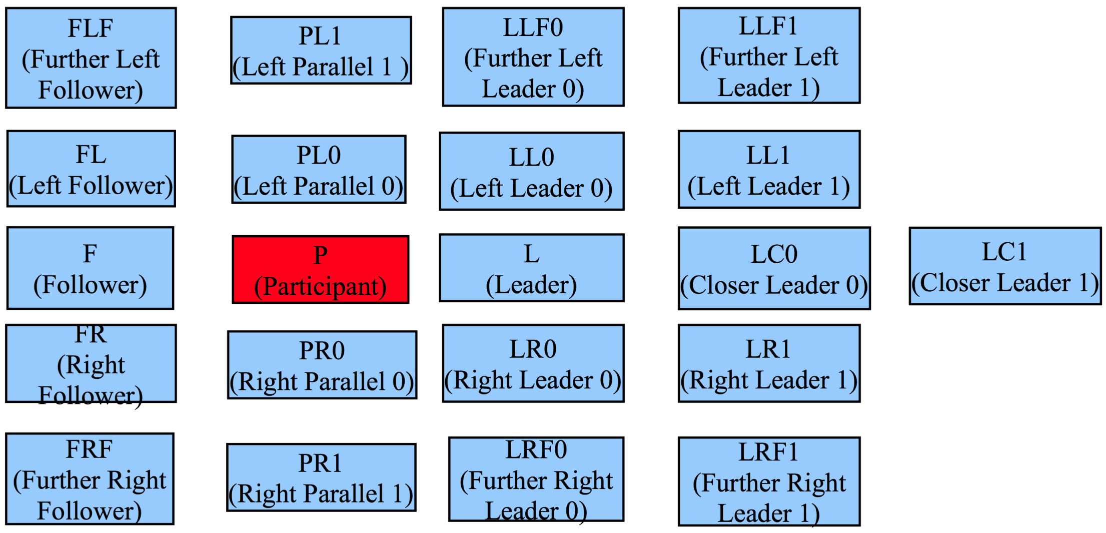


Figure 5.7: Formation Position

If a simulated vehicle can be found in the first step, Smith will then match the vehicle model with specified parameters regarding its model, its manufacturer, its max speed etc. As the max speed is always set to the speed limit, vehicle model along with manufacturer is used to be the vehicle restriction, e.g., Volvo S40.

If the vehicle model has been satisfied, Smith will proceed to Matching of Gr_{α}.

Let us use the Example 1 on Page 111 to describe how the temporal constraints can be used for Role Matching. As illustrated in Figure 5.6(a) on Page 113, the temporal constraints are fixed. For instance, the Assignment γ_1 should occur within 255 seconds of simulation starts. What Smith needs to do, therefore, is to make sure that he can drive the Ego-vehicle/flock to the pre-defined Formation Position before the constraint expires. This is done by:

1) checking the Monitor in the oncoming Assignment. If the Monitor specifies that Smith needs to dictate a position, e.g., a specific distance value in a road segment, Smith will consider the Instant when the participant's vehicle reaches that position as the finish time of the Action "Generate-formation", which is β_1 in Figure 5.6(a);

2) checking the speed limit of the road segment. This is to see if he can reach the position in time by considering the maximum speed as a normal driver[1]. This is evaluated by calculating the time needed to reach the position:

$$t_p = \frac{\Delta X}{v} \tag{5.10}$$

Where,

- v is the current speed of the Ego-vehicle/flock;
- ΔX is the distance between the threshold position and the current position of Ego-vehicle/flock;

By comparing t_p with the temporal constraints, e.g., number 255 in Example 1, Smith will know if this Ego-vehicle/flock can reach the proposed Formation Position in time. Hence, if $t_p > 255 - t$ (t is the current time), then Smith needs to consider a higher speed by calculating t_p as:

$$t_p = \frac{\Delta X}{v_{limit}} \tag{5.11}$$

Where,

- v_{limit} is the speed limit of current road segment.

If t_p can satisfy the constraint, he will then choose the Ego-vehicle/flock that has just been evaluated.

If a failure is reported in any step, Smith will first try to spot any simulated vehicle near the proposed position based on the Neighbourhood information, such as "NSL" until he finds an alternative, after which the other two steps will be invoked again: matching of vehicle model and matching of Gr_α. If Smith fails in finding an Ego-vehicle/flock, he will broadcast "Failure" or request to create a new simulated vehicle or Flock according to the vehicle restrictions. Moreover, if the simulated vehicle in the proposed position changed during Regulating procedure, Smith will do another Role Matching in order to make sure that the present Ego-vehicle/flock is the most suitable one.

[1]In this research, personalities of drivers are not modelled, so it is assumed that the speed limit is obeyed at all circumstances.

5.3.2.3 Targeting

Targeting is straightforward by adopting a pre-defined route which usually implies a sequential set of road segments. Therefore, Smith will not interfere with the Targeting module in simulated vehicles, i.e., the “Route” information in the Memory reflects the one stored in Ego-vehicle/flock in the Sim. This layer has been preserved in SAIL/NAUSEA for future development.

5.3.2.4 Regulating

This procedure controls the Ego-vehicle/flock identified in Role Matching. By utilizing the real-time traffic conditions and the present Assignment-action in Memory, it can output a speed, a lane or an overtaking indention that the Ego-vehicle/Flock should adopt. The speed, lane or desired overtaking intention will be added into the General Plan as the Recipe for β_1.

In general, three behaviours are adapted from the Sim: lane changing, speed adaptation and overtaking. They are adapted because either 1) some modifications are required according to the needs of the Assignment execution (lane changing and speed adaptation) or 2) the status of some behaviours should be monitored, especially overtaking. That is, the Regulating layer in SAIL/NAUSEA will incorporate with the existing behaviours in the simulated vehicles in the Sim.

Lane changing is used to get to the lane that the Formation Position requires, while overtaking is used to overtake slower vehicles. Speed adaptation is mainly used to make Smith obey the speed limit of the road segment by considering its desired speed and maintain a realistic speed trajectory when performing a turning movement.

In general, lane changing has been enhanced based on the needs from Assignment. It will be invoked to satisfy the Formation Position and act as the Recipe for β_1.

As shown in Figure 5.8, a simulated vehicle that is in the “Leader” position is needed, however, there is only one qualified simulated vehicle in the position of “Left Leader 0” (which is vehicle no.1 in Figure 5.8). Smith will therefore treat

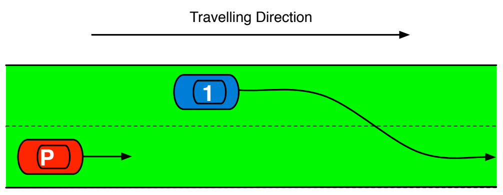


Figure 5.8: Lane Changing Example

the vehicle no.1 as the Ego-vehicle and instruct the Ego-vehicle to change lane in order to reach the "Leader" position.

Overtaking has been enhanced with flags indicating its running status: When did it start? Is it being undertaken? Has it finished?

Speed adaptation has not been changed, but when being driven for some particular Assignment, the Ego-vehicle can be instructed to adopt a slow speed or higher speed to meet with the participant's vehicle. Moreover, speed adaptation has been enhanced with the longitudinal transportation strategy proposed by Olstam *et al.* (2011), which is included in the third experiment with VTI's simulation software in Chapter 11.3.3. More details regarding the algorithm will be given in Chapter 11.3.3 as it was not developed within this research.

The Regulating procedure is only active in phase β_1, which is to arrive at the Formation Position required by an Assignment. In phase β_2, which is to perform Assignment-actions, lane changing and overtaking are both forbidden while speed-adaptation is allowed if there is no speed requirement in Assignment-actions.

The finish time of Regulating is the start time of the coming Assignment(s) as the Monitors will be dictated during the Regulating phase. However, the satisfaction of a particular Formation Position according to specific definitions from, e.g., scenario designers, can also be used to indicate the finish time of Regulating. For instance, the finish time can be the Instant when the time headway between the participant's vehicle and the Ego-vehicle is less than six seconds. In the latter case, the Regulating layer is actually forbidden in advance.

5.3.2.5 Assignment Assessment

After the Ego-vehicle/flock has been identified and driven to the proposed formation by Smith, Assignment Assessment will be invoked. It takes the Triggers of checking Assignment-action as inputs and checks the status of corresponding Assignment-action: finished, failed or succeed. It has four tasks:

- trigger any Assignment-action according to its Monitor or precedence order, record the time as the online release time r_{β}^{online};
- execute any Assignment-action based on its r_{β}^{online};
- dictate the result of Assignment-action based on Success and Failure Conditions during its Duration, record the adjusted deadline d_{β}^{adj};
- check if the Assignment-action has been sent successfully, if not, mark it as "unfinished" and report "failure".

Algorithm 3 shows the mechanism of Assignment Assessment procedure in NAUSEA. The Assignment Checker procedure is covered in Lines 2 to 8. The Action Execution procedure is covered in Lines 9 to 19. The Action Checker procedure is covered in Lines 20 to 53.

The Assignment-checker procedure will dictate the Monitors stored in the Assignments, which are the oncoming Assignments in Gr_{α}. As shown in Example 1 on Page 111, γ_0, which is associated with s_{γ_0} and f_{γ_0}, should be executed right after the Formation Position has been reached. However, due to the existence of Monitor(s) in its parent Assignment, γ_0 will be dictated until the Monitor(s) are satisfied, in which case, the Assignment-actions will be released and time will be recorded as the online release time r_{β}^{online}.

r_{β}^{online} will then be handled by a procedure named "Action-execution", which will check if r_{β}^{online} is consistent with Gr_{α}. If the answer is yes, this Action will be executed by sending out relevant Smith Orders stored in the definition of Assignment-action, e.g., decelerate with an acceleration rate of $1m/s^2$. This Action will then be passed to the following procedure, which is "Action-checker".

"Action-checker" checks if an Assignment-action has succeeded or failed according to

Algorithm 3 Mechanism of Assignment Assessment in NAUSEA (*function* $Assignment\_Assessment\_Procedure()$)

Require: Gr_α and the Assignment that is being monitored CA in Memory, i.e., the first successive Assignment(s)

Output: $Order\_To\_Be\_Sent$ that represents the Actions that need to be executed and $Assignment\_Failure$ that presents if CA has failed to be finished.

```
1.  $Action\_Execution\_Queue, Action\_Monitor\_Queue, Order\_To\_Be\_Sent \leftarrow \emptyset$
2.  $CA\_is\_Triggered \leftarrow$ check if $CA$ has been triggered based on its Monitors.
3.  if $CA\_is\_Triggered$ and tried times of $CA$ has not exceeded the threshold then
4.      $r_\beta^{online} \leftarrow$ online release time of $CA$
5.      $Action\_Execution\_Queue \leftarrow CA$
6.      increment of tried times of $CA$
7.      return [$Action\_Execution\_Queue$, false]
8.  end if
9.  if $Action\_Execution\_Queue \neq \emptyset$ then
10.     for each Assignment in $Action\_Execution\_Queue$ do
11.         if $r_\beta^{online} >$ corresponding refined metric constraints in Memory then
12.             return [$\emptyset$, true]
13.         else
14.             $Action\_Monitor\_Queue \leftarrow CA$
15.             erase this Assignment from $Action\_Execution\_Queue$
16.             Continue
17.         end if
18.     end for
19. end if
20. if $Action\_Monitor\_Queue \neq \emptyset$ then
21.     for each Assignment in $Action\_Monitor\_Queue$ do
22.         if present time - $r_\beta^{online} >$ required Duration then
23.             $d_\beta^{adj} \leftarrow$ present time
24.             mark this Assignment as $Finished$ in Memory
25.             if $d_\beta^{adj} >$ corresponding refined metric constraints in Memory then
26.                 return [$\emptyset$, true]
27.             else
28.                 erase this Assignment from $Action\_Monitor\_Queue$
29.                 mark this Assignment as $Successful$ in Memory
30.             end if
31.         end if
32.         if present time - $r_\beta^{online} <$ required Duration then
33.             if Failure Conditions are satisfied then
34.                 empty $Action\_Monitor\_Queue$
35.                 return [$\emptyset$, true]
36.             end if
37.             if Success Conditions are satisfied then
38.                 $d_\beta^{adj} \leftarrow$ present time
39.                 mark this Assignment as $Finished$ in Memory
40.                 if $d_\beta^{adj} >$ corresponding refined metric constraints in Memory then
41.                     return [$\emptyset$, true]
42.                 else
43.                     erase this Assignment from $Action\_Monitor\_Queue$
44.                     mark this Assignment as $Successful$ in Memory
45.                 end if
46.             end if
47.         end if
48.     end for
49.     if $Action\_Monitor\_Queue = \emptyset$ then
50.         $CA \leftarrow$ the successive Assignment(s) from $Gr_\alpha$
51.         return [$\emptyset$, false]
52.     end if
53. end if
```

- pre-defined Success/Failure Conditions and,
- Durations of the Assignment-action.

Pre-defined Success/Failure Conditions are specified by OSO users. They will be checked during the entire Duration of the Assignment-action; if Success Conditions are not satisfied, this Assignment-action and corresponding Assignments will fail.

The Durations of the Assignment-actions are checked when there are no Success Conditions available. Hence, if the Failure Conditions are not satisfied and the Durations of Assignment-actions have expired, the Assignment-actions and corresponding Assignments will be marked as finished; satisfaction of Failure Conditions will cause the failure of the Assignment-action and corresponding Assignments. It should be noted that all the Triggers, i.e., Monitors, Success Conditions and Failure Conditions, can have more than one instance in order to supervise more than one state variables in the Sim.

An Assignment-action that has been marked as finished will be provided with an adjusted deadline d_{β}^{adj}, which is the Instant when the Assignment-action finishes. d_{β}^{adj} will be checked with the metric constraints in Gr_{α} to make sure that it is consistent with the General Plan. If it is consistent, then it its parent Assignment will be marked as successful.

If any failure is found during Assignment Assessment, e.g., d_{β}^{adj} is not consistent with the metric constraints, the Replanning procedure will be invoked.

In addition, Action-check also needs to check if an order has been sent successfully by making sure that the order has been cleared after the execution. If the order has not been cleared, Action-check will return failure.

5.3.2.6 Replanning

Failures in either step of the algorithm will force Smith to find another Ego-vehicle/flock, provided that changing the Ego-vehicle/flock is allowed.

The potential cause of failures can be:

- Plan Evaluation failure: Smith cannot find a feasible plan according to the metric constraints;

- Role Matching failure: Smith cannot find a suitable Ego-vehicle/flock in the Sim to finish the Assignment and he is not allowed or it is impossible to create a new vehicle, e.g., the road segment is not long enough for the Assignment;

- Assignment Execution failure: an Assignment has failed because of the satisfaction of Failure Conditions or the release time/deadline are not consistent with the refined metric constraints.

What Smith will do in most cases is to keep the failed Assignment in Gr_α and perform another Role Matching if it is allowed. Hence, the General Plan will not change throughout the scenario unless a lane-changing behaviour is needed for Assignment compensation, in which case the failed Assignment will be replaced by this lane-changing Action that is the Recipe for β_1. However, the remaining refined metric constraints excluding the failed Assignment should be the same as before.

Requesting a new vehicle to be created has been covered by NAUSEA, however, this feature has not yet been fully considered, as speed adaptation in the Regulating layer has not been fully examined in this research. Creation of new simulated vehicles may result in long-distance speed adaptations, as they are created out of the participant's sight. Therefore, Assignments are always used to indicate when the right time and place to create new simulated vehicles should be.

To sum up, Replanning can be carried out with the help of Role Matching. However, Replanning can be forbidden by 1) specifying that Smith is not allowed to perform Role Matching more than once; 2) specifying that the maximum tried time of the Assignment is one time or 3) both.

5.4 Summary

By utilizing relevant context in Assignments, NAUSEA addresses the run-time issues in driving simulation in two directions: 1) the simulated vehicles can actively engage the participant to avoid failure by navigating Ego-vehicle/flock with required context for interactions based on Regulating procedure and 2) if failures happen, the simulated vehicles can be re-driven to generate the proposed interactions based on replanning capability. The main procedures in NAUSEA are as follows:

Plan evaluation is used to generate a refined metric constraints based on user's definitions in order to indicate the permitted release time and deadline ranges of some Assignments.

Role Matching is used to find suitable Ego-vehicle/flock in this research by matching not only required Formation Position and vehicle models, but also the potential travelling time spent by the Ego-vehicle/flock from the present position to the required Formation Position.

Targeting is straightforward by adopting pre-defined route, while Regulating is used to monitor the overtaking process and modify the lane-changing & speed adaptation behaviours in order to prepare the Ego-vehicle/flock for a particular Assignment.

Assignment Assessment is used to trigger/fail/finish Assignments by considering the precedence constraints between Assignments according to the General Plan or Triggers used to indicate Monitors, Failure Conditions and Success Conditions.

By carrying out the procedures above according to the General Plan and dynamic driving context, NAUSEA is able to replan Actions by going back to Role Matching if necessary.

SAIL was proposed to support NAUSEA by supplying relevant data regarding dynamic driving context and relevant interfaces for communications between NAUSEA and the Sim. Therefore, SAIL has been designed with existing driver models but enhanced to adopt NAUSEA. This also adopts a context-understanding and task-commitment driver model for driving simulation.

In Chapter 6, details regarding the implementation of SOAV will be provided. It will focus on two components: Smith and the Sim equipped with SMM. From Chapter 7 to 9, SOAV will be used to run three different experiments with a focus on verifying the design and implementation. It should noted that, as suggested in Chapter 3 and the algorithm described in this chapter, NAUSEA will concentrate on one Smith controlling one vehicle.

Chapter 6

Implementation

She told you exactly what you needed to hear. That's all. Sooner or later, Neo, you're going to realize just like I did the difference between knowing a path and walking a path.
-The Matrix

If it's not in the computer, it doesn't exist.
-One of "Murphy's Technology Laws"

6.1 Introduction

SOAV has four online components: Smith, SMM, the Sim and Scenario Observer. In this chapter, details regarding the implementation of Smith and the Sim equipped with SMM will be introduced.

The Sim and SMM have been developed with the simulation software of two driving simulator facilities: UoLDS (University of Leeds Driving Simulator) and VTI[1]'s Simulators, which are called Sim platform 1 and Sim platform 2 respectively.

Smith has been developed as standalone software using the C++ language. It is equipped with SAIL/NAUSEA.

Scenario Observer has been developed based on an Open Source library called

[1]The Swedish National Road and Transport Research Institute

LCM (Lightweight Communication & Marshalling)[1], which is also the library used for communications among the online components of SOAV.

In the following content, the implementation of Smith and two platforms, namely Sim platform 1 and Sim platform 2 will be discussed first. The three verification experiments that have been carried out in this research are also briefly introduced. SMM, as a component in each platform, will not be discussed because it is included in every platform as a module being used to receive, interpret and execute orders with relevant methods in each platform, e.g., set the target speed of some simulated vehicle. It does not contain any decision-making algorithm.

6.2 Smith

Smith is the implementation of the SAIL driver model. It has been implemented with C++ and the following Open Source libraries: Boost, Lightweight Communication & Marshalling (LCM) and OpenroadED[2]. There are two versions of Smith. In version one, the Cognition and Perception layers were implemented in different threads and thus ran in parallel, while in version two, the two layers were implemented in a single thread and thus ran in sequence.

In version one, Perception and Cognition layers were implemented in different threads and shared two sets of Memory data based on Mutex, which is a mutual exclusion mechanism used to ensure that the two layers are not possessing the shared data at the same time so that each layer can modify the shared data without conflicts. A detailed description of Mutex can be found in the manual of Boost Thread Library [3]. This feature was originally designed in the hope of making the two layers operate at different frequencies in order to adopt more advanced features in the future, e.g., different data packages can be received by the Perception layer with different arrival frequencies, whilst the Cognition layer should not be affected.

[1] https://code.google.com/p/lcm/

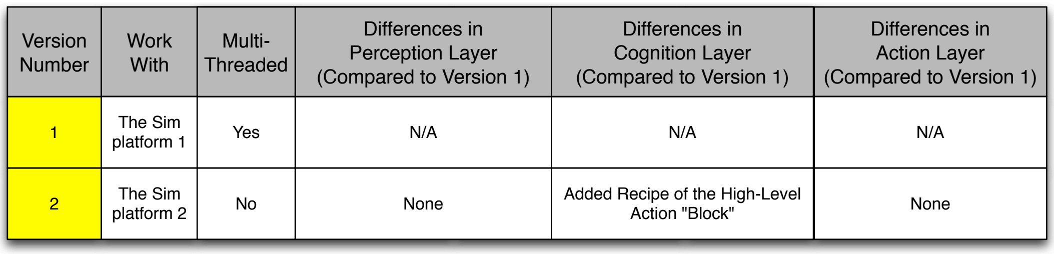
[2]Boost: http://http://www.boost.org;
OpenroadED: http://openroaded.sourceforge.net.

[3]http://www.boost.org/doc/libs/1_52_0/doc/html/thread.html

The two layers share two sets of data: simulated vehicles' information around the participant's vehicle (Neighbourhood) and participant's vehicle's information. When the Perception layer receives a new data package, it will 1) lock the two sets of Memory data by monopolizing Mutex, 2) interpret the data package and 3) write interpreted information to the two sets of Memory data. The Perception layer will finally unlock Mutex and allow the Cognition layer to lock it. When the Cognition layer is able to lock Mutex and occupy the Memory data, it will then make decisions and modify Smith's Memory accordingly. The Cognition layer will unlock the two sets of Memory data after the Action layer has executed orders by releasing the Mutex.

As multi-threaded architecture adopts completion between two layers, single-threaded Smith was tried in verification experiment three in order to see if the delay could be consistent with single threading and be reduced to one decision loop. In this version of Smith, which is version two, the Perception and Cognition layers were implemented in the same thread and the arrival of new data packages will invoke the Cognition layer. In this case, the Sim and Smith shares the same running frequency. As a result, the orders from Smith may experience less time lag and less variation in the time spent on decision-making than those sent from version one. Moreover, the High-Level Action "Block" was also implemented in version two.

The differences between the two versions of Smith are summarised in Table 6.1:



Version Number	Work With	Multi-Threaded	Differences in Perception Layer (Compared to Version 1)	Differences in Cognition Layer (Compared to Version 1)	Differences in Action Layer (Compared to Version 1)
1	The Sim platform 1	Yes	N/A	N/A	N/A
2	The Sim platform 2	No	None	Added Recipe of the High-Level Action "Block"	None

Table 6.1: Differences Between two Versions of Smith

6.3 Sim Platform 1

"Sim platform 1" or "Sim1" for short was based on the simulation software of UoLDS (University of Leeds Driving Simulator). Sim1 runs at a frequency of 60 Hz, corresponding to the frequency of the monitor used for visual display. In order to communicate with Smith, Sim1 has been modified to adopt two new components, which are SMM and the Information Broadcast Module. SMM was adopted in Sim1 to interpret orders from Smith. The Information Broadcast Module is used to 1) collect every vehicle's information, 2) package it and 3) send it out using the LCM library in each update step. The details of this process are as followed (Figure 6.1):

- Smith receives information of vehicles' states and makes decisions, after which corresponding orders will be sent out;
- SMM receives and interprets orders. Relevant Ego-vehicle/flock in Sim1 will then be found and their behaviours will be changed, e.g., target lane, desired speed, accelerate rate. Also, SMM can interpret traffic flow requests and invoke the relevant traffic flow controller in Sim1, which produces ambient traffic flow with pre-defined parameters, e.g., time headway between simulated vehicles in the traffic flow. SMM can be also instructed to carry out some pre-defined tasks such as putting cones in some lane;
- Vehicle State Update uses some tactical behaviours (see Table 6.2), e.g., overtaking and local parameters, e.g., desired speed, to update vehicles' states. During this phase, SMM may change some local parameters to 1) configure Ego-vehicle/flock; 2) turn on/off some pre-defined traffic flow or 3) perform some non-driving Actions such as putting cones. Sim1 will then update the states and visualize them on the display of a PC used for the participant's driving (i.e., PC No.1 in Figure 6.4);
- Information Broadcast packs vehicles' updated information and send them to the network.

In Sim1, the ID of some road segments can act as the threshold values for some Triggers. An ID, e.g., "r1.5", means a specific road segment in the scenario.

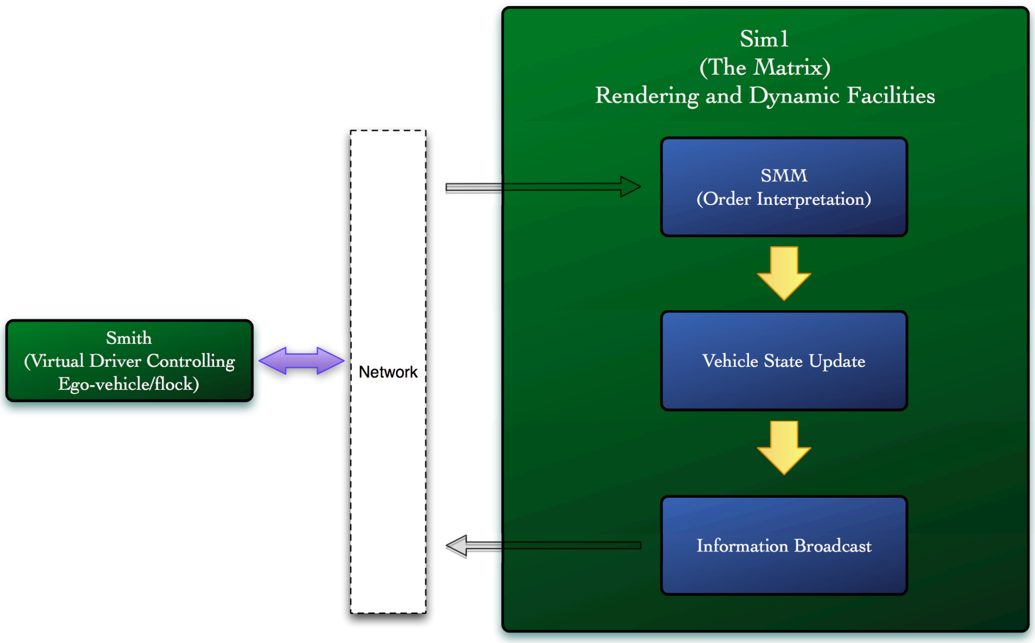


Figure 6.1: Sim Platform 1 (Sim1)

"r" is the keyword for road segment; the first number is the identifier of the road segment; the second number is the identifier of a section in a road segment. A road segment can have several sections because 1) the road curvature changes so different sections are needed or 2) some sections are needed to connect with junctions. When the vehicles are travelling in the right direction, the identifiers of both the road segments and sections will increase. A scenario can have more than one road segment and a road segment can have more than one section. Sections can have different lengths.

6.4 Sim Platform 2

The simulation software being used by VTI's driving simulators has a distributed architecture. The Sim Platform 2 or Sim2, in short, uses the kernel, visual system and the data as illustrated in Figure 6.2.

The Kernel runs at a frequency of 200 Hz and sends out information, e.g.,

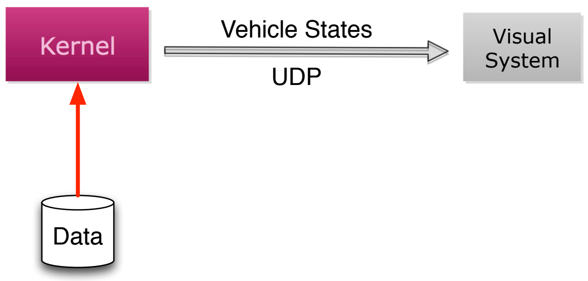


Figure 6.2: Architecture of VTI Simulation Software

vehicles' positions, every decision loop (1/200 seconds) to the visual system that displays the simulation separately. Data contains logical road representations based on the OpenDRIVE format (Dupuis, 2011).

Sim2 has the same updating procedure as Sim1: 1) SMM interprets orders; 2) every vehicle updates its parameters, e.g., desired speed or lane-changing intentions, based on some tactical behaviours or orders, e.g., adopt a specified desired speed and 3) vehicles' states on the road network are updated accordingly.

Sim1 and Sim2 have identical interfaces for simulated vehicle control. A comparison between Sim1 and Sim2 regarding the simulated vehicles is summarised in Table 6.2. It should be noted that the Operational behaviours will not be included, as Smith is not designed to interfere with behaviours used to fulfil lane or speed decisions from Tactical behaviours.

Table 6.2: Comparison of Simulated Vehicles in two Platforms

Platform	Strategical Behaviours	Tactical Behaivours	Used Interfaces for Autonomous Vehicle Control
The Sim platform 1 (Sim1)	No Path Planning	Speed Adaptation Overtaing Lane Changing Gap Acceptance Obstacle Avoidance	Set Desired Speed Set Target Speed Set Action Acceleration Rate Set Target Lane Forbid Lane-changing Set Overtaking Intention
The Sim platform 2 (Sim2)	No Path Planning	Speed Adaptation Overtaing Lane Changing Gap Acceptance Obstacle Avoidance	Set Desired Speed Set Action Speed Set Action Acceleration Rate Set Target Lane Forbid Lane-changing

6.5 Framework Verification

In general, SAIL/NAUSEA should be able to help Smith plan his Actions based on Assignments so that Smith can 1) find Ego-vehicle/flock, 2) “drive” the Ego-vehicle/flock to the proposed Formation Position and 3) execute Assignment-actions as required. He should also be able to replan if necessary. Moreover, OSO should be evaluated to test its expressiveness and demonstrate its usage in describing relevant context for interactions along with potential reactions from participants. Finally, SOAV, as a framework, should also be evaluated to see if its implementation requires improvement for future application. The verification process has followed a four-step procedure as illustrated in Figure 6.3:

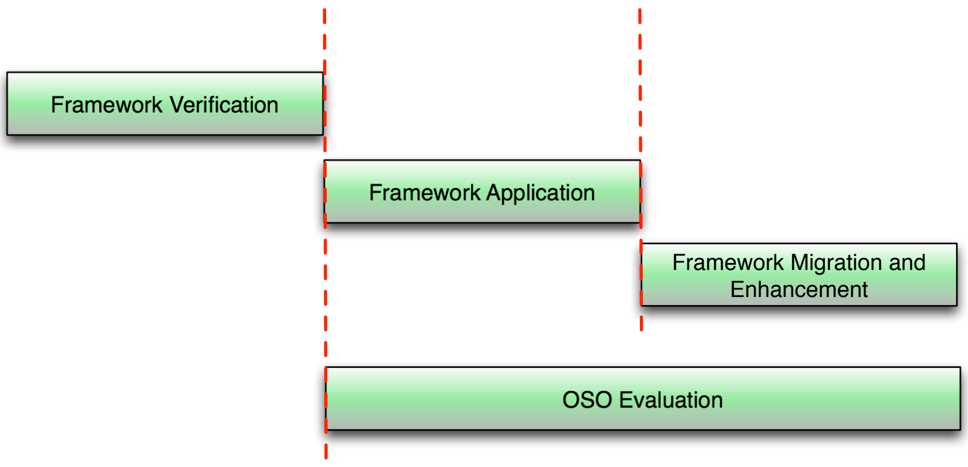


Figure 6.3: Four-step Verification Procedure

In Framework Verification, a verification experiment was carried out to see if SAIL/NAUSEA could be used to orchestrate a scenario by committing to Assignments with the capability of replanning. It was also used to test the implementation of SOAV. Five human participants were involved in experiment one. Smith version one and Sim1 were used in this experiment. The scenario used in this experiment was based on some interactions designed in an ongoing PhD research in Leeds (Horrobin & Carsten, 2011). The details of this verification experiment are introduced in Chapter 7 on Page 135.

In Framework Application, another verification experiment was carried out and a scenario from a previous project was used to observe the functionality of SOAV in "real" applications. Ten human participants were involved. Smith version one and Sim1 were also used in this experiment. The details of this step is introduced in Chapter 8 on Page 151.

In Framework Migration and Enhancement, based on the findings from experiment two, verification experiment three was used to enhance the speed adaptation behaviour in NAUSEA based on the findings from Olstam *et al.* (2011) and demonstrate that SOAV can be adopted into other simulation software. The third experiment also included a demonstration of the recipe of High-Level Action "Block", which involves three simulated vehicles preventing the participant from overtaking. Smith version two and Sim2 were used in this experiment. The scenario used in this experiment was based on a scenario in Olstam *et al.* (2011).

OSO Evaluation started after the Framework Verification. This step was undertaken to see if OSO could be used to describe scenarios. Moreover, the criteria from Krummenacher & Strang (2007) regarding context modelling and ontology engineering were also used to identify OSO's advantages and disadvantages in order to guide future research. The evaluation was also used to demonstrate the information that can be expressed using OSO, i.e., its role in scenario orchestration. Details of the evaluation will be included in Chapter 10 on Page 181.

In experiment one and two, SOAV was run with the same set-up shown in Figure 6.4. The human participant used one of the machines to experience the scenario; the experimenter used another to run Smith and collect data. Hence, they were based on the desktop version of UoLDS. In experiment three, all components in SOAV were run on a single machine because no human participants were involved. The machine, therefore, acted as a desktop version of VTI's simulators.

6.6 Summary

Smith was developed from scratch to adopt SAIL/NAUSEA and the Sim was modified in order to adopt SMM, which contains interfaces for executing Actions sent from Smith. Scenario Observer was directly derived from the LCM library.

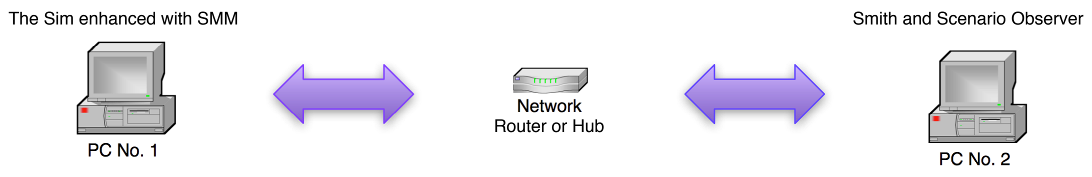


Figure 6.4: Experiment Set-up

There are several concerns regarding both the system design and the implementation: 1) can SAIL/NAUSEA provide desired output based on the information from Assignments in SDF? 2) as a computing environment that is used to orchestrate scenarios, can SOAV produce the scenarios as desired and 3) how about the implementation and further enhancement? The three verification experiments and OSO evaluation that will be detailed in the next three chapters were, therefore, proposed to deal with those concerns and see if the three objectives of this research (Chapter 3) have been achieved.

Chapter 7

Experiment One - Driving with Smith and Results

Agent Smith: *As you can see, we've had our eye on you for some time now, Mr. Anderson.*
-The Matrix

7.1 Introduction

Verification experiment one was designed for Framework Verification, which was to see if SAIL/NAUSEA could be used to plan a Virtual Driver's Actions according to Assignments, with the capability of dynamic Role Matching and replanning. SOAV was also tested to ascertain if the system can be used to orchestrate scenarios and to see if the implementation needed improvement.

In order to test SAIL/NAUSEA, a scenario was designed based on some interactions generated in an ongoing PhD project in Leeds (Horrobin & Carsten, 2011) to cover all of its aspects, e.g., Assignment compensation based on lane-changing generated by the Regulating layer. A scenario containing a rural road was used because on rural roads, failed interactions that require lane-changing to compensate are easy to design, as the participant has only one lane to choose from.

In addition, although failures can be caused for a variety of reasons and reconstructing failures on purpose could be hard to achieve, the failures reconstructed in this verification experiment were designed to imitate some specific reasons of failures, e.g., the participant does not want to be engaged or the participant exhibits unpredicted reactions such as overtaking instead of decelerating. However, it should be noted that, they are all the same to Smith as he treats all as Failure Conditions and replans his Actions based on the same decision making algorithm NAUSEA.

7.2 Equipment

Sim1 and Smith version one were used in this experiment.

A laptop was used to run Smith. It was equipped with an Intel® T2130 CPU and 2GB of memory and ran Ubuntu Linux 11.10 32bit. The communication between Sim1 and Smith was based on a wired 10Mb hub. A video camera was also used to record the animation on the screen so that the driving activities could be reviewed. The experiment contained one scenario and two phases, which are described in the following sections.

7.3 Experiment

7.3.1 Scenario Description

The scenario contained a 13.7 mile long (22 km) stretch of a rural road with some curved road segments. There were three villages and five junctions along the road. The speed limit on the open road was 60 mph but in villages was 30 mph. In order to be concise, the Assignments will be used to represent the interactions generated by corresponding Assignments.

In this experiment, the participants were asked to drive in the scenario twice. In the first drive, the participants were asked to drive freely while in the second drive, the participants were encouraged to fail some of the Assignments in the scenario.

Because participants were able to sabotage Assignments and the implemented Virtual Driver Smith had the ability to generate extra Actions to compensate, participants could experience different number of Assignments, depending on whether or not the failed Assignment could cause the whole scenario to fail and whether it could be reattempted. Moreover, the metric constraints of the scenario were generated by manual estimation and values used can be found in Appendix C.1.5 on Page 254.

Assignments that a participant could experience are listed below and illustrated in Figure 7.1:

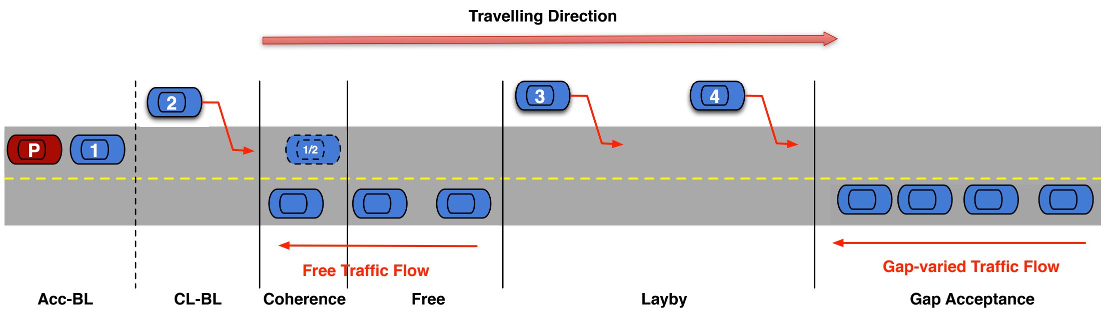


Figure 7.1: Illustration of the Scenario for Experiment One

The four candidate vehicles, whose IDs range from 1 to 4, have been illustrated in Figure 7.1. They were placed 402 m, 2016 m, 14116 m and 16399 m away from the start position of the participant's vehicle respectively.

Vehicle one and the participant were both placed on the left lane in the beginning, while the other three vehicles were placed in the leftmost lane, which was not a driving lane.

7.3.1.1 Assignment "Acc-BL"

"Acc-BL" is short for "Accelerate and Be Participant's Leader". This was the first Assignment that participants experienced. This Assignment required a Formation Position of "Leader", so the simulated vehicle with ID "1" should be chosen at the beginning as the Ego-vehicle.

When the participant's vehicle's time headway to the Ego-vehicle changed from greater than to less than 6 seconds, Smith should accelerate vehicle 1, which was the Ego-vehicle, and maintain a speed of 30 mph.

When the participant's travelling road was "r5.0" (3248m from the start position of the participant's vehicle) and the participant's leader before "r5.0" was always vehicle 1, this Assignment would be regarded as successful. If the participant's leader changed before "r5.0", the Assignment would fail and replanning should be invoked.

This Assignment was used to adopt a leader for the Assignment "Coherence" described later because the speed adaptation algorithm in Regulating layer was lacking.

7.3.1.2 Assignment "CL-BL"

"CL-BL" is short for "Change Lane and Be Participant's Leader". It was an Assignment/Action generated by the Regulating layer in order to navigate vehicle 2 to the position of "Leader" and compensate for the failure of Assignment "Acc-BL" if it failed, so that the participant could still have a leader after he/she had failed "Acc-BL". Vehicle 2 was the Ego-vehicle in this Assignment. The failure of "CL-BL" led to the failure of the whole scenario.

This Assignment used the Monitor and Success/Failure Conditions of the Assignment "Acc-BL". The Assignment-action was to force vehicle 2 to change to the participant's lane. This Action lasted for five seconds.

7.3.1.3 Assignment "Coherence"

After the adoption of a leader by performing "Acc-BL" or "CL-BL", the Assignment "Coherence" would start after a village was passed. This Assignment required a Formation Position of "Leader", so vehicle 1 should be the Ego-vehicle if "Acc-BL" succeeded. If "Acc-BL" failed, vehicle 2 should be the Ego-vehicle for this Assignment. This Assignment was derived from Brookhuis *et al.* (1994) and is a test of car following capability.

When the participant's travelling road was "r5.0" (right after a village), the Ego-vehicle, which was the leader should vary its speed sinusoidally for 60 seconds. Because there can be some delay in assigning and checking the release time/deadline of a particular Assignment-action due to the communication mechanism between Perception and Cognition layers, the Duration was set to 70 seconds in the

General Plan in order to make sure that Smith had enough time to dictate the Assignment and would not trigger failure accidentally.

When the Ego-vehicle's leader changed to the participant's vehicle during the Assignment, this Assignment would fail and replanning should be invoked.

The participant was told to match the leader's speed and maintain his/her favoured distance from the leader. Two sub-Assignments were generated: "Coherence1" and "Coherence2", which are short for "Coherence performed by vehicle 1" and "Coherence performed by vehicle 2" respectively. The failure of "Coherence" led to the failure of the whole scenario.

This Assignment was used to test the Assignment Assessment module in NAUSEA so Duration/Monitor/Success Conditions/Failure Conditions were all specified. It was also used to imitate the situation in which the participant did not want to be engaged. Duration of 60 seconds was used to make sure that the participant was able to overtake and the value of Duration was fixed beforehand by testing.

7.3.1.4 Assignment "Free Traffic Flow"

This Assignment was used to generate traffic flow in order to prevent the participant from overtaking. Because the oncoming traffic flow was of low density, the participant still had the chance to overtake. This Assignment started with "Coherence" and stopped with "Layby" that is defined below.

This Assignment was used to imitate the situation in which the experiment designer failed to foresee the "overtaking" reaction from the participant and thus, a traffic flow with low density was used. This traffic flow would make the overtaking feasible.

7.3.1.5 Assignment "Layby"

In this Assignment, Smith was required to find an Ego-vehicle and pull it into the participant's lane suddenly without indication when the participant was entering a village, in which case the participant might accidentally overtake the Ego-vehicle. This Assignment required a Formation Position of "LeftLeader0", so vehicle 3 should be chosen as the Ego-vehicle first and if this Assignment performed by vehicle 3 failed, vehicle 4 should be chosen to carry out this Assignment again.

When the participant's vehicle's time headway to the Ego-vehicle changed from greater than, to less than 3 seconds, Smith should force the Ego-vehicle to change into the participant's lane (which was the offside lane to the Ego-vehicle) with an acceleration rate of $5m/s^2$ and a target speed of 13.3 m/s. This Assignment lasted for 45 seconds, which meant that Smith stayed idle for 45 seconds. Smith used 65 seconds for plan evaluation.

When the Ego-vehicle's leader changed to the participant's vehicle, i.e., when the participant overtook, this Assignment would fail.

Two sub-Assignments were generated: Layby-V3 and Layby-V4, which are short for "Layby performed by vehicle 3" and "Layby performed by vehicle 4" respectively. The failure of Layby-V4 led to the failure of the whole scenario.

This Assignment was also used to test the Assignment Assessment module in NAUSEA so Duration/Monitor/Success Conditions/Failure Conditions were all specified. It was used to imitate the situation in which the participant brought in unpredicted reactions. The Action profile regarding the acceleration rate was set to imitate a situation in which an inappropriate value was used and Duration of 45 seconds was used to make sure that the participant was able to overtake. The Duration value was fixed beforehand by testing.

7.3.1.6 Assignment "Gap Acceptance"

After the Assignment "Layby", the participant arrived at a junction with oncoming simulated vehicles, the gap between which was increasing. Participants were instructed to turn right if the gap was considered safe. This Assignment was not supposed to fail so the participant was instructed to turn right.

In addition, the junction could be the one near road segment "r10.1" (17178 m from the start position of the participant's vehicle) or the one near "r12.1" (20300 m from the start position of the participant's vehicle) depending on whether the participant missed the one near "r10.1" after "Layby-V4". Hence, this Assignment could be triggered after "r10.1".

Because it is not currently possible to dynamically create simulated vehicles or road segments/junctions in Sim1, this Assignment was used to demonstrate the possibility of arranging Assignment locations dynamically. The dynamical environment creation feature was not fully examined.

7.3.1.7 Summary and Test Cases

In total, there were eight Assignments ("Coherence", "CL-BL", "Coherence1", "Coherence2", "Layby-V3", "Layby-V4", "Free Traffic Flow" and "Gap Acceptance"). However, as some of the Assignments were dynamically generated by Smith, there were actually five Assignments in OSO for the whole scenario and two of them were traffic flow-related. Smith controlled simulated vehicles in Sim1 according to the three Assignments ("Acc-BL", 'Coherence1" and "Layby") or requested traffic flows for the other two Assignments ("Free Traffic Flow" and "Gap Acceptance"). The Assignment "CL-BL" was created dynamically to compensate for the failure of Assignment "Acc-BL", so "CL-BL" and "Acc-BL" were actually related to the same Assignment, which mainly stated that 1) an Ego-vehicle that was the leader of the participant's vehicle was needed and 2) the initial target speed of the Ego-vehicle should be 30 *mph*. Hence, "CL-BL" was only created to make the Ego-vehicle (left to the participant's vehicle) become the leader in order to fulfil the requirement of "Leader" position. The three Assignments performed by Ego-vehicle were represented in OSO. The details of the representation can be found in Appendix C.1 on Page 239. The essential information in each Assignment have been represented by directed graphs.

This scenario had eight Assignments, but not all of them were exposed to the participant in each trial. Hence, a Test Case has been used to represent a specific set of Assignments that a participant could experience, i.e., a Test Case is a set of Assignments that a participant would experience or sabotage and the corresponding information Smith received, there were in total 9 Test Cases in this experiment. This number was generated by considering all the combinations of the seven Assignments that the participants could experience except the "Free Traffic Flow" and the fact that 1) "Free Traffic Flow" and "Gap Acceptance" did not need to be failed, 2) failures of "Coherence1"/"Coherence2"/"Layby-V4"/"CL-BL" led to the failure of the whole scenario and 3) some Test Cases were actually the same. For instance, in order to perform "CL-BL", "Acc-BL" should be failed first. Hence, the Test Case of "fail 'Acc-BL' and then fail 'CL-BL' " was actually the Test Case of "fail 'CL-BL' ". See Table 7.1 for all the 9 Test Cases and corresponding desired output.

Table 7.1: Test Case List and Desired Output of Each Test Case

Test Case		Total Trigger Number/ Order Number	Acc-BL			CL-BL			Coherence1			Coherence2			Layby-V3			Layby-V4			Gap Acceptance			Free Traffic Flow		
			M	S	F	M	S	F	M	S	F	M	S	F	M	S	F	M	S	F	M	S	F	M	S	F
1	Normal	6/5	√	√	×				√	D	×				√	D	×				√	N	N	√	N	N
2	Fail **Acc-BL**	8/6	√	×	√	√	√	×				√	D	×	√	D	×				√	N	N	√	N	N
3	Fail **CL-BL**	4/2	√	×	√	√	×	√																		
4	Fail **Coherence(1)**	5/3	√	√	×				√	×	√													√	N	N
5	Fail **Layby-V3**	8/6	√	√	×				√	D	×				√	×	√	√	D	×	√	N	N	√	N	N
6	Fail **Layby-V4**	8/5	√	√	×				√	D	×				√	×	√	√	×	√				√	N	N
7	Fail **Acc-BL** and **Coherence2**	7/4	√	×	√	√	√	×				√	×	√										√	N	N
8	Fail **Acc-BL** and **Layby-V4**	10/6	√	×	√	√	√	×				√	D	×	√	×	√	√	×	√				√	N	N
9	Fail **Acc-BL** and **Layby-V3**	10/7	√	×	√	√	√	×				√	D	×	√	×	√	√	D	×	√	N	N	√	N	N

M: Monitor

S: Success Condition

F: Failure Condition

√ **: Triggered**

× **: Not Triggered**

N **: Not Available**

D **: Duration-Based Success Condition**

As shown in Table 7.1, in the normal Test Case, "Acc-BL" should be triggered (M: ✓) and succeed (S: ✓). The Failure Conditions of "Acc-BL" should not be triggered (F: ✗). "Coherence1" should be triggered and succeed by considering its Duration (S: D). The Failure Conditions of "Coherence1" should not be triggered. "Layby-V3" should be triggered and succeed by considering its Duration. The Failure Conditions of "Layby-V3" should not be triggered. "Gap Acceptance" and "Free Traffic Flow" should both be triggered and since there were no Success or Failure Conditions (S: N; F: N), Smith would not check the status of the two Assignments after they were triggered. Moreover, in the Normal Test Case, "CL-BL", "Coherence2" and "Layby-V4" should not be triggered.

Moreover, details regarding the metric constraints can be found in Section C.1.5 on Page 254. The refined metric constraints generated by Plan Evaluation will be given when examining relevant procedures in NAUSEA.

7.3.2 Experimental Procedure

This experiment had two phases:

In Phase one, five participants were recruited from within the Institute for Transport Studies (ITS) to act as software testers. All of them were male and four of them had taken part in driving simulation experiments before. Every participant drove in the scenario two or three times in two stages.

- In stage one, the participant drove Test Case 1, which meant that the participant drove normally and experienced "Acc-BL", "Coherence1", "Layby-V3" and "Gap Acceptance" sequentially.
- In stage two, the participant drove the other two Test Cases randomly in two sections. Overtaking the leader was allowed according to the requirements of each Test Case. The participants would try Test Case 3, 4 and 7 in section one and 2,5,6,8 and 9 in section two, so two participants did not try any Test Cases in section one. The rule was that the Test Cases tried by the previous participant would not be considered for the following participants. The Test Cases each participant tried are given in Table 7.2. The input to Smith, which is the information of every vehicle in Sim1, was recorded in a Data-Log during each participant's drive.

Table 7.2: Test Case Tried by Each Participant in Phase One

Participant No.	Test Case in Stage One	Test Case in Stage Two	
		Section One	Section Two
1	1	7	9
2	1		8
3	1	4	2
4	1		6
5	1	3	5

In phase two, an automatic software test was performed by using the Data-Log recorded in Phase one. In this phase, every record of Test Case was played by a Log player from LCM library in order to re-construct every participant's drive. Every log was played 10 times, so there were in total 90 tests (9 $\times$ 10). The test of the Normal Test Case used the Data-Log of participant one in phase one.

The Instant when Smith made the decision of executing or finishing a particular Assignment-action were termed "(online) release time" and "(adjusted) deadline" respectively as specified in Section 5.3.2.

7.4 Results

7.4.1 Algorithm Examination

This section analyses each procedure in NAUSEA in order to examine if NAUSEA has provided the desired output and if each procedure satisfied the design goal. This examination can also suggest if SAIL has achieved its design goal by supporting NAUSEA as desired.

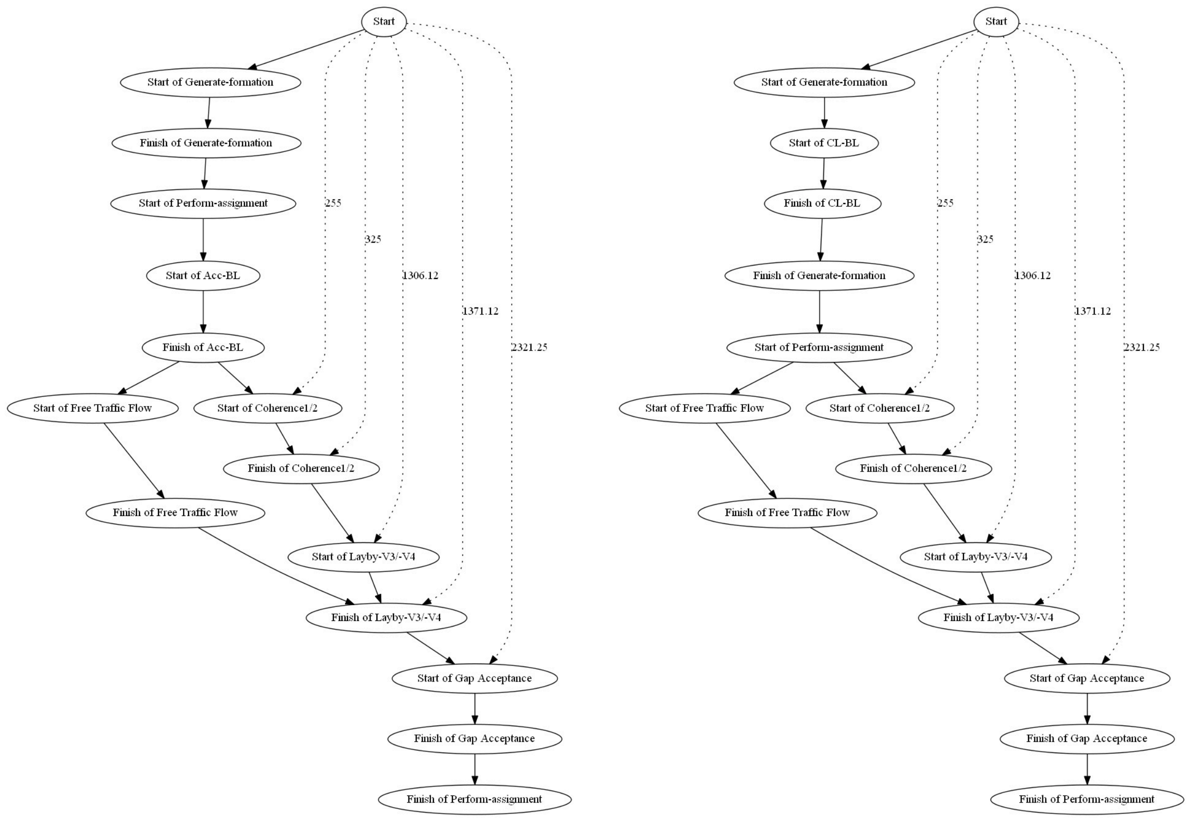
7.4.1.1 Plan Evaluation

Smith built the General Plan Gr_α and performed relevant Actions as desired. The output of the evaluation can be found in Figure 7.2. Figure 7.2(a) shows the output of the Normal Test Case and Figure 7.2(b) shows the output when "Acc-BL" failed (Test Case 2,3,7,8,9). Numbers in the graph are refined metric constraints, e.g., 255 means that the maximum release time of "Coherence1"/ "Coherence2" should be less than 255 seconds from the start of the simulation. The minimum release times of Assignments have been ignored by Smith, as Smith cares if the Assignment can be started or finished in time only.

7.4.1.2 Situation Assessment

All Assignments were executed as desired and all the release times complied with the metric constraints. Details regarding each sub-procedure are given below:

1) Assignment Checker: The release times and the the actual states of corresponding conditions in the Sim are given in Tables A.1 to A.5 on Pages 212 to



(a) (b)

Figure 7.2: Output of Plan Evaluation Procedure in Normal (7.2(a)) and Failure (7.2(b)) Test Case

214 in Appendix A. For instance, in Table A.1, "r5.0, 1.4460(m)" means that when Sim1 received the order of Assignment "Coherence1", the participant's vehicle was at 1.4460 metres from the beginning of the road segment "r5.0", which ideally should be zero metres from the beginning of the road segment "r5.0". "3.2137(s)" relates to the time headway from the participant's vehicle to the Ego-vehicle, which ideally should be three seconds if there was no delay in decision making, network transmission or order execution. All numbers have been rounded to less than 4 digits after the decimal point.

As shown in Tables A.1 to A.5 , the Assignment Checker was working properly and triggered Assignments according to corresponding Monitors. However, delays were found.

2) Action Executer: As shown in Tables A.1 to A.5, the release times of the three Assignments, namely "Coherence", "Layby" and "Gap Acceptance", satisfied the refined metric constraints, which were:

- "Coherence": the release time of "Coherence" $\leqslant 255$
- "Layby": the release time of "Layby" $\leqslant 1306.12$
- "Gap Acceptance": the release time of "Gap Acceptance" $\leqslant 2321.25$

3) Action Checker: Because all Test Cases were successfully accomplished, proposed Failure and Success Conditions were both evaluated correctly. In order to test if the Action Checker handled Durations and Success Conditions properly, the release times and deadlines of four Assignments, namely "Acc-BL", "CL-BL", "Coherence1/2" and "Layby-V3/4", have been summarised in Tables A.6 to A.10 on Pages 214 to 216 in Appendix A.

All Durations of Assignment-actions were evaluated properly and Success Conditions, e.g., the one for "Acc-BL", were also evaluated properly. The numbers within brackets are the differences between the release times and deadlines of the Assignments that had Duration requirements in their Assignment-actions.

Moreover, the deadline of each Assignment was also consistent with the metric constraints, which were:

- "Coherence": the deadline of "Coherence" $\leqslant 325$
- "Layby": the deadline of "Layby" $\leqslant 1371.12$

7.4.1.3 Role Matching

As all proposed Assignments were executed in every test, Smith was able to find the right simulated vehicle to carry out the Assignments.

7.4.1.4 Regulating

As Smith was able to find the appropriate simulated vehicle and guide it to the leader position if the Assignment "Acc-BL" failed, Regulating was considered successful. For instance, as listed in Tables A.1 to A.3, "Coherence" was successful in Test Case two, seven, eight and nine. This means that Regulating was working properly by adopting a new leader with lane-changing instruction.

7.4.2 General Analysis

Generally speaking, NAUSEA generated the desired output and worked properly. SAIL also worked as desired and supported NAUSEA as designed. All components in SOAV also worked well. They succeeded in planning Actions for the scenario in all test cases and replanning according to Assignments, i.e., compensate failed Assignment if necessary by finding suitable Ego-vehicle in Test Case 2,3,5,6,7,8,9. However, as suggested by the results, delays were found in evaluating Durations or executing orders. Hence, in Phase two, the stability of Smith was examined through automatic tests.

Although the time spent on decision-making is platform-dependent, it can still reflect whether Smith is working stably in a specific platform with the same scenario. Smith should spend almost the same time to trigger a particular Assignment so that every participant experiences the same context when this Assignment is triggered.

7.4.2.1 Can Smith Generate the Desired Output? (Is the Algorithm Working?)

In Phase one, Smith generated the desired output (13 Participant-based tests) and in phase two, Smith generated the desired output with a success rate of 100% in all 90 tests. The plan evaluation procedure was also working properly.

7.4.2.2 How well did Smith Generate the Output? (Is the Implementation Good?)

Order lag was measured in Phase one by calculating the difference between the Instant Sim1 receives an order from Smith and the Instant when Smith made the decision. The latter is the release time. As shown in Table 7.3, Smith needs 2 ± 1 frame(s) on average to execute Assignments, which is supposed to be a reference value as the time is a platform-dependent value.

Table 7.3: Statistics of Order Lag in Phase One (s)

Minimum	Maximum	Mean	Std. Deviation
.0000163	.1000000	.031511634	.0148830995

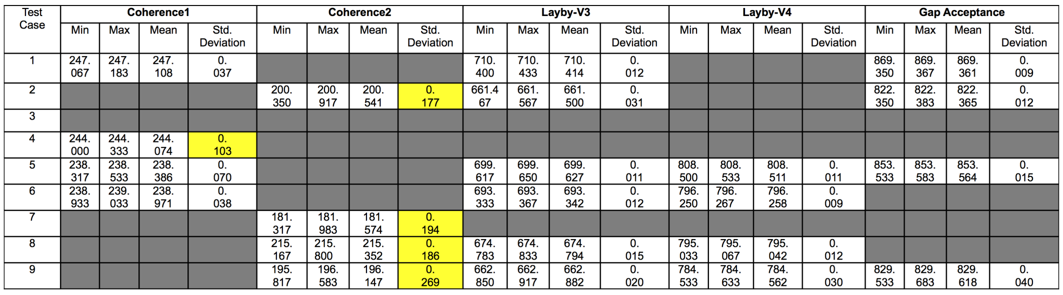
7.4.2.3 Is Smith Stable?

The release time was recorded in Phase two and is shown in Table 7.4, it exhibits that Smith was not stable enough, as the Instant when he made decision in the same test varied from less than one frame to more than ten frames compared to the mean release time. Since there was no difference regarding algorithms or data structures when performing the same Assignment on the same machine, the cause of this variance must be the contention for Mutex between the Perception and Cognition layers.

7.4.2.4 Comments from Participants

Participants were encouraged to give some general comments after their drives in order to provide references for future development. It has been found that all

Table 7.4: Statistics of the Release Times in Phase Two (s)



Test Case	Coherence1				Coherence2				Layby-V3				Layby-V4				Gap Acceptance			
	Min	Max	Mean	Std. Deviation	Min	Max	Mean	Std. Deviation	Min	Max	Mean	Std. Deviation	Min	Max	Mean	Std. Deviation	Min	Max	Mean	Std. Deviation
1	247.067	247.183	247.108	0.037					710.400	710.433	710.414	0.012					869.350	869.367	869.361	0.009
2					200.350	200.917	200.541	0.177	661.467	661.567	661.500	0.031					822.350	822.383	822.365	0.012
3																				
4	244.000	244.333	244.074	0.103																
5	238.317	238.533	238.386	0.070					699.617	699.650	699.627	0.011	808.500	808.533	808.511	0.011	853.533	853.583	853.564	0.015
6	238.933	239.033	238.971	0.038					693.333	693.367	693.342	0.012	796.250	796.267	796.258	0.009				
7					181.317	181.983	181.574	0.194												
8					215.167	215.800	215.352	0.186	674.783	674.833	674.794	0.015	795.033	795.067	795.042	0.012				
9					195.817	196.583	196.147	0.269	662.850	662.917	662.882	0.020	784.533	784.633	784.562	0.030	829.533	829.683	829.618	0.040

their comments were not related to the algorithms developed in this research, but how the algorithms can be used (e.g., what acceleration rate should be used in some circumstance) and how the simulation software can be improved. Some of them noted that 1) in "Coherence1" and "Cohehence2", the acceleration rate of the leader was relatively high; 2) vehicle 3 or vehicle 4 in "Layby-V3" or "Layby-V4" pulled out without advance indication (although they were designed to be a surprise Assignment) and 3) the trajectories of the simulated vehicles were not realistic enough, e.g., lane changing trajectory and turning movement at a junction.

7.5 Summary

In general, SAIL/NAUSEA was working as desired by committing to Assignments and planning Actions towards Assignments. It can be used to orchestrate scenarios with replanning capability. SOAV was also working properly by generating required scenarios. However, the implementation of Smith needed enhancement in terms of the communication mechanism between the Perception and Cognition layers. Moreover, NAUSEA has been designed to handle scenario-related Actions, so more work can be undertaken in the future regarding Operational behaviours to make Smith handle them as well, e.g., lane changing trajectory. However this may increase the network workload as Smith will need to send relevant parameters in every update step. In addition, in future experiments, *IntervalBefore* should be used according to the plan evaluation procedure in NAUSEA and Post

Conditions can be provided for any parallel Assignments in order to control the deadlines.

When it comes to multiple Monitors, an event-based Monitor (WHEN or EVERY) can be problematic because the satisfaction of multiple Monitors at the same time may be difficult. As a result, the Monitor will concentrate on periodic Monitors, i.e., WHENEVER or ASLONGAS, so that multi-Monitor can be satisfied at the same time, although WHENEVER and ASLONGAS imply the repetition of Assignment-actions, which is not handled by NAUSEA.

In addition, with a modular architecture, NAUSEA can be enhanced in the future with some specific algorithms to make SOAV more powerful in dealing with scenarios:

- a longitudinal actor preparation algorithm that can be adopted into the Regulating layer to make Smith choose an appropriate acceleration rate when navigating the Ego-vehicle to the Formation Position. This is the speed adaptation behaviour;
- a Role Matching algorithm that can be adopted to enhance the Role Matching procedure with the capability of evaluating all Assignments at the same time and thus generating a Role Matching plan based on all Assignments and,
- a Flock control algorithm that can be adopted into every simulated vehicle to perform Flock manoeuvre such as merging or leaving a Flock.

Chapter 8

Experiment Two - Driving in the Matrix and Results

Have you ever had a dream, Neo, that you were so sure it was real? What if you were unable to awake from that dream? How would you know the difference between the dream world, and the real world?

-The Matrix

8.1 Introduction

Experiment two was designed for Framework Application, and was used to ascertain the functionality of SOAV in a "real" application from a previous research project.

The scenario was derived from the ITERATE project because it had failed several times as sometimes the participant was too far behind the proposed vehicle for interactions (Forsman *et al.*, 2011). Hence, SOAV was used to create the contexts needed and trigger the interactions.

Durations/Monitors/Failure Conditions/Success Conditions/Action Profiles for each Assignment were specially crafted based on original values in order to make sure that the interactions were obvious for observation and the corresponding Assignments would not fail to be triggered because the Monitor conditions were in conflict with any safety configurations, e.g., accepted gap for overtake.

Hence, larger values could be adopted, e.g., large time headway, large acceleration rate or large Durations.

The experiment contained one scenario without metric constraints as no metric constraints were included in the original definition. Because there were no extra road segments for retrying scenarios and the final triggering location of some Assignments could not be determined in advance, the replanning of Assignments failed by Failure Conditions was not allowed, but Role Matching was allowed in order to change the Ego-vehicle.

In addition, Smith was not changed to single thread in this experiment because delays were acceptable, when the Monitors used to trigger the main Assignments (Assignment one, two and four that will be elaborated later) were based on state, not event. As a result, since a single threaded Smith was not needed, the multi-threaded Smith was not changed.

8.2 Equipment

Sim1 and Smith version one were used in this experiment.

The same laptop was used to run Smith and was equipped with an Intel® T2130 CPU and 2GB of memory running Ubuntu Linux 11.10 32bit. Communication between Sim1 and Smith was based on a wired 10Mb hub.

8.3 Experiment

8.3.1 Scenario Description

The scenario contained a two-lane motorway with some curved road segments. It took a participant a maximum of 33 minutes to complete. The speed limit was 110 *km/h* and participants were told to drive freely, as if they were going to a destination with a purpose.

A second task was also adopted because the original scenario had one. While each participant was driving, an automated message instructed him/her to carry out a counting-back task. For example, he/she would hear the following instruction:

"*Please count backwards in 7 from 949. Start when you are ready*"

His/her task was therefore to announce the following numbers in sequence: 949, 942, 935, 928, 921 etc.

He/she carried on doing so until he/she heard the instruction:

"*Please stop the counting-back task*"

This task was kept, but the analysis of this experiment ignored its potential effect on participants, as this research concentrates on the examination of Assignment execution during the scenario and the proposed behaviours of surrounding vehicles.

After about 6 minutes' free drive, every participant was exposed to four Assignments in sequence as illustrated in Figure 8.1.

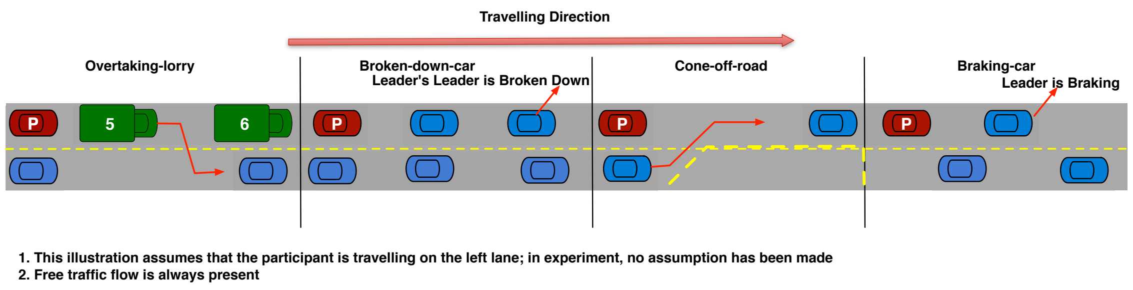

Travelling Direction
Overtaking-lorry
Broken-down-car
Leader's Leader is Broken Down
Cone-off-road
Braking-car
Leader is Braking
P
5
6
1. This illustration assumes that the participant is travelling on the left lane; in experiment, no assumption has been made
2. Free traffic flow is always present

Figure 8.1: Illustration of the Scenario for Experiment Two

8.3.1.1 Assignment "Overtaking-lorry"

This Assignment involved two lorries; both were travelling in Lane 1 (left lane) as illustrated in Figure 8.1.

When the participant's vehicle's time headway to the Ego-vehicle, which should be lorry 5, was less than 4 seconds, and the time-to-collision with the Ego-vehicle was greater than 6 seconds (i.e., close enough but no possibility of collision), lorry 5 should start to overtake lorry 6 as lorry 5 wanted to achieve a higher desired speed. In this Assignment, those two lorries would eventually block the two lanes and force the participant to decelerate. The Duration of this

Assignment was set to 80 seconds in order to guarantee that the overtaking could be finished.

8.3.1.2 Assignment “Broken-down-car”

This Assignment involved a car in the Formation Position of “CloseLeader0”, which indicates “Leader’s Leader” as illustrated in Figure 8.1 and implies that a leader and a leader’s leader should both exist.

When 1) the participant passed or was on the road segment “r4.0” (14637 m from the start position of the participant’s vehicle), 2) the participant’s headway to its leader was less than 4 seconds, and 3) the participant’s time-to-collision with its leader was greater than 5 seconds, the car in the position of “CloseLeader0” should break down. The Assignment-action would last for 70 seconds, which meant the lane would be blocked for 70 seconds.

This Ego-vehicle would force the leader of the participant to decelerate, in which case the participant had to overtake or brake and wait for the leader’s leader to recover.

In order to test and demonstrate other Formation Positions, this Assignment was modified by using cars to be the leader’s leader instead of using lorries, because using lorries would narrow Smith’s choice during Role Matching.

8.3.1.3 Assignment “Cone-off-road”

This Assignment needed to cone off the lane that the participant was not travelling in; if the participant was travelling in Lane 1, then Lane 2 (right lane) would be coned-off. If the participant was travelling in Lane 2, then Lane 1 would be coned-off. The Lane that was left for traffic would last for 1 km. The Monitor for this Assignment was that as long as the participant’s vehicle reached or passed the road “r5.2”, which was 26063 m from the start position of the participant’s vehicle, the Assignment should be triggered.

8.3.1.4 Assignment “Braking-car”

This Assignment involved a car in the Formation Position of “Leader”, which is the leader position.

When 1) the participant reached the road "r6.0" (34883 m from the start position of the participant's vehicle), 2) the time headway to the leader, which is the Ego-vehicle, was less than 4 seconds, and 3) the time-to-collision with the Ego-vehicle was greater than 5 seconds, the Ego-vehicle should brake with an acceleration rate of -5.5 m/s^2 and force the participant to overtake or decelerate. This Assignment would block the lane for 20 seconds, which was the Duration of the Assignment-action.

8.3.1.5 Other Information

There were also some minor Assignments that were used to assist Smith by providing relevant parameters:

- Assignment "Create-lorry": This Assignment was added to tell Smith when the right time to create the candidate lorries was. SMM had been instructed to create two lorries after the free drive when the participant reached the road segment "r3.0" (3715 m from the start position of the participant's vehicle). Two lorries would be created about 1300 m ahead of the participant's vehicle in the left lane (lane one) with an initial distance difference of 20 m. Their initial desired speeds were 25.1 m/s and 25.0 m/s respectively;

- Assignment "Clear-lane": this Assignment was used to clear the lane that the participant was travelling in before the execution of the Assignment "Overtaking-lorry". It was used to make sure that the participant could meet with the lorry and would not be interrupted by any other simulated vehicles. This Assignment would be triggered when the participant reached the road segment "r3.1", which is 1850 m away from the road segment "r3.0";

- Assignment "Restore": this Assignment was used to notify SMM which behaviours of the simulated vehicles should be restored according to specific needs, e.g., whether or not overtaking should be allowed. "Restore" would be invoked sequentially after each main Assignment, e.g., "Overtaking-lorry".

Moreover, the relationships between those Assignments were specified with *IntervalBefore* in OSO only, because all Assignments were designed to be triggered sequentially and no parallel Assignments were presented.

The four main Assignments were represented in OSO. The details of the representation can be found in Appendix C.2 on Page 257. The essential information in each Assignment have been represented by directed graphs.

8.3.2 Experimental Procedure

Ten participants were recruited to act as framework testers. Eight of them were male and two female. One of them had just obtained his/her driving license. Every participant drove in the scenario once.

Before the experiment, the participants were told that:

- they should drive freely with a speed limit of “110 kmh” and imagine that they were driving to a destination with a desired purpose;
- they were warned with the in-vehicle collision warning system if a potential collision with the lead simulated vehicle was about to happen;
- when they were driving, as specified in the previous section, they were instructed to carry out a counting-back task.

During the experiments, as usual, the following data were recorded: every vehicle’s information in Sim1 and orders from Smith. Details regarding the package containing every vehicle’s information can be found on Page 98.

In addition, the Instants when the Sim/SMM received orders were used in this experiment to do analysis because delays were investigated in experiment one already. The states of simulation at those Instants were the actual conditions that the participants could experience, which reflected the actual functionality of SOAV. Because those Instants were not the (online) release time or the (adjusted) deadline, some general terms, which are start/finish times, will be used in this experiment. The start times indicated the Instants when the Sim/SMM executed/started some Assignments; the finish times indicated the Instants when the Sim/SMM finished some Assignments and the “Restore” Assignments were executed after them, which reflected the actual Durations that the participants experienced.

8.4 Results

As participant nine had an unstable lane tracking record, it appears that he/she was not familiar with driving or the desktop driving simulator. His/her data were therefore marked as invalid and were not included in the analysis. However, in order to demonstrate his/her driving style, some data were included when examining the Regulating procedure.

Because no metric constraints were given in this experiment, an overview of the plan evaluation procedure is not covered as the evaluation was always consistent.

8.4.1 Algorithm Evaluation

8.4.1.1 Role Matching

In the Assignment “Broken-down-car”, Smith had to find a leader’s leader of the participant’s vehicle to carry out the Assignment, so a small formation containing two vehicles was anticipated. In the Assignment “Braking-car”, Smith had to find a lead vehicle of participant’s vehicle to carry out the Assignment. In the Assignment “Overtaking Lorry”, Smith had to find a leader to perform an overtaking Action, but eventually found a lorry in nearside lane in all experimental cases. The results of the Role Matching in Assignments one, two and four are shown in Table 8.1.

8.4.1.2 Regulating

Speed adaptation for Assignments was straight-forward and worked properly in Assignment one - “Overtaking-lorry”.

In this research, Smith focused on controlling one vehicle, so SMM directly set the vehicle 6’s (lead lorry 6) Regulating speed based on the Ego-vehicle’s (vehicle/lorry 5) in Assignment “Overtaking-lorry”. As a result, the Ego-vehicle

Table 8.1: Summary of Role Matching Results

Participant	Assignment One-Overtaking-lorry		Assignment Two-Broken-down-car		Assignment Four-Braking-car	
	Found Vehicle	Times of Successful Role Matching	Found Vehicle	Times of Successful Role Matching	Found Vehicle	Times of Successful Role Matching
1	5	1	9	1	20	5
2	5	1	14	3	3	13
3	5	1	14	1	6	2
4	5	1	19	1	15	2
5	5	1	13	1	13	4
6	5	1	9	1	12	2
7	5	1	19	2	3	6
8	5	1	17	4	3	4
10	5	1	13	2	18	5

and its lead lorry 6 were instructed to 1) drive at a constant speed (20 m/s) to meet with the participant's vehicle and 2) adopt a higher speed when the Assignment was about to be started (less than 6 seconds time headway). Data have been extracted by applying the following filter:

From the road segment "r3.0" , which was also the condition of a Monitor in Assignment "Overtaking-lorry", to the road segment where the Assignment was executed, all data regarding simulated vehicles within a 1500 m radius of the participant's vehicle were extracted from the raw data. This extracted set of data contains the following information of each vehicle including the participant's: vehicle ID, time stamp, lane offset and travelled distance compared to the start point of "r3.0". It should be noted that vehicle 0 was the participant's vehicle and the lane offset indicates which lane the vehicle was travelling in.

The extracted data have been illustrated in 20 figures in Appendix B:

- from Figure B.1 on Page 218 to Figure B.10 on Page 227, the lane travel trajectories of the vehicles are shown;
- from Figure B.11 on Page 228 to Figure B.20 on Page 237, the longitudinal travel trajectories of each vehicle are shown.

In addition, some vehicles that were 1300 m ahead or 325 m behind the participant's vehicle would be deleted or added from time to time to maintain the traffic flow, so their trajectories would not be used for illustrations because of two reasons: 1) their trajectories were not continuous and the corresponding representations on the Figures can be unclear and 2) other vehicles are sufficient for demonstrations. However, trajectories of vehicle 5 and 6 (lorry 5 and its lead lorry 6) were always kept even if their trajectories could be discontinuous at some Instants. For example, as is illustrated in Figure B.13 on Page 230, vehicle 5 and 6 were both deleted and added several times between 286.117 seconds and 340.4 seconds.

The Regulating layer did not play the same role as it did in experiment one, because Smith did not need to regulate the Ego-vehicle with lane-changing behaviour. In addition, lane-changing during Regulating was forbidden in Assignment one, because Smith found an overtaking Action in this Assignment.

8.4.1.3 Situation Assessment

The results of the scenario execution are shown in Table 8.2. Assignments with Durations have been listed. The actual value of Monitor-related parameters after Sim1 had executed the Smith Order following one decision loop (1/60 seconds) are shown in Tables 8.3 and 8.4.

As shown in Tables 8.2 to 8.4, all Assignments were executed and finished successfully, although delays were found in some Assignments and resulted in differences between stated Trigger conditions and actual execution conditions. For instance, the stated Trigger condition in one of the Monitors of Assignment two indicated the time headway should be less than 4 seconds, but participant two experienced a time headway of 4.12879 seconds. Because the triggering conditions in this experiment were very loose, i.e., some of them were based on state conditions, this delay or difference is acceptable provided that the Assignment was successfully executed, and delays within 1 - 3 frames were found in experiment one.

When participant four was driving, Smith encountered a run-time fault just before triggering the Assignment "Overtaking-lorry". Further investigation was

Table 8.2: Summary of Scenario Execution Results

Participant	Assignment One Overtaking-lorry		Assignment Two Broken-down-car		Assignment Three Coned-off-Road		Assignment Four Braking-car	
	Start Time (*s*)	Finish Time/ Duration (*s*)	Start Time (*s*)	Finish Time/ Duration (*s*)	Start Time (*s*)	Finish Time (*s*)	Start Time (*s*)	Finish Time /Duration (*s*)
1	362.367	442.733/ 80.366	668.717	739.133/ 70.416	1069.05	N/A	1509.15	1529.43/ 20.28
2	328.333	408.533/ 80.2	661.3	731.517/ 70.217	1013.03	N/A	93.55	113.567/ 20.017
3	460.45	540.85/ 80.4	792.9	863.4/ 70.5	1097.82	N/A	1581.53	1602.1/ 20.57
4	Not Executed	N/A	632.383	702.667/ 70.284	1033.2	N/A	1413.18	1433.43/ 20.25
5	342.85	423.1/ 80.25	641.65	712.017/ 70.367	1003.93	N/A	1394.93	1415.17/ 20.24
6	353.55	433.8/ 80.25	660.267	730.517/ 70.25	1021.43	N/A	1354.15	1374.67/ 20.52
7	434.433	514.733/ 80.3	853.55	923.867/ 70.317	1135.75	N/A	1580.38	1600.9/ 20.52
8	358.083	438.233/ 80.15	762.317	823.717/ 70.4	1077.3	N/A	1488.88	1509.08/ 20.2
10	359.9	440.2/ 80.3	773.817	843.933/ 70.116	1026.57	N/A	1382.33	1402.53/ 20.2

Table 8.3: Summary of Scenario Execution Conditions for Assignment One and Two

Participant	Assignment One Overtaking-lorry					Assignment Two Broken-down-car				
	Start Time (*s*)	Ego-Vehicle	Participant's Position	TTC and Time HW from the Participant(*s*)		Start Time (*s*)	Ego-Vehicle	Participant's Position	Leader's TTC and Time HW from the Participant (*s*)	
				TTC	THW				TTC	THW
1	362.367	5	R3.8, 22.0335(*m*)	11.837	2.9356	668.717	9	R4.1, 15.6923(*m*)	21.7599	2.78012
2	328.333	5	R3.6, 125.284(*m*)	10.555	2.8581	661.3	14	R4.3, 246.573(*m*)	33.1702	4.12879
3	460.45	5	R3.13, 67.4482(*m*)	14.709	3.0395	792.9	14	R4.9, 69.1635(m)	32.8794	3.53379
4	N/A	5	N/A	N/A	N/A	632.383	19	R4.1, 9.24009(*m*)	Inf	2.97597
5	342.85	5	R3.7, 44.3262(*m*)	13.019	3.0395	641.65	13	R4.1, 13.9901(*m*)	117.045	1.6769
6	353.55	5	R3.7, 51.4248(*m*)	12.073	2.9798	660.267	9	R4.1, 13.6097(*m*)	15.6334	1.29995
7	434.433	5	R3.12, 118.449(*m*)	50.041	3.6757	853.55	19	R4.16, 116.707(*m*)	Inf	2.13465
8	358.083	5	R3.8, 21.5241(*m*)	12.728	3.0031	762.317	17	R4.14, 31.4533(*m*)	79.5794	4.08382
10	359.9	5	R3.8, 59.1571(*m*)	9.2003	2.7402	773.817	13	R4.15, 42.4593(*m*)	27.1356	4.10151

Table 8.4: Summary of Scenario Execution Condition for Assignment Three and Four

Participant	Assignment Three Coned-off-Road					Assignment Four Braking-car				
	Start Time (*s*)	Ego-Vehicle	Participant's Position	TTC and Time HW from the Participant (*s*)		Start Time (*s*)	Ego-Vehicle	Participant's Position	TTC and Time HW from the Participant (*s*)	
				TTC	THW				TTC	THW
1	1069.05	N/A	R5.2, 11.1795(*m*)	N/A	N/A	1509.15	20	R6.1, 17.1133(m)	Inf	2.21234
2	1013.03	N/A	R5.2, 13.6889(*m*)	N/A	N/A	93.55	3	R6.3, 135.262(*m*)	34.1812	4.12553
3	1097.82	N/A	R5.2, 14.025(*m*)	N/A	N/A	1581.53	6	R6.13, 245.052(*m*)	22.7350	4.05222
4	1033.2	N/A	R5.2 10.8499(*m*)	N/A	N/A	1413.18	15	R6.1, 13.5367(*m*)	40.2678	2.61181
5	1003.93	N/A	R5.2, 9.49668(*m*)	N/A	N/A	1394.93	13	R6.1, 6.30467(*m*)	Inf	1.68058
6	1021.43	N/A	R5.2, 14.9132(*m*)	N/A	N/A	1354.15	12	R6.1, 20.6451(*m*)	8.8721	2.46182
7	1135.75	N/A	R5.2, 13.3518(*m*)	N/A	N/A	1580.38	3	R6.8, 149.303(*m*)	14.1568	2.55677
8	1077.3	N/A	R5.2, 17.3984(*m*)	N/A	N/A	1488.88	3	R6.9, 212.744(*m*)	18.1457	4.05322
10	1026.57	N/A	R5.2, 4.71521(*m*)	N/A	N/A	1382.33	18	R6.1, 10.8702(*m*)	31.6703	1.67951

started after this incident, but no reason could be found. Moreover, no run-time fault has been found after this experiment, so this fault might have been caused by the LCM library.

In addition, participant two encountered two run-time faults that were caused by SMM. At the same time as SMM was placing cones for the Assignment "Cone-off-road", the participant was changing lane, resulting in a lane number that was not the right or left, so a null lane was assigned to place those cones. Therefore, he/she did the experiment three times. In all three runs, all Assignments were executed. The start/finish time of Assignment three was recorded in run no.1, and the start/finish time of Assignment four was recorded in run no.3. This bug had been fixed in the preceding experiments and because the four main Assignments were independent from each other, all data are still valid for further analysis. However, this means that the framework requires more tests regarding stability, especially on the implementation of SMM, which is used to execute orders from Smith.

8.4.2 General Analysis

Primarily, as shown in Table 8.2, Smith succeeded in finding suitable vehicles and executing Assignments, although failures occurred in Assignment one. It should be noted that Assignment one was intentionally designed to avoid replanning because 1) the original scenario description did not specify that replanning was allowed and most importantly, 2) there was no dynamic environment construction mechanism, so if the participant failed in triggering the Assignment, the road segment might be too short to implement another Assignment for compensation.

As shown in Table 8.1, Role Matching had a success rate of 100% in Assignments two and four.

Moreover, in Assignment one, because of the adoption of vehicle constraints, Smith had no choice but to regulate the selected lorry to reach the proposed Formation Position. Two incidents occurred in Assignment one:

1) Incident One: when participant four was driving, Smith encountered a run-time fault just before triggering the Assignment, although the preparation was a success;

2) Incident Two: the data from participant nine was excluded as he/she had a very unstable driving style, i.e., unstable lane tracking and speed choice. The lane-tracking of participant nine can be found in Figure B.9 on Page 226. However, his/her driving style also suggested that NAUSEA may need some enhancement in speed adaptation when reaching the Formation Position, as the participant was still able to miss the Assignment, especially when encountering some “unusual” participants.

From the longitudinal trajectories in Assignment one, the lorry blocked the simulated vehicles behind it, i.e., lower speed had to be adopted by those vehicles, so the traffic flow was affected. If blocked vehicles were allowed to overtake, this Assignment could be failed because only a lorry was proposed to overtake the slower lorry ahead.

As a result, some conclusions can be drawn:

- Role Matching without vehicle constraint showed a success rate of 100% because Smith had more choices and there was no need to manipulate the Ego-vehicle's speed profile;

- Role Matching with vehicle constraints may cause long-time speed adaptation. This can be solved by adopting more simulated vehicles with the same model and allowing replanning. However, the adoption of those specific simulated vehicles can change the distribution of vehicle types and may affect the traffic flow in general;

- Preparing an Ego-vehicle with vehicle constraints can be problematic because it may block the traffic flow if it has to travel slower than the participant and meet him/her in a particular position ahead. Without replanning permission and more candidate vehicles with the same vehicle type, the Assignment can be open to failures and even if the permission can be granted, the Assignment is still liable to failures because it involves a restricted vehicle type.

8.5 Summary

From the analysis above, Smith can be used to deal with "real" scenarios, but it could also be found that:

1) more work should be undertaken in speed adaptation algorithms in terms of both vehicle restrictions and specific scenario constraints, e.g., only a restricted type of vehicle is allowed to perform particular Actions;

2) this research focuses on the specification and scheduling of Actions for the virtual driver. Adoption of Monitors and Action profiles has been ignored. In this experiment, the ignorance of Action profile had caused a problem as one of the participants noticed that the overtaking was slow. Hence, Action profiles should be considered in the future. Moreover, because the Regulating behaviour, especially the speed adaptation, was lacking, change of Monitor conditions may fail some Assignments because the participant could miss the

Monitor conditions as the Ego-vehicle cannot meet with the participant's vehicle;

3) further software tests may be needed to avoid any run-time fault of Sim1/SMM, especially regarding the implementation of SMM. As changing delays were caused by the multi-threaded architecture, single-threaded Smith was tried in verification experiment three in order to see if the delay could be consistent with single threading and be reduced to one decision loop.

After two experiments, the limitations of the simulation software were also noticed, especially regarding the perception of the surrounding environment. For instance, the cones in Assignment two should have been noticed by Smith so that he could notify the SMM to clear the lane or the cones should be "seen" by the simulated vehicles so that they would not travel in the coned-off lane. However, cones were invisible to both simulated vehicles and Smith, so a separate obstacle was placed just before the coned lane. Simulated vehicles were able to "see" the obstacle but some of them just stopped in front of it. In order to test SOAV with its full power, the limitation regarding environment perception should be taken care of.

In addition, interfaces for object creation or even dynamic vehicle creation should be considered for future testing of SOAV in order to avoid system faults such as those found in this experiment.

Chapter 9

Experiment Three - Driving at VTI and Results

I know exactly what you mean. Let me tell you why you're here. You're here because you know something. What you know you can't explain, but you feel it. You've felt it your entire life, that there's something wrong with the world. You don't know what it is, but it's there, like a splinter in your mind, driving you mad. It is this feeling that has brought you to me. Do you know what I'm talking about?

-The Matrix

9.1 Introduction

As is shown in experiment two, when vehicle restrictions were presented and no replanning was allowed, actor preparation based on speed adaptation could be hard to achieve.

With support from HEIF 5[1], another verification experiment was carried out with the simulation software from VTI in order to adopt the actor preparation algorithm in Olstam *et al.* (2011), which can enhance the speed adaptation in the NAUSEA. In this experiment, SMM was used to control simulated vehicles only.

[1]Higher Education Innovation Funding, UK

Experiment three was designed for Framework Migration & Enhancement. A scenario was designed to test SOAV with new enhancements. This final procedure was also used to demonstrate that the framework can be extended. In total there were three purposes:

1) To demonstrate and test the integration of the actor preparation algorithm from Olstam *et al.* (2011) with NAUSEA;

2) To demonstrate and test the establishment of SOAV based on VTI's simulation software;

3) To demonstrate the recipe of a High-level Action"Block" and form a Situation named "Blocking"

A scenario was designed based on Play[1] three of the scenario in Olstam *et al.* (2011), because it involved a Flock of three simulated vehicles plus a lead vehicle ahead of the participant's. It was a suitable interaction for this experiment as speed adaptation could be used along with a High-Level Action "Block". Furthermore, Durations/Monitors/Failure Conditions/Success Conditions/Action Profiles for each Assignment were different from the original values in Olstam *et al.* (2011) in order to make the interactions observable within the visual display. Changes of those values had no influence on the results of this verification step.

In this experiment, only one road segment was used so replanning after the failure of Assignments was not allowed. No vehicle restrictions or metric constraints were included and changing Ego-vehicle was allowed.

9.2 Equipment

Sim2 and Smith version two were used in this experiment.

[1]The definition of Play has been described by Assignment in this research, so it has been newly defined as a pre-defined situation to which the scenario designers would like the participant be exposed. It is generated by Smith according to requirements in the Assignments. A scenario can contain several Plays. A Play may contain several interactions generated by several Actions.

A laptop was used to run Sim2 and Smith. It was equipped with a 2.3 GHz Intel® Core i7 CPU and 16 GB memory. The threading mechanism in the kernel was changed to make it runnable under OS X. The visual system was run under OS X using Wine[1], which is used to run Windows applications on OS X.

9.3 Experiment

9.3.1 Scenario Description

The scenario consisted of a two-lane motorway with some curved road segments. It was 12.4 *km* long and could take simulator drivers a maximum of 7 minutes to complete. The speed limit was 110 km/h and no extra instructions were applied. At the beginning of the experiment, a dense traffic flow was presented using an existing interface from Sim2, so nine simulated vehicles were placed around the participant's vehicle as illustrated in Table 9.1.

Table 9.1: Initial Traffic Conditions

Vehicle Number	Autonomous Simulator Driver	Vehicle Model	Lane (Fast or Slow)	Distance to the Start of the Road Segment(*m*)	Initial Desired Speed (*m/s*)
0	Yes	Volvo S40	Slow	300	29.167 - 31.667
132	No	Volvo S40	Slow	400.791	30.5556
133	No	Toyota Corolla	Slow	600.791	28.1111
134	No	Mercedes Lorry	Slow	750.791	21.75
135	No	Volvo S40	Slow	202.458	32.0833
136	No	Audi TT	Fast	272.458	31.4722
137	No	Volvo S40	Slow	2.45757	33.6111
138	No	Renault Master	Slow	1054.12	29.6389
139	No	Mercedes Lorry	Slow	1254.12	21.25
140	No	Audi TT	Fast	904.124	31.1667

Autonomous simulator drivers were used in this experiment due to the tight schedule and the fact that it is easier to adopt different desired speeds using

[1]http://www.winehq.org

autonomous simulator drivers than using human drivers. Ten autonomous simulator drivers were used and shared the same behaviour model as the autonomous vehicles in Olstam *et al.* (2011). The desired speeds of each autonomous simulator driver were set to 105, 106, 107, 108, 109, 110, 111, 112, 113 and 114 kmh respectively from less than the speed limit to greater than the speed limit. In this experiment, participants were called directly as simulator driver (DS) unless an algorithm is described for general participants in Section 9.3.2.1.

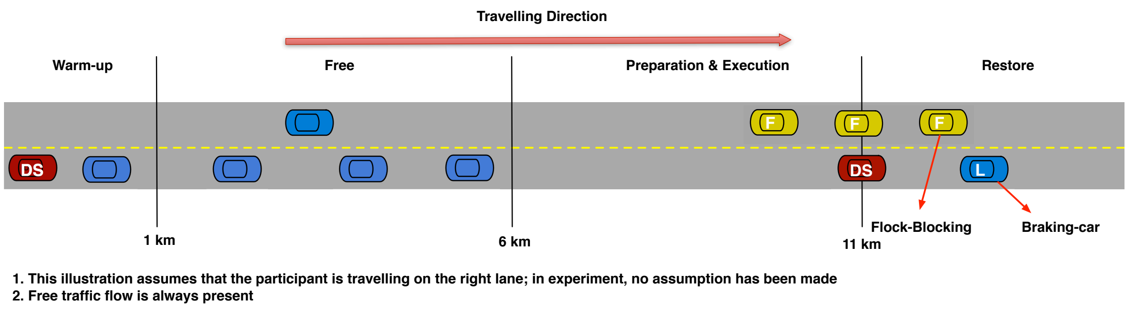

Travelling Direction
Warm-up
Free
Preparation & Execution
Restore
DS
F
F
F
DS
L
1 km
6 km
11 km
Flock-Blocking
Braking-car
1. This illustration assumes that the participant is travelling on the right lane; in experiment, no assumption has been made
2. Free traffic flow is always present

Figure 9.1: Illustration of the Scenario for Experiment Three

As illustrated in Figure 9.1, this experiment involved four phases: 1) Warm-up phase was used to initiate vehicles' speeds; 2) Free phase was a free driving period lasting 5000 m; 3) Preparation & Execution phase involved actor preparation based on Regulating layer, after which the Assignments were performed and 4) Restore Phase was used to restore the traffic flow after the Assignments were executed. Moreover, in the four phases, there were two Assignments for interactions: "Braking-car" and "Flock-Blocking". An extra Assignment was used to indicate when the Role Matching should be invoked.

9.3.1.1 Assignment "Braking-car"

This Assignment involved a simulated vehicle in the Formation Position of "Leader", which is the leader position.

When the simulator driver reached 11000 m of the road segment, and the simulator driver's distance headway to the Ego-vehicle, which is the leader, was less than 200 meters, the Ego-vehicle should decelerate with an acceleration rate of -1 m/s^2 and hold this rate for 18 seconds.

This Assignment would force the simulator driver to decelerate with the help from the Assignment below, as the simulator driver should be unable to perform overtaking because of a Flock travelling in the adjacent lane.

9.3.1.2 Assignment "Flock-Blocking"

This Assignment shared the same Monitors as the last Assignment and involved a Flock that was used to block the simulator driver when the Assignment "Braking-car" was being carried out. The Assignment-action was a High-Level Action "Block", whose recipe and related information are provided in Section 9.3.3. General speaking, this High-Level Action included several sub-Actions to 1) create a Flock containing three simulated vehicles, 2) clear simulated vehicles that are not Ego-vehicle/flock before the interaction, 3) maintain the speed of the Flock for 18 seconds when the Monitors specified in this Assignment become true and 4) restore the configurations of all simulated vehicles.

9.3.1.3 Assignment "Role-Matching"

This Assignment notified Smith when Role Matching should be invoked. In this experiment, when the driving simulator reached 6000 m of the road network, Smith started to look for an appropriate Ego-vehicle/flock.

The three Assignments were represented in OSO. The details of the representation can be found in Appendix C.3 on Page 275. The essential information in each Assignment has been represented by directed graphs.

9.3.2 Additional Information of the Experiment

9.3.2.1 Actor Preparation Algorithm

9.3.2.1.1 Actor Preparation for Ego-vehicle

By adopting the actor preparation algorithm from Olstam *et al.* (2011), Smith enhanced his speed adaptation in the Regulating layer, which was used to approach the simulator driver according to the Formation Position as close as possible in an inconspicuous manner. This was handled by the Action named β_1 (Generate-formation). The following content includes details regarding the calculation of acceleration rates for actor preparation. Let us suppose that the first successive Assignment for Smith is A_0.

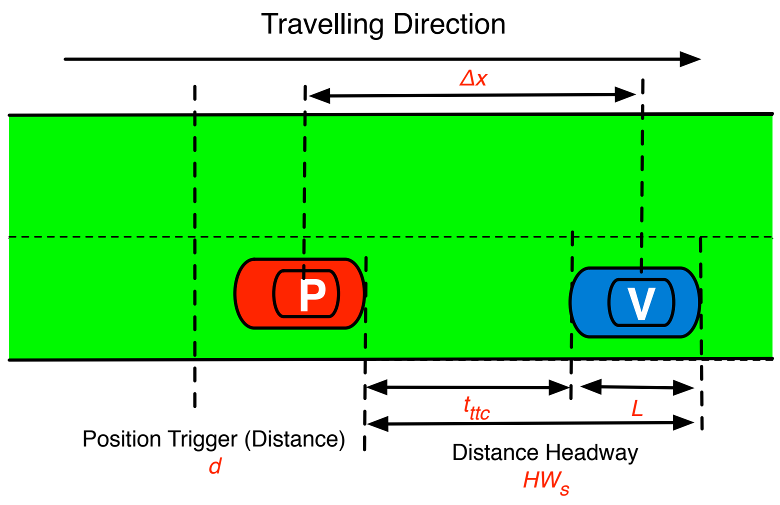


Figure 9.2: Illustration of Monitor Conditions of A_0

In Figure 9.2, Ego-vehicle V's final position for A_0 is illustrated; it is proposed to be the participant's leader. Smith should carry out A_0 when the participant has passed the position of d, i.e., $x_P \geqslant d$ and $d = x_{play}$, where x_{play} is used in Olstam *et al.* (2011) to specify the start position of a Play. Moreover, its time-to-collision with the vehicle V (t_{ttc}) should be large enough to guarantee that the participant's vehicle will not collide with the simulated vehicle V. The distance between the two vehicles (Δx) should be less than the upper distance headway HW_{su}, but greater than the lower distance headway HW_{sl} if the two vehicles share the same length. If the Assignment will be triggered at $\hat{t}$, the start conditions of A_0 are:

$$x_P^{\hat{t}} \geqslant d \tag{9.1}$$
$$HW_{sl} < \Delta x^{\hat{t}} < HW_{su} \tag{9.2}$$
$$t_{ttc}^{\hat{t}} = \inf \tag{9.3}$$

where $x_P^{\hat{t}}$ is the position of the participant's vehicle when the Assignment is triggered; $\Delta x^{\hat{t}}$ is the distance between the Ego-vehicle and the participant's vehicle when the Assignment is triggered; $t_{ttc}^{\hat{t}}$ is the participant's time-to-collision with the vehicle V when the Assignment is triggered.

If $v^{\hat{t}}$ is the speed of Ego-vehicle V when the Assignment is triggered and $v_P^{\hat{t}}$ is the participant's vehicle's speed when the Assignment is triggered, then in order to guarantee that $t_{ttc}^{\hat{t}} = \inf$, we should have:

$$\begin{aligned} x_P^{\hat{t}} &\geqslant d && (9.4)\\ HW_{sl} < \Delta x^{\hat{t}} &< HW_{su} && (9.5)\\ v^{\hat{t}} &\geqslant v_P^{\hat{t}} && (9.6) \end{aligned}$$

If v_R is the proposed role speed for the Ego-vehicle, which should be met when A_0 is triggered, a set of parameters for the Regulating layer can be derived from the inequalities mentioned above:

$$\begin{aligned} \Delta x_R &= D \quad (HW_{sl} < D < HW_{su}) && (9.7)\\ v_R &= v_P^{\hat{t}} && (9.8) \end{aligned}$$

where Δx_R is the proposed role distance between the Ego-vehicle and the participant when A_0 is triggered.

To sum up, Smith will guide the Ego-vehicle to achieve the following states for A_0 based on the participant's real-time position x_P and speed v_P, which replaces $v_P^{\hat{t}}$ in this research:

$$\begin{aligned} \hat{t} &= (d - x_P)/v_P + t && (9.9)\\ \Delta x_R &= \frac{HW_{sl} + HW_{su}}{2} && (9.10)\\ v_R &= v_P && (9.11) \end{aligned}$$

where t is the current time, so Smith can regulate the Ego-vehicle's speed with acceleration rate of:

$$a = \begin{cases} \frac{v_a - v}{t_c}, & \text{if } sign(v - v_a) \neq sign(v_R - v_a) \text{ and } t_c \leqslant 0.5 \cdot \tilde{t} \\ \frac{v_a - v_R + v_a - v}{0.3 \cdot \tilde{t}}, & \text{otherwise} \end{cases} \tag{9.12}$$

where,

$sign$ checks if a number is positive or negative;

$\tilde{t}$ is the time left for the Play: $\hat{t} - t$;

v is the current speed of the Ego-vehicle;

v_R is the proposed role speed specified in Equation 9.11;

v_a is the required average speed of the Ego-vehicle in order to reach the required position for the Assignment, which can be calculated by adopting Δx_R and $\hat{t}$ from Equation 9.10 and 9.9 respectively:

$$v_a = v_P + \frac{\Delta x_R - \Delta x}{\tilde{t}} \tag{9.13}$$

where Δx is the present distance between the Ego-vehicle and participant's vehicle.

t_c is the time when the vehicle should pass the required average speed in order to reach not only the role speed, but also the proposed start position of Play. It is calculated by:

$$t_c = \frac{v_a - v_R}{v - v_R} \cdot \tilde{t} \tag{9.14}$$

When preparing the Ego-vehicle, Smith will use the defined $\hat{t}$, ΔX_R and v_R to regulate the Ego-vehicle's speed based on the acceleration rate obtained from Euqation 9.12. When the Monitors of the Assignment are satisfied, Smith will trigger the Assignment even if the values of the three defined parameters are not the same as specified in Equations 9.9 to 9.11, because Smith just needs to make sure that when the Assignment is triggered, the conditions specified in Monitors, which are represented from Inequalities 9.1 to 9.3, have been satisfied.

If it comes to the specific Assignment "Braking-car" in experiment three, the following values have been used:

$$\Delta x_R = (150 + 200)/2 = 175(m) \tag{9.15}$$

$$v_R = v_{DS} \tag{9.16}$$

where,

200 m is HW_{su} defined in the Monitor of A_0, which is "Braking-car" in our case;

150 m is HW_{sl} defined as $(HW_{su} - 50)$ (m) in Smith;

v_{DS} is the speed of autonomous simulator driver.

9.3.2.1.2 Actor Preparation for Ego-flock

As for Ego-flock, the following values were provided directly:

$$\Delta x_R = 30(m) \tag{9.17}$$

$$v_R = 1.05 \cdot v_{DS} \tag{9.18}$$

9.3.2.2 Flock Control in Experiment Three

As the control of Flock is not the concern of this research, formation generation involving the Flock that was used to perform "Flock-blocking" was not dynamical. In this experiment, the Flock was created by Smith as a part of the Recipe for the High-Level Action "Block". Smith was fed with the Flock's ID and controlled its leader. Other members adopted the same Action profile, e.g., the same acceleration rate.

Moreover, simulated vehicles that might interrupt the Flock were instructed to adopt a higher speed in the fast lane and simulated vehicles travelling in parallel with the Flock were instructed to ignore their lane-changing behaviours.

9.3.3 Recipe of “Block” in Smith

There are four pre-defined Assignments in Smith when the Assignment-action of an Assignment is a High-Level Action “Block”:

- Assignment “Create-flock”: when the simulator driver reached the position of 6000 m, a Flock would be requested and thus created by SMM in Sim2. This Assignment was present because a Flock manoeuvre algorithm is lacking. The Assignment-action was used to indicate that a Flock containing three simulated vehicles needed to be created. Their initial positions were determined by the fact that the initial average speed should be less than 1.1×110 kmh, where 110 kmh is the Flock member’s desired speed. This is the rule of being inconspicuous based on the findings from Olstam *et al.* (2011);

- Assignment “Clearing”: when the simulator driver was 2000 m away from the proposed Assignment position (11000 m in the Assignments “Braking-car” and “Flock-blocking”), Smith would start to clear other simulated vehicles that were not members of the Flock or the autonomous simulator driver’s lead vehicle. This Assignment was present in order to avoid any simulated vehicles that could interfere with the coming interaction. The threshold of 2000 m was set based on the results from Olstam *et al.* (2011). The Assignment-action was used to indicate that a clearing process should be invoked. This clearing Action had been implemented in SMM directly and what Smith needed to do was to send an indicator to invoke the whole process without any details. The clearing Action would set other simulated vehicles that were not Ego-vehicles or members of Ego-flocks with a higher (vehicles that were travelling ahead of the Ego-vehicle/flock, 36 m/s) or lower (vehicles that were behind the Ego-vehicle/flock, 30 m/s) desired speed; This desired speed had proven to be adequate as clearing simulated vehicles in front could be within 100 meters in some tests without a sudden change of acceleration rate;

- Assignment "Maintain-speed": this was to maintain the speed of the Flock when the Assignment was triggered. Smith would send the Flock ID and an indicator to make SMM pass the required speed to the Ego-flock and,

- Assignment "Restore": this was to restore the configurations of all simulated vehicles so that they could return to autonomy with the original desired speed. Because there was no need to send parameters, Smith would just send an indicator to SMM and let SMM carry out restoring activities.

The recipe of the top-Action α was constructed as shown in Figure 9.3. In this Figure, the Assignment-actions are represented by their parent Assignments' names, e.g., "Create-flock".

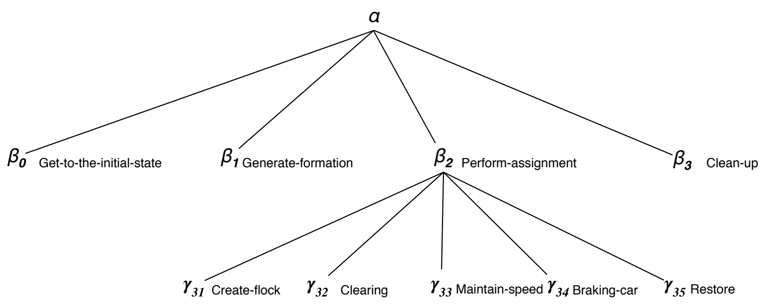


Figure 9.3: Illustration of Recipe for α ("Perform-scenario") in Experiment Three

The precedence constraints were also included:

γ_{31} $IntervalBefore$ γ_{32};

γ_{32} $IntervalBefore$ γ_{34};

γ_{32} $IntervalBefore$ γ_{33};

γ_{33} $IntervalBefore$ γ_{35};

γ_{34} $IntervalBefore$ γ_{35};

By using the recipe and constraints above, the generated General Plan included five Assignment-actions as illustrated in Figure 9.4. Moreover, no metric constraints were specified in this recipe except for the Duration of the Actions γ_{33} and γ_{34}. In this General Plan, Assignment-action γ_{33} and γ_{34} were parallel. A Situation named "Blocking" can be finally generated.

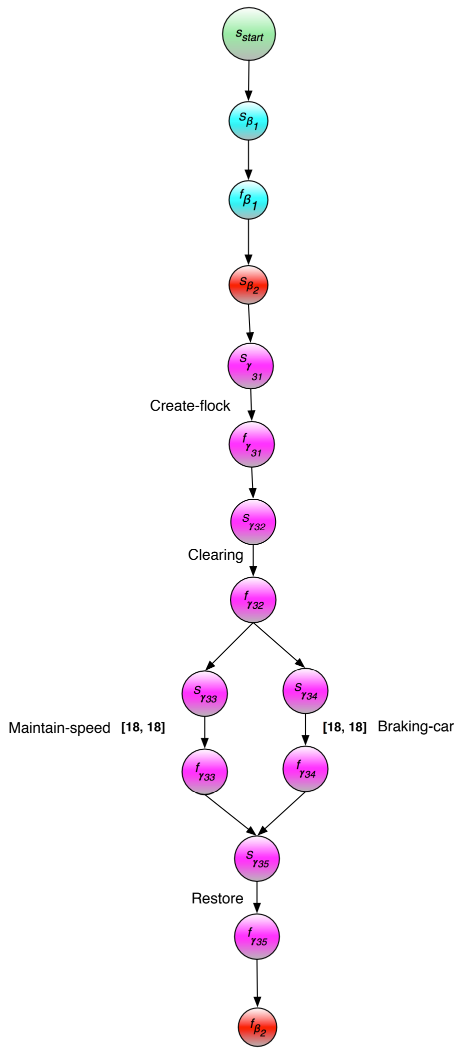


Figure 9.4: Illustration of General Plan Gr_α for "Perform-scenario" (α) in Experiment Three)

9.3.4 Experimental Procedure

Ten tests were performed sequentially without breaks and in each test, the following data were recorded:1) Role Matching commands, 2) the start times of the Assignments “Braking-car”, “Maintain-speed”, “Restore” in Sim2 and 3) positions and speeds of Ego-vehicles and Ego-flocks when orders were executed.

In addition, the Instants when the Sim/SMM received orders were also used in this experiment, so some generals terms, which are start/finish times, were used in this experiment.

9.4 Results

9.4.1 Algorithm Examination

Because no metric constraints were given in this experiment, an overview of the plan evaluation procedure is not covered as the evaluation was always consistent.

9.4.1.1 Role Matching

In each test, Smith needed to look for a leader in order to execute “Braking-car”. In every test, Smith always found a simulated vehicle for the Assignment “Braking-car” as shown in Table 9.2 (As Role Matching for Flock was not covered in this research, Smith was fed with the Flock information directly in the “Flock-blocking” Assignment).

9.4.1.2 Regulating

As Smith was able to find the appropriate lead vehicle, regulating based on lane-changing was not triggered. As for the longitudinal transportation of the Ego-vehicle/flock, which is called speed adaptation in NAUSEA, the results are shown in Table 9.3.

The analysis of the speed adaptation algorithm will not be covered, not only because it is not the concern of this research, but also because the criteria used for speed adaptation, i.e., Formation Position, role distance and role speed are just reference values for Smith. What Smith attempts to do is to regulate his

Table 9.2: Role Matching Statistics of the Assignment “Braking-Car”

Driver Number	Desired Speed	Times of Role Matching for "Braking-car"	Vehicle Found for "Braking-car"
1	105	6	132
2	106	4	132
3	107	5	135
4	108	4	132
5	109	2	136
6	110	10	138
7	111	4	138
8	112	4	138
9	113	4	138
10	114	4	138

Table 9.3: Statistics of Start Time, Position and Speed of Ego-vehicle/flock when “Braking-car” is Triggered

Driver Number	Desired Speed (kmh)	Start Time	Position of Driver	Speed of Driver	Position of Leader Vehicle	Speed of Leader Vehicle	Position of Flock Leader	Speed of Flock Leader
1	105	380.36	11000.2	29.1483	11175.2	29.0527	11050.1	30.5703
2	106	377.97	11000.2	29.4444	11175.2	29.4962	11050.1	30.9711
3	107	375.715	11000.2	29.7222	11175.2	29.7831	11050.1	31.1674
4	108	374.095	11000.2	30	11175.2	29.9046	11050	31.4989
5	109	363.825	11000.3	30.2778	11175.3	30.2148	11050.1	31.7851
6	110	360.665	11000.2	30.5555	11175	30.5465	11050.1	32.0844
7	111	357.62	11000.3	30.8333	11167.9	30.4737	11050.1	32.3096
8	112	354.925	11000.2	31.1111	11167.4	30.7855	11050.1	32.5991
9	113	352.24	11000.2	31.3889	11175.2	31.3885	11050.1	33.0217
10	114	349.885	11000.2	31.6666	11173.1	31.6118	11050	33.2344

Ego-vehicle/flock towards that criteria, but it does not mean that Smith needs to satisfy the criteria without any errors. The Assignment-action can, therefore, be triggered depending on loose conditions, e.g., when the distance headway is less than 200 m, i.e., errors within the conditions in Monitors are all acceptable. Hence, since Assignment "Braking-car" was fired in each test and every driver did not overtake, the Regulating layer was working properly.

9.4.1.3 Situation Assessment

The two major Assignments in Smith's Memory (Recipe) were executed: "Braking-car" and "Maintain-speed". As shown in Table 9.3, "Braking-car" was carried out within the 200 m distance headway defined in the Monitor after 11000 metres of the road segment.

As shown in Table 9.3, the delay of executing the Action concerned a decision loop. For instance, the difference between the actual trigger position (11000.2) and the proposed position (11000) was 0.2 m, the speed of the participant was 29.1483 m/s, so the delay was 0.0069 s. The delay was acceptable as the interpretation needed a decision loop to take effect.

Moreover, "Restore" was triggered 18.005 seconds after "Maintain-speed" and "Braking-car" as is shown in Table 9.4, so the Durations of the Assignment took effect. Another decision loop was needed to send out the order from Smith, so the start time difference between Assignments "Restore" and "Maintain-speed" was 0.005 seconds

9.4.2 General Analysis

The integration of VTI's simulation software with SOAV was successful. The speed adaptation algorithm derived from Olstam *et al.* (2011) along with the information from the Assignments were used to construct the Monitor condition in every test actively and helped NAUSEA to prepare the Ego-vehicle/flock. It ensured that Smith could have a leader, as preparation can last for a long period.

Situation "Blocking" along with the High-Level Action "Block" also worked, but a Flock manoeuvre algorithm should be considered to enhance simulated vehicles in maintaining Flocks.

Table 9.4: Statistics of the Start Times of Assignment “Braking-car”, “Maintain-speed” and “Restore”

Driver Number	Start Time of “Braking-car”	Start Time of “Maintain-speed”	Start Time of “Restore”
1	380.36	380.36	398.365
2	377.97	377.97	395.975
3	375.715	375.715	393.72
4	374.095	374.095	392.1
5	363.825	363.825	381.83
6	360.665	360.665	378.67
7	357.62	357.62	375.625
8	354.925	354.925	372.93
9	352.24	352.24	370.245
10	349.885	349.885	367.89

Smith is able to incorporate other algorithms as demonstrated in the actor preparation algorithm.

9.5 Summary

The integration was successful and the results were promising as Assignments can provide relevant contextual information for Regulating, and SOAV also worked properly with new integration. However, it should be noted that the application of Regulating could be limited because the estimation of the start time of Play was not reliable as it was based on real-time participant’s speeds. In this case the Ego-vehicle may miss the participants’ vehicle and cause failure.

Further research is also needed in Flock control in order to manage the members of a Flock with the following behaviours: 1) merging into the Flock; 2) cutting-in to the Flock and 3) leaving the Flock. Some example algorithms can be found in the CyberCar2 project[1]. Those Flock-related algorithms can be adopted by each simulated vehicle and be dictated or requested by Smith.

Moreover, single-threaded Smith showed that the delays for executing orders can be reduced to relatively constant values compared to the ones obtained in the previous two experiments, so single-threaded Smith should be used until multi-threaded is needed.

[1]http://cybercars2.paris-rocquencourt.inria.fr

Chapter 10

Evaluation of OSO

The Oracle: *We're all here to do what we're all here to do. I'm interested in one thing, Neo, the future. And believe me, I know: the only way to get there is together.*

- *The Matrix Reloaded*

10.1 Introduction

In order to model context for scenario orchestration, this research used an ontology for three reasons, 1) ontologies have been used in context modelling for DAS already as mentioned in Chapter 2, which makes it possible to merge other DAS-oriented ontologies with OSO or use OSO directly for DAS; 2) ontologies are programming-independent and logic based and 3) regular human-oriented knowledge repositories as discussed in Section 2.4 on Page 31 cannot be used for machine processing.

Furthermore, OSO has been designed based on findings from experts in the area of driving simulation, so it has been developed according to not only a "practice" methodology, but also a "philosophy" methodology, which can be regarded as stepwise and modelling methodologies respectively according to Jarrar (2005). Therefore, the development of OSO complies with the On-To-Knowledge methodology (Sure, 2003): feasibility study, kickoff, refinement, evaluation and application & evolution. For instance, from the three verification experiments, classes are being added (e.g., $MatchRole$, $Play$) or deleted (e.g., $TimePoint$ that is deleted because of the adoption of *Instant*) according to specific requirements. Moreover, the development of OSO also complies with the OntoClean (Guarino & Welty, 2002) methodology by identifying the meaning of the properties, classes and their relations in OSO.

OSO is still in an early phase in standardising formal scenario descriptions. Instead of focusing on the normativeness, this research focused on its expressiveness by using it to describe scenarios for experiments by using both a practice-oriented and a logic-based methodologies. However, because OSO was developed with a focus on scenario orchestration, i.e., representing scenarios with essential contextual information for virtual drivers, evaluation is therefore based on application and has two aspects: 1) identify its role in scenario orchestration and demonstrate its expressiveness in experiments and 2) identify its advantages and disadvantages for future development.

10.2 Role of OSO

In SOAV, OSO can provide the information regarding the context of required interactions. This is achieved by using the information encapsulated in Assignments, which is used by Smith to generate interactions. The following information has been included in Assignments:

1) the proposed formation of simulated vehicles around the participant's vehicle when the Assignment is triggered and interaction is generated;

2) the proposed condition for triggering the Assignment, i.e., the Monitor for that Assignment;

3) the proposed Success Conditions of the Assignment, i.e., what should be true if the Assignment is performed successfully;

4) the proposed Failure Conditions of the Assignment, i.e., what can be true if the Assignment is failed;

5) the proposed measurement of the Assignment, which is included by relating an individual of *Assignment* to some individual of *Measure*. Experimenters can

have several candidate Assignments that have the same measurement output but with different Failure Conditions.

In addition, Success Conditions/Failure Conditions can be used to indicate the participant's potential reactions. Success Conditions can indicate the ones that are anticipated, while the other can indicate the ones that are not desired, e.g., the Failure Condition of "the Ego-vehicle's leader changed to participant's vehicle" may imply that the participant may choose to overtake during the Assignment, which could fail the Assignment.

As a result, with the simulated vehicles required by the Assignments, temporal constraints, Triggers and proposed measurements, contextual information regarding potential interactions can be obtained. In addition, an Assignment can be selected from several Assignments that have the same measurement and similar contexts, however, the Assignment with fewest Failure Conditions could be chosen first.

For instance, the Assignment "Overtaking-lorry" in experiment two on Page 151 may have Failure Conditions of "the Ego-vehicle reaches the road segment for next Assignment", which suggests that the participant may miss the lorry/truck. In order to avoid this situation, the simulator users can be notified that some Assignments have the same measurement output, which can provide suggestions for particular Assignments' alternatives. As an example, Assignment of "Overtaking-lorry" *hasMeasurement some DriverReactionTime* and Assignment of "Braking-car" also *hasMeasurement some DriverRectionTime*. Both Assignments are sub-classes of an anonymous class that relates to some individual of *DriverRectionTime* via an object property *hasMeasurement*, which means that they may be exchangeable. However, the comparison and validation should be undertaken before writing the similarity into OSO.

Moreover, as part of standardization, limitations of simulation software should be modelled in OSO in order to be able to notify simulator users of these. This could be a long-term process and was not covered fully in this research as limitations can vary a lot among different platforms, but OSO can be extended to adopt concepts and properties regarding limitations. For instance, a sub-class of *SimLimitation* named *IgnoredObject* was created to indicate all invisible objects in the Sim, which can, of course, include cones (*Cone*, a subclass of *IgnoredObject*).

10.3 Evaluation of OSO

As demonstrated in the three experiments and the analysis in Section 10.2, OSO is expressive enough in describing scenarios. Its capacity was also demonstrated in providing 1) driving context for interactions; 2) participant's potential reactions in some interactions and 3) description regarding the capabilities of the simulation software, especially its limitations. Moreover, OSO has been checked by the HermiT reasoner (Section 3.5.1.1) in Protégé and is consistent.

In addition, OSO has been evaluated based on some context modelling and ontology engineering criteria inspired by Krummenacher & Strang (2007), in order to see OSO's advantages and disadvantages.

- Applicability: OSO can be used in areas that need to model driving context, so it is not only limited to scenario orchestration in driving simulation;
- Comparability: in OSO, only SI units can be used and are assumed by default. Information regarding driving context are regarded to be world-wide comparable, e.g., vehicle's states. However, the information regarding Assignment and related concepts cannot, which are virtual concepts developed in this research;
- Traceability: in OSO, all interpretations regarding sensory data processing, which is focused on simulated vision, are based on quantitative numbers without mapping components, e.g., speed of a vehicle. This is helpful when OSO is open to other researchers;
- History, logging: in OSO, temporal concepts are included for history logging in Smith, so History and logging are supported in OSO by specifying corresponding timestamps, although it is not currently being used;
- Quality: in SOAV, every state variable in the Sim can be observed without ambiguity, so quality of information is not modelled in OSO;

- Satisfiability: data types are not restricted and any reasoning regarding the satisfiability is done in Smith;
- Inference: in order to be descriptive, OSO has been designed to guarantee expressiveness, so the ability of inference has not been covered. It is done by Smith at present.

OSO has been also compared to some ontology engineering criteria:

- Re-usability, standardization: OSO, as a model for scenario orchestration in driving simulation, can be also used when driving context or Virtual Driver's tasks need to be defined and represented;
- Flexibility, extensibility: new concepts can be added into OSO with relevant classes and properties, without the need of changing the existing model in OSO. For instance, in order to indicate when the Role Matching should be started, the concept *MatchRole*, which has been modelled as a subclass of *Action*, was adopted to represent the Action of Role Matching so that relevant Monitors can be associated with this Action;
- Granularity: OSO is built upon concepts identified in this research as discussed in Section 3.3 on Page 42. As a result, OSO has been developed to reflect the hierarchical architecture of NAUSEA. It was also developed to reflect the hierarchical driver model SAIL. Further refinement can be undertaken, e.g., refinement of High-Level Actions;
- Completeness: with all three experiments, it is clear that OSO is able to describe common scenarios in driving simulation. Moreover, OSO is extensible if new concepts are found or it needs to be modified;
- Redundancy: redundant information was included in order to reflect the development history of OSO and could be deleted without any problems. However, after the third experiment, all redundant information has been deleted in order to avoid inconsistency in the future;

- Readability: vocabularies and naming conventions being used in OSO are based on the understanding of relevant concepts, which are extracted from personal communication with relevant experts or documents authored by them with the intention of being understandable to humans, so OSO is designed to be comprehensible to humans;
- Language, formalism: RDF/OWL is used to model OSO and can be changed in the future. As a model focusing on concepts and their relationships, OSO can be recreated or represented by other ontology engineering tools or languages.

10.4 Summary

In this chapter, what OSO can provide and what its advantages and disadvantages are have been presented. In general, as an early attempt of modelling driving context along with scenario requirements in a programming language-independent and formal manner, OSO has achieved its design goals. Further development making OSO more widely applicable through the adoption of concepts from other platforms in the driving simulation community would be desirable.

Chapter 11

Conclusion

The Architect: *The first Matrix I designed was quite naturally perfect, it was a work of art, flawless, sublime. A triumph equaled only by its monumental failure. The inevitability of its doom is apparent to me now as a consequence of the imperfection inherent in every human being...*
-The Matrix Reloaded

11.1 Introduction

This chapter serves the purpose of summarising this thesis by providing some discussions regarding 1) achievements and drawbacks of this research through the examinations of fulfilment of each objective; 2) contributions of this research summarised from the presented achievements; 3) future research or enhancements inspired by the drawbacks of this research and 4) some conclusions of this research with a short summary of the thesis.

11.2 Thesis Summary

Scenario orchestration in driving simulation is a difficult problem not only because both realism and repeatability are needed, but also because scenarios are not just computer animations. Therefore, scenario orchestration is liable to pre-run issues

(Section 2.2.3 on Page 2.2.3) due to the complexity of designing scenarios and a lack of Knowledge Bases for both humans and machines. It is also liable to run-time issues (Section 2.2.3 on Page 2.2.3) due to the dynamic features of driving simulation and a lack of mechanisms for generating interactions actively. However, attempts have been made in this research to deal with these issues by achieving some objectives listed in Section 3.2 on Page 42:

11.2.1 First Objective - Encoding Contextual Information

Assignments were used to achieve the first objective. They were developed to encode the contextual information regarding interactions in scenarios in order to answer the following questions: 1) which simulated vehicle should be involved in the coming interaction? (Formation Position and vehicle restriction) 2) where should the simulated vehicles be driven to? (Formation Position) 3) what should be undertaken in order to generate the interactions (Assignment-action)? 4) when should the interactions be generated, succeed or failed? (Monitor, Success Conditions and Failure Conditions) and 5) what measurements should be collected? (Measurement)

This is motivated by the fact that programming languages are always used to define the Action sequence simulated vehicles should follow in a scenario, and they focus on what Actions should be performed and when to perform those Actions (see e.g., Leitao *et al.* 1999), not the relevant contexts for each Action.

As shown in the three verification experiments, by providing relevant context, Assignments can be used not only by a Virtual Driver to generate required interactions with the capability of re-generating those interactions, but also by experimenters to describe interaction context along with outcomes (measurements or Actions needed). They can also predict the reactions from participants by providing Success/Failure Conditions, so the pre-run issues can be dealt with by utilising Assignments to encode and notify 1) the interaction outcomes along with corresponding context and 2) the potential reactions from participants.

Although Assignments have covered some essential context for interaction generation, more work should be done to include more contextual information in

Assignments, e.g., disjunctive Triggers as mentioned in Section 4.3.9 and contextual information regarding Flocks:

1) as shown in the three verification experiments, multiple Triggers, especially multiple Monitors, were assumed to be satisfied at the same time, so further research should be done regarding disjunctive Triggers when needed.

2) flock-related algorithms are lacking in this research and thus in the three experiments, Assignments do not carry specific contextual information regarding Flocks, e.g., required leader of a Flock or required Flock that an Ego-vehicle should belong to. More work should be done to adopt contextual information regarding Flocks so that Flock-related algorithms can utilize relevant information.

11.2.2 Second Objective - Knowledge Base

The Ontology for Scenario Orchestration (OSO) was used to achieve the second objective. It is the medium used to record Assignments and relevant contexts.

OSO was proposed in order to record the Assignment, as programming languages, e.g., SSL (Wolffelaar *et al.*, 1999), always record platform-dependent details together with scenario information. They are not suitable for contextual modelling and scenario sharing in the future.

As a programming language-independent and logic-based scenario representation mechanism, OSO is human-readable as it is a collection of concepts and their relationships in the domain of scenario orchestration in driving simulation; it is also machine-processable as it can be recorded in an XML file for machine processing.

As evaluated with some context modelling and ontology engineering criteria, OSO has some advantages, e.g., readable, extensible, applicable to other areas, re-usable and flexible (details can be found in Chapter 10). As demonstrated in the three verification experiments, OSO can be fed to and thus processed by the Virtual Driver Smith. It is also expressive enough for scenario description by modelling Assignments and related contexts.

As shown in Appendix C on Page 239, OSO is human-readable but whether or not it is user-friendly has not been examined and how to present OSO to different audiences, e.g., scenario designers, simulator developers, needs further investigation. OSO users may need some time to become familiar with the concepts and relationships. Moreover, it has not been fully tested regarding its normativeness and its expressiveness with other possible interactions. To sum up, further evaluation will be needed when different requirements are presented, e.g., its re-usability in other areas rather than driving-oriented ones.

11.2.3 Third Objective - Plan and Replan

An algorithm named NAUSEA (autoNomous locAl manoeUvre and Scenario orchEstration based on automated action plAnning) was generated to utilise Assignments in order to achieve the third objective. With corresponding supporting facilities, i.e., SAIL, Sim(SMM) and OSO, and the contextual information encoded in Assignments, NAUSEA can be used to generate interactions.

NAUSEA was proposed to deal with the run-time issues in two directions: 1) the simulated vehicles can actively engage the participant to avoid failure and 2) if failures happen, the simulated vehicles can be re-driven to generate the proposed interactions.

As tested in the three verification experiments, the Virtual Driver Smith, equipped with SAIL/NAUSEA, can recruit simulated vehicles dynamically by using Role Matching procedure and prepare for interactions actively by using Regulating procedure. Failed interactions, generated by corresponding Assignments, can be regenerated through the ability of replanning. However, more work may be needed in order to cover the following aspects that were not fully examined in this research, especially regarding some procedures in NAUSEA, i.e., Regulating, Role Matching, Plan Evaluation:

1) As shown in experiment one and two, a Regulating algorithm is lacking regarding speed adaptation and lane-changing manoeuvres. NAUSEA, therefore, has been designed to handle local manoeuvres that cover less than 200 metres around the participant's vehicle (six seconds time-headway as in Assignment "Overtaking-lorry" in experiment two or 200 metres distance headway as in

Assignment “Braking-car” in experiment three). By providing some suitable Regulating algorithm, not only can the Formation Positions be defined with absolute locations, but Regulating (Action β_1) and replanning after failure can be enhanced also. In verification experiment three, a speed adaptation algorithm was adopted but it is still problematic because 1) the algorithm did not consider the lane changing manoeuvre and 2) the basis for preparation, i.e., prediction of participant’s speed, considers limited factors (see Olstam *et al.* 2008). Moreover, Regulating of Ego-flock was not handled because the Flock manoeuvre algorithm is not implemented or developed in this research. Moreover, Regulating with vehicle-restriction also needs further research as suggested by experiment two;

2) Role Matching was not full examined in this research, as it focused on finding simulated vehicles for the coming Assignment(s) as demonstrated in the three experiments;

3) the Assignments were pre-defined and pre-scheduled, so the plan evaluation procedure was not fully examined. Although it is sufficient for the three experiments, future research regarding plan evaluation could still be suggested by the driving simulation community or some future applications in order to utilise the spatial and temporal constraints for scenario orchestration.

4) as shown in experiment one, NAUSEA cannot plan in the Operational layer to make Smith handle operational action as well when preparing Ego-vehicle, i.e., Regulating. For instance, lane changing trajectory was generated by the Sim or specifically, by the autonomous simulated vehicles in this research. More work would be done regarding this issue, however, this may increase the network workload as Smith will need to send relevant parameters in every update step;

5) as suggested by experiment two, how Assignment-actions should be performed, i.e., what particular Action profiles (e.g., acceleration rate) should be adopted, have been ignored due to the focus of this research and time schedule. Moreover, in order to make the interactions obvious for observation, the Action profiles used in the three experiments have been designed with values that can

serve this purpose. However, because the profiles may affect how participants perceive the scenario, what Action profiles to use in a particular Assignment should be considered in the future and the standardization of scenarios based on Assignments should also consider the standardization of Action profiles;

6) how to design the conditions of Triggers, including the Monitors, should also be examined in the future as suggested by experiment two, in order to standardize the conditions for executing/finishing/failing Assignments and examine complex scenarios with a series of conditions, e.g., scenarios with multi-lane road networks;

7) as shown in the three experiments, SOAV has not been either tested in terms of its potential influence on traffic flow, or equipped with relevant modules to handle traffic flow dynamically, which could be a problem when using SOAV for human behaviour research.

11.3 Contributions of this Research

Based on the achievements discussed in the previous section, contributions of this research can be identified by considering two aspects: decision making and the knowledge base. In this section, the main contribution, which is SAIL/NAUSEA used for decision making will be discussed first, followed by the contribution regarding knowledge base - OSO.

11.3.1 SAIL/NAUSEA

Using pre-scheduled Action plans for scenario orchestration is actually a good solution for driving simulation as they can guarantee repeatability, that's why they have been used since as early as 1990s. However, because those plans are crafted by humans and fed to the simulated vehicles passively, the behaviours of simulated vehicles cannot be interrupted, so Run-Time Issues may arise as mentioned in Section 2.2.3 on Page 21:

- participants do not want to be engaged in some interactions and,

- some interactions never happen due to design or system issues.

If the simulated vehicles, or precisely, the controllers of those vehicles, know what should be done in a scenario, the vehicles can try to 1) prepare actively and engage the participant and 2) re-generate their Action plans after failure according to proposed interactions and dynamic driving contexts.

After the identification of the concept “Assignment”, a way of guiding the simulated vehicles was found: contextual information and corresponding algorithms.

The mechanism of scenario orchestration in this research was developed based on a Hierarchical Task Network (HTN) with additional before/after and temporal constraints in order to handle Assignments. The Assignment performer, which is the Virtual Driver Smith, needs to finish the Assignments by finding appropriate Ego-vehicle/flock, regulating them to proposed Formation Positions and carrying out Assignments. Hence, NAUSEA, which is short for autoNomous locAl manoeUvre and Scenario orchEstration based on automated action plAnning, was proposed. A driver model named SAIL, which is short for Scenario-Aware drIver modeL, was developed to support NAUSEA. As a result, NAUSEA is in charge of the Decision-making layer in SAIL. Other layers in SAIL provide NAUSEA with a World Model that contains relevant driving context for decision making and an Action layer that is used to send out orders.

Simulated vehicles can then be driven by Virtual Drivers equipped with SAIL/-NAUSEA and used to orchestrate scenarios by considering Assignments and temporal constraints. Finally, failures can be dealt with by replanning Actions based on the Assignments. In this case interactions can be actively prepared in order to engage the participants in some interactions and if something went wrong during the scenario because of, e.g., unintentional reactions from the participants or Monitor fault, interactions can be retried by Smith with the dynamic Role Matching capability and the Action Plan - the General Plan Gr_{α}.

11.3.2 OSO

Scenario orchestration in driving simulation has gained a great deal of attention since the early 1990s. More recently it has received less attention in the scientific community as existing methodologies appear enough for present experiments. Generally speaking, the characteristics of those methodologies are:

1) scenarios have been described in natural language in many papers, articles and reports. However, when it comes to the knowledge sharing of scenarios, this can be time-consuming as a standardized scheme is lacking;

2) when implementing scenarios, those descriptions based on plain language are always transferred to programming languages, which always include information about the action sequence only. Context for interactions can be hard to describe as those descriptions concentrate on the required Action sequences for interactions.

These two aspects can be the cause of the pre-run issues mentioned in Section 2.2.3 on Page 21:

- the experimenter describes interaction outcomes without corresponding context and,
- the experimenter predicts the wrong reactions from participants.

As a result, establishing a standardized, programming language-independent and logic-based formal Knowledge Base may not only be able to deal with those pre-run issues but also be used as the data source for the algorithm developed in this research. It can concentrate on the knowledge regarding contexts of the interactions and thus can be a communication protocol between different driving simulator platforms so that knowledge can be shared worldwide in an easy manner, inspiring each other from time to time.

Another contribution of this research is, therefore, the development of a formal Knowledge Base for scenario orchestration in driving simulation - Ontology for Scenario Orchestration (OSO). OSO places its focus on the major information in a scenario and ignores the implementation details of a specific simulator platform. This major information should include: what should be done and when it should be done. As a result, some concepts have been included in OSO. For instance:

- in scenarios, simulated vehicles always need to perform some tasks, so those tasks have been called as “Assignments”. These include all the necessary context for generating interactions, e.g., when to trigger an Action, which simulated vehicle should be involved, etc. The performers of Assignments are Virtual Drivers and,

- in scenarios, simulated vehicles always need to consider some conditions of triggering those tasks, so “Triggers” have been adopted by the Assignments. Triggers can also provide information such as the potential reactions of the participants.

Those concepts have been modelled in an Ontology for Scenario Orchestration (OSO) along with their relationships. Scenarios were thereafter described based on OSO, which can provide the Virtual Drivers with driving context and scenario requirements (i.e., Assignments). As an abstract model of scenarios in driving simulation, OSO can be also used to standardise scenario definitions and driving context. It can be shared among different platforms. Scenarios can therefore easily be maintained and reused based on this standardised scenario description.

11.3.3 Cross-platform Standardization

Based on the aforementioned contributions, the third contribution of the thesis is a mechanism of cross-platform standardization of scenario descriptions and implemetnation. As demonstrated in experiment three in Chapter , when two simulator teams need to work together with the compatibility of their scenario(s), what they have to do will involve five steps, in which case, the Sim will refer to two different simulation platforms:

1. examine the Ontology for Scenario Orchestration (OSO) and identify the state variables in the Sim. Those variables are what can be observed in the Sim by Smith, e.g., how to represent roads, lanes and vehicle’s speeds, they should be standardised between the two platforms first by identifying and merging relevant concepts in OSO;

2. modify the raw data published by the Sim according to the new state variable list in OSO; the raw data should include the state variables identified in the first step and look similar to the ones described in the Tables 5.1 and 5.2 in the Section 5.2.1;

3. examine the Actions needed to modify the state of objects in the Sim, especially the behaviours of simulated vehicle; the Actions can be similar to the ones in Table 6.2 on Page 130. OSO can be updated to reflect new Actions needed, e.g., "place cones on a lane", which can be a class named *setmodelswitch_3_2* as in experiment two or a new class named *PutObject* for standardized and this class can be related with a class named *Cone* with a property named *hasPutObject*;

4. add a Scenario Management Module (SMM) into each Sim to interpret relevant orders that reflect Actions identified in the previous step. An example module can be a c++ method that finds a simulated vehicle with the required ID in the Action and execute a lane-changing Action by modifying the desired lane of that vehicle, which can implement the Action *GotoLN* in OSO;

5. describe the interactions by using the concept of Assignment. This will involve the potential Action required and corresponding context: Triggers and Formation Positions. Examples can be found in Appendix C

That is, the two teams need to 1) have the same OSO for exchanging Assignments and 2) have the same interfaces for publishing raw data of vehicles' states and interpreting Smith's Orders, i.e., executing required Assignment-actions.

In order to standardize more detailed information, the two teams can also enrich OSO with the behavioural models in the Sim by identifying the model used and relevant values for parameters. For instance, team A can propose a new class in OSO to represent the car-following model being used by simulated vehicles and this new class will be related to several numbers to represent, e.g., the maximum acceleration rate the model should use.

11.4 Future Research

The drawbacks discussed in Section 11.2 have not been handled in this research, but they can be dealt with in the future by carrying out tests using "real" scenarios from previous projects. This verification process can include a combination of development and verification. The former can suggest which procedure in NAUSEA or component in SOAV should be enhanced and how; the latter can examine the validation of the framework, which can be performed by comparing the data collected with the existing simulation software against the data obtained from SOAV. SOAV's potential influence on human behaviour can also be examined during this verification phase.

In this section, some research directions will be introduced in order to deal with the main drawbacks mentioned above: a suitable simulation platform for future research, Regulating, traffic flow manipulation, Role Matching and Regulating based on more than one Assignment and OSO development.

11.4.1 Environment, Vehicles and Interfaces for Controlling

The first research direction relates to the basis of future research, which focuses on providing a suitable simulation platform for testing SOAV.

Environmental representation in driving simulation has been focused on paths that a simulated vehicle or a pedestrian could follow. For instance, in Willemsen (2000), EDF (Environment Description Framework) was developed to represent complex paths on roads and at junctions. In Lacroix *et al.* (2007), in order to fix some graphical glitches as illustrated in Figure 11.1, a data description format named RND (Road Network Description) was developed to make the road surface continuous. However, less attention has been paid to the descriptions of other objects in the simulation such as cones, trees, etc.

The lack of object representation can affect the decision making of simulated vehicles and Smith, as they will not be aware of those objects. That is why some workarounds are needed to inform them what should be done. For instance, an obstacle may need to be placed in front of a coned lane to indicate the lane's

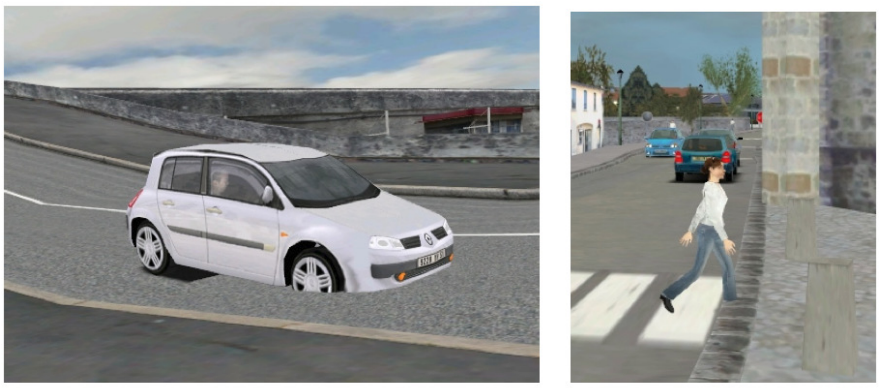


Figure 11.1: Graphical Glitches from Lacroix *et al.* (2007)

unavailability. However, after simulated vehicles have passed that obstacle, new workarounds will be needed to prevent them from returning to the coned lane. This will not be a problem in applications, but could bring trouble to the standardization of scenario descriptions and implementation of SOAV. Moreover, the decision-making of simulated vehicles that Smith will need to drive can be problematic. For instance, in experiment two, simulated vehicles may brake just in front of the obstacle.

Apart from the problems raised by object representation and decision making of simulated vehicles, how a simulated vehicle can be designed should also be considered in future research. Basically, some compromise should be made regarding 1) the underlying environment representation scheme being used by the simulated vehicles and the simulation software; 2) available Actions for controlling those vehicles, i.e., interfaces for controlling simulated vehicles; 3) available state variables for Triggers; 4) available Action profiles that can be used, e.g., acceleration rate; 5) available tactical behaviours and corresponding configurations, e.g., the gap choice for overtaking and 6) the realism of operational behaviours, e.g., lane-changing trajectory.

Furthermore, the literature was focused on designing realistic simulated vehicles (e.g., Cremer *et al.* 1995), but attempts to develop a standardized autonomous simulated vehicle for driving simulation have not been made yet. Standardising behaviours and corresponding interfaces for controlling should be the initial attempt.

If an Open Source version of autonomous simulated vehicles, environment representation scheme and related interfaces can be established, then an Open Source SOAV can be provided to be the standard version to be tested and enhanced. It can also be used to share knowledge in the driving simulation community.

11.4.2 Regulating - Action β_1

11.4.2.1 Participant Behaviour Prediction

Predicting what the participant or other simulated vehicles may do should be examined in order to adopt feasible Regulating algorithms. Only with these predictions can Smith plan more accurately in driving simulation, e.g., what their desired speeds and desired lanes could be? How possible it is that they will adopt new speeds or new lanes?

In Olstam *et al.* (2008), participants' speeds have been predicted based on road conditions (e.g., road width), average speed of traffic flow, etc. but more should be done. For instance, let us consider a situation called "Discovered Check" from Sukthankar (1997) (illustrated in Figure 11.2):

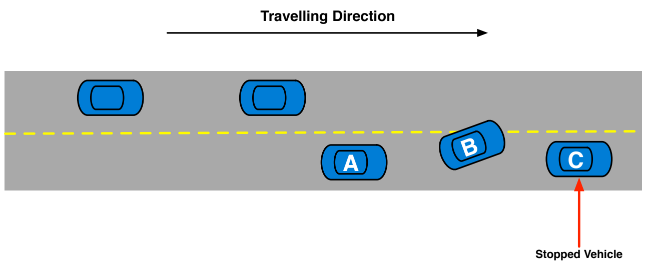


Figure 11.2: Illustration of "Discovered Check" in a 2-lane one way road (Re-illustrated from Sukthankar (1997))

In this situation, not only speed limit, road conditions and average speed of traffic flow, but also some individual entity in the dynamic traffic flow is needed to predict a vehicle's speed, including the participant's one.

In Figure 11.2, vehicle A is travelling at 30 m/s and following vehicle B. Vehicle C stops for some reason so B changes its lane to avoid a collision. However, A is unable to change lane because of the traffic flow on its left and A is also unable to decelerate and avoid vehicle C.

If A is the participant's vehicle, in order to predict its speed, the traffic flow near A should be examined, so predicting A's speed will involve examining C's state as soon as C's speed changes. Hence, this situation should be anticipated by Smith beforehand even if A has not yet noticed C, in which case Smith needs to lower the estimated driving speed of A, even if A is still travelling at a higher speed and approaching B and C.

Participants' lane choice in scenarios can be predicted based on lane choice models from, e.g., Toledo *et al.* (2005), and the preferred lane obtained from those models can be used to predict which lane the participant may choose with the corresponding probabilities.

Future research can, therefore, put a focus on the behaviour prediction of participants and consider their potential lane choices along with speed trajectories. By predicting participants' behaviours, Smith is able to compensate in advance in order to make sure that Assignments will be carried out successfully in a realistic manner, without failures or retries.

In addition, if the participant changed his/her speed or lane suddenly, Smith should not consider the new speed or lane as the preparation goal. For instance, if the participant chose to brake in the situation in Figure 11.2, the Ego-vehicle/flock should not brake, but keep travelling with a speed that will not catch the participant's attention. In this case, Smith should report failure and start to prepare again with a new Role Matching procedure.

11.4.2.2 Preparing the Ego-flock

When it comes to Flocks, two situations can occur as illustrated in Figures 11.3 and 11.4:

In this research, the first situation in Figure 11.3 was used in experiment three (Page 195). Smith needed to handle the leader of the Flock F1. Other members will follow F1 with the same Action profiles specified by Smith, e.g., acceleration rate, so Smith is actually controlling just one vehicle. However, the situation specified in Figure 11.4 can be more common in a "realistic" situation in which

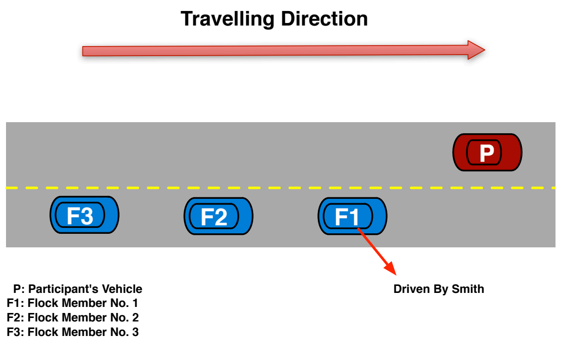


Figure 11.3: Situation One of Flock Control

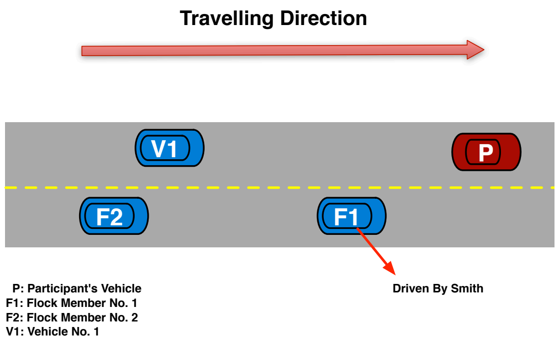


Figure 11.4: Situation Two of Flock Control

no assumption has been made regarding how the simulated vehicles could evolve, unless Smith needs to prepare them for some Assignment. There will be two questions: 1) what should be done if situation two is needed? and 2) what should be done if situation one needs to be generated from situation two.

The first question can be answered by adopting several Smiths in parallel and allowing them to handle every simulated vehicle and Flock, so several Formation Positions around the participant's vehicle can be occupied by different Ego-vehicles or Ego-flocks. However, in this case, the coordination between Smith(s) needs further examination.

The second question can be answered by adopting just one Smith for the Flock, but autonomous simulated vehicles need Flock-related algorithms to handle such situations. Merging into a Flock (Figure 11.5) is therefore possible when no junctions are present, so that members of Flocks can be dynamically recruited. As a result, Assignment one in experiment two (Page 151) can be further supported in a more standardized manner by adopting a Smith to control the Flock containing the two lorries. The Assignment-action will just be specified to control the second member of the Flock to overtake, no matter which vehicle will be the second one.

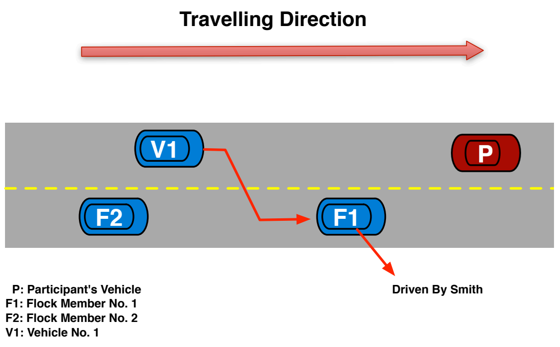


Figure 11.5: Flock Merging in Situation Two

11.4.3 Traffic Flow Manipulation

Assignments can provide contextual information for proposed interactions. Smith can regulate relevant vehicles around the participant's vehicle to prepare and generate such interactions. An assumption has been made in this research that the traffic flow is sufficient for selecting Ego-vehicle/flock and generating ambient traffic flows. However, further research is needed to explore the possibility of manipulating traffic flow based on not only required traffic flow rate, but also the proposed interactions.

For instance, a traffic flow with a rate of 1200 vehicles/h/direction and 92% cars, 6% trucks and 2% buses (Olstam *et al.*, 2011) may be needed in a "Blocking" Situation, which means one vehicle is needed for braking event, three vehicles are needed for "Block" Action. The traffic flow manipulator in SOAV should consider the contextual information from the Assignments, including 1) the required Formation Positions ("Leader" for the lead vehicle and "LeftParallel" or "RightParallel" for the Flock) around the participant's vehicle, 2) required TTC (e.g., TTC should be infinite), 3) required distance headway (e.g., 200 m for the lead vehicle) and 4) required trigger position of Assignment (e.g., 11000 m down the road segment for "Braking-car"). The traffic flow, including the four vehicles, should evolve gradually to clear spaces or preparing for "Blocking". In this case, SOAV's influence on the traffic flow rate/distribution and corresponding data validity should be examined.

11.4.4 Global Optimization vs Local Optimization

Before Smith can carry out any Assignments, he needs to find out which simulated vehicle he should control by using Role Matching, after which he needs to drive the Ego-vehicle/flock to the proposed Formation Position. As elaborated in previous chapters, Smith makes decisions based on the first Assignment(s), which means, he optimizes his choices in Ego-vehicle/flock and corresponding preparation trajectories based on just one Assignment involving the SMM, one Ego-vehicle or one Ego-flock, so he performs local optimization. However, can he perform global optimization that involves more than one vehicle or Assignment?

11.4.4.1 Role Matching

If an Assignment involves several simulated vehicles, Smith should try to find Ego-vehicles based on global criteria so that the total time spent to drive them to several Formation Positions can be minimized.

In Zu *et al.* (2004), a multi-vehicle multi-target pursuit problem was studied and a Global Cost Function(GCF) was proposed to indicate the time spent by one vehicle to pursue one target (they form one vehicle-target-pair). In order to find the optimized vehicle-target-pairs, this GCF was used to minimize the maximum pursuit time spent by all the vehicle-target-pairs instead of just one vehicle-target-pair. Inspired by this paper, a step could be added into Role Matching to evaluate the quality of Role Matching results based on global criteria. By doing so, Smith will try to minimize the maximum time spent for actor preparation from the very beginning.

11.4.4.2 Regulating

When Smith has several Assignments to perform, he prepares for the successive Assignment. However, let us assume that there are two Assignments in sequence: 1) an Ego-vehicle should brake in front of the participant's vehicle and 2) an Ego-vehicle should push the participant's vehicle as a follower. Moreover, the delay between those two Assignments should be less than one minute and of course, some Monitor(s) should be assigned to both Assignments. At present, Smith will find a leader, finish Assignment one, and then he will find a follower, finish Assignment two according to Monitor(s). If Smith can find a follower and assign that follower an appropriate speed behind the participant during Assignment one, then immediately after that Assignment, he will have at least one candidate follower in the proposed Formation Position (if another simulated vehicle becomes the candidate's leader, this new simulated vehicle will be assigned as Ego-vehicle in Assignment two, which is why Smith has at least one candidate.).

As a result, instead of considering the actor preparation immediately prior to each Assignment, Smith can choose to prepare all or some of the Assignments rather than the coming one. By doing so, the time spent in the actor preparation phase, i.e., Regulating, can be shortened.

11.4.5 OSO Development

In this research, OSO has been mainly developed to handle interactions involving an Ego-vehicle and be as descriptive as possible, so OSO has not been fully examined:

- Flock-related concepts and negative assertions:

  OSO's expressiveness can be further investigated, especially regarding the following two aspects:

  First, Flock members were not dynamically recruited in this research so Flock-related concepts were not fully specified in OSO, e.g., how to represent a specific member of an Ego-flock and how to represent the property of an Ego-flock. Those concepts can be added into OSO after the Flock-related algorithms mentioned in Section 11.4.2.2 have been developed and more tests have been undertaken.

Second, all properties are positive in OSO, indicating the characteristics that a particular individual should possess. Negations were not examined and specified. However, negations can be investigated in the future to act as extra Failure Conditions or situations that Smith should avoid. For instance, negations can be used to indicate that the Ego-vehicle should not reach the road segment whose ID is “r4.0”.

- Offline scenario verification that includes:
  - inconsistency checking regarding classes that have no individuals;
  - subsumption checking in order to classify individuals, e.g., generate a group of Smiths that have the same Virtual Driver personalities;
  - inconsistency checking regarding the scenario information. This concentrates on profile reasoning, e.g., there is a road that is 20 miles long; the participant should drive at a speed of 20 mph and the whole scenario should last for a total of 2 hours. In this case the reasoning mechanism should notice that this scenario is inconsistent or impossible to generate. In fact, this can be implemented by the inconsistency checking regarding classes that have no individuals.

  The first two listed items have been covered by existing OWL-DL reasoner HermiT. However, the last item has not yet been covered, so when scenario designers are generating scenario descriptions using OSO, there is no possibility that inconsistency caused by scenario profiles can be found before the experiment commences.

  This feature is desirable in order to aid the scenario designers and bring offline verification before each experiment, but further examination regarding OSO’s normativeness may be needed.

- GUI development based on OSO:

A GUI (Graphical User Interface) can be developed based on OSO to provide visual aid in designing scenarios and test OSO's expressiveness with humans in a user-friendly manner.

To sum up, OSO can be a start point in bringing a formal, standardized and thus shareable Knowledge Base for the driving simulation community. However, further development may be needed to examine OSO's normativeness, expressiveness and potential usage in a user-friendly manner.

11.5 Summary

This research used "The Matrix" metaphor to design an intelligent Virtual Driver Smith. An Ontology for Scenario Orchestration (OSO) was also developed to represent scenarios. A framework SOAV was finally formed to orchestrate scenarios with autonomous simulated vehicles.

Compared to the existing methodologies in scenario orchestration, SOAV used 1) a programming language-independent and logic-based ontology to describe scenarios, including potential Actions and relevant contextual information encoded in Assignments; 2) a Monitor system derived from (Willemsen, 2000) to model a standardized event-driven mechanism in OSO to form Triggers, in order to indicate when the Actions or Assignments should be executed for interactions; 3) a hierarchy of Actions in order to standardize the Actions available for scenario orchestration and describe scenarios in different details, e.g., scenarios can be orchestrated with detailed Action such as "decelerate" or with abstract Actions such as "Block" and 4) an algorithm NAUSEA along with its user SAIL for scenario orchestration, which can provide a scenario-driven way of generating interactions with the capability of replanning. In general, the contributions of this work are:

- A medium for scenario and driving context sharing/understanding/reusing was developed based on the Ontology for Scenario Orchestration (OSO);
- A methodology of adopting human driver's task into an autonomous vehicle was proposed. It can make the simulated vehicles carry out the tasks as

required in a controllable manner. In driving simulation, this controllable manner can be used to guarantee repeatability.

This research is promising and has provided some insights into the solutions to some scenario orchestration issues. However, due to the research focus and some limitations from existing simulation platforms, the objectives of this research have not been fully achieved and SOAV, especially NAUSEA, still has some drawbacks that need to be taken care of in the future, as discussed in previous sections. In general, this research calls for greater attention on the following aspects:

1) the simulated vehicles should be enhanced to increase their fidelity regarding not only the behaviour richness, but also the behaviour realism. Their available behaviours should also include Flock-related manoeuvres;

2) the simulated world should be fully observable by Smith or the simulated vehicles and the manipulation of physical objects in the world should be standardized, e.g., the relevant interfaces for creating road segments dynamically;

3) collaboration within the driving simulation community, regarding 1) the simulation platforms and corresponding simulated vehicle design, 2) environment representation scheme and 3) scenario orchestration mechanism, are anticipated in order to not only distribute the efforts needed to tackle some existing problems in scenario orchestration, but also build a standardised scheme for knowledge sharing, especially regarding scenarios.

To sum up, the computing environment SOAV, which was created to orchestrate scenario with autonomous simulated vehicles in driving simulation, worked as desired. Its major components, namely OSO and Smith (SAIL/NAUSEA), can be the keys of bringing scenario orchestration a new future, when knowledge is more widely shared in different simulator platforms and failure-free scenarios can be orchestrated. Moreover, SOAV can also benefit some other areas:

- a standardized framework for designing the controller of autonomous vehicles can be proposed based on SOAV. Autonomous vehicles can interact with human drivers and focus on cooperation. Also, The Sim will be replaced with a physical vehicle and the SMM will be replaced with some physical interface, e.g., wheel or pedal controllers. Data collected in autonomous vehicles can be directly integrated into a simulation platform with SOAV;

- similarly, in-vehicle devices that can understand human drivers' tasks can be also designed based on SAIL/NAUSEA;
- OSO can be used to standardize information exchanged in vehicular communication systems, e.g., vehicle to vehicle or vehicle to infrastructure communications.

When I used to look out at this world, all I could see was its edges,
its boundaries, its rules and controls, its leaders and laws.
But now, I see another world.
A different world where all things are possible.
A world of hope. Of peace.
- The Matrix

Appendix A

Results of Assignment Checker and Action Checker Procedures in Experiment One

Table A.1: The Release Times and Actual Corresponding State in Sim1 (Participant One)

Assignment	**Test Case 1**		**Test Case 7**		**Test Case 9**	
	Release Time (*s*)	Condition When Executing	Release Time (*s*)	Condition When Executing	Release Time (*s*)	Condition When Executing
Acc-BL	24.1	6.1775(*s*)	16.4	6.0657(*s*)	17.5167	5.9347(*s*)
CL-BL			83.6333	6.2672(*s*)	97.1167	6.3253(*s*)
Coherence1	247.167	R5.0, 1.4460(*m*)				
Coherence2			181.633	R5.0, 4.6873	196.533	R5.0, 10.028(*m*)
Layby-V3	710.45	3.2137(*s*)			662.906	3.3334(*s*)
Layby-V4					784.578	3.1078(*s*)
Gap Acceptance	869.375	R10.1, 0.7425(*m*)			829.656	R10.1, 223.124(m)
Free Traffic Flow	247.15	R5.0, 1.2296(*m*)	181.617	R5.0, 4.4543(*m*)	196.517	R5.0, 9.7930(*m*)

Table A.2: The Release Times and Actual Corresponding State in Sim1 (Participant Two)

Assignment	**Test Case 1**		**Test Case 8**	
	Release Time (*s*)	Condition When Executing	Release Time (*s*)	Condition When Executing
Acc-BL	22.9333	5.8320(*s*)	35.5	6.0158(*s*)
CL-BL			117.2	6.2384(*s*)
Coherence1	245.783	R5.0 1.43427		
Coherence2			215.3	R5.0, 2.0145(*m*)
Layby-V3	714.094	3.3237341	674.812	3.1548(*s*)
Layby-V4			795.078	3.1134(*s*)
Gap Acceptance	869.359	R10.1 1.16649		
Free Traffic Flow	245.75	R5.0 1.00164	215.3	R5.0, 2.0145(*m*)

Table A.3: The Release Times and Actual Corresponding State in Sim1 (Participant Three)

Assignment	Test Case 1		Test Case 4		Test Case 2	
	Release Time (*s*)	Condition When Executing	Release Time (*s*)	Condition When Executing	Release Time (*s*)	Condition When Executing
Acc-BL	24.3667	6.0876(*s*)	20.1167	5.8537(*s*)	17.1	5.9746(*s*)
CL-BL					100.383	6.2844(*s*)
Coherence1	248.767	R5.0, 1.1642(*m*)	244.117	R5.0, 1.7406(*m*)		
Coherence2					200.45	R5.0, 2.2784(*m*)
Layby-V3	719.883	3.2870(*s*)			661.5	3.2431(*s*)
Layby-V4						
Gap Acceptance	884.641	R10.1, 0.6325(*m*)			822.4	R10.1, 1.5363(*m*)
Free Traffic Flow	248.733	R5.0, 0.7062(*m*)	244.083	R5.0, 1.2981(*m*)	200.433	R5.0, 2.0290(*m*)

Table A.4: The Release Times and Actual Corresponding State in Sim1 (Participant Four)

Assignment	Test Case 1		Test Case 6	
	Release Time (*s*)	Condition When Executing	Release Time (*s*)	Condition When Executing
Acc-BL	15.5167	6.0729(*s*)	15.7833	5.9165(*s*)
CL-BL				
Coherence1	238.533	R5.0, 0.7567(*m*)	238.967	R5.0, 0.5498(*m*)
Coherence2				
Layby-V3	693.033	3.2604(*s*)	693.35	3.2347(*s*)
Layby-V4			796.283	3.1073(*s*)
Gap Acceptance	846.383	R10.1, 0.7060(*m*)		
Free Traffic Flow	238.517	R5.0, 0.5278(m)	238.967	R5.0, 0.5498(m)

Table A.5: The Release Times and Actual Corresponding State in Sim1 (Participant Five)

Assignment	Test Case 1		Test Case 3		Test Case 5	
	Release Time (*s*)	Condition When Executing	Release Time (*s*)	Condition When Executing	Release Time (*s*)	Condition When Executing
Acc-BL	21.5	5.5399(*s*)	21.1667	6.0590(*s*)	16.15	5.9435 (*s*)
CL-BL			109.033	6.2812 (*s*)		
Coherence1	244.067	R5.0, 0.8440(*m*)			238.35	R5.0, 0.6188 (*m*)
Coherence2						
Layby-V3	709.817	3.2254(*s*)			699.65	3.1533 (*s*)
Layby-V4					808.531	3.2320 (*s*)
Gap Acceptance	868.203	R10.1, 1.1070(*m*)			853.583	R10.1, 210.197(*m*)
Free Traffic Flow	244.05	R5.0, 0.6228(*m*)			238.333	R5.0, 0.3966 (*m*)

Table A.6: The Release Times and Deadlines of Assignments with Success Condition or Duration (Participant One)

Assignment (Proposed Duration)	Test Case 1		Test Case 7		Test Case 9	
	Release Time (*s*)	Deadline (*s*)	Release Time (*s*)	Deadline (*s*)	Release Time (*s*)	Deadline (*s*)
Acc-BL	24.1	247.133	16.4	Fail	17.5167	Fail
CL-BL			83.6333	181.6	97.1167	196.5
Coherence1(60)	247.167	307.18(60.013)				
Coherence2 (60)			181.633	Fail	196.533	256.55(60.017)
Layby-V3 (45)	710.45	755.469(45.019)			662.906	Fail
Layby-V4 (45)					784.578	829.583(45.005)

Table A.7: The Release Times and Deadlines of Assignments with Success Condition or Duration (Participant Two)

Assignment (Proposed Duration)	Test Case 1		Test Case 8	
	Release Time (*s*)	Deadline (*s*)	Release Time (*s*)	Deadline (*s*)
Acc-BL	22.9333	245.733(*s*)	35.5	Fail
CL-BL			117.2	215.283
Coherence1(60)	245.783	305.797(60.014)		
Coherence2 (60)			215.3	275.317(60.017)
Layby-V3 (45)	714.094	759.1(45.006)	674.812	Fail
Layby-V4 (45)			795.078	Fail

Table A.8: The Release Times and Deadlines of Assignments with Success Condition or Duration (Participant Three)

Assignment (Proposed Duration)	Test Case 1		Test Case 4		Test Case 2	
	Release Time (*s*)	Deadline (*s*)	Release Time (*s*)	Deadline (*s*)	Release Time (*s*)	Deadline (*s*)
Acc-BL	24.3667	248.717	20.1167	244.067	17.1	Fail
CL-BL					100.383	200.417
Coherence1(60)	248.767	308.781(60.014)	244.117	Fail		
Coherence2 (60)					200.45	260.45(60)
Layby-V3 (45)	719.883	764.891(45.008)			661.5	706.517(45.017)
Layby-V4 (45)						

Table A.9: The Release Times and Deadlines of Assignments with Success Condition or Duration (Participant Four)

Assignment (Proposed Duration)	Test Case 1		Test Case 6	
	Release Time (*s*)	Deadline (*s*)	Release Time (*s*)	Deadline (*s*)
Acc-BL	15.5167	238.517	15.7833	238.95
CL-BL				
Coherence1(60)	238.533	298.547(60.014)	238.967	298.977(60.01)
Coherence2 (60)				
Layby-V3 (45)	693.033	738.05(45.017)	693.35	Fail
Layby-V4 (45)			796.283	Fail

Table A.10: The Release Times and Deadlines of Assignments with Success Condition or Duration (Participant Five)

Assignment (Proposed Duration)	Test Case 1		Test Case 3		Test Case 5	
	Release Time (*s*)	Deadline (*s*)	Release Time (*s*)	Deadline (*s*)	Release Time (*s*)	Deadline (*s*)
Acc-BL	21.5	244.033	21.1667	Fail	16.15	238.333
CL-BL			109.033	Fail		
Coherence1(60)	244.067	304.078(60.011)			238.35	298.367(60.017)
Coherence2 (60)						
Layby-V3 (45)	709.817	754.828(45.011)			699.65	Fail
Layby-V4 (45)					808.531	853.533(45.002)

Appendix B

Vehicles' Trajectories in Assignment One of Experiment Two

B. VEHICLES' TRAJECTORIES IN ASSIGNMENT ONE OF EXPERIMENT TWO

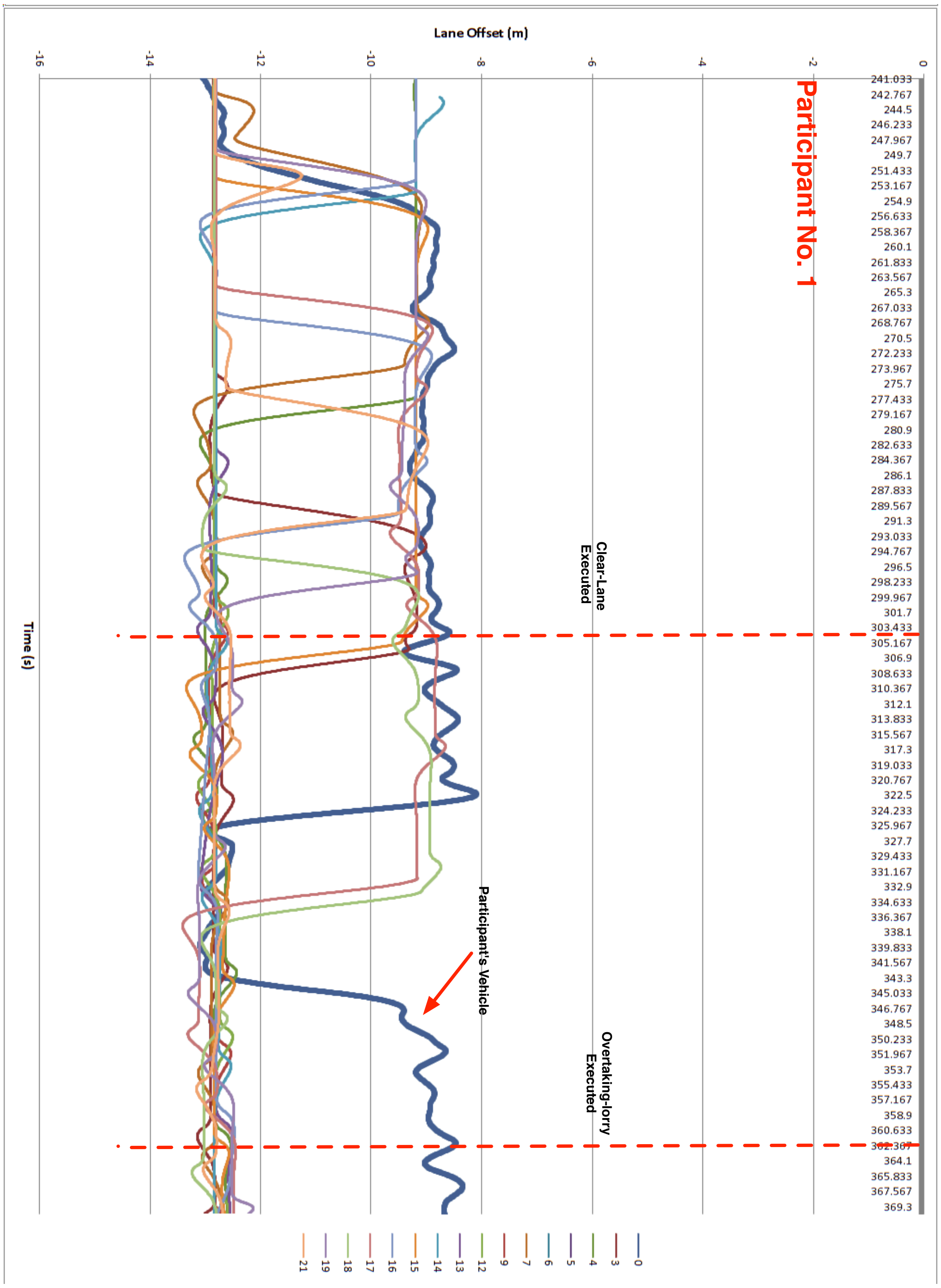


Figure B.1: Vehicles' Lane Trajectories in Assignment One - Participant One

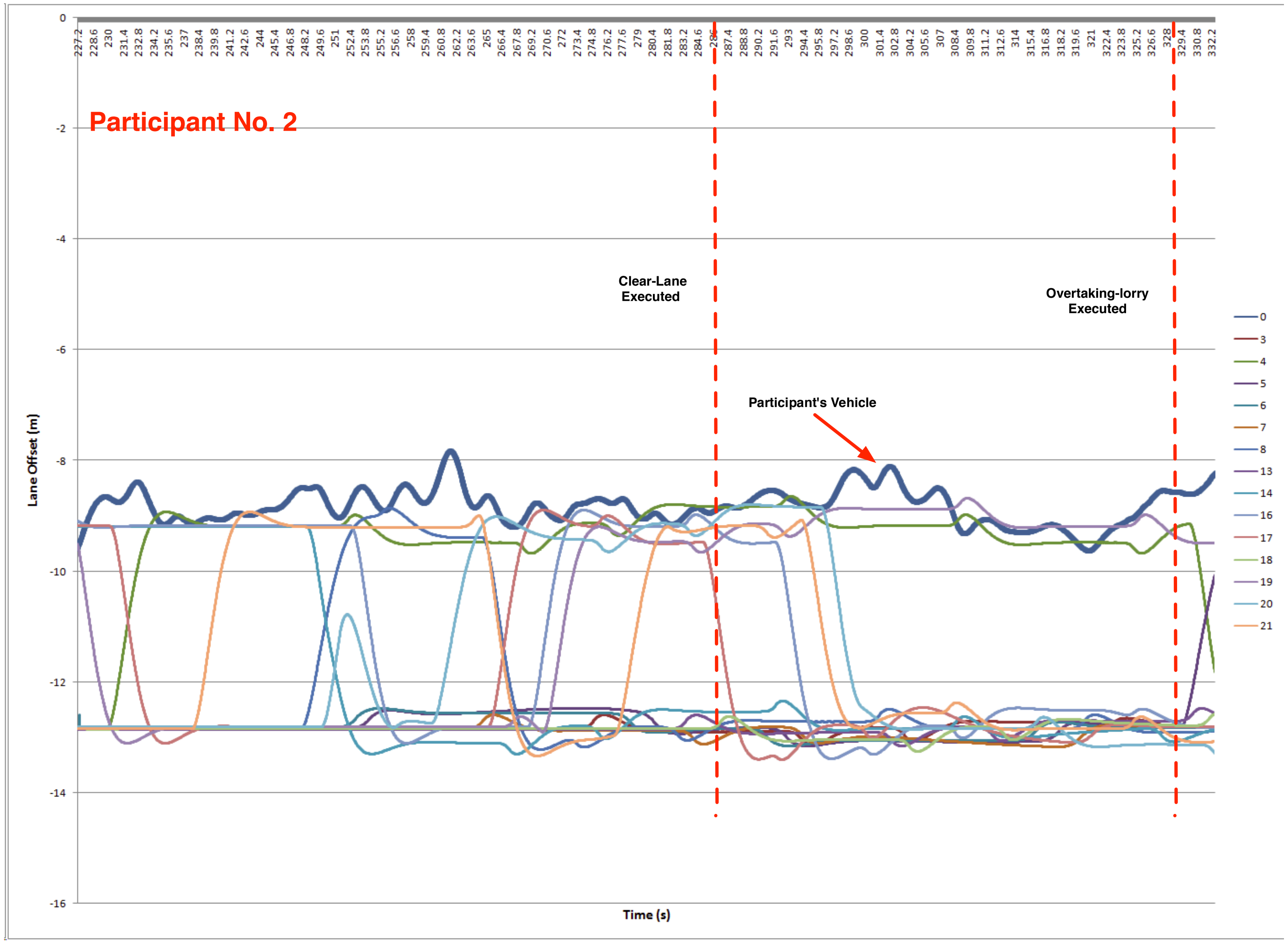


Figure B.2: Vehicles' Lane Trajectories in Assignment One - Participant Two

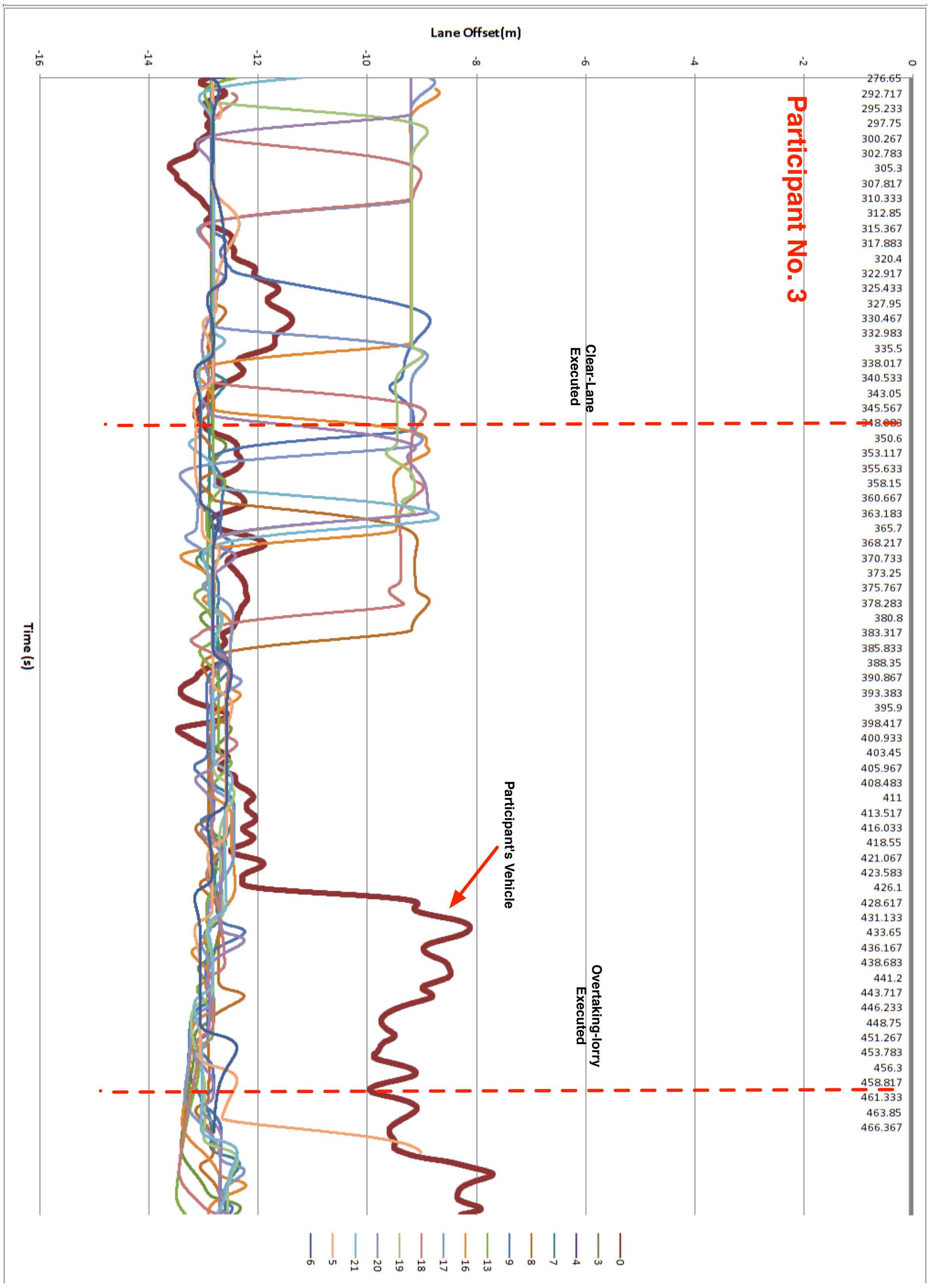


Figure B.3: Vehicles' Lane Trajectories in Assignment One - Participant Three

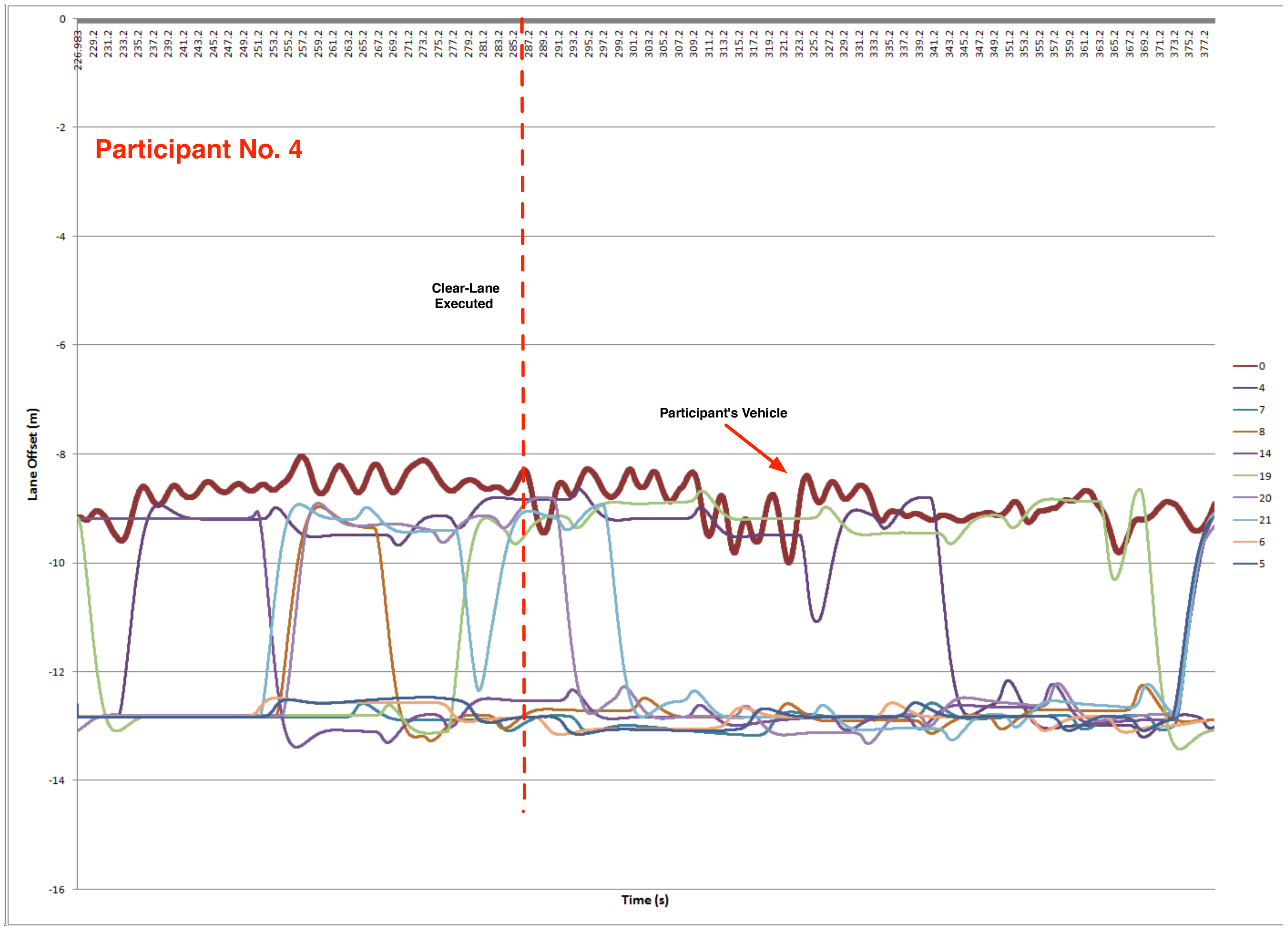


Figure B.4: Vehicles' Lane Trajectories in Assignment One - Participant Four

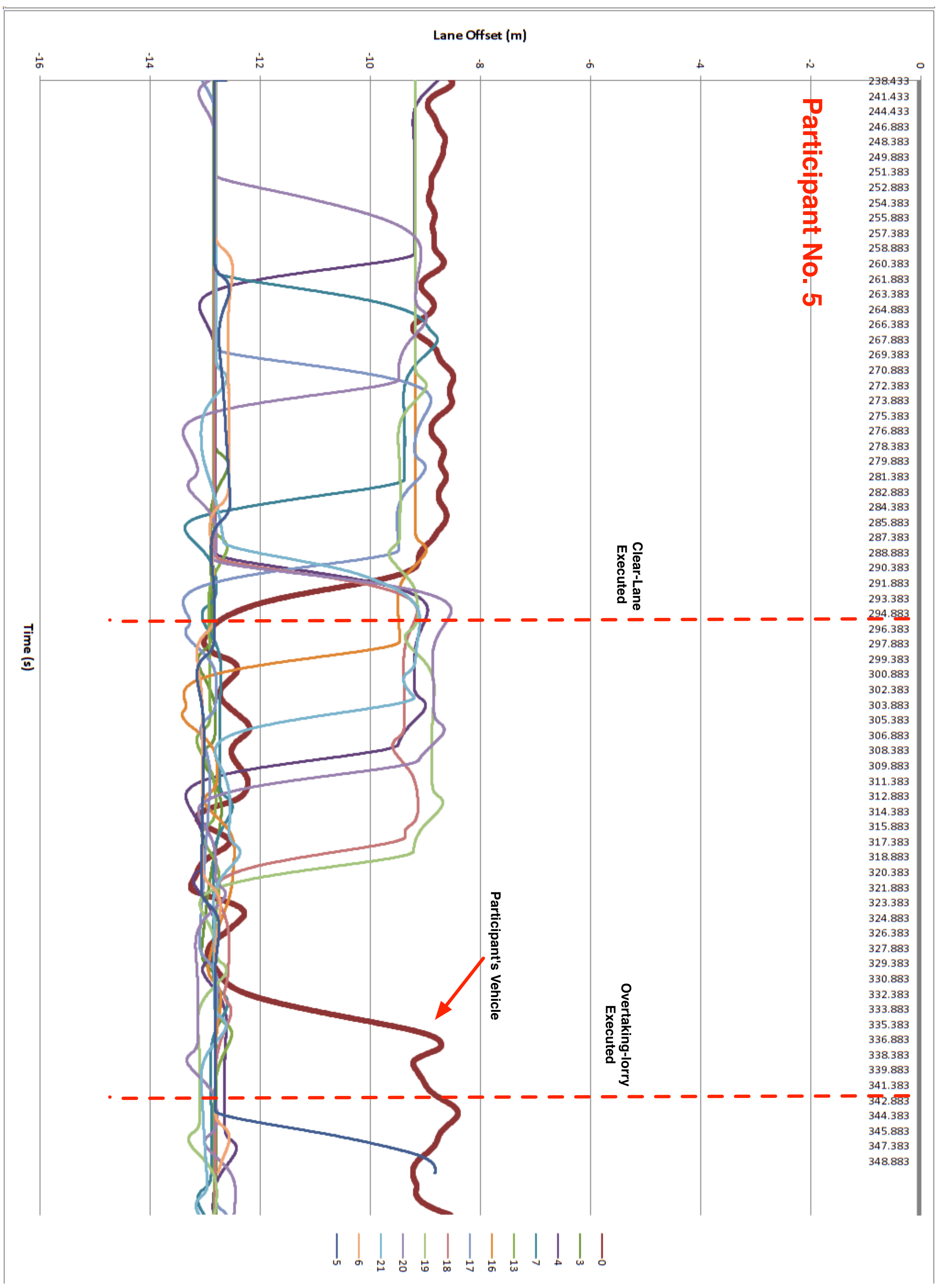


Figure B.5: Vehicles' Lane Trajectories in Assignment One - Participant Five

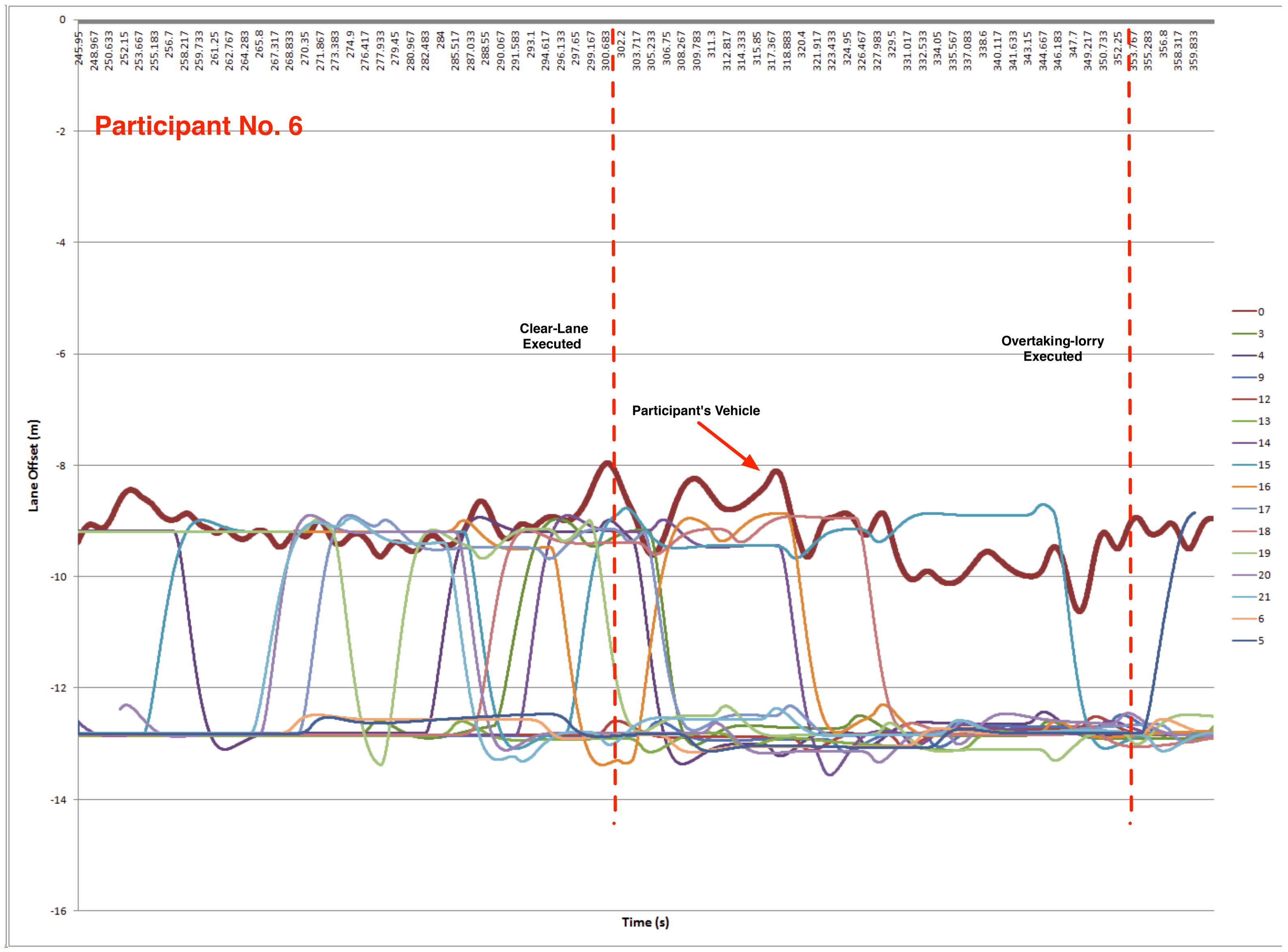


Figure B.6: Vehicles' Lane Trajectories in Assignment One - Participant Six

B. VEHICLES' TRAJECTORIES IN ASSIGNMENT ONE OF EXPERIMENT TWO

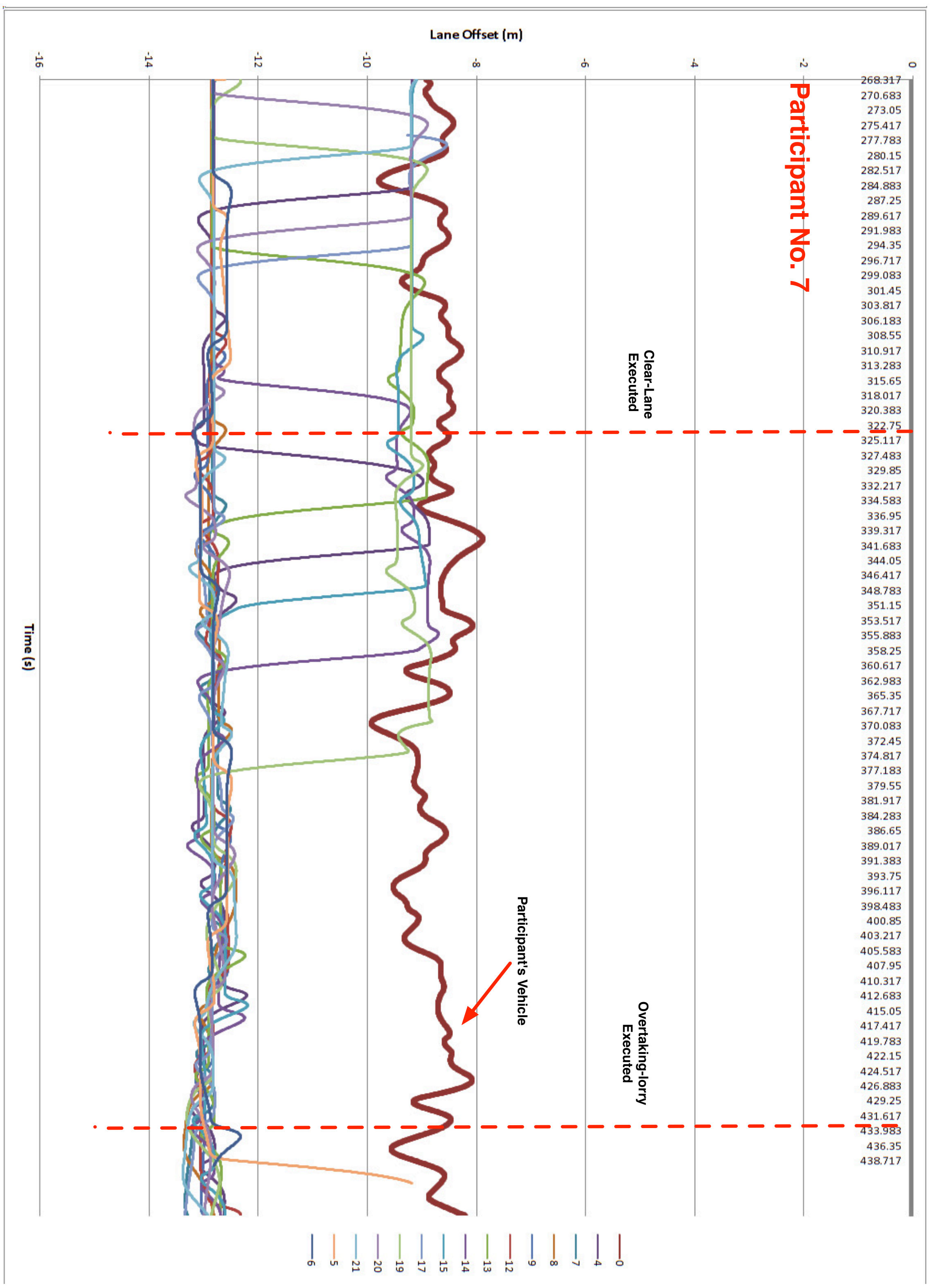


Figure B.7: Vehicles' Lane Trajectories in Assignment One - Participant Seven

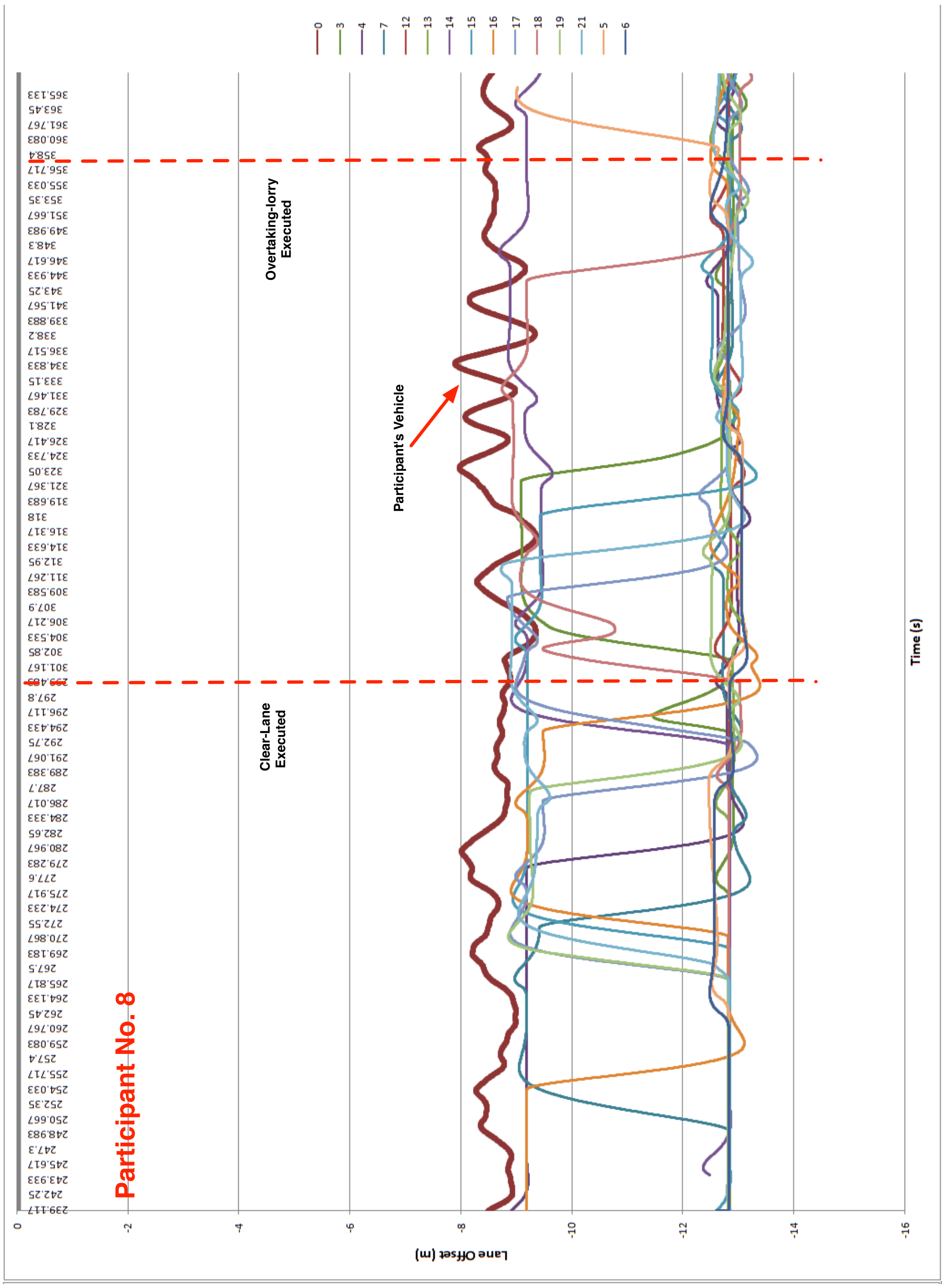


Figure B.8: Vehicles' Lane Trajectories in Assignment One - Participant Eight

B. VEHICLES' TRAJECTORIES IN ASSIGNMENT ONE OF EXPERIMENT TWO

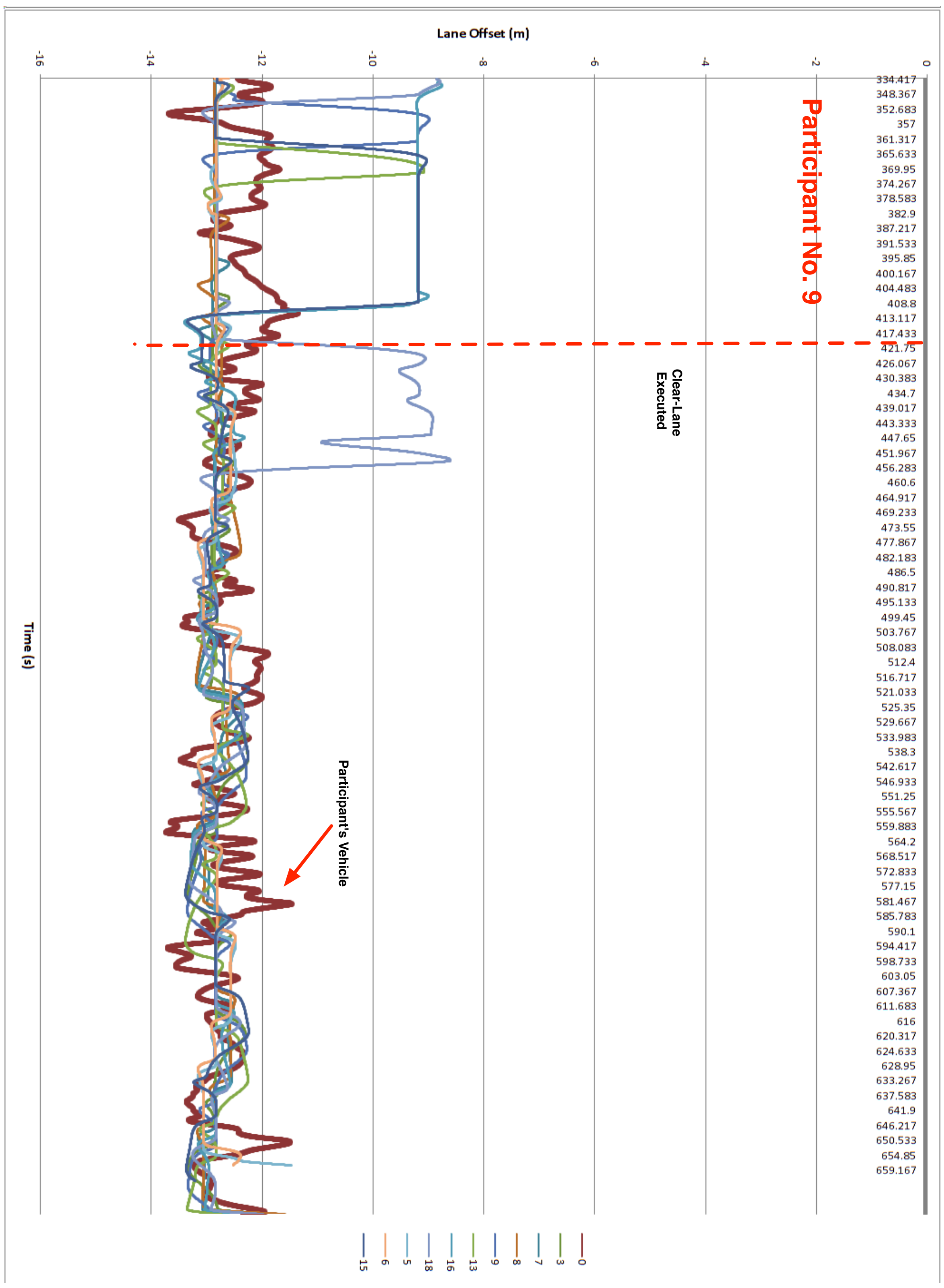


Figure B.9: Vehicles' Lane Trajectories in Assignment One - Participant Nine

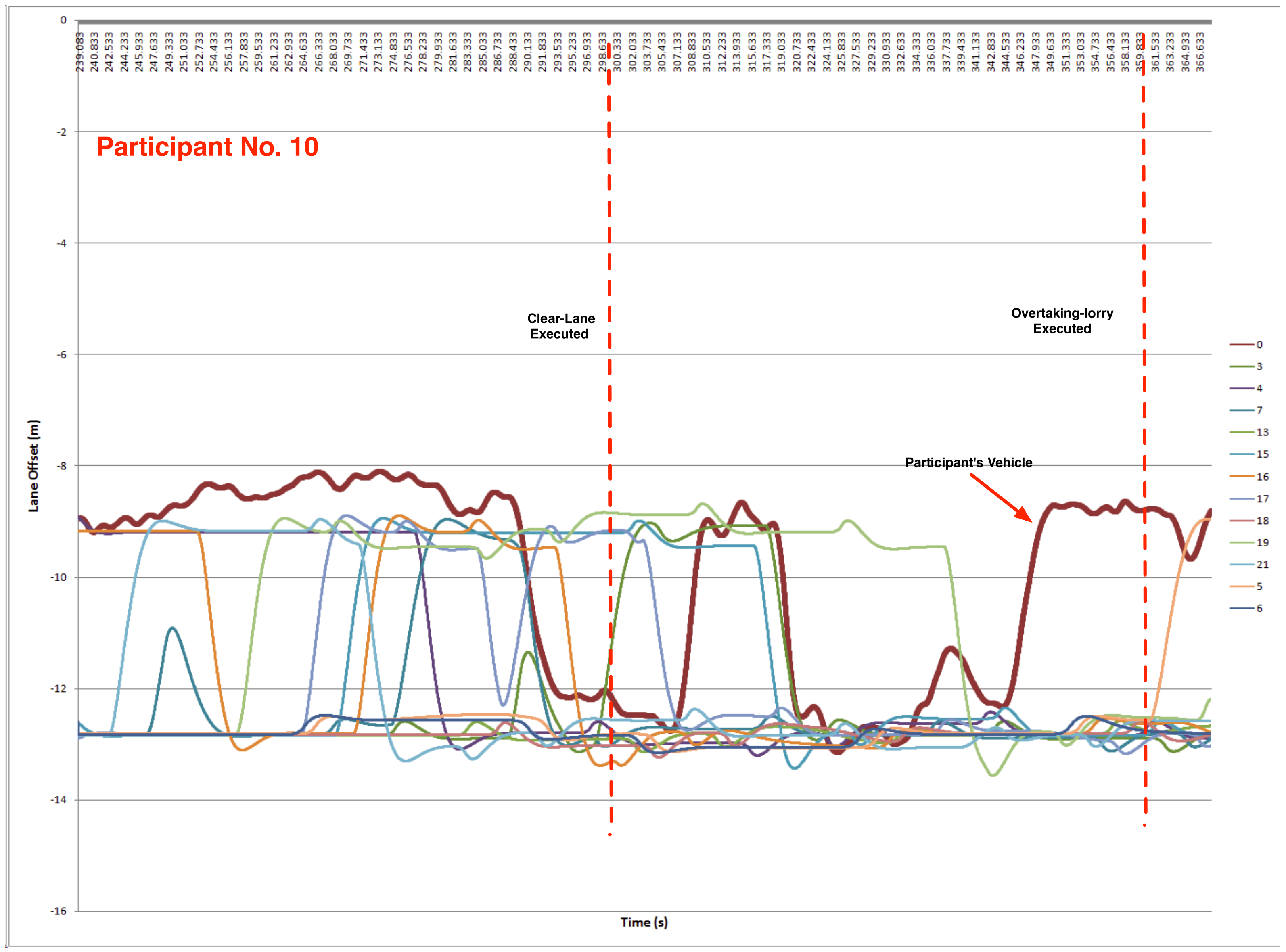


Figure B.10: Vehicles' Lane Trajectories in Assignment One - Participant Ten

B. VEHICLES' TRAJECTORIES IN ASSIGNMENT ONE OF EXPERIMENT TWO

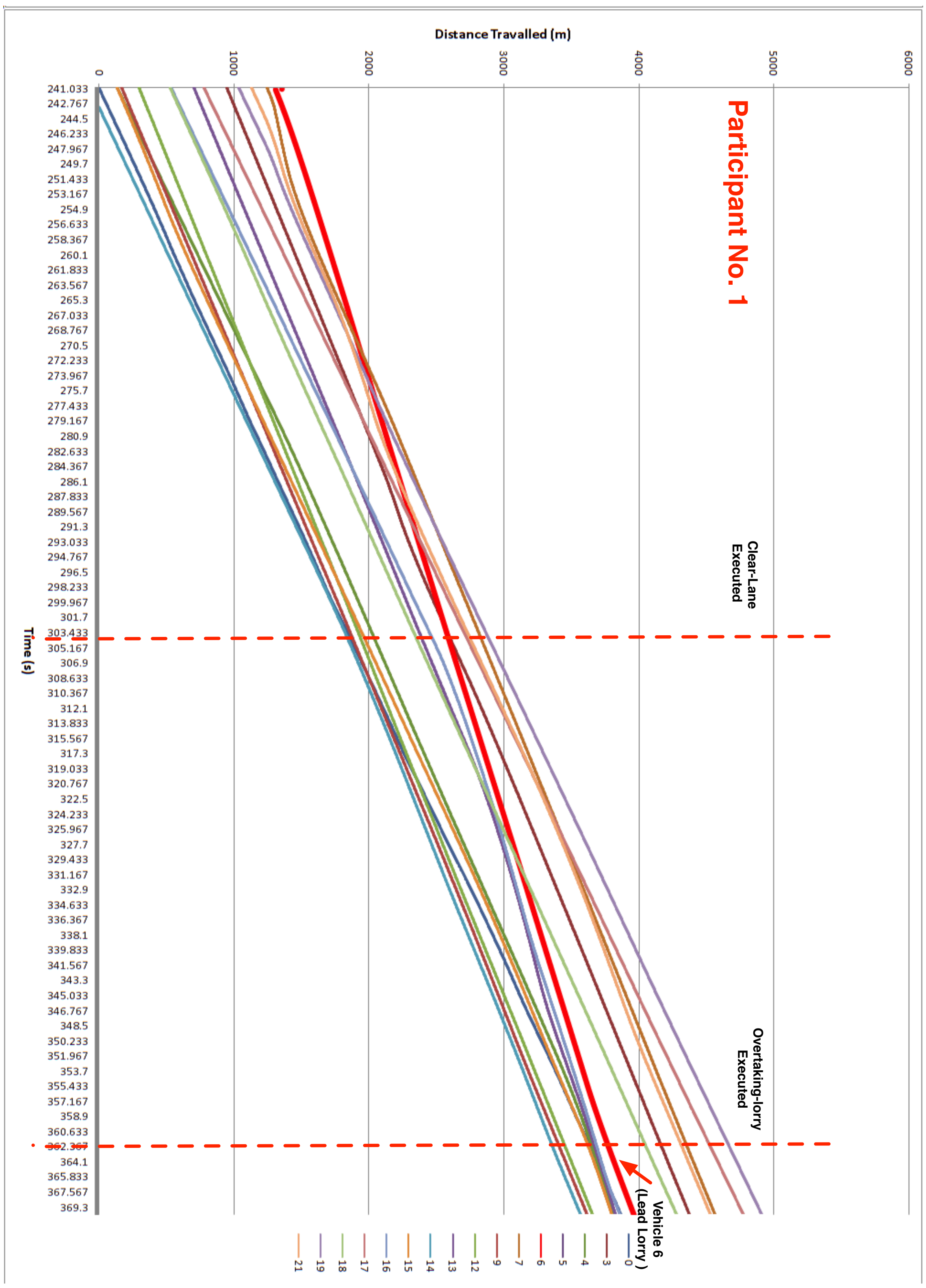


Figure B.11: Vehicles' Longitudinal Trajectories in Assignment One - Participant One

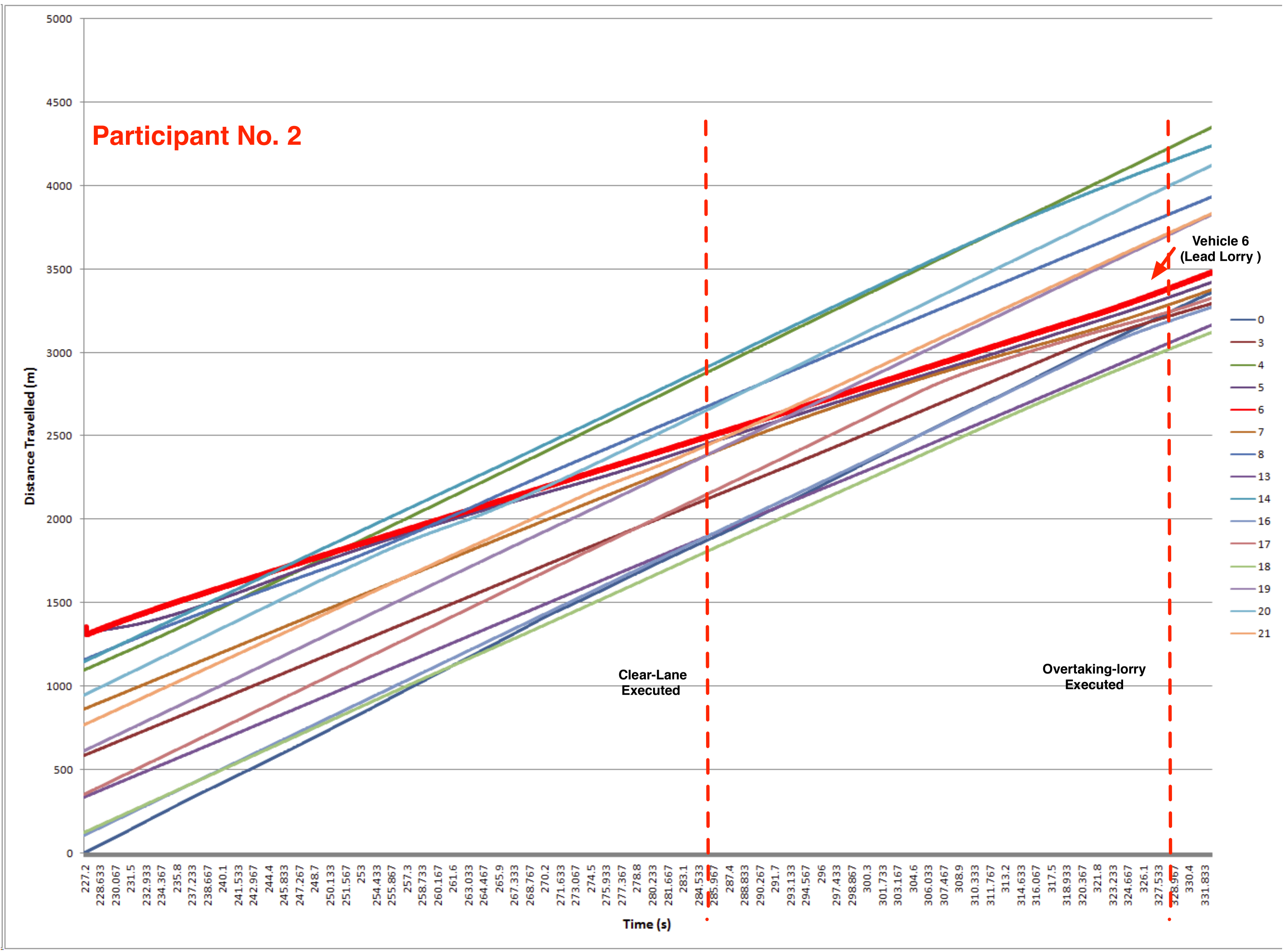


Figure B.12: Vehicles' Longitudinal Trajectories in Assignment One - Participant Two

B. VEHICLES' TRAJECTORIES IN ASSIGNMENT ONE OF EXPERIMENT TWO

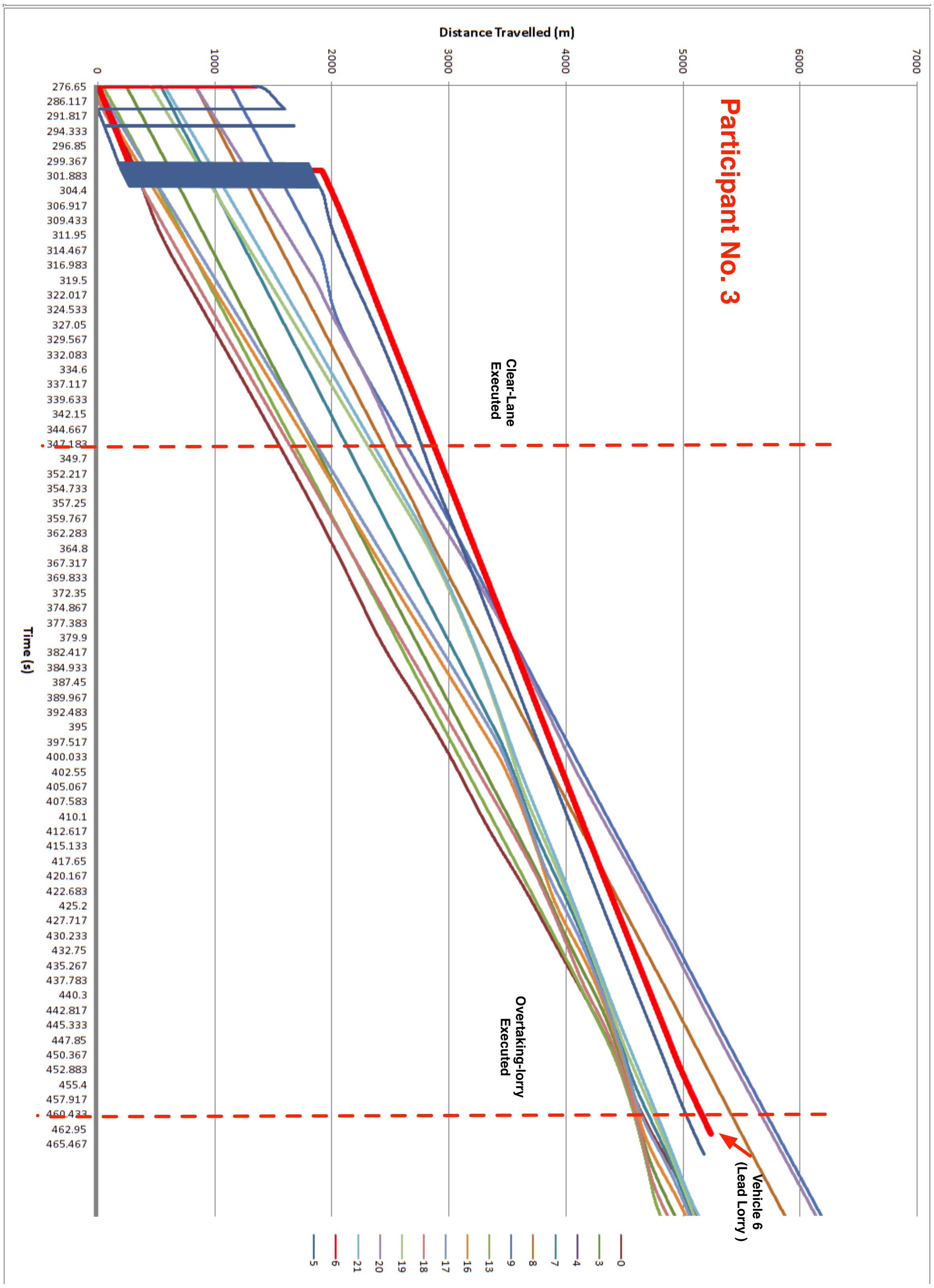


Figure B.13: Vehicles' Longitudinal Trajectories in Assignment One - Participant Three

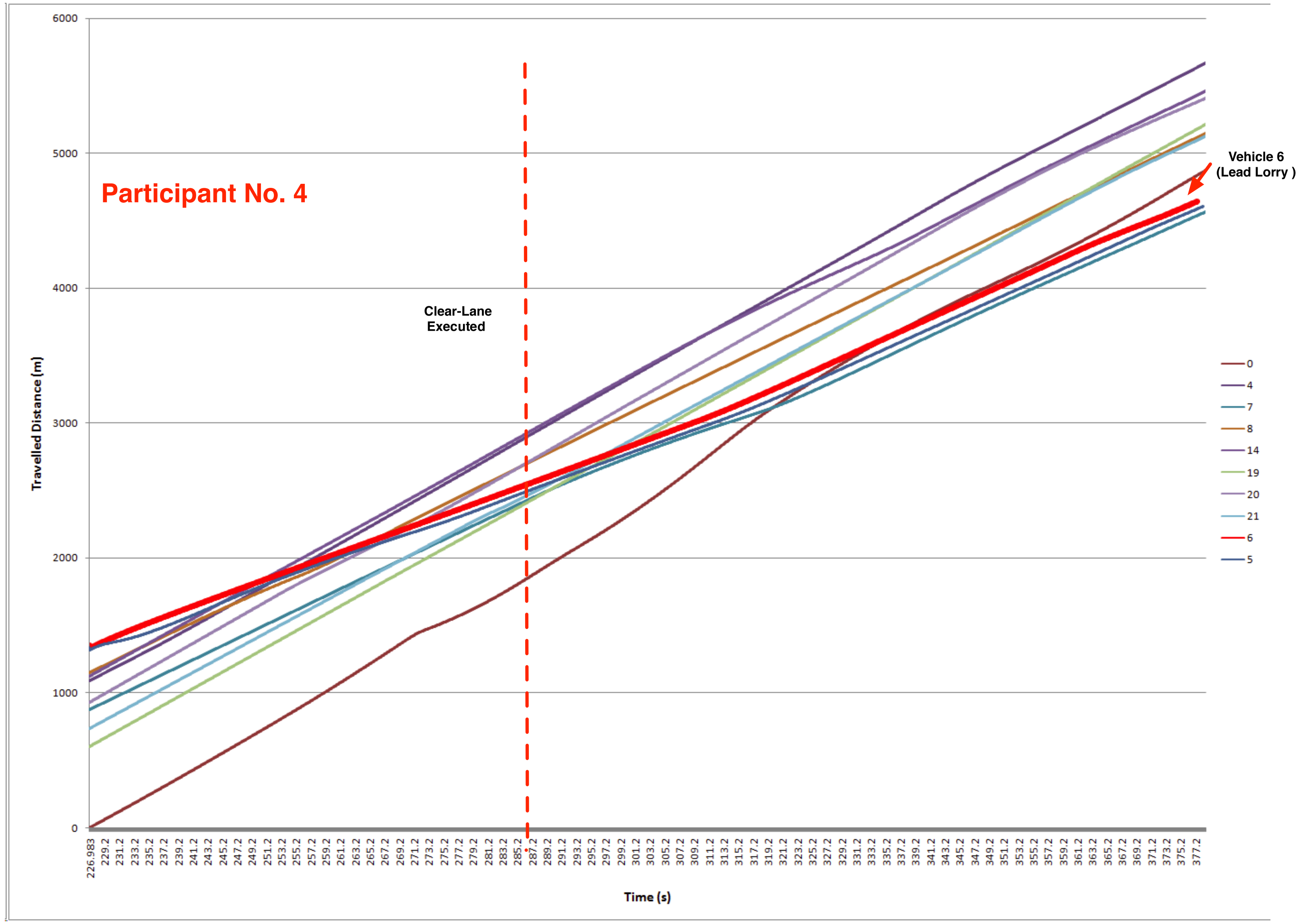


Figure B.14: Vehicles' Longitudinal Trajectories in Assignment One - Participant Four

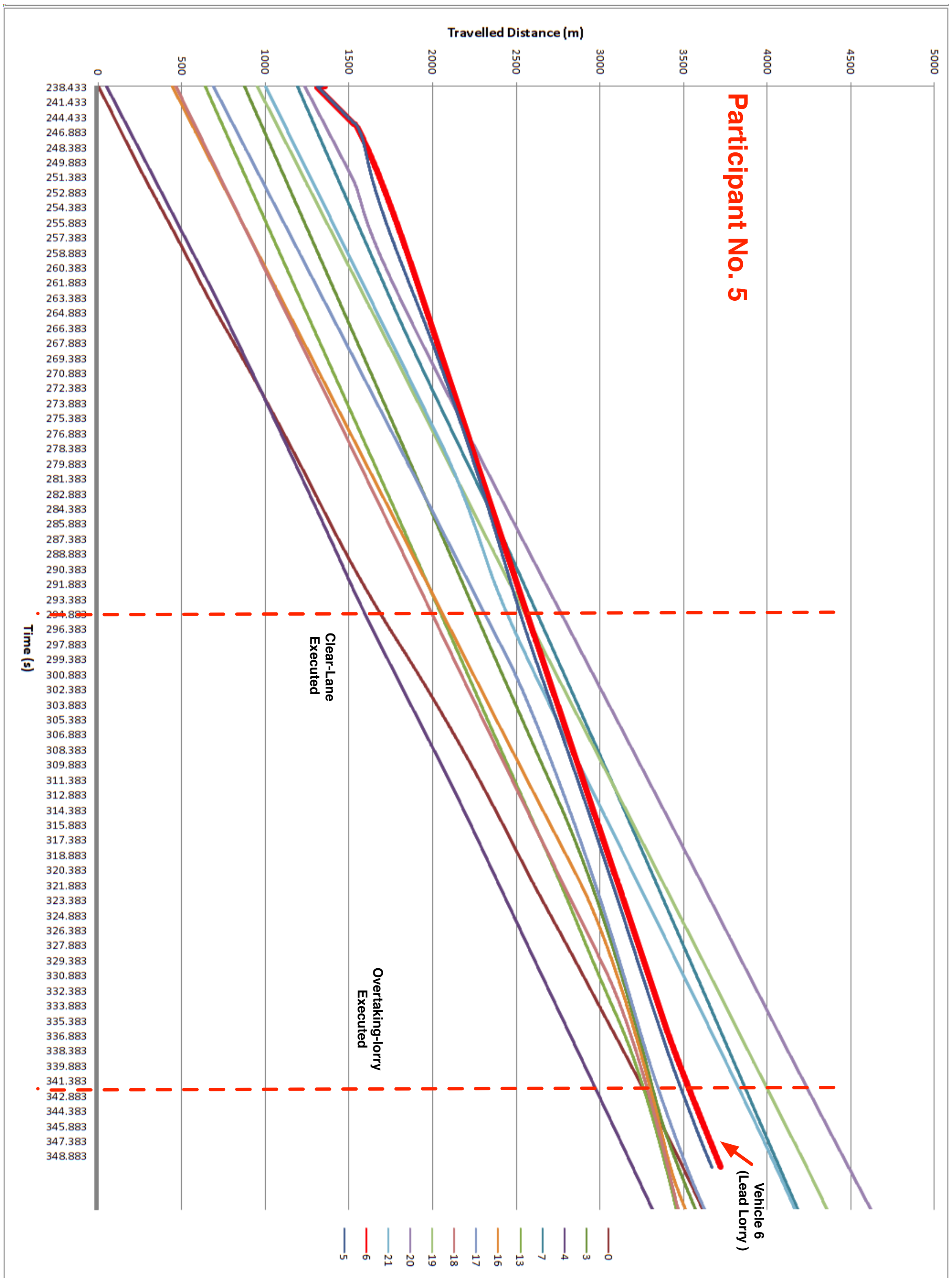


Figure B.15: Vehicles' Longitudinal Trajectories in Assignment One - Participant Five

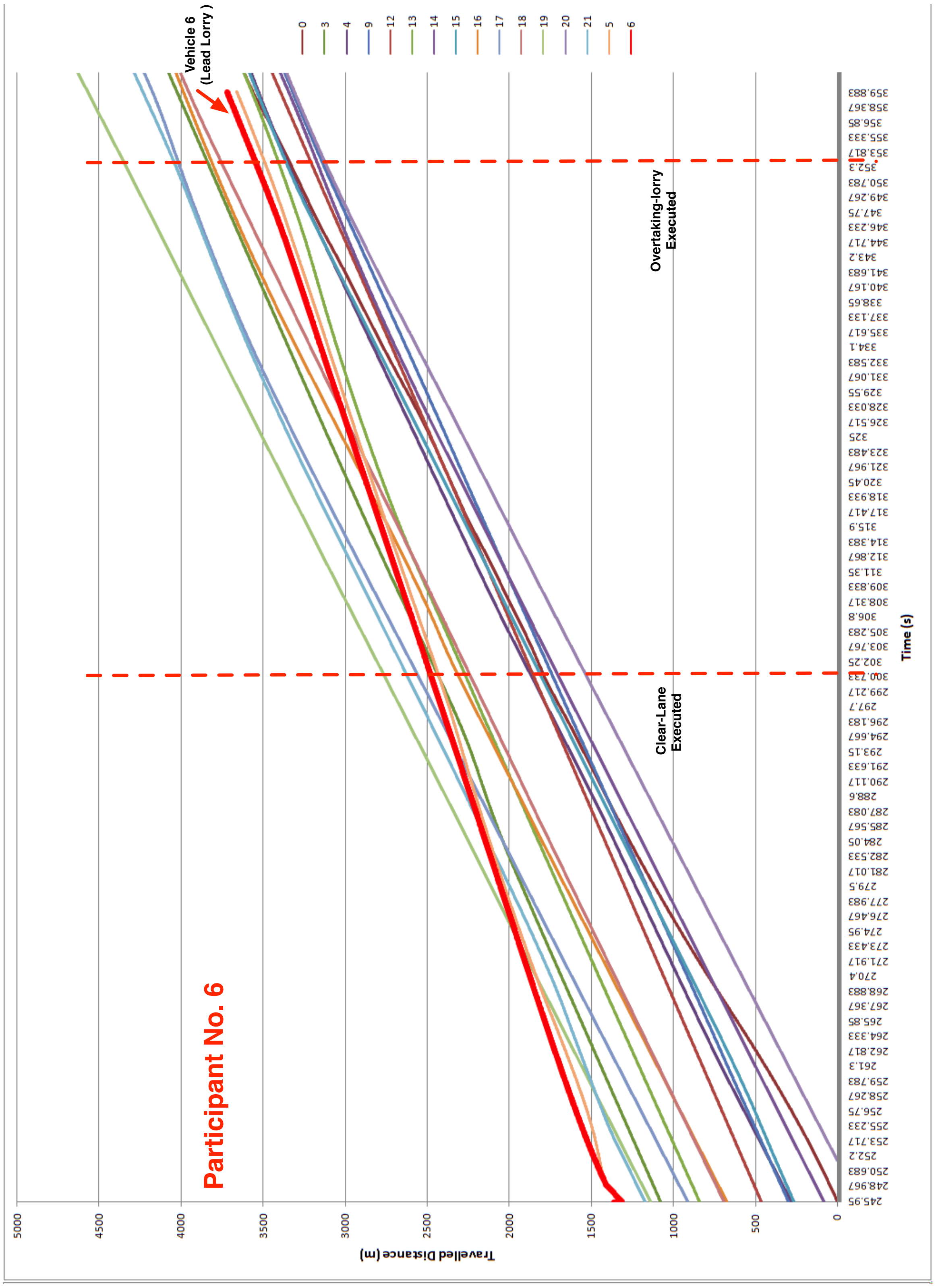


Figure B.16: Vehicles' Longitudinal Trajectories in Assignment One - Participant Six

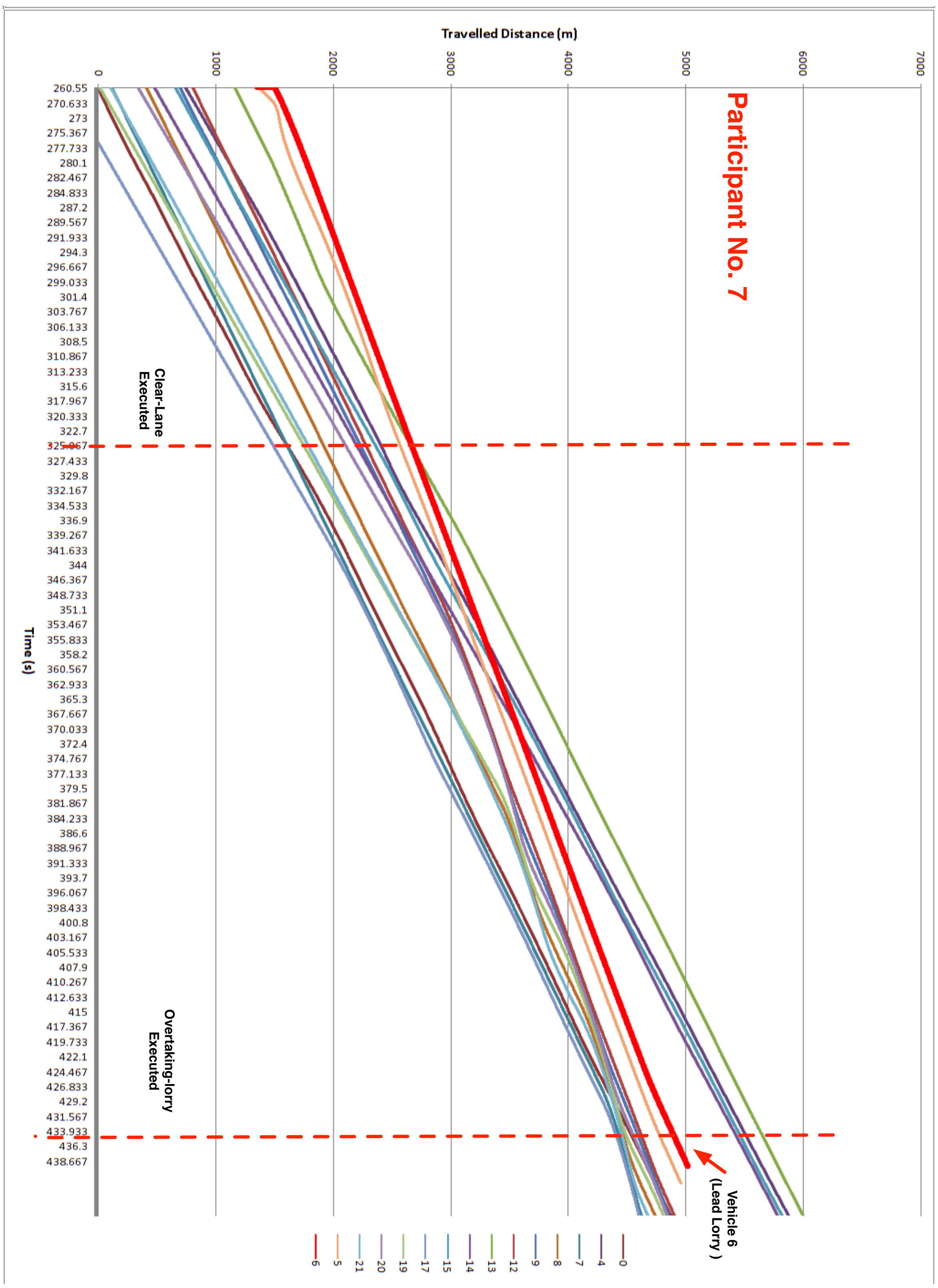


Figure B.17: Vehicles' Longitudinal Trajectories in Assignment One - Participant Seven

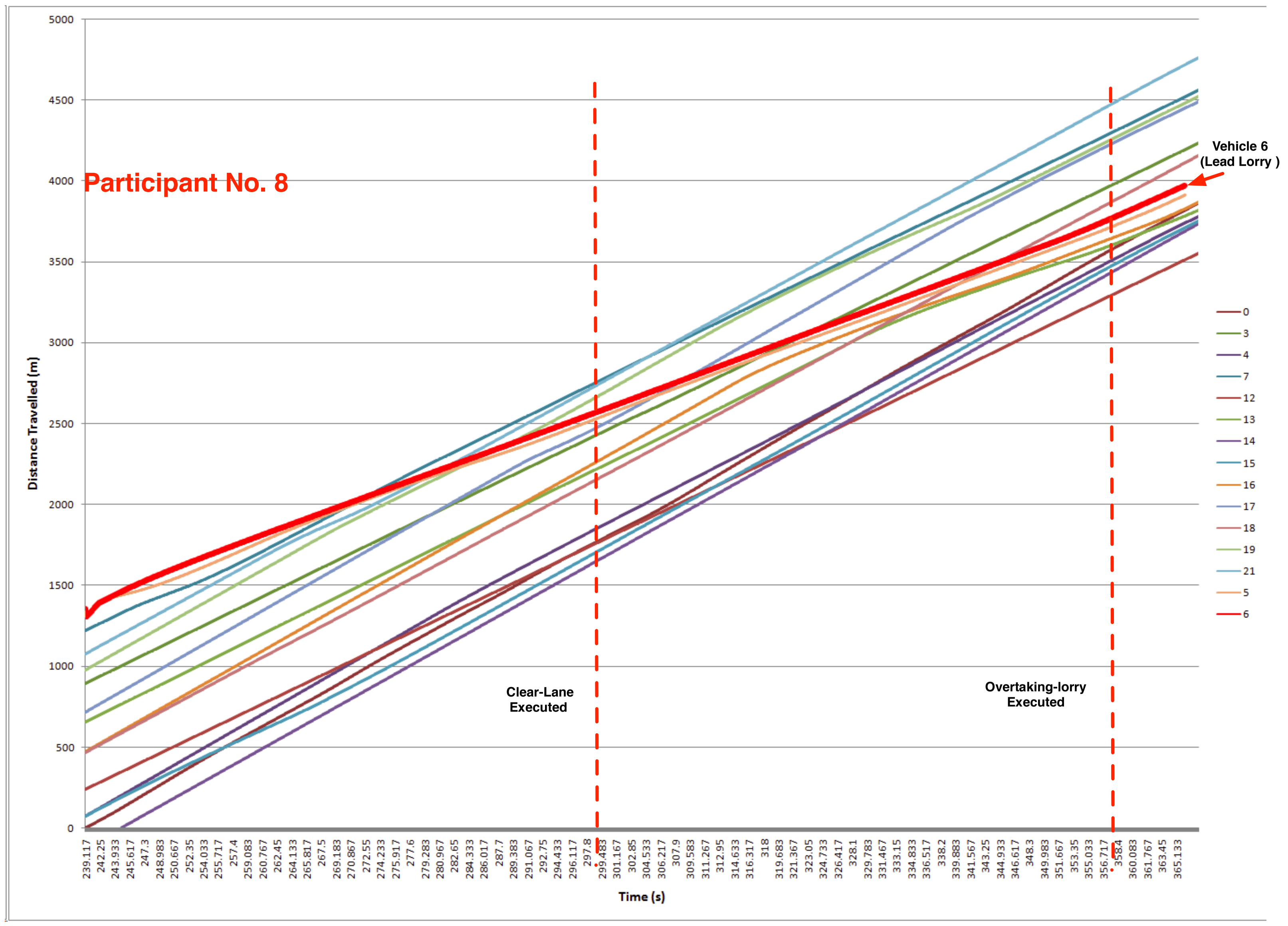


Figure B.18: Vehicles' Longitudinal Trajectories in Assignment One - Participant Eight

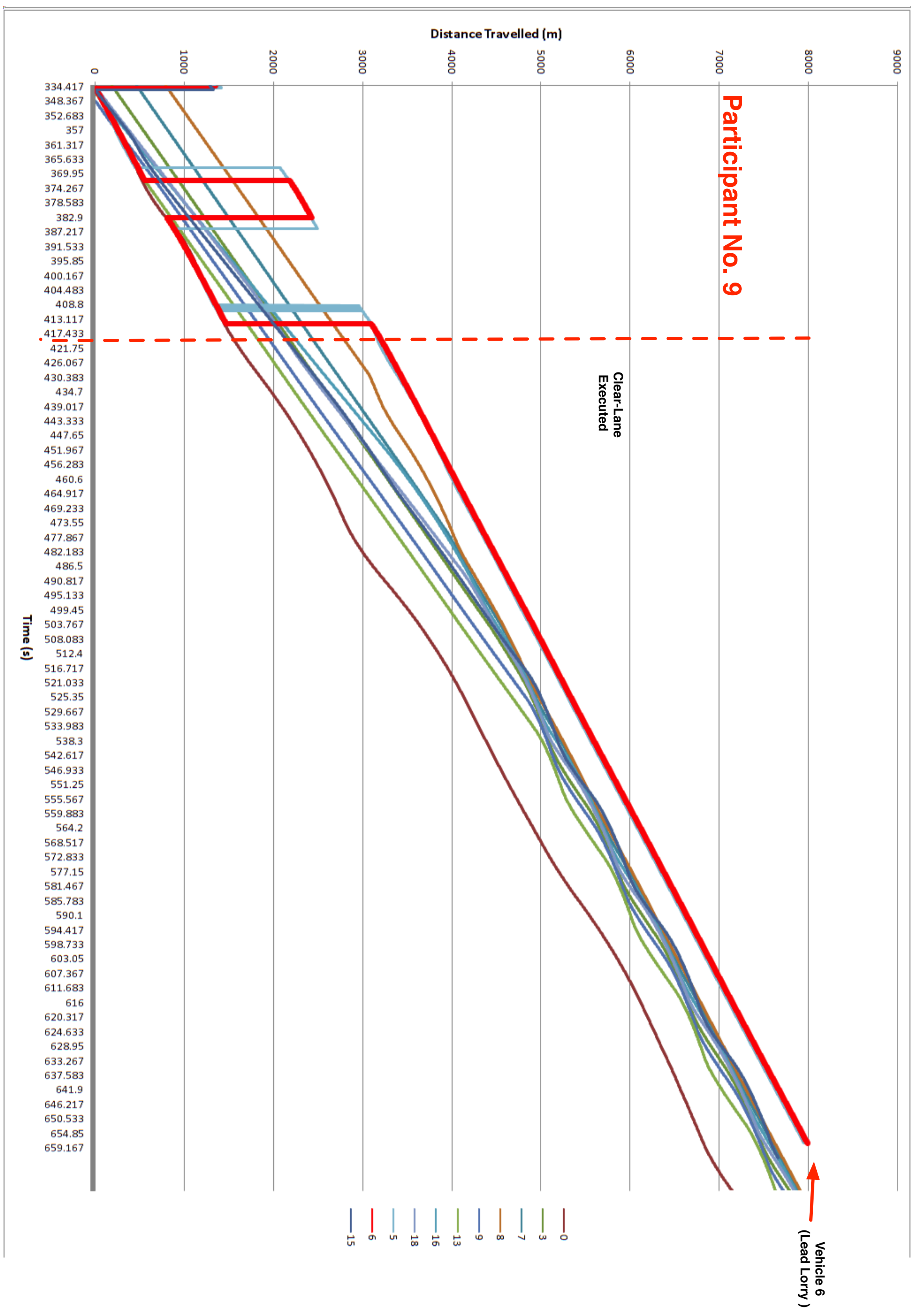


Figure B.19: Vehicles' Longitudinal Trajectories in Assignment One - Participant Nine

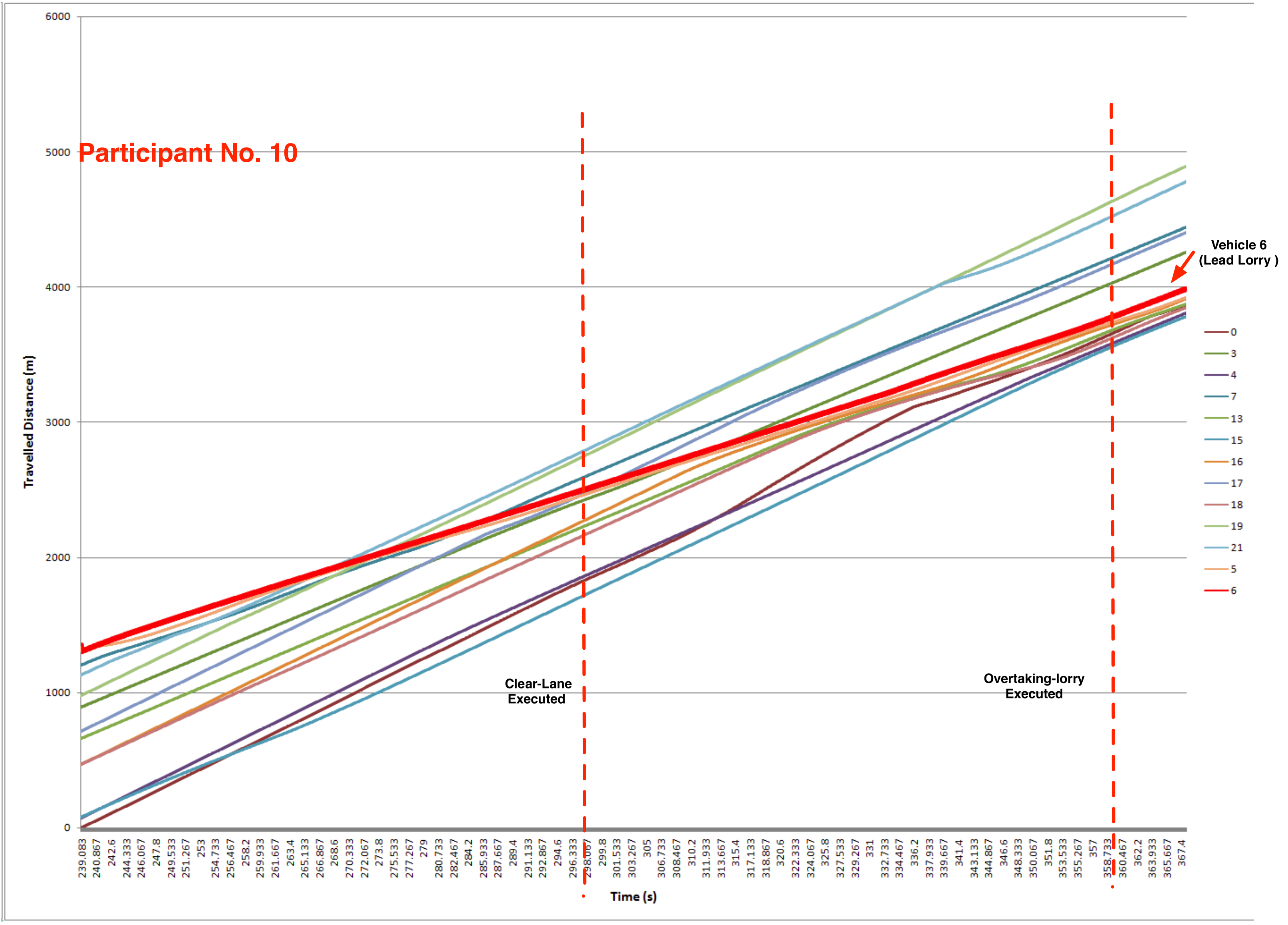


Figure B.20: Vehicles' Longitudinal Trajectories in Assignment One - Participant Ten

Appendix C

Scenario Representation with OSO

C.1 Experiment One

C.1.1 Assignment "Acc-BL"

First of all, an individual of class *Assignment* named *coherence_beleader_1* was created in OSO, containing the following main information:

C.1.1.1 Formation Position

This Assignment needed a leader, so the Formation Position should be "Leader", i.e., *hasFormationPosition* some *Leader*.

C.1.1.2 Monitor

"**When the participant's vehicle's (*subject*) time headway to the Ego-vehicle, which is the leader (*SV_TimeHW_L*), changes from greater than, to less than 6 seconds.**"

An individual of class *ReferenceValue* named *value_bl_monitor* was created to represent the reference value mentioned: "from greater than to less than 6 seconds". Its properties are shown in Table C.1.

Table C.1: Properties of *value_bl_monitor* for "Acc-BL" in Experiment One

Property	**Related to**
hasRefValueType	some *NUMBER*
hasRefValueRange	some *SEQUAL* ($\leqslant$)
hasGeneralValue	6.0

An individual of class *Monitor* named *bl_monitor* was created to represent this Monitor. Its properties are shown in Table C.2.

Table C.2: Properties of *bl_monitor* for "Acc-BL" in Experiment One

Property	**Related to**
hasMonitorType	some *Event*
hasEventType	some *ET_Threshold*
hasMonitorOperator	some *WHEN*
hasReference	*subject*
hasStateVariable	some *SV_TimeHW_L*
hasRefValue	*value_bl_monitor*

C.1.1.3 Success Condition

"When the participant's (*subject*) travelling road (*SV_RdID*) changes to 'r5.0'."

An individual of the class *ReferenceValue* named *value_bl_post* was created to represent the reference value mentioned, which is "changes to 'r5.0' " (from not equal to equal to "r5.0"). Its properties are shown in Table C.3.

An individual of the class *Monitor* named *bl_post* was created to represent this Success Condition. Its properties are shown in Table C.4.

Table C.3: Properties of *value_bl_post* for “Acc-BL” in Experiment One

Property	**Related to**
hasRefValueType	some *NUMBER*
hasRefValueRange	some *NOTOEQUAL*
hasGeneralValue	5.0

Table C.4: Properties of *bl_post* for “Acc-BL” in Experiment One

Property	**Related to**
hasMonitorType	some *Event*
hasEventType	some *ET_Threshold*
hasMonitorOperator	some *WHEN*
hasReference	*subject*
hasStateVariable	some *SV_RdID*
hasRefValue	*value_bl_post*

C.1.1.4 Failure Condition

“When the participant’s (*subject*) leader’s ID (*SV_Nei_L*) changes from 1 to another number (the participant’s leader changes before ‘r5.0’).”

An individual of the class *ReferenceValue* named *value_bl_fail* was created to represent the reference value mentioned, which is “from 1 to another number”. Its properties are shown in Table C.5.

Table C.5: Properties of *value_bl_fail* for “Acc-BL” in Experiment One

Property	**Related to**
hasRefValueType	some *NUMBER*
hasRefValueRange	some *EQUALTONO*
hasGeneralValue	1.0

An individual of the class *Monitor* named *bl_fail* was created to represent this Failure Condition. Its properties are shown in Table C.6.

Table C.6: Properties of *bl_fail* for "Acc-BL" in Experiment One

Property	**Related to**
hasMonitorType	some *Event*
hasEventType	some *ET_Threshold*
hasMonitorOperator	some *WHEN*
hasReference	*subject*
hasStateVariable	some *SV_Nei_L*
hasRefValue	*value_bl_fail*

C.1.1.5 Action

When the Monitor became true, an Action of "accelerate to 13 *mph*" was triggered. Smith would force the Ego-vehicle to reach a target speed of 13 *mph*. An individual of the class *Action* named *beleader* was created, specifying that the Target speed (*DesiredSpeed*) should be 13.3 *m/s* or 30 *mph*. Its properties are shown in Table C.7.

Table C.7: Properties of *beleader* for "Acc-BL" in Experiment One

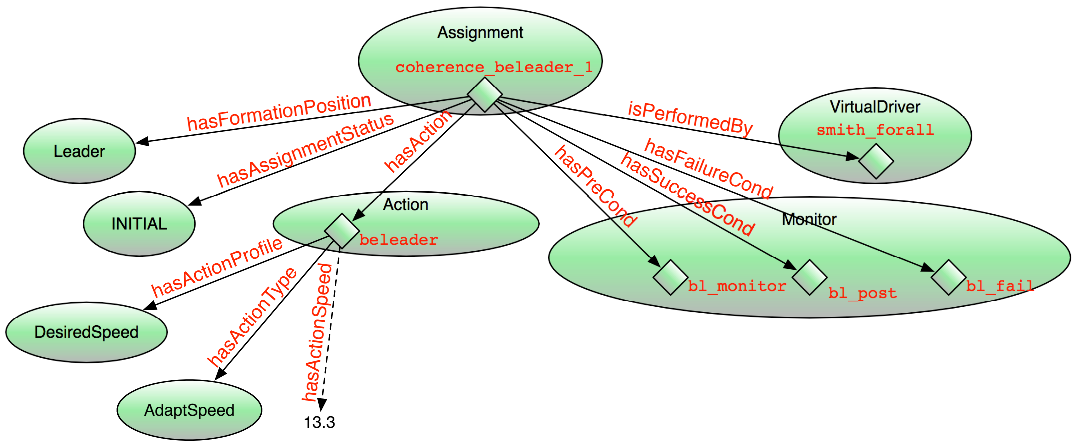
Property	**Related to**
hasActionProfile	some *DesiredSpeed*
hasActionType	some *AdaptSpeed*
hasActionSpeed	13.3

C.1.1.6 Performer

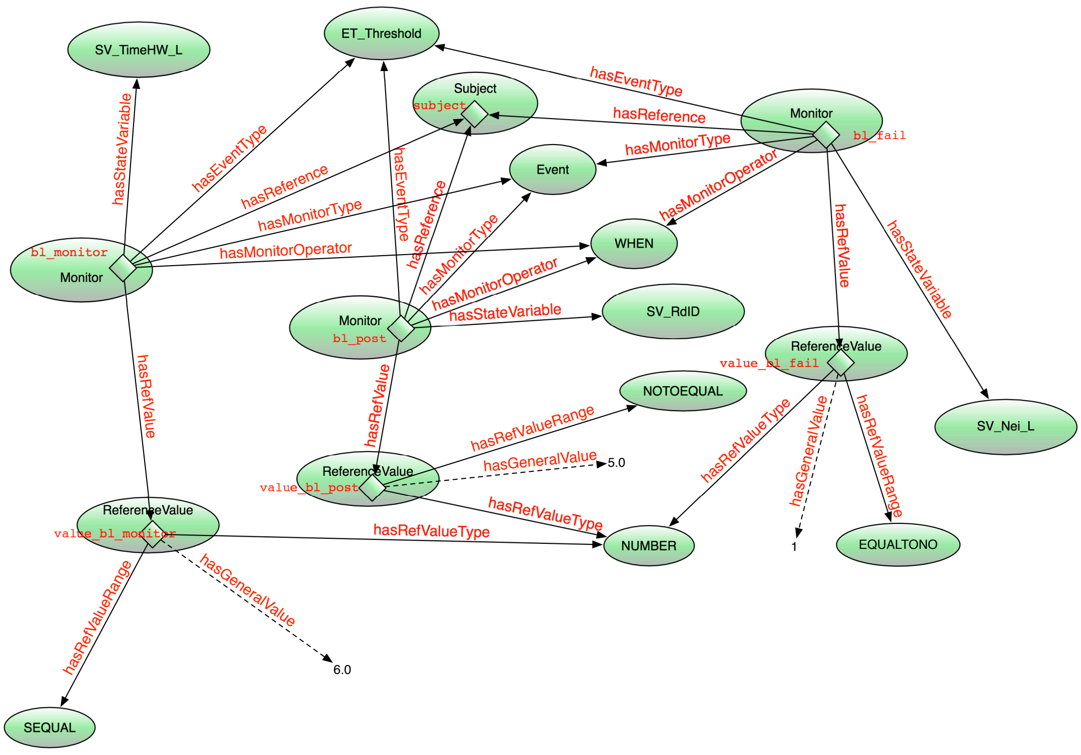
This Assignment should be carried out by Smith, not SMM. An individual of the class *VirtualDriver* named *smith_forall* was created to represent this Smith, i.e., *isPerformedBy smith_forall*

C.1.1.7 Representation By Directed Graph

By adopting the individuals mentioned above and some other information, e.g., the initial status of Assignment *coherence_beleader_1*, the main information of this Assignment can be illustrated as in Figure C.1. This Assignment can be tried twice (*hasMaxtriedTime 2*).



(a)



(b)

Figure C.1: Illustration of *coherence_beleader_1* in Experiment One

C.1.2 Assignment “Coherence”

First of all, an individual of the class *Assignment* named *coherence_2* was created in OSO, containing the following main information:

C.1.2.1 Formation Position

This Assignment needed a leader, so the Formation Position should be “Leader”, i.e., *hasFormationPosition* some *Leader*.

C.1.2.2 Monitor

“When the participant’s (*subject*) travelling road (*SV_RdID*) changes to ‘r5.0’.”

An individual of the class *ReferenceValue* named *value_c_monitor* was created to represent the reference value mentioned, which is “changes to ‘r5.0’ ”. Its properties are shown in Table C.8.

Table C.8: Properties of *value_c_monitor* for “Coherence” in Experiment One

Property	**Related to**
hasRefValueType	some *NUMBER*
hasRefValueRange	some *NOTOEQUAL*
hasGeneralValue	5.0

An individual of the class *Monitor* named *coherence_monitor* was created to represent this Monitor. Its properties are shown in Table C.9.

C.1.2.3 Failure Condition

“When the Ego-vehicle’s (*smith_forall*) leader’s ID (*SV_Nei_L*) changes from greater than, to less than 2 (the Ego-vehicle’s leader changes to the participant’s vehicle during the Assignment).”

An individual of the class *ReferenceValue* named *value_c_fail* was created to represent the reference value mentioned, which is “greater than to less than 2”. Its properties are shown in Table C.10.

Table C.9: Properties of *coherence_monitor* for "Coherence" in Experiment One

Property	**Related to**
hasMonitorType	some *Event*
hasEventType	some *ET_Threshold*
hasMonitorOperator	some *WHEN*
hasReference	*subject*
hasStateVariable	some *SV_RdID*
hasRefValue	*value_c_monitor*

Table C.10: Properties of *value_c_fail* for "Coherence" in Experiment One

Property	**Related to**
hasRefValueType	some *NUMBER*
hasRefValueRange	some *SEQUAL* ($\leqslant$)
hasGeneralValue	2

An individual of class *Monitor* named *coherence_fail* was created to represent this Failure Condition. Its properties are shown in Table C.11.

Table C.11: Properties of *coherence_fail* for "Coherence" in Experiment One

Property	**Related to**
hasMonitorType	some *Event*
hasEventType	some *ET_Threshold*
hasMonitorOperator	some *WHEN*
hasReference	*smith_forall*
hasStateVariable	some *SV_Nei_L*
hasRefValue	*value_c_fail*

C.1.2.4 Action

When the Monitor became true, an Action was triggered, in which case, Smith would force the Ego-vehicle to change its speed sinusoidally for 60 seconds based on a desired speed of 26.8 m/s. An individual of the class *Action* named *coherence* was created. Its properties are shown in Table C.12.

Table C.12: Properties of *coherence* for "Coherence" in Experiment One

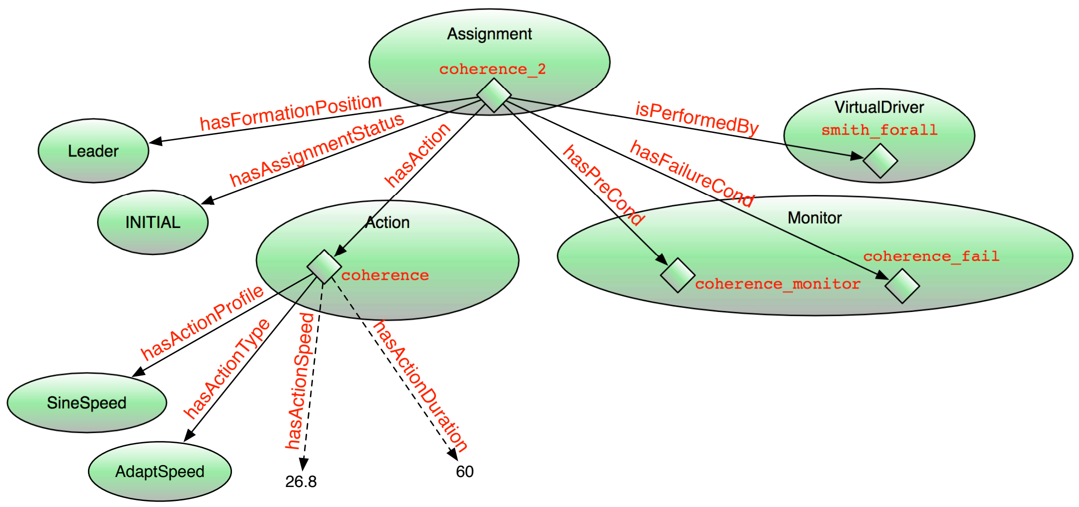
Property	**Related to**
hasActionProfile	some *SineSpeed*
hasActionType	some *AdaptSpeed*
hasActionSpeed	26.8
hasActionDuration	60

C.1.2.5 Performer

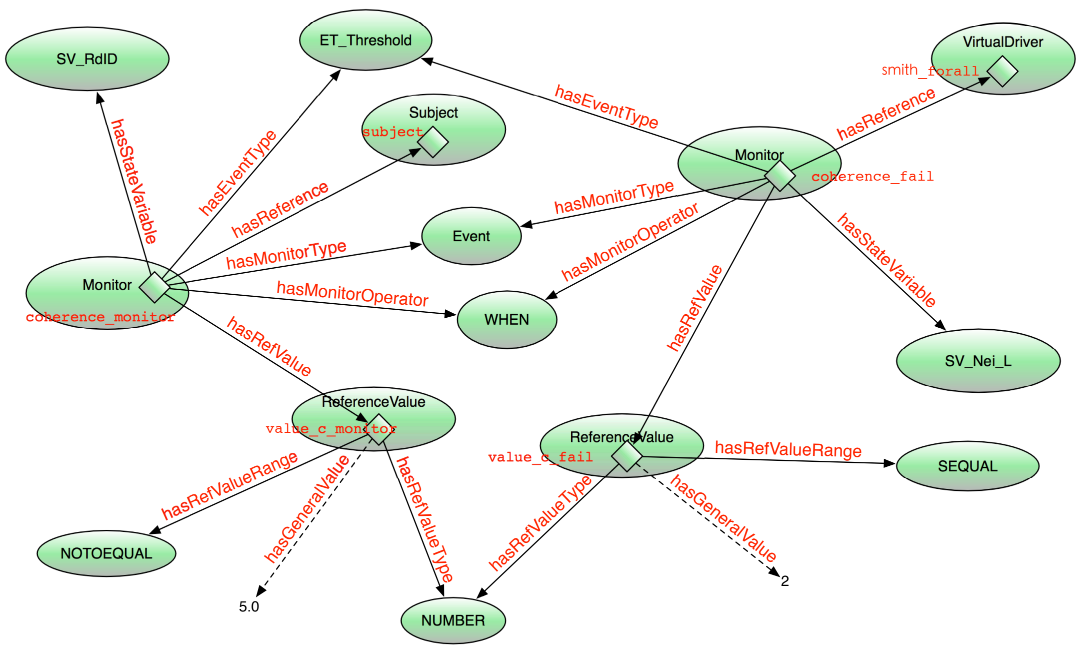
This Assignment should be carried out by Smith, not SMM. As a result, *smith_forall* was used, i.e., *isPerformedBy smith_forall*

C.1.2.6 Representation by Directed Graph

The main information of this Assignment can be illustrated in Figure C.2. This Assignment can be tried once (*hasMaxtriedTime 1*)



(a)



(b)

Figure C.2: Illustration of *coherence_2* in Experiment One

C.1.3 Assignment "Layby"

As usual, an individual of the class *Assignment* named *layby_3* was created in OSO, containing the following main information:

C.1.3.1 Formation Position

This Assignment needed a nearside leader, so the Formation Position should be "LeftLeader0", i.e., *hasFormationPosition* some *LeftLeader*0.

C.1.3.2 Monitor

"When the participant's vehicle's(*subject*) time headway to its nearside leader (*SV_TimeHW_NSL*), which is the Ego-vehicle, changes from greater than, to less than 3 seconds."

An individual of the class *ReferenceValue* named *value_layby_monitor* was also created to represent the reference value mentioned, which is "greater than, to less than 3 seconds". Its properties are shown in Table C.13.

Table C.13: Properties of *value_layby_monitor* for "Layby" in Experiment One

Property	**Related to**
hasRefValueType	some *NUMBER*
hasRefValueRange	some *SEQUAL* ($\leqslant$)
hasGeneralValue	3.0

An individual of the class *Monitor* named *layby_monitor* was created to represent this Monitor. Its properties are shown in Table C.14.

C.1.3.3 Failure Condition

"When the Ego-vehicle's (*smith_forall*) leader's ID (*SV_Nei_L*) changes to 0, which represents the participant's vehicle's ID."

An individual of the class *ReferenceValue* named *value_layby_fail* was created to represent the reference value mentioned, which is "changes to 0". Its properties are shown in Table C.15.

Table C.14: Properties of *layby_monitor* for “Layby” in Experiment One

Property	**Related to**
hasMonitorType	some *Event*
hasEventType	some *ET_Threshold*
hasMonitorOperator	some *WHEN*
hasReference	*subject*
hasStateVariable	some *SV_TimeHW_NSL*
hasRefValue	*value_layby_monitor*

Table C.15: Properties of *value_layby_fail* for “Layby” in Experiment One

Property	**Related to**
hasRefValueType	some *NUMBER*
hasRefValueRange	some *NOTOEQUAL*
hasGeneralValue	0

An individual of the class *Monitor* named *layby_fail* was created to represent this Failure Condition. Its properties are shown in Table C.16.

Table C.16: Properties of *layby_fail* for “Layby” in Experiment One

Property	**Related to**
hasMonitorType	some *Event*
hasEventType	some *ET_Threshold*
hasMonitorOperator	some *WHEN*
hasReference	*smith_forall*
hasStateVariable	some *SV_Nei_L*
hasRefValue	*value_layby_fail*

C.1.3.4 Action

When the Monitor became true, Smith would force the Ego-vehicle to change into the participant's lane (*offside* to the Ego-vehicle) with an acceleration rate of 5 m/s^2 and a target speed of 13.3 m/s. This Assignment lasted for 45 seconds, which meant that Smith stayed idle for 45 seconds. An individual of the class *Action* named *layby* was created, which specified the acceleration rate, target speed and target lane. Its properties are shown in Table C.17.

Table C.17: Properties of *layby* for "Layby" in Experiment One

Property	**Related to**
hasActionProfile	some *TargetLane*
hasActionType	some *GotoLN*
hasActionSpeed	13.3
hasActionAcc	5.0
hasActionDuration	45
hasActionLane	offside

C.1.3.5 Performer

This Assignment should be carried out by Smith, not SMM. As a result, *smith_forall* is used, , i.e., *isPerformedBy smith_forall*

C.1.3.6 Representation by Directed Graph

The main information of this Assignment can be illustrated in Figure C.3:

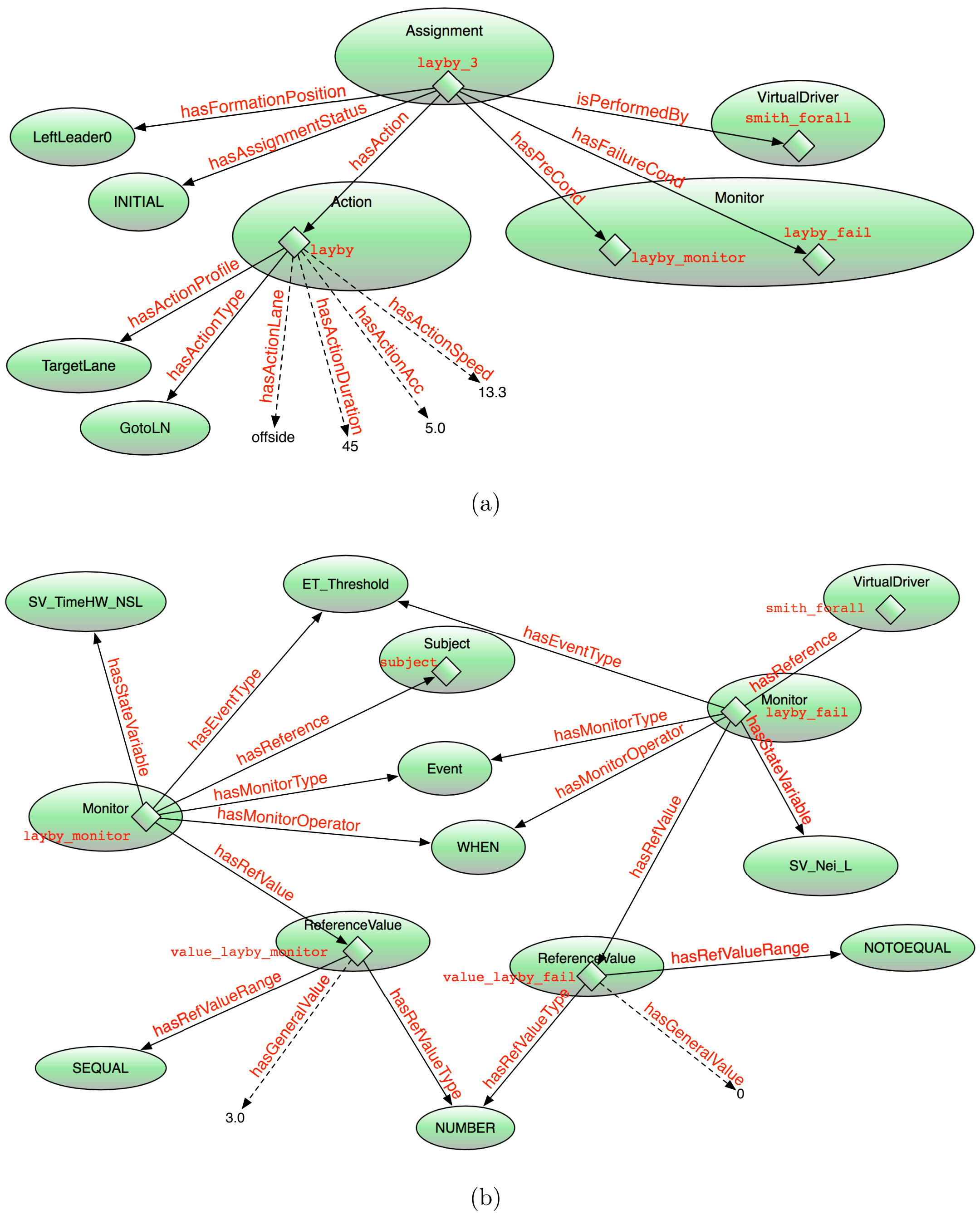


Figure C.3: Illustration of *layby_3* in Experiment One

C.1.4 Other Assignments

Moreover, two individuals of the class *Assignment* named *gap_tf_4* and *free_tf_5* were created to represent the Assignment “Gap Acceptance” and “Free Traffic Flow” respectively. Because both traffic flows used in these two Assignments have been implemented in Sim1 prior to this experiment, Smith simply triggered these two traffic flows via pre-defined indicators that were fed to relevant controllers. In this experiment, the indicators were “low” and “gap”, which corresponded to Flock No. 4 and Flock No. 5 respectively. “Gap Acceptance” was triggered after the road segment with the ID “r10.1” and after “Layby”. “Free Traffic Flow” was triggered along with “Coherence” and stopped with “Layby”.

C.1.5 Temporal Constraints

Precedence constraints were specified as (Figure C.4):

$$\begin{aligned}
coherence\_beleader\_1 &\quad IntervalBefore \quad coherence\_2;\\
coherence\_beleader\_1 &\quad IntervalBefore \quad free\_tf\_5;\\
coherence\_2 &\quad IntervalBefore \quad layby\_3;\\
layby\_3 &\quad IntervalBefore \quad gap\_tf\_4;\\
layby\_3 &\quad IntervalFinishes \quad free\_tf\_5;
\end{aligned}$$

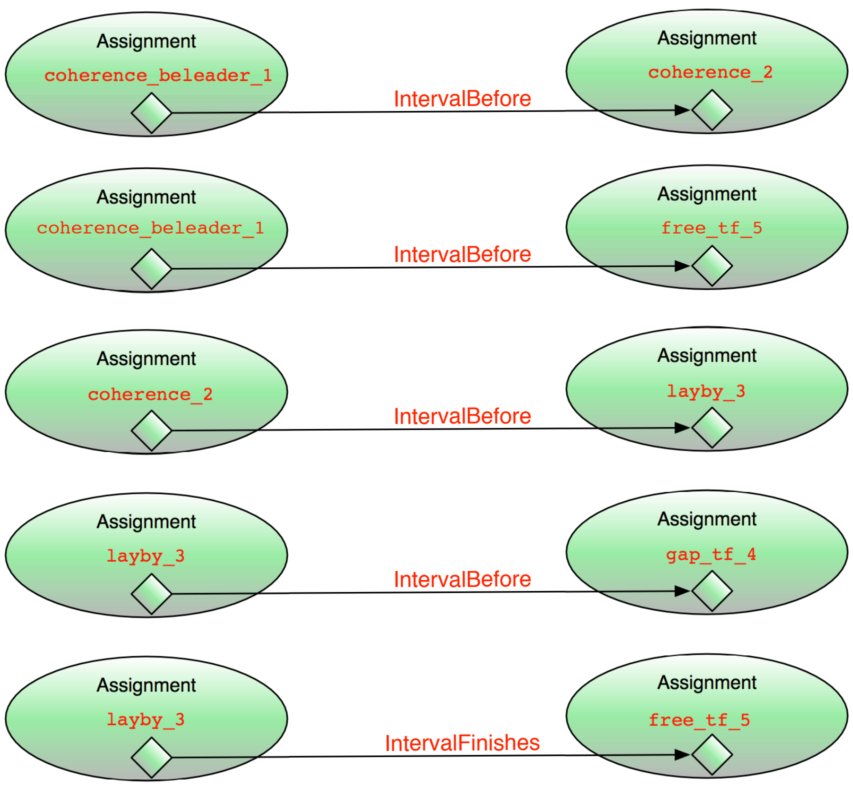


Figure C.4: Metric Constraints for Experiment One

Metric constraints were estimated by considering the maximum or minimum time spent for participants to reach the places where Assignments would take place. Minimum and maximum speeds were estimated by taking into account the speed limit. The minimum speed was the speed limit in villages, which was 30 *mph*; the maximum speed was the speed limit on open roads, which was 60 *mph*. A set of metric constraints could be specified as:

$$117.6 \leqslant s_{coherence\_2} - s_{alpha} \leqslant 255.0 \tag{C.1}$$

$$614.5 \leqslant f_{beta2} - s_{coherence\_2} \leqslant 1527.8 \tag{C.2}$$

$$414.37 \leqslant s_{layby\_3} - s_{coherence\_2} \leqslant 1051.12 \tag{C.3}$$

$$105.78 \leqslant s_{gap\_tf\_4} - s_{layby\_3} \leqslant 1015.13 \tag{C.4}$$

Which state that

- Constraint C.1: the start time of the Assignment "Coherence" (*coherence_*2) minus the start time of the simulation (i.e., the start time of top-Action α (*alpha*)) should be greater than 117.6 seconds, but less than 255.0 seconds;

- Constraint C.2: the finish time of all the Assignments (i.e., the finish time of pre-defined Action β_2 (*beta*2)) minus the start time of the Assignment "Coherence" (*coherence_*2) should be greater than 614.5 seconds, but less than 1527.8 seconds;

- Constraint C.3: the start time of the Assignment "Layby" (*layby_*3) minus the start time of the Assignment "Coherence" (*coherence_*2) should be greater than 414.37 seconds, but less than 1051.12 seconds;

- Constraint C.4: the start time of the Assignment "Gap Acceptance" (*gap_tf_4*) minus the start time of the Assignment "Layby" (*layby_*3) should be greater than 105.78 seconds, but less than 1015.13 seconds.

The individuals created in OSO based on the information above are illustrated in Figure C.5:

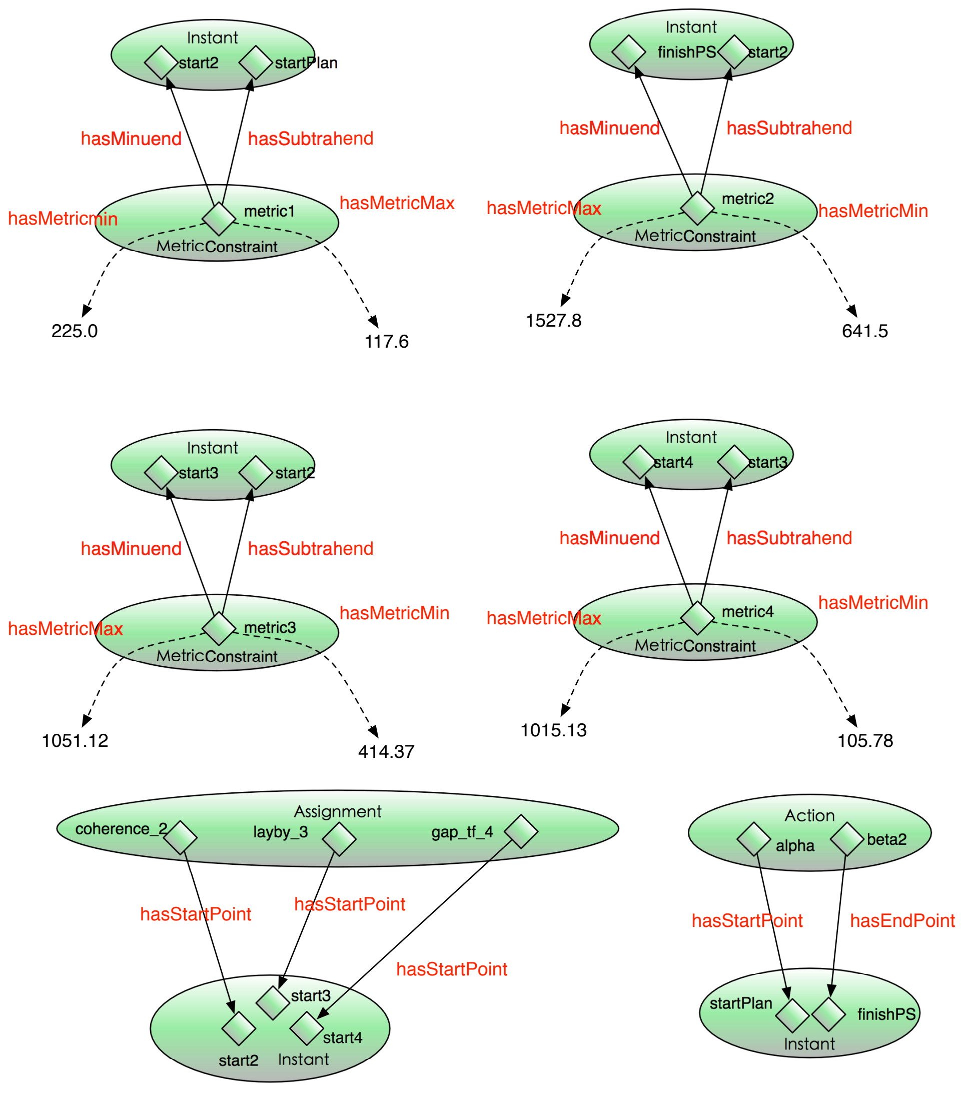


Figure C.5: Illustration of Metric Constraints for Experiment Two in OSO

C.2 Experiment Two

C.2.1 Assignment "Overtaking-lorry"

An individual of the class *Assignment* named *smith_overtake_1_3* was created in OSO, containing the following main information:

C.2.1.1 Formation Position

This Assignment needed a leader, so the Formation Position should be "Leader", i.e., *hasFormationPosition* some *Leader*.

C.2.1.2 Vehicle Restriction

This Assignment required a truck to be the Ego-vehicle, so the vehicle restriction should be specified. As a result, an individual of the class *SingleVehicle* named *overtake_truck* was created in order to specify that *smith_overtake_1_3* *hasVehicleRestriction overtake_truck*.

overtake_truck should be a LGV, OGV1 or OGV2 as specified in OSO, so *overtake_truck* should be related to some individual of the class *LGV*, *OGV1* or *OGV2*, , i.e., *overtake_truck hasVehicleRestriction* some *OGV1* or *OGV2* or *LGV*.

C.2.1.3 Monitor

There were two Monitors for this Assignment. When they were met at the same time, the Assignment-action would be triggered.

C.2.1.3.1 Monitor One

"**Whenever the Ego-vehicle's (*smith_forall*) time headway to the participant's vehicle's (*SV_TimeHW*) is less than 4 seconds.**"

An individual of the class *ReferenceValue* named *value_1_3_monitor_hw* was created to represent the reference value mentioned, which is "less than 4 seconds". Its properties are shown in Table C.18.

An individuals of the class *Monitor* named *monitor_1_3_hw* was created to represent this Monitor. Its properties are shown in Table C.19.

Table C.18: Properties of *value_1_3_monitor_hw* for "Overtaking-lorry" in Experiment Two

Property	**Related to**
hasRefValueType	some *NUMBER*
hasRefValueRange	some *SEQUAL* ($\leqslant$)
hasGeneralValue	4.0

Table C.19: Properties of *monitor_1_3_hw* for "Overtaking-lorry" in Experiment Two

Property	**Related to**
hasMonitorType	some *State*
hasMonitorOperator	some *WHENEVER*
hasReference	*smith_forall*
hasStateVariable	some *SV_TimeHW*
hasRefValue	*value_1_3_monitor_hw*

C.2.1.3.2 Monitor Two

"**Whenever the Ego-vehicle's (*smith_forall*) time-to-collision with the the participant's vehicle (*SV_TTC*) is greater than 6 seconds.**"

An individual of the class *ReferenceValue* named *value_1_3_monitor* was created to represent the reference value mentioned, which is "greater than 6 seconds". Its properties are shown in Table C.20.

Table C.20: Properties of *value_1_3_monitor* for "Overtaking-lorry" in Experiment Two

Property	**Related to**
hasRefValueType	some *NUMBER*
hasRefValueRange	some *BEQUAL* ($\geqslant$)
hasGeneralValue	6.0

An individuals of the class *Monitor* named *monitor_1_3* was created to represent this Monitors. Its properties are shown in Table C.21.

Table C.21: Properties of *monitor_1_3* for "Overtaking-lorry" in Experiment Two

Property	**Related to**
hasMonitorType	some *State*
hasMonitorOperator	some *WHENEVER*
hasReference	*smith_forall*
hasStateVariable	some *SV_TTC*
hasRefValue	*value_1_3_monitor*

C.2.1.4 Action

When the Monitors became true, an Action of "setting the overtaking intention to true" was triggered. SMM interpreted this Action and forced the Ego-vehicle to adopt a higher desired speed (28.1 m/s) and kept for 80 seconds in order to guarantee that the overtaking could be finished.

An individual of the class *Action* named *overtake_1_3* was created, which specified that the overtaking intention should be true. Its properties are shown in Table C.22.

Table C.22: Properties of *overtake_1_3* for "Overtaking-lorry" in Experiment Two

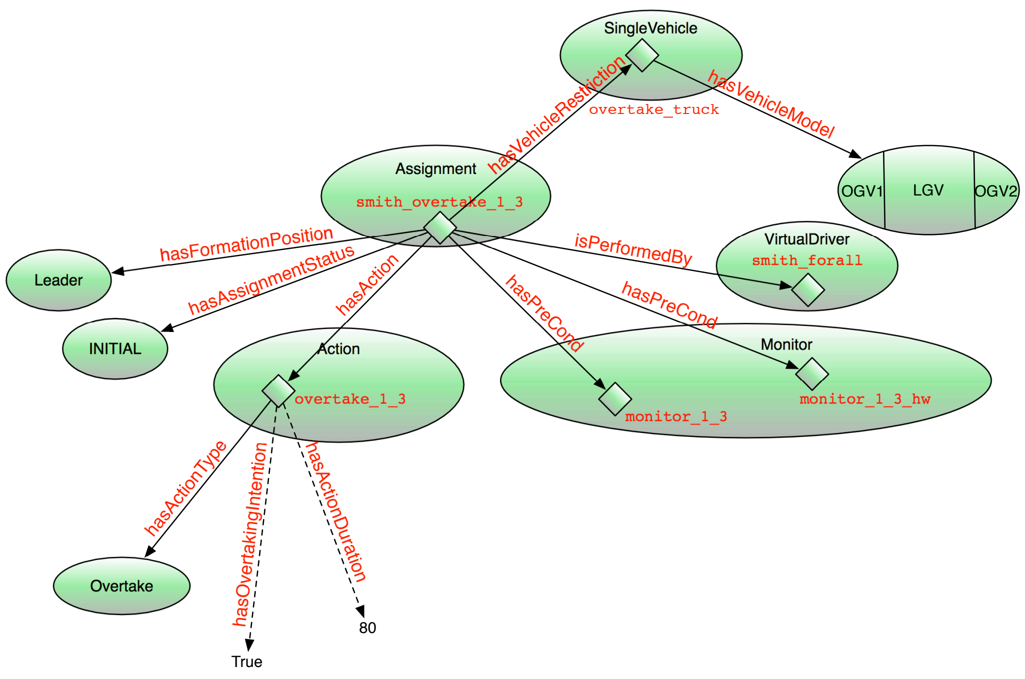
Property	**Related to**
hasActionType	some *Overtake*
hasOvertakingIntention	True
hasActionDuration	80.0

C.2.1.5 Performer

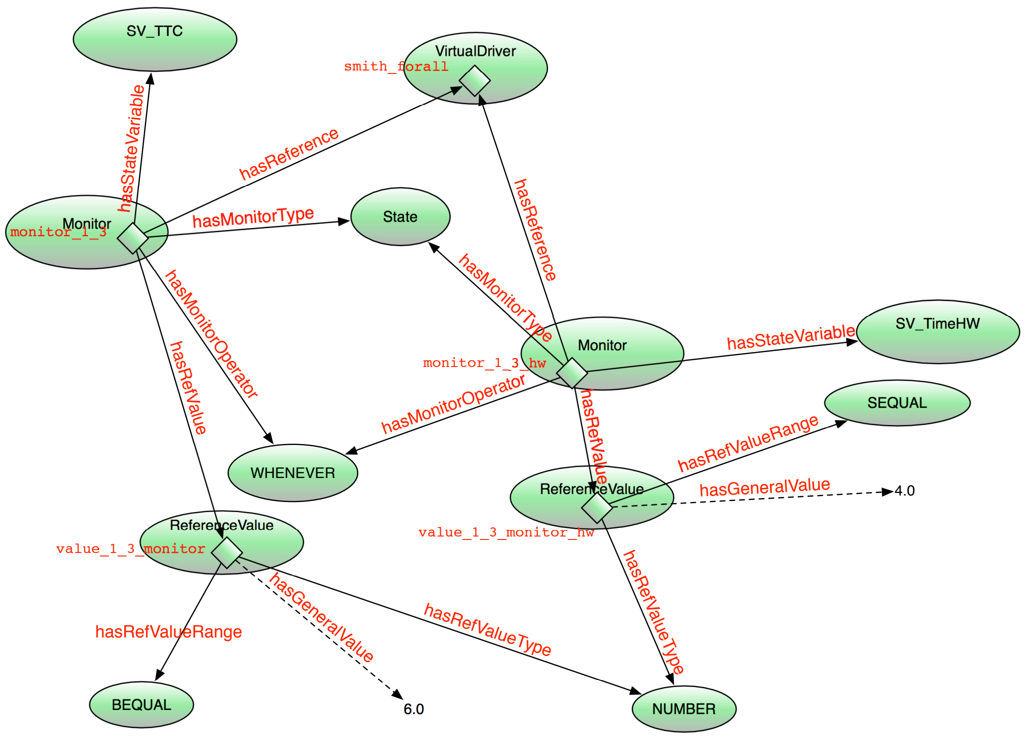
This Assignment should be carried out by Smith, not SMM. Hence, *smith_forall* was included in this Assignment and was an individual of the class $VirutalDriver$, i.e., *isPerformedBy smith_forall*

C.2.1.6 Representation by Directed Graph

By adopting the individuals mentioned above and some other information, e.g., initial status of Assignment *coherence_beleader*_1, the main information of this Assignment can be illustrated as in Figure C.6. This Assignment can be tried once (*hasMaxtriedTime 1*)



(a)



(b)

Figure C.6: Illustration of *smith_overtake_1_3* in Experiment Two

C.2.2 Assignment “Broken-down-car”

First of all, an individual of the class *Assignment* named *smith_breakdown_2_5* was created in OSO, containing the following main information:

C.2.2.1 Formation Position

This Assignment required a leader’s leader, so the Formation Position should be “CloseLeader0”, i.e., *hasFormationPosition* some *CloseLeader*0.

C.2.2.2 Monitor

There were three Monitors for this Assignment. When they were met at the same time, the Assignment-action would be triggered.

C.2.2.2.1 Monitor One

“As long as the participant (*subject*) passes or is on the road segment with an ID (*SV_RdID*) of ‘r4.0’. ”

An individual of the class *ReferenceValue* named *value_2_5_monitor_road* was created to represent the reference value mentioned, which is “passes or is on the road segment whose ID is ‘r4.0’ ”. Its properties are shown in Table C.23.

Table C.23: Properties of *value_2_5_monitor_road* for “Broken-down-car” in Experiment Two

Property	**Related to**
hasRefValueType	some *NUMBER*
hasRefValueRange	some *BEQUAL* ($\geqslant$)
hasGeneralValue	4.0

An individuals of the class *Monitor* named *monitor_2_5_road* was created to represent this Monitor. Its properties are shown in Table C.24.

Table C.24: Properties of *monitor_2_5_road* for "Broken-down-car" in Experiment Two

Property	**Related to**
hasMonitorType	some *State*
hasMonitorOperator	some *ASLONGAS*
hasReference	*subject*
hasStateVariable	some *SV_RdID*
hasRefValue	*value_2_5_monitor_road*

C.2.2.2.2 Monitor Two

"Whenever the participant's (*subject*) time headway to its leader (*SV_TimeHW_L*) is less than 4 seconds."

An individual of the class *ReferenceValue* named *value_1_3_monitor_hw* was used to represent the reference value mentioned, which is "less than 4 seconds". It was adopted from previous Assignment as shown in Table C.18 on Page 258.

An individuals of the class *Monitor* named *monitor_2_5_hw* was created to represent this Monitor. Its properties are shown in Table C.25.

Table C.25: Properties of *monitor_2_5_hw* for "Broken-down-car" in Experiment Two

Property	**Related to**
hasMonitorType	some *State*
hasMonitorOperator	some *WHENEVER*
hasReference	*subject*
hasStateVariable	some *SV_TimeHW_L*
hasRefValue	*value_1_3_monitor_hw*

C.2.2.2.3 Monitor Three

"Whenever the participant's (*subject*) time-to-collision with its leader (*SV_TTC_L*) is greater than 5 seconds."

An individual of the class *ReferenceValue* named *value_2_5_monitor* was created to represent the reference value mentioned, which is “greater than 5 seconds”. Its properties are shown in Table C.26.

Table C.26: Properties of *value_2_5_monitor* for “Broken-down-car” in Experiment Two

Property	**Related to**
hasRefValueType	some *NUMBER*
hasRefValueRange	some *BEQUAL* ($\geqslant$)
hasGeneralValue	5.0

An individuals of the class *Monitor* named *monitor_2_5* was created to represent this Monitor. Its properties are shown in Table C.27.

Table C.27: Properties of *monitor_2_5* for “Broken-down-car” in Experiment Two

Property	**Related to**
hasMonitorType	some *State*
hasMonitorOperator	some *WHENEVER*
hasReference	*subject*
hasStateVariable	some *SV_TTC_L*
hasRefValue	*value_2_5_monitor*

C.2.2.3 Action

When the Monitors became true, an Action of ”set the desired speed to 0” was triggered, in which case, Smith would force the Ego-vehicle to reach a speed of 0.

An individual of the class *Action* named *breakdown_2_1* was created, which specified that the Action speed should be 0 mph and the whole process should last for 70 seconds in order to block the lane for that period of time. Its properties are shown in Table C.28.

Table C.28: Properties of *breakdown_2_1* for "Broken-down-car" in Experiment Two

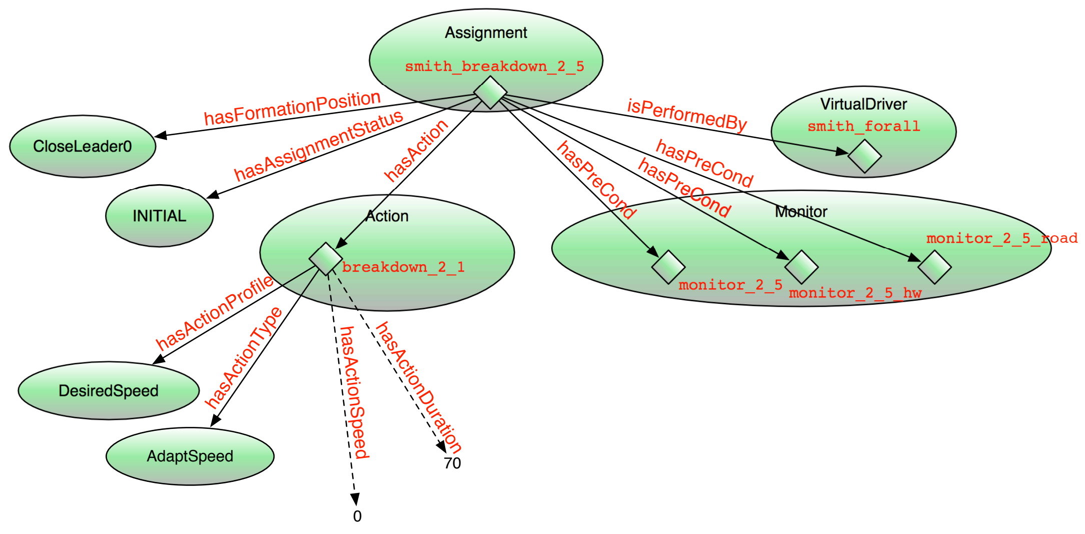
Property	**Related to**
hasActionProfile	some *DesiredSpeed*
hasActionType	some *AdaptSpeed*
hasActionSpeed	0
hasActionDuration	70

C.2.2.4 Performer

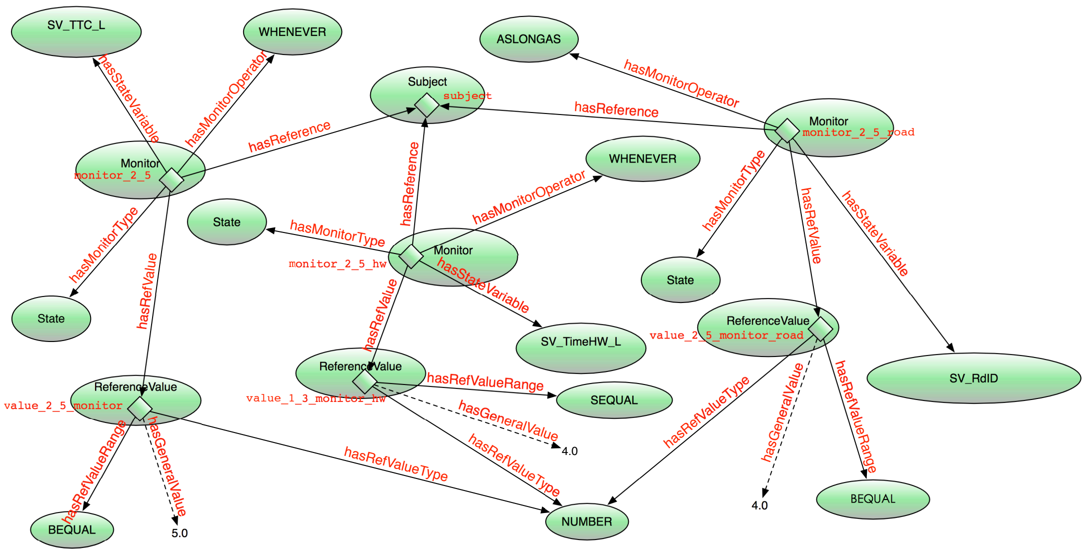
This Assignment should be carried out by Smith, not SMM. *smith_forall* is therefore used, i.e., *isPerformedBy smith_forall*

C.2.2.5 Representation by Directed Graph

By adopting the individuals mentioned above and some other information, e.g., initial status of the Assignment *coherence_beleader_1*, the main information of this Assignment can be illustrated as in Figure C.7. This Assignment can be tried once (*hasMaxtriedTime 1*)



(a)



(b)

Figure C.7: Illustration of *smith_breakdown_2_5* in Experiment Two

C.2.3 Assignment "Cone-off-road"

An individual of the class *Assignment* named *smm_setmodelswitch_3_8* was created in OSO, containing the following main information:

C.2.3.1 Monitor

"As long as the participant (*subject*) reaches or passes the road segment whose ID (*SV_RdID*) is 'r5.2'."

An individual of the class *ReferenceValue* named *value_3_2_monitor* was created to represent the reference value mentioned, which is "reaches or passes the road segment whose ID is 'r5.2' ". Its properties are shown in Table C.29.

Table C.29: Properties of *value_3_2_monitor* for "Cone-off-road" in Experiment Two

Property	**Related to**
hasRefValueType	some *NUMBER*
hasRefValueRange	some *BEQUAL*
hasGeneralValue	5.2

An individual of class *Monitor* named *monitor_3_2* was created to represent this Monitor. Its properties are shown in Table C.30.

Table C.30: Properties of *monitor_3_2* for "Cone-off-road" in Experiment Two

Property	**Related to**
hasMonitorType	some *State*
hasMonitorOperator	some *ASLONGAS*
hasReference	*subject*
hasStateVariable	some *SV_RdID*
hasRefValue	*value_3_2_monitor*

C.2.3.2 Action

When the Monitor became true, an Action of "cone off the specified lane" would be triggered, in which case, Smith would request SMM to cone off a lane. An individual of the class *Action* named *setmodelswitch_3_2* was created, which indicated that SMM should cone off the lane where the participant's vehicle was not travelling in. Its properties are shown in Table C.31.

Table C.31: Properties of *setmodelswitch_3_2* for "Cone-off-road" in Experiment Two

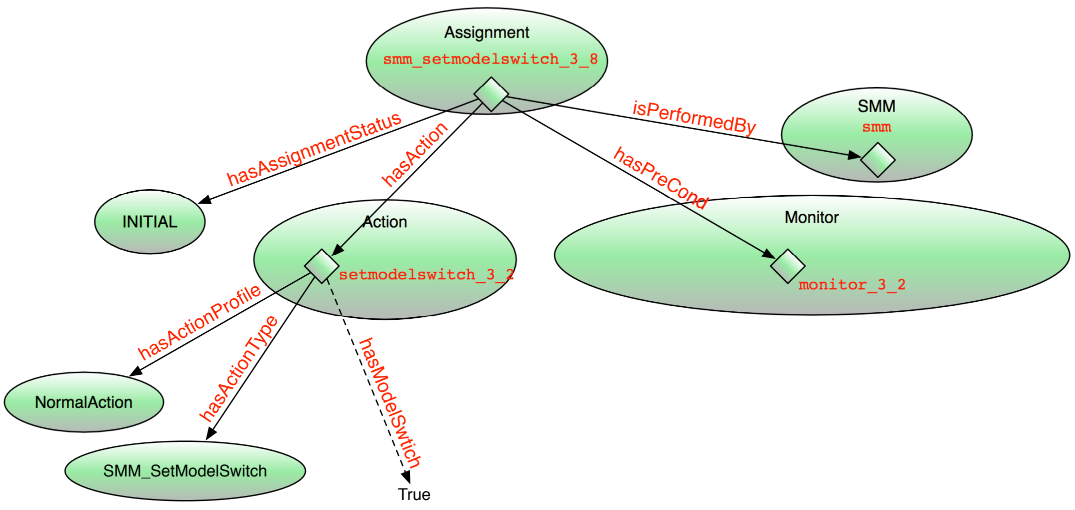
Property	**Related to**
hasActionProfile	some *NormalAction*
hasActionType	some *SMM_SetModelSwith*
hasModelSwitch	True

C.2.3.3 Performer

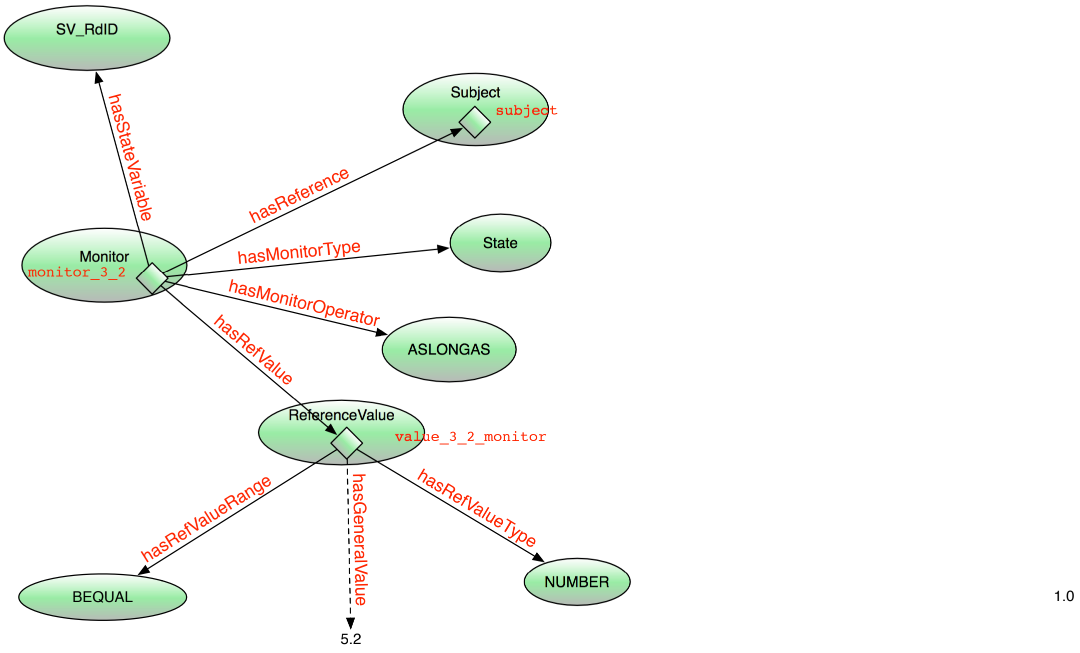
This Assignment should be carried out by SMM, but requested by Smith. Hence, the performer is an individual of class *SMM* named *smm*, i.e., *isPerformedBy smm*

C.2.3.4 Representation by Directed Graph

By adopting the individuals mentioned above and some other information, e.g., initial status of Assignment *coherence_beleader_1*, the main information of this Assignment can be illustrated as in Figure C.8. This Assignment can be tried once (*hasMaxtriedTime 1*)



(a)



(b)

Figure C.8: Illustration of *smm_setmodelswitch_3_8* in Experiment Two

C.2.4 Assignment "Braking-car"

An individual of the class *Assignment* named *smith_stop_4_10* was created in OSO, containing the following main information:

C.2.4.1 Formation Position

This Assignment needed a leader, so the Formation Position should be "Leader", i.e., *hasFormationPosition* some *Leader*.

C.2.4.2 Monitor

There were three Monitors for this Assignment. When they were met at the same time, the Assignment-action would be triggered.

C.2.4.2.1 Monitor One

"Whenever the participant's vehicle's (*subject*) time headway to the leader (*SV_TimeHW_L*), which is the Ego-vehicle, is less than 4 seconds."

This Monitor utilised the one from Assignment as shown in Table C.25 on Page 263.

C.2.4.2.2 Monitor Two

"As long as the participant (*subject*) passes to is on the road 'r6.0' (*SV_RdID*)."

An individual of the class *ReferenceValue* named *value_4_1_monitor_road* was created to represent the reference value mentioned, which is "passes or is on the road segment whose ID was 'r6.0' ". Its properties are shown in Table C.32.

An individuals of the class *Monitor* named *monitor_4_1_road* was created to represent this Monitor. Its properties are shown in Table C.33.

C.2.4.2.3 Monitor Three

"Whenever the participant's vehicle's (*subject*) time-to-collision with the Ego-vehicle, which should be the leader (*SV_TTC_L*), is greater than 5 seconds."

Table C.32: Properties of *value_4_1_monitor_road* for “Braking-car” in Experiment Two

Property	**Related to**
hasRefValueType	some *NUMBER*
hasRefValueRange	some *BEQUAL* (⩾)
hasGeneralValue	6.0

Table C.33: Properties of *monitor_4_1_road* for “Braking-car” in Experiment Two

Property	**Related to**
hasMonitorType	some *State*
hasMonitorOperator	some *ASLONGAS*
hasReference	*subject*
hasStateVariable	some *SV_RdID*
hasRefValue	*value_4_1_monitor_road*

An individual of the class *ReferenceValue* named *value_4_1_monitor* was created to represent the reference value mentioned, which is “greater than 5 seconds”. Its properties are shown in Table C.34.

Table C.34: Properties of *value_4_1_monitor* for “Braking-car” in Experiment Two

Property	**Related to**
hasRefValueType	some *NUMBER*
hasRefValueRange	some *BEQUAL* (⩾)
hasGeneralValue	5.0

An individuals of the class *Monitor* named *monitor_4_1* was created to represent this Monitor. Its properties are shown in Table C.35.

Table C.35: Properties of *monitor_4_1* for “Braking-car” in Experiment Two

Property	**Related to**
hasMonitorType	some *State*
hasMonitorOperator	some *WHENEVER*
hasReference	*subject*
hasStateVariable	some *SV_TTC_L*
hasRefValue	*value_4_1_monitor*

C.2.4.3 Action

When the Monitors became true, an Action of “decelerate to speed of 0 with a deceleration rate of -5.5 m/s^2” was triggered, in which case Smith would force the Ego-vehicle to reach a speed of 0 with the specified deceleration rate. An individual of the class *Action* named *stop_4_1* was created, which specified that the desired acceleration rate should be -5.5 m/s^2 and hold for 20 seconds. Its properties are shown in Table C.36.

Table C.36: Properties of *stop_4_1* for “Braking-car” in Experiment Two

Property	**Related to**
hasActionProfile	some *DesiredAccRate*
hasActionType	some *AdaptAcc*
hasActionSpeed	-5.5
hasActionDuration	20.0

C.2.4.4 Performer

This Assignment should be carried out by Smith, not SMM. *smith_forall* is used, i.e., *isPerformedBy smith_forall*

C.2.4.5 Representation by Directed Graph

By adopting the individuals mentioned above and some other information, e.g., initial status of the Assignment *coherence_beleader_1*, the main information of this Assignment can be illustrated as in Figure C.9. This Assignment can be tried once (*hasMaxtriedTime 1*)

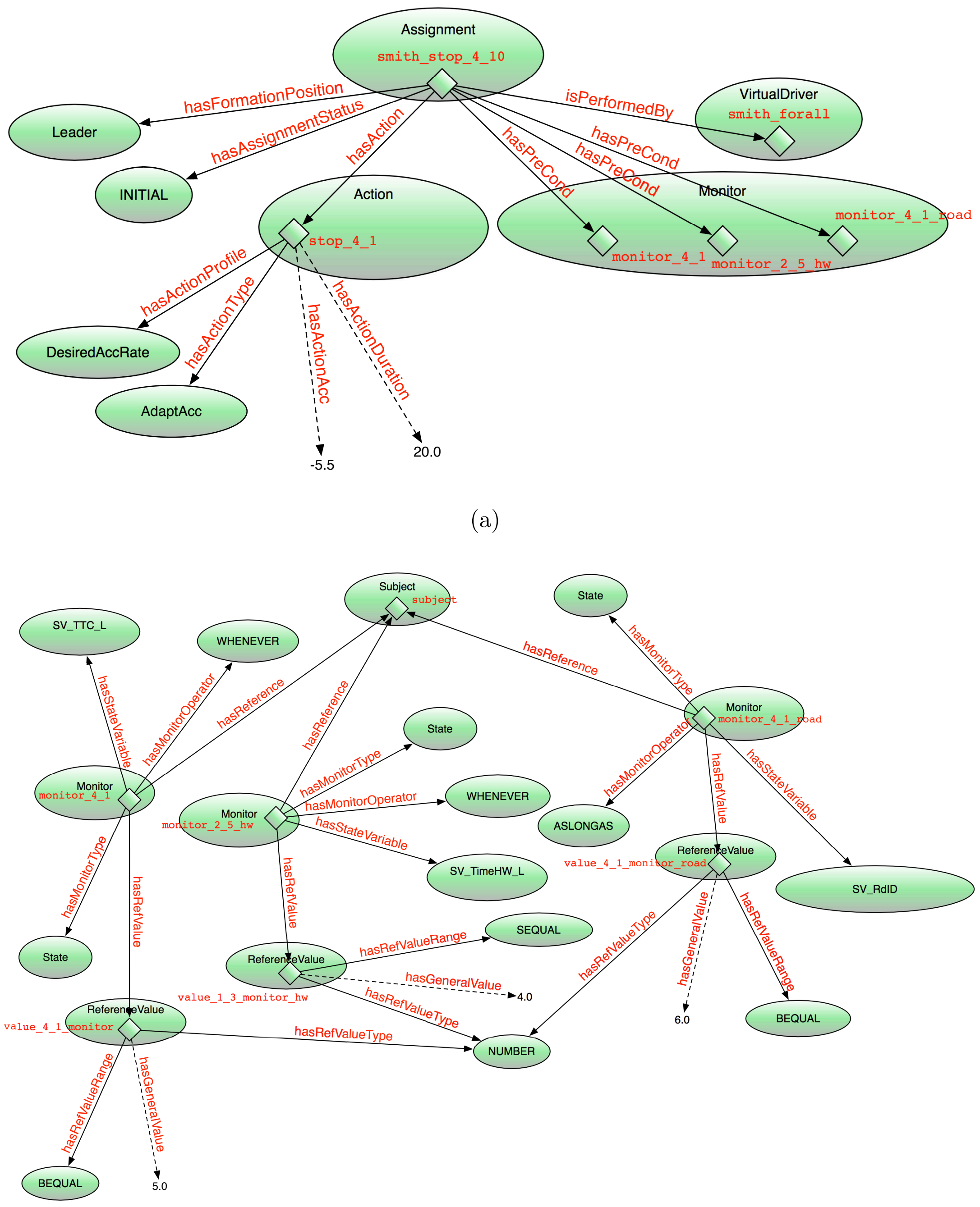


Figure C.9: Illustration of *smith_stop_4_10* in Experiment Two

C.3 Experiment Three

C.3.1 Assignment “Braking-car”

An individual of the class *Assignment* named *smith_dec_1_1* was created in OSO, containing the following main information:

C.3.1.1 Formation Position

This Assignment needed a leader, so the Formation Position should be “Leader”, i.e., *hasFormationPosition* some *Leader*.

C.3.1.2 Monitor

There were two Monitors for this Assignment. When they were met at the same time, the Assignment-action would be triggered.

C.3.1.2.1 Monitor One

“As long as the simulator driver (*subject*) reaches or passes 11000 *m* of the road segment (*SV_Distance*).”

An individual of the class *ReferenceValue* named *value_1_1_monitor_road* was created to represent the reference value mentioned, which is “reaches or passes 11000 *m*”. Its properties are shown in Table C.37.

Table C.37: Properties of *value_1_1_monitor_road* for “Braking-car” in Experiment Three

Property	**Related to**
hasRefValueType	some *NUMBER*
hasRefValueRange	some *BEQUAL* ($\geqslant$)
hasGeneralValue	11000

An individuals of the class *Monitor* named *monitor_road_1_1* was created to represent this Monitor. Its properties are shown in Table C.38.

Table C.38: Properties of *monitor_road_1_1* for “Braking-car” in Experiment Three

Property	**Related to**
hasMonitorType	some *State*
hasMonitorOperator	some *ASLONGAS*
hasReference	*subject*
hasStateVariable	some *SV_Distance*
hasRefValue	*value_1_1_monitor_road*

C.3.1.2.2 Monitor Two

“Whenever the simulator driver’s (*subject*) distance headway to the leader (*SV_SpaceHW*), which is the Ego-vehicle, is less than 200 meters.”

An individual of the class *ReferenceValue* named *value_1_1_monitor* was created to represent the reference value mentioned, which is “less than 200 meters”. Its properties are shown in Table C.39.

Table C.39: Properties of *value_1_1_monitor* for “Braking-car” in Experiment Three

Property	**Related to**
hasRefValueType	some *NUMBER*
hasRefValueRange	some *SEQUAL* ($\leqslant$)
hasGeneralValue	200

An individuals of the class *Monitor* named *monitor_1_1* was created to represent this Monitor. Its properties are shown in Table C.40.

C.3.1.3 Action

When the Monitor became true, an Action of “decelerate to a speed of 0 with a deceleration rate of -1 m/s^2” should be triggered and last for 18 seconds.

Table C.40: Properties of *monitor_1_1* for "Braking-car" in Experiment Three

Property	**Related to**
hasMonitorType	some *State*
hasMonitorOperator	some *WHENEVER*
hasReference	*subject*
hasStateVariable	some *SV_SpaceHW*
hasRefValue	*value_1_1_monitor*

An individual of the class *Action* named *dec_1_1* was created, which specified that the desired acceleration rate should be -1 m/s^2 and the Duration of this Action should be 18 seconds. Its properties are shown in Table C.41.

Table C.41: Properties of *dec_1_1* for "Braking-car" in Experiment Three

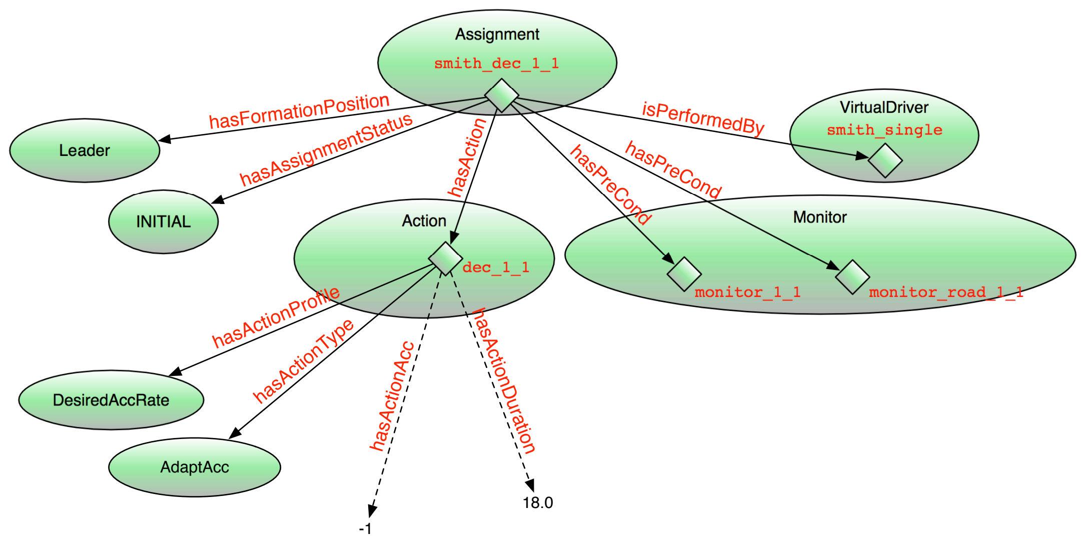
Property	**Related to**
hasActionProfile	some *DesiredAccRate*
hasActionType	some *AdaptAcc*
hasActionAcc	-1
hasActionDuration	18.0

C.3.1.4 Performer

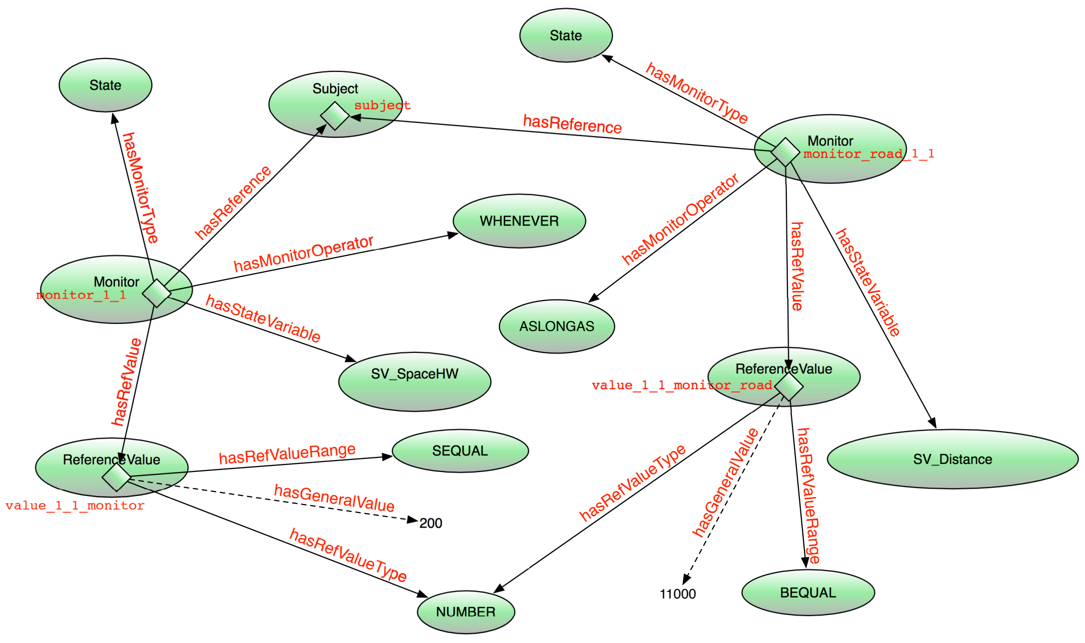
This Assignment should be carried out by Smith, not SMM, so an individual of the class *VirtualDriver* named *smith_single* was created to represent this Smith, i.e., *isPerformedBy smith_single*

C.3.1.5 Representation by Directed Graph

By adopting the individuals mentioned above and some other information, e.g., the initial status of the Assignment, the main information of this Assignment can be illustrated as in Figure C.12. This Assignment can be tried once (*hasMax-triedTime 1*)



(a)



(b)

Figure C.10: Illustration of *smith_dec_1_1* in Experiment Three

C.3.2 Assignment "Flock-Blocking"

An individual of the class *Assignment* named *smith_flock_1_2* was created in OSO, which should contain the following main information:

C.3.2.1 Formation Position

This Assignment requires a parallel vehicle to block the driver, so the Formation Position should be $LeftParallel0$ or $RightParallel0$, i.e., *hasFormationPosition* some $LeftParallel0$ or $RightParallel0$.

C.3.2.2 Monitor

This Assignment shared the same Monitors as the last one.

C.3.2.3 Action

A High-Level Action called "Block" was needed (*Block*), in which case Smith needed to use a corresponding Recipe in Memory to perform this Action.

An individual of class *Action* named *flk_2_5* was created to represent this Action. Its properties are shown in Table C.42.

Table C.42: Properties of *flk_2_5* for "Flock-Blocking" in Experiment Three

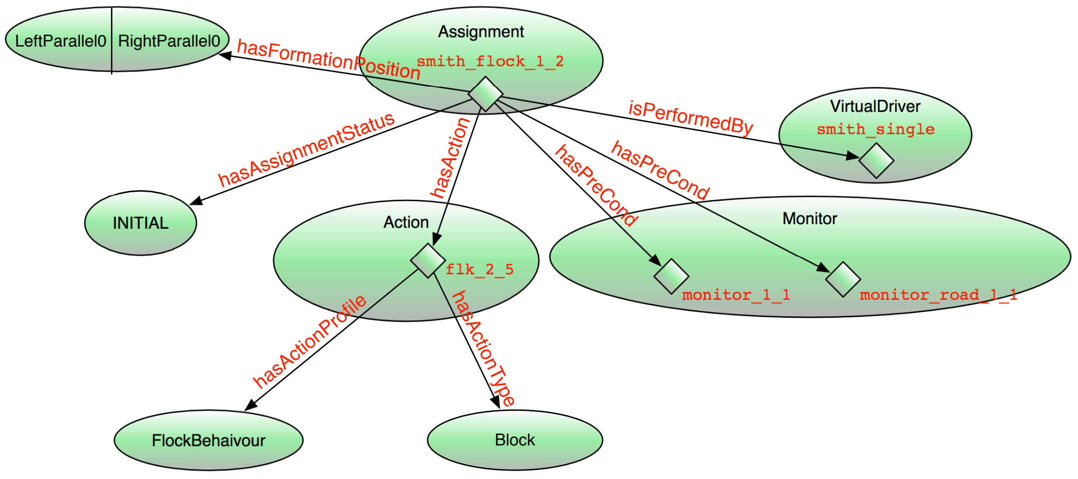
Property	**Related to**
hasActionProfile	some *FlockBehaviour*
hasActionType	some *Block*

C.3.2.4 Performer

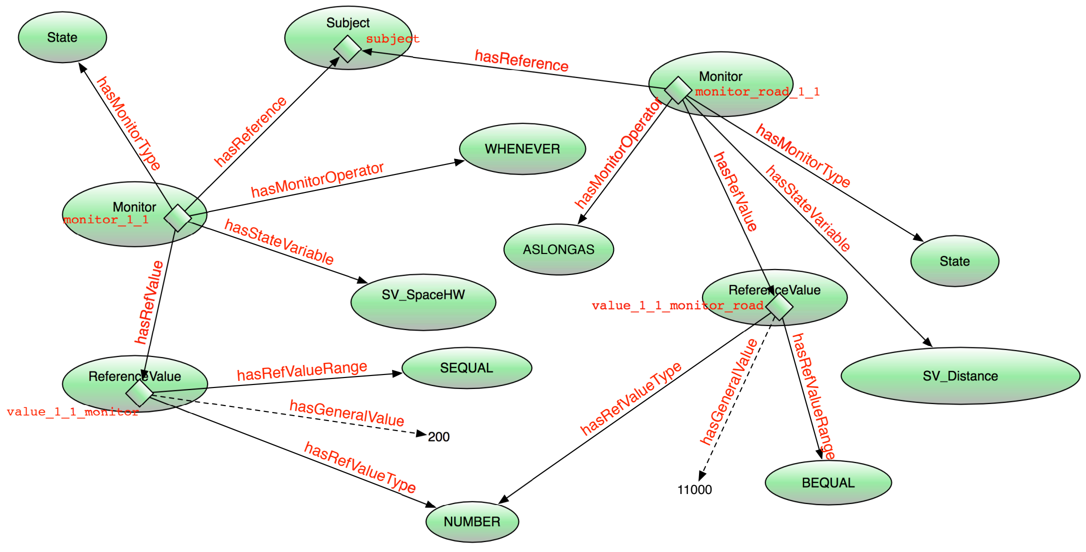
This Assignment should be carried out by Smith, not SMM, so *smith_single* was used, i.e., *isPerformedBy smith_single*

C.3.2.5 Representation by Directed Graph

By adopting the individuals mentioned above and some other information, e.g., the initial status of Assignment, the main information of this Assignment can be illustrated as in Figure C.11. This Assignment can be tried once (*hasMaxtriedTime 1*)



(a)



(b)

Figure C.11: Illustration of *smith_flock_1_2* in Experiment Three

C.3.3 Assignment "Role-Matching"

An individual of the class *Assignment* named *smith_matchrole* was created in OSO, which contains the following information:

C.3.3.1 Monitor

"As long as the simulator driver (*subject*) passes or arrives at the position of 6000 *m* in the road segment (*SV_Distance*)."

An individual of the class *ReferenceValue* named *value_matchrole_monitor* was created to represent the reference value, which is "passes or arrives at the position of 6000 *m* in the road segment". Its properties are shown in Table C.43.

Table C.43: Properties of *value_matchrole_monitor* for "Role-Matching" in Experiment Three

Property	**Related to**
hasRefValueType	some *NUMBER*
hasRefValueRange	some *BEQUAL* ($\geqslant$)
hasGeneralValue	6000

An individual of the class *Monitor* named *monitor_matchrole* was created to represent this Monitor. Its properties are shown in Table C.44.

Table C.44: Properties of *monitor_matchrole* for "Role-Matching" in Experiment Three

Property	**Related to**
hasMonitorType	some *State*
hasMonitorOperator	some *ASLONGAS*
hasReference	*subject*
hasStateVariable	some *SV_Distance*
hasRefValue	*value_matchrole_monitor*

C.3.3.2 Action

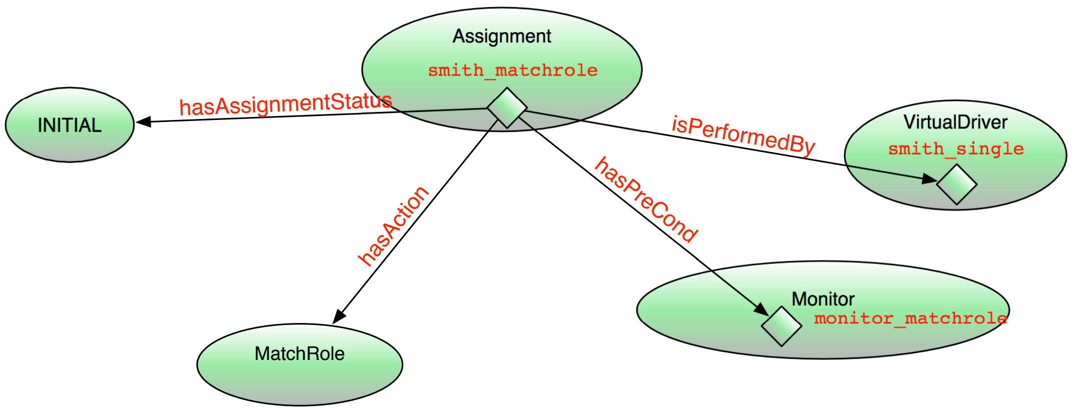
When the Monitor became true, a pre-defined Action of "MatchRole" was triggered, which was represented by the sub-class of *Action* named *MatchRole*, i.e., *hasAction* some *MatchRole*.

C.3.3.3 Performer

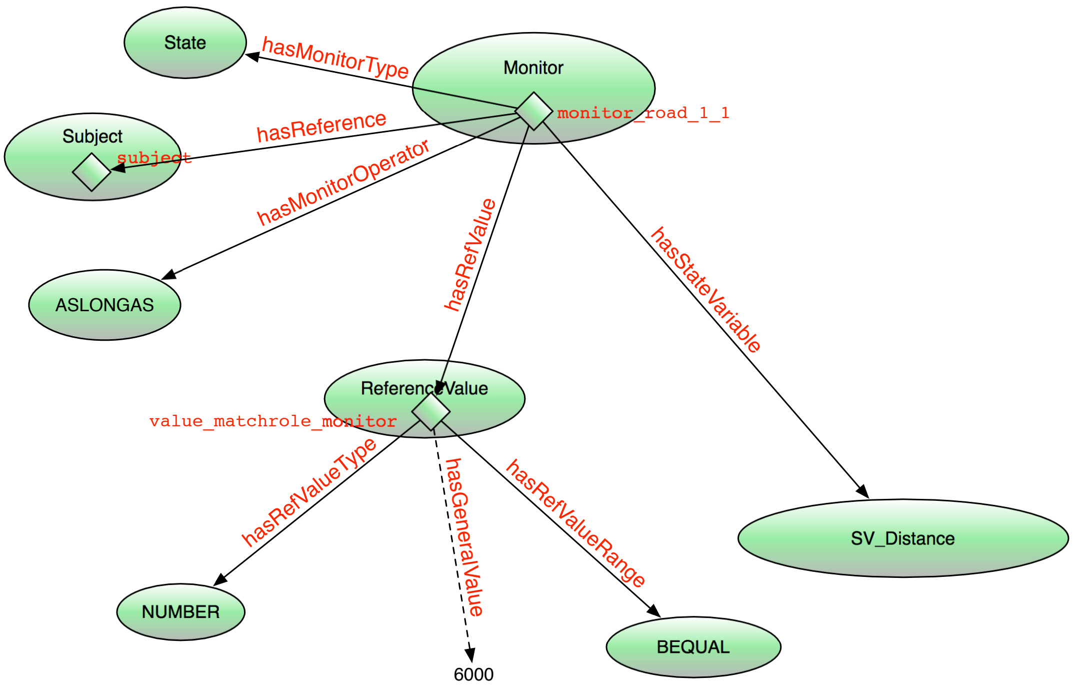
This Assignment should be carried out by Smith, not SMM, so an individual of class *VirtualDriver* named *smith_single* was used, i.e., *isPerformedBy smith_single*

C.3.3.4 Representation by Directed Graph

By adopting the individuals mentioned above and some other information, e.g., the initial status of Assignment, the main information of this Assignment can be illustrated as in Figure C.6. This Assignment can be tried once (*hasMaxtriedTime 1*)



(a)



(b)

Figure C.12: Illustration of *smith_matchrole* in Experiment Three

References

AL-SHIHABI, T. & MOURANT, R. (2003). Toward more realistic driving behavior models for autonomous vehicles in driving simulators. *Transportation Research Record: Journal of the Transportation Research Board*, **1843**, 41–49.

ALBUS, J.S. (1999). The Engineering of Mind. *Information Sciences*, **117**, 1–18.

ALLEN, J. (1981). An interval-based representation of temporal knowledge. In *Proc. 7th International Joint Conference on Artificial Intelligence, Vancouver, Canada*, 221–226.

BROOKHUIS, K., WAARD, D.D. & MULDER, B. (1994). Measuring driving performance by car-following in traffic. *Ergonomics*, **37:3**, 427 – 434.

CHAMPION, A., ÉSPIÉ, S. & AUBERLET, J. (2001). Behavioral road traffic simulation with ARCHISIM. In *Proceedings of Summer Computer Simulation Conference*, 359–364, Society for Computer Simulation International; 1998.

COBA (2006). *Traffic Input to COBA*, vol. 13. Department for Transport, UK.

CORCHO, O., FERNÁNDEZ-LÓPEZ, M. & GÓMEZ-PÉREZ, A. (2003). Methodologies, tools and languages for building ontologies: where is their meeting point? *Data Knowl. Eng.*, **46**, 41–64.

CORMEN, T., LEISERSON, C., RIVEST, R. & STEIN, C. (2001). *Introduction to algorithms*. The MIT Press, Cambridge, MA.

CREMER, J., KEARNEY, J. & PAPELIS, Y. (1995). HCSM: a framework for behavior and scenario control in virtual environments. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, **5**, 242–267.

Currie, K. & Tate, A. (1991). O-plan: the open planning architecture. *Artificial Intelligence*, **52**, 49–86.

Dahlgren, K. (1995). A linguistic ontology. *International Journal of Human-Computer Studies*, **43**, 809 – 818.

Dechter, R., Meiri, I. & Pearl, J. (1991). Temporal constraint networks. *Artif. Intell.*, **49**, 61–95.

Devillers, F. & Donikian, S. (2003). A scenario language to orchestrate virtual world evolution. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 275, Eurographics Association.

Dewar, R. (2002). Individual differences. *Human Factors in Traffic Safety*, 111–142.

Doniec, A., Espié, S., Mandiau, R. & Piechowiak, S. (2006). Multi-agent coordination and anticipation model to design a road traffic simulation tool. In *Proceedings of the fourth European Workshop on Multi-Agent Systems (EUMAS'06), Lisbon, Portugal*, Citeseer.

Doniec, A., Mandiau, R., Piechowiak, S. & Espié, S. (2008). Anticipation based on constraint processing in a multi-agent context. *Autonomous Agents and Multi-Agent Systems*, **17**, 339–361.

Donikian, S. (2001). HPTS: a behaviour modelling language for autonomous agents. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, 401–408, ACM, New York, NY, USA.

Drabble, B., Tate, A. & Dalton, J. (1997). Repairing Plans On-the-fly. In *Proceedings of the NASA Workshop on Planning and Scheduling for Space*, Oxnard, CA.

Driving Wiki (2010a). Driving Wiki. Retrieved July 2010. From http://www.simusers.com/.

Driving Wiki (2010b). Standard Measures. Retrieved July 2010. From http://www.simusers.com/tiki-index.php?page=Standard+Measures.

DUPUIS, M. (2011). Opendrive format specification, rev. 1.3.

EL HADOUAJ, S. & ESPIÉ, S. (2002). A generic road traffic simulation model. In *Proceedings of International Conference on Traffic and Transportation Studies*.

EL HADOUAJ, S., DROGOUL, A. & ESPIÉ, S. (2001). How to combine reactivity and anticipation: the case of conflicts resolution in a simulated road traffic. In *MABS 2000: Proceedings of the second international workshop on Multi-agent based simulation*, 82–96, Springer-Verlag New York, Inc., Secaucus, NJ, USA.

ENGSTRÖM, J. & HOLLNAGEL, E. (2007). A general conceptual framework for modelling behavioural effects of driver support functions. In P. Cacciabue, ed., *Modelling Driver Behaviour in Automotive Environments*, 61–84, Springer London.

EROL, K., HENDLER, J. & NAU, D.S. (1994). UMCP: A sound and complete procedure for hierarchical task-network planning. In *Proc. 2nd Intl. Conf. on AI Planning Systems*, 249–254.

ESPIÉ, S., AUBERLET, J.M. *et al.* (2007). Archisim: a behavioral multi-actors traffic simulation model for the study of a traffic system including its aspects. *International Journal of ITS Research*.

FISHER, D., RIZZO, M., CAIRD, J. & LEE, J., eds. (2010a). *Handbook of driving simulation for engineering, medicine, and psychology*. CRC.

FISHER, D., RIZZO, M., CAIRD, J. & LEE, J., eds. (2010b). *Scenario Authoring*, chap. 6, 6.1 – 6.12. CRC.

FORSMAN, ÅSA., VADEBY, A., YAHYA, M.R., TAPANI, A., ENJALBERT, S., CAS-SANI, M., AMANTINI, A., LAI, F., KECK-LUND, L. & ARVIDSSON, M. (2011). D5.1 - results from the analysis and input to the develop- ment and validation of the statistical models. Second revised ec submission, The ITERATE Project (FP7 Collaborative Project, Grant Agreement Number: 218496).

FUCHS, S., RASS, S. & KYAMAKYA, K. (2008). Integration of ontological scene representation and logic-based reasoning for context-aware driver assistance systems. *Electronic Communications of the EASST*, **11**.

Ghallab, M., Nau, D. & Traverso, P. (2004). *Automated Planning: Theory & Practice*. Morgan Kaufmann, San Francisco, CA, USA.

Guarino, N. & Welty, C. (2002). Evaluating ontological decisions with ontoclean. *Communications of the ACM*, **45**, 61–65.

Gugerty, L. (2011). Situation awareness in driving. *Handbook for driving simulation in engineering, medicine and psychology*.

Hadad, M., Kraus, S., Gal, Y. & Lin, R. (2003). Temporal reasoning for a collaborative planning agent in a dynamic environment. *Annals of Mathematics and Artificial Intelligence*, **37**, 331–379.

Hamilton, A., González, E., Acosta, L., Arnay, R. & Espelosín, J. (2013). Semantic-based approach for route determination and ontology updating. *Engineering Applications of Artificial Intelligence*, **23**, 1174 – 1184.

Hitzler, P., Krotzsch, M. & Rudolph, S. (2011). *Foundations of semantic web technologies*. Chapman and Hall/CRC.

Hobbs, J. & Pan, F. (2004). An ontology of time for the semantic web. *ACM Transactions on Asian Language Information Processing (TALIP)*, **3**, 66–85.

Horridge, M. *et al.* (2009). A practical guide to building owl ontologies using protégé 4 and co-ode tools edition1. 2. *The University of Manchester*.

Horrobin, T. & Carsten, O. (2011). Personal communication.

Jarrar, M. (2005). *Towards Methodological Principles for Ontology Engineering*. Phd thesis, Vrije Universiteit Brussel.

Kearney, J., Willemsen, P., Donikian, S., Devillers, F., de Beaulieu, C. & Rennes, F. (1999). Scenario languages for driving simulation. In *Proceedings of Driving Simulation Conference,DSC'99*, 377–393.

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *Int. J. Rob. Res.*, **5**, 90–98.

Krummenacher, R. & Strang, T. (2007). Ontology-based context modeling. In *Proceedings Third Workshop on Context-Aware Proactive Systems (CAPS 2007)(June 2007)*.

Lacroix, B., Mathieu, P., Rouelle, V., Chaplier, J., Gallée, G. & Kemeny, A. (2007). Towards traffic generation with individual driver behavior model based vehicles. In *Proceedings of Driving Simulation Conference, Iowa City, USA*, 144–154.

Lacroix, B., Rouelle, V., Kemeny, A., Mathieu, P., Laurent, N., Millet, G. & Gallee, G. (2009). Informal rules for autonomous vehicles in scaner. In *Driving Simulation Conference*, 58–69, Monaco, conference web site: http://dsc-europe.imagina.mc/.

Leitao, M., Sousa, A. & Ferreira, F. (1999). A scripting language for multi-level control of autonomous agents in a driving simulator. In *Proceedings of Driving Simulation Conference,DSC'99*, vol. 99, 339–351.

Michon, J. (1985). A critical view of driver behavior models: What do we know, what should we do. In L. Evans & R. Schwing, eds., *Human behavior and traffic safety*, 485–520, Plenum Publishing Corporation.

Miller, I., Campbell, M., Huttenlocher, D., Nathan, A., Kline, F.R., Moran, P., Zych, N., Schimpf, B., Lupashin, S., Garcia, E., Catlin, J., Kurdziel, M. & Fujishima, H. (2009). Team cornell's skynet: Robust perception and planning in an urban environment. In M. Buehler, K. Iagnemma & S. Singh, eds., *The DARPA Urban Challenge*, vol. 56 of *Springer Tracts in Advanced Robotics*, 257–304, Springer Berlin Heidelberg.

Morisset, B. & Ghallab, M. (2008). Learning how to combine sensory-motor functions into a robust behavior. *Artificial Intelligence*, **172**, 392 – 412.

Musliner, D.J., Hendler, J.A., Agrawala, A.K., Durfee, E.H., Strosnider, J.K. & Paul, C. (1995). The challenges of real-time ai. *Computer*, **28**, 58–66.

Nau, D., Au, T.C., Ilghami, O., Kuter, U., Murdock, J.W., Wu, D. & Yaman, F. (2003). SHOP2: an HTN planning system. *J. Artif. Int. Res.*, **20**, 379–404.

Noy, N.F. & Mcguinness, D.L. (2001). Ontology development 101: A guide to creating your first ontology. Tech. rep., Stanford University, Stanford, CA, USA.

Olstam, J. (2009). Simulation of surrounding vehicles in driving simulators,phd thesis summary. Tech. rep., Department of Science and Technology, Linköping University.

Olstam, J., Espié, S., Måardh, S., Jansson, J. & Lundgren, J. (2011). An algorithm for combining autonomous vehicles and controlled events in driving simulator experiments. *Transportation Research Part C: Emerging Technologies*, **19**, 1185–1201.

Olstam, J.J., Lundgren, J., Adlers, M. & Matstoms, P. (2008). A framework for simulation of surrounding vehicles in driving simulators. *ACM Trans. Model. Comput. Simul.*, **18**, 1–24.

OpenDRIVE (2013). OpenDRIVE — Background.

Papelis, Y. & Ahmad, O. (2001). A comprehensive microscopic autonomous driver model for use in high-fidelity driving simulation environments. In *National Research Council (US). Transportation Research Board. Meeting (80th: 2001: Washington, DC). Preprint CD-ROM*.

Papelis, Y., Ahmad, O. & Schikore, M. (2001). Scenario definition and control for the national advanced driving simulator. In *International Conference on the Enhanced Safety of Vehicles (ESV)*, SAE International.

Papelis, Y., Ahmad, O. & Watson, G. (2003). Developing scenarios to determine effects of driver performance: Techniques for authoring and lessons learned. In *Proceedings of Driving Simulation Conference North America,DSC'03*.

Peters, B. & Nilsson, L. (2007a). Modelling the driver in control. In P. Cacciabue, ed., *Modelling Driver Behaviour in Automotive Environments*, 85–104, Springer London.

Peters, B. & Nilsson, L. (2007b). Modelling the driver in control. In P. Cacciabue, ed., *Modelling Driver Behaviour in Automotive Environments*, 85–104, Springer London.

Protégé (2012). Protégé-OWL Ontology Editor. http://protege.stanford.edu.

Provine, R., Schlenoff, C., Balakirsky, S., Smith, S. & Uschold, M. (2004). Ontology-based methods for enhancing autonomous vehicle path planning. *Robotics and Autonomous Systems*, **49**, 123–133.

Sacerdoti, E.D. (1975). The nonlinear nature of plans. In *Proceedings of the 4th international joint conference on Artificial intelligence - Volume 1*, IJCAI'75, 206–214, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Salvucci, D., Boer, E. & Liu, A. (2001). Toward an integrated model of driver behavior in cognitive architecture. *Transportation Research Record: Journal of the Transportation Research Board*, **1779**, 9–16.

Sukthankar, R. (1997). *Situation Awareness for Tactical Driving*. doctoral dissertation, Robotics Institute, Carnegie Mellon University.

Sure, Y. (2003). *Methodology, Tools and Case Studies for Ontology based Knowledge Management*. Phd thesis, University of Karlsruhe, Department of Economics and Business Engineering.

Tate, A. (1977). Generating project networks. In *Proceedings of the 5th international joint conference on Artificial intelligence - Volume 2*, IJCAI'77, 888–893, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Tate, A., Drabble, B. & Kirby, R. (1994). O-plan2: an open architecture for command, planning and control. In *Intelligent Scheduling*, Morgan Kaufmann.

Toledo, T., Choudhury, C. & Ben-Akiva, M. (2005). Lane-changing model with explicit target lane choice. *Transportation Research Record: Journal of the Transportation Research Board*, **1934**, 157–165.

Treiber, M., Hennecke, A. & Helbing, D. (2000). Congested traffic states in empirical observations and microscopic simulations. *Physical Review E*, **62(2)**, 1805–1824.

Treiber, M., Kesting, A. & Helbing, D. (2006). Delays, inaccuracies and anticipation in microscopic traffic models. *Physica A: Statistical Mechanics and its Applications*, **360**, 71 – 88.

Wikipedia (2011). Agent Smith — Wikipedia, The Free Encyclopedia.

Wilkins, D.E. (1991). Can AI planners solve practical problems? *Comput. Intell.*, **6**, 232–246.

Wilkins, D.E. *et al.* (2001). A call for knowledge-based planning. *AI magazine*, **22**, 99.

Willemsen, P. (2000). *Behavior and scenario modeling for real-time virtual environments*. Ph.D. thesis, The University of Iowa.

Wolffelaar, P., Bayarri, S. & Coma, I. (1999). Script-based definition of complex driving simulator scenarios. In *Proceedings of Driving Simulation Conference,DSC'99*, 353–536.

Wright, S. (2000). *Supporting intelligent traffic in the Leeds driving simulator*. Ph.D. thesis, School of Computing Studies, University of Leeds.

Wright, S., Ward, N. & Cohn, A. (2002). Enhanced presence in driving simulators using autonomous traffic with virtual personalities. *Presence: Teleoperators & Virtual Environments*, **11**, 578–590.

Zu, D., Han, J. & Campbell, M. (2004). Artificial potential guided evolutionary path plan for multi-vehicle multi-target pursuit. In *Robotics and Biomimetics, 2004. ROBIO 2004. IEEE International Conference on*, 855 –861.