



Université
de Toulouse

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)

Présentée et soutenue par :

Ghasan Bhatti

Le vendredi 21 novembre 2014

Titre :

User-centered multi-layer programming approach to model scenarios on driving simulators.

ED MITT : Image, Information, Hypermedia

Unité de recherche :

UT3-IRIT : Institut de Recherche en Informatique de Toulouse

Directeur(s) de Thèse :

Professeur Jean-Pierre Jessel

Rapporteurs :

Professeur Frédéric Merienne

Professeur Ronan Querrec

Autre(s) membre(s) du jury :

Dr. Roland Brémond (Co-directeur de thèse)

Professeur Andras Kemeny

Dr. Corinne Brusque

M. Guillaume Millet

Acknowledgements

One of the joys of completion is to look over the journey and remember all the people who have contributed and supported me all along to achieve this milestone in my life. So I am using this opportunity to express my gratitude to these people. First of all, I would like to thank the company **OKTAL** and the organizers of the ADAPTATION project for financing this work and allowing me to become a part of their team and work on this subject.

I would like to express my utmost gratitude to my Advisor **Professor Jean-Pierre Jessel** for allowing me to join him. I am thankful to him for his patience, his valuable support during this time and useful advices which allowed me to complete this work. It would have been difficult to complete this work without his support. I would also thank him for his help all along in the administrative work.

My special thanks to my co-advisor **Dr. Roland Brémont**, who has been an utmost help during all the stages of my thesis work. I appreciate him for his patience, support and valuable suggestions and discussions. He has been a great mentor. And I would not hesitate to say that, it would have been difficult to complete my work without his support.

I wish to thank **Guillaume Millet** and **Dr. Fabrice Vienne**. It was a pleasant experience to work with them and I would like to thank them for all the support during this time and especially the technical assistance they have provided during the time of my PhD thesis. I am grateful to **Dr. Nguyen-Thong Dang** for his assistance and the valuable discussions we had during my thesis work. . I am also thankful to **Gilles Gallée** for his support.

I would like to extend my appreciation to my manuscript reviewers, **Professor Frédéric Merienne** and **Dr. Ronan Querrec**, for taking out time to read my thesis and for their valuable feedback. I would like to extend my gratitude to other Jury members **Dr. Andras Kemeny** and **Dr. Corinne Brusque** for becoming the part of the jury.

Special thanks are due to my colleagues at **IFSTTAR** and **OKTAL** for providing me a wonderful and a friendly working environment. I have spent a memorable time with these people. Here I would like to mention Ferhat and Karine, Abdurrehmane and Abdarrehmane, with

Acknowledgements

whom I have spent a memorable time. I would also like to thank my colleagues in the ADAPTATION project. I am also grateful to all the participants, who participated with patience in my experimental studies.

I would like to direct my most sincere thanks to my friends around the globe and especially in France Ayyaz, Hassan, Zakir, and Sohail.

I am forever indebted to my parents and sisters for their unconditional love and faith in me. It is due to their prayers that I have been able to achieve what I have. I wish you both a very good health.

In the end, I would like to mention about the most important person of my life, my wife Zainab. I am really thankful to her for her support during the last days of my work, I love you a lot!

Last but not the least; I am grateful to almighty ALLAH for his blessings and for giving me the courage and strength to complete my work.

Champs sur Marne, 21 Novembre 2014

G. B.

Abstract

Driving simulators are useful tools for researcher in order to study the drivers' behaviour, to analyze road safety features and to evaluate ADAS (Advance Driving Assistance Systems). Modeling scenarios on driving simulators is a critical and complex task for behavioral researcher. It requires specific technical and programming skills, for which researchers are not formally trained. One of the main reasons why designing scenarios is a complex task is the lack of User-Centered Design (UCD) of the scenario authoring tools, which could account for the skills they lack in order to achieve their objectives with driving simulators. The user interfaces are thus not very intuitive and user-friendly in most driving simulators. A User-Centered and Multilayer programming approach is proposed in order to fill the gap between the end-user's skills and the goals they want to achieve with driving simulators.

A user study was conducted to gather the user's needs and requirements to model scenarios on driving simulators. Different steps have been identified, followed by the end-users while designing an experimental protocol; moreover, different types of users who interact with driving simulators have been identified. We propose that the interface to develop an experimental protocol should be split into three sub-interfaces used by different end-users who interact with driving simulators: the Template Builder for Technical persons, the Experiment Builder for Researchers, and the Experiment Interface, for Experiment operators. Using the Experiment Builder, researchers can develop scenarios at high-level exploiting the programming primitives. An evaluation of the approach was conducted on a semi-functional and a functional prototype. During the evaluation, end-users (behavioral researchers) developed experimental protocols on driving simulators. The results have shown that the proposed approach has empowered the non-programmers to model scenarios on driving simulators without any technical or programming help from technical persons.

Besides, every driving simulator has a different execution platform. An interoperability framework and a Scenario-Meta Language (SML) is proposed and developed to port the scenarios from one platform to another. The scenarios developed using the multi-layer programming approach can be executed on different driving simulators. The interoperability framework and the meta-language has been successfully tested by integrating them with the SCANeR software.

Key words: Driving simulators, Scenario modelling, User-centered design, Multi-layer pro-

Acknowledgements

gramming, End-user software engineering, Interoperability framework.

Résumé

Le simulateur de conduite est un outil très utilisé par les chercheurs, Il leurs permet d'étudier le comportement des conducteurs, d'analyser certains aspects de la sécurité routière et d'évaluer des Systèmes d'Aide à la Conduite (SAC). La modélisation des scénarios pour ces simulateurs de conduite est une tâche cruciale et complexe pour le chercheur. Elle exige des compétences techniques et de programmation spécifiques, pour lesquelles les chercheurs ne sont pas nécessairement formés. Une des principales raisons est le manque de conception centrée sur l'utilisateur (UCD), ce qui pourrait expliquer la difficulté des chercheurs à atteindre leurs objectifs avec des simulateurs de conduite. L'interface utilisateur n'est donc pas très intuitive et conviviale dans la plupart des simulateurs de conduite. Afin de combler l'écart entre les compétences des utilisateurs et les objectifs qu'ils souhaitent atteindre en utilisant des simulateurs de conduite, une approche de programmation multicouche centrée sur l'utilisateur est proposée.

Une étude a été menée sur des utilisateurs afin de recueillir leurs besoins et leurs exigences pour modéliser des scénarios sur simulateur de conduite. Les différentes étapes que suit l'utilisateur final lors de la conception d'un protocole expérimental et les différents types d'utilisateurs qui interagissent avec les simulateurs de conduite ont été identifiés. L'interface pour le développement d'un protocole expérimental a été divisée en trois sous-interfaces, qui sont utilisées par les différents utilisateurs qui interagissent avec les simulateurs de conduite : le 'Template Builder' pour le personnel technique, le 'Experiment Builder' pour les chercheurs, et le 'Experiment Interface' pour les opérateurs expérimentés. L'utilisation de cet 'Experiment Builder' peut permettre aux chercheurs de développer des scénarios à haut niveau tout en exploitant les primitives de programmation.

Une évaluation de l'approche a été effectuée sur un prototype semi-fonctionnel et fonctionnel. Lors de l'évaluation, les utilisateurs finaux (les chercheurs) ont développé un protocole expérimental sur simulateurs de conduite. Les résultats obtenus ont montré que l'approche proposée permet aux chercheurs non-programmeurs de modéliser des scénarios sur les simulateurs de conduite sans aucune aide technique ou de programmation. En outre, chaque simulateur de conduite à une plate-forme d'exécution différente. Un cadre d'interopérabilité et un Scénario-Meta Langage (SML) ont été proposés et développés afin de porter les scénarios d'une plateforme à une autre. Les scénarios développés avec l'approche de programmation multicouches peuvent ainsi être exécutées sur différents simulateurs de conduite. Le

Acknowledgements

cadre d'interopérabilité et le métalangage ont été testés avec succès en les intégrant dans le logiciel SCANeR.

Contents

Acknowledgements	i
Abstract	iii
List of figures	xi
List of tables	xiii
1 Introduction	1
1.1 End users programming	1
1.2 Driving simulators	3
1.3 Driving simulator scenarios	5
1.4 Challenges in modeling scenarios	7
1.5 Driving simulation platforms	9
1.6 Overview of the thesis	9
2 State of the art	11
2.1 Scenario authoring in driving simulators	11
2.2 Scenario modeling approaches	14
2.3 Scenario authoring language	16
2.4 Driving simulators and other domains	19
2.4.1 End-user programming Systems	20
2.5 Dealing with designing interfaces for end-users	22
3 End-user requirements	25
3.1 User-Centered Design	25
3.2 User Interviews	26
3.3 Method	26
3.3.1 Participants	26
3.3.2 Procedure	27
3.4 Results	27
3.4.1 General problems	28
3.4.2 Concept-to-script translation problems	29
3.4.3 Ideal approach to model scenarios	30
	vii

3.4.4	Improvement of the driving Simulator in use	30
3.5	Discussion and conclusion	31
4	Proposed Multi-Layer programming approach	33
4.1	Key challenges for researchers	33
4.1.1	Technical challenges	34
4.2	User roles	34
4.3	Experimental Protocol development Steps	35
4.4	Proposed solution	36
4.4.1	Multi-layer Programming	36
4.4.2	Empowering the end-users	37
4.4.3	Scenario modeling process using Multi-layered approach	37
4.4.4	Discussion on user roles	39
4.4.5	Movie theater metaphor of the approach	40
4.5	Prototype Building	41
4.5.1	Step 1: Experiment Description	41
4.5.2	Step 2: Terrain selection	42
4.5.3	Step 3: Configure subject vehicle	42
4.5.4	Step 4: Configure Autonomous Traffic	43
4.5.5	Step 5: Configure environment	43
4.5.6	Step 6: Select dependent variables	43
4.5.7	Step 7a: Construct critical events and scenarios	43
4.5.8	Step 7b: Specify Template parameters	44
4.5.9	Step 7c: Overview of the experiment	44
4.6	Significance of the approach	47
4.6.1	Overcoming barriers	47
4.6.2	Support higher-level goals	48
4.6.3	Reusability	49
4.6.4	Is it another new System?	49
4.7	Comparison with the existing system	49
4.7.1	Interface	50
5	User Experience	61
5.1	Preliminary study	61
5.1.1	Procedure	61
5.1.2	Interviews	62
5.1.3	Results	62
5.1.4	Discussion	63
5.2	Main study	63
5.2.1	Participants' selection and participation	63
5.2.2	Experiment Setup and Procedure	64
5.2.3	Experiment Task	64
5.2.4	Data Collection	66

5.2.5	Data Analysis	68
5.2.6	Results	69
5.3	Discussion and hypotheses exploration	76
5.3.1	User selection	77
5.3.2	User performance	77
5.3.3	Problems with the users	78
5.3.4	Hypotheses Exploration	78
5.4	Conclusion	79
6	Interoperability of the solution	81
6.1	Need for interoperability of scenarios	81
6.2	Challenges to develop an interoperability framework	82
6.3	Scenario modelling process	82
6.4	Scenario Meta-Language (SML)	83
6.4.1	Functional requirements	83
6.4.2	General requirements	83
6.4.3	Description of the SML	83
6.5	Interoperability framework	91
6.5.1	Implementation of the Interoperability Framework	92
6.6	Conclusion	94
7	Conclusion	95
7.1	Concluding discussion	95
7.2	Contributions	98
7.3	Prospective	99
	Publications	101
	Bibliography	109
A	Appendix A	111
B	Appendix B	115

List of Figures

1.1	High-fidelity driving simulator (credit: Tongji university, China).	4
1.2	Low-fidelity driving simulator (credit: Ludoscience.com).	5
1.3	Scenario modeling process followed by researchers.	6
1.4	Basic steps for driving simulation study.	6
1.5	SCANeR Interface (OKTAL SA).	7
1.6	ARCHISIM Interface (IFSTTAR).	8
2.1	SDL in STI SIM.	12
2.2	STIM SIM PDE.	13
2.3	ISAT Scene Authoring GUI [Kearney and Timofey F, 2011].	14
2.4	SCANeR (top) and ARCHISIM (bottom) GUI for scenario authoring.	15
2.5	Assembling scene from tiles in Simvista [Kearney and Timofey F, 2011].	17
4.1	Multi-layer Programming Architecture.	37
4.2	Experiment Building Environment.	39
4.3	User roles corresponding to scenario levels.	40
4.4	Experiment description (Step 1).	41
4.5	Terrain selection (Step 2).	42
4.6	Subject Vehicle Configuration (Step 3).	42
4.7	Autonomous traffic configuration (Step 4).	43
4.8	Top: Configure environment. Bottom: Weather zone configuration (Step 5).	44
4.9	Variable Selection (Step 6).	45
4.10	Trigger for the template (Step 7a).	45
4.11	End-user specifying template parameters (Step 7b).	46
4.12	Visualization of the experiment on the Time line and distance line (Step 7c).	46
4.13	Comparison of Participant vehicle configuration.	52
4.14	Comparison of Autonomous traffic configuration.	54
4.15	Comparison of environment configuration.	55
4.16	Comparison of selecting the Variables.	57
4.17	Comparison of constructing the critical events.	59
5.1	Exercise to model scenario.	62
5.2	5-point Likert scale.	67

List of Figures

5.3	Task performance data.	70
5.4	Learning Experience of all users.	71
5.5	Mean SUS score rating [Bangor et al., 2009].	72
5.6	Task performance data (Experienced vs inexperienced users).	73
5.7	Learning experience of Traffic zone (Experienced vs inexperienced).	73
5.8	Learning experience of Situation 1 (Top) and 2 (bottom).	74
5.9	SUS score (Experienced vs inexperienced users).	75
6.1	Typical scenario modeling process.	82
6.2	SML Schema.	84
6.3	Typical scenario modeling process.	91
6.4	Interoperability Framework.	92
6.5	Interoperability Framework using multi-layer programming.	93

List of Tables

3.1	Driving Simulators.	27
3.2	User Profile.	27
3.3	Programming Language Knowledge of the Users.	28
3.4	Solution to problem mapping.	32
4.1	Terrain Selection comparison.	50
4.2	Participant vehicle comparison.	51
4.3	Comparison of configuration of autonomous traffic.	53
4.4	Comparison of configuration of Environment.	56
4.5	Comparison of the process of selecting the dependent variables.	56
4.6	Comparison of construction of critical events.	58
5.1	SUS score of all users.	71
5.2	Task easiness rating (All users).	72
5.3	Task easiness rating (Experienced User).	75
5.4	Task easiness rating (Inexperienced User).	76

Chapter 1

Introduction

The work presented in this thesis was carried out under the framework of European FP7 project "ADAPTATION". The project consists of several industrial partners, universities and research organizations including IFSTTAR and OKTAL. ADAPTATION aims at studying the whole range of adaptation processes in response to ADAS (Advance Driving Assistance Systems) including not only observable behavioral changes, but also changes in regulatory, cognitive and motivational processes. Researchers use different tools in the projects like adaptation to study drivers' behaviour and to propose road safety feature. Driving simulator is one of the tools used by researchers to perform different studies. The efficient and effective use of driving simulators plays a vital role in obtaining the results of their studies. This thesis aims at improving the user experience while using driving simulators.

1.1 End users programming

Software technology has revolutionized today's world. The increasing use of software technologies in different platforms (desktop, web, mobile) has enabled the users to adopt software programmers as their profession. But nowadays software development is not only done by professional programmers but sometimes also by non-professional programmers, which we call here end-user. End-users are domain experts who do programming to support their business goals. For example, Accountants use spreadsheet, teacher use excel sheets for grading. Scientists develop programs to support their analysis and research work. Gartner's report in 2011 indicated that by the end 2014, 25% of the new business application will be developed by end-users, which indicates the role of end-users in developing the future applications.

"End-User Development is a set of methods, techniques and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify, or extend a software artifact" Lieberman et al. [2006]. End-User development is a multi-disciplinary process which integrates the knowledge from different domain which is Human-Computer Interaction, Software Engineering (SE) and Artificial Intelligence. A better

integration of knowledge from these field can help to develop concepts like configurability, end-user programming, usability, visual programming, natural programming and provides a considerable knowledge but they need to be better integrated.

End-user development is different from software development because in software development, software developers develop application by following the need and requirements of the end-users. End-users are not proficient in programming, but they have domain knowledge and they use this knowledge to develop application in order to support their work so what makes end-users different from professional programmers are their goals Ko et al. [2011].

End-user programming has been one of the key research areas in the domain of human-computer interaction, where the focus is to develop the system to do programming for end-users who do not have programming skills or who have never programmed before. One of the ways to tackle this problem is to study the profile of the end-users and propose solutions which could fulfill the barriers of programming using these systems. Many approaches and notions have been used in past decades in order to tackle this problem. Ko (Ko, Myers et al. 2004) has identified 6 barriers which end-users faces while learning a end-user programming system. These barriers are:

1. Design barriers ("I do not know what I want my computer to do").
2. Selection barriers ("I think I know what I want computer to do but I do not know what to use").
3. Coordination barriers ("I think I know what things to use, but I do not know how to make them work together")
4. Use barriers ("I think I know what to use, but I do not know how to use it")
5. Understanding barriers ("I knew how to use it, but I did not know what I expected")
6. Information barriers ("I think I know why it did not do what I expected but I did not know how to check it")

End-user systems should address these barriers so that end-users could achieve the tasks using the tools. End-users are not like programmers, they know what they want to do, and what are their goals, but they may not know what they need (support from the end-user systems) in order to achieve these goals. The end-user programming systems provide support to end-users to achieve these goals and to remove the barriers they face while learning those systems. So many issues need to be addressed while designing those systems. Repenning and Ioannidou [2006] has proposed 13 design guidelines to design the usable programming systems for end-users. Besides these guidelines, there is a need to do the task analysis of the users: what the users want to do, what support do they need while performing the programming tasks using these systems. End-user programming systems should take in

to account that how end-users can be empowered to achieve their goals using these systems by considering their profile and capabilities.

End-user software engineering has been used in almost all the domains to support the tasks of the end-users, i.e. from daily use (using macros in excels and word) to the advanced use of these systems, for example statistical use and many other scientific uses. In the domain of driving simulators, they are used by researchers to configure their experiment for the study they want to conduct.

1.2 Driving simulators

A driving simulator is a useful tool for behavioral researchers in order to study the driver's behaviour, to analyze road safety features, and to study and evaluate ADAS (Advance Driving Assistance Systems) without any safety risk. In driving simulators, real driving environments are simulated and displayed to the drivers in order to achieve the study goal.

There are many advantages of using driving simulators over conducting a study on real roads and real cars. One advantage is that the experiment can be conducted in a controlled manner by easily setting up different environments and different parameters, which is one of the key requirements of most study: all participants must encounter the same driving conditions. The second advantage is the reproducibility of situations. If a researcher wants to study the behavior of drivers in a given situation, it is possible to simulate this situation for all the participants of the study. There are many dangerous driving situations which are risky to study in a real environment, for example accident situations, performing secondary tasks while driving etc. These situations can be studied in driving simulators without any safety risk. In the case of designing the new features or systems such as during the early design phases of an ADAS, it is cost-effective to test those systems on driving simulator rather than developing physical prototype of these systems. While performing a behavioral study, data is collected regarding the study, it is fairly easy and cost-effective to collect the data on driving simulator rather than using physical sensors in instrumented cars. They are also used for drivers training: drivers can be trained to drive in certain driving environment and situations, such as snow, motorway etc.

However, there are also some weak points which come along with driving simulators. The validity of driving simulators is a topic that has been addressed long ago, and significant work has been done to validate driving simulators for various tasks Blana [1996], Kaptein et al. [1996], Törnros [1998], Godley et al. [2002]. Simulator sickness is also a problem which may be encountered by some participants on driving simulators, presumably due to motion cues. The fidelity of the driving simulator also plays a role in the realism. Low-fidelity driving simulators are less realistic but cheaper as compared to high-fidelity and medium fidelity driving simulators. Figure 1.1 and 1.2 show examples of high-fidelity and low-fidelity driving simulators.



Figure 1.1: High-fidelity driving simulator (credit: Tongji university, China).

Designing and implementing an experiment on a driving simulator is a quite critical and complex task, which requires careful considerations and controlled environment. It is often carried out by behavioral researchers with an educational and research background in psychology or ergonomics, while implementing scenarios in the experimental protocol requires technical and programming skills which those researchers lack in most of the cases; so they take help from technical persons of their organization, who program scenarios for them.

Figure 1.3 shows the typical process followed by the researchers in order to develop an experimental protocol. A researcher designs an experimental protocol on the paper or using any software. In most of the cases he gives the design (which includes the scenario specification) to the technical persons who implement the scenarios on the driving simulators. In a next run, this scenario is validated or re-negotiated by the researchers. This process continues until the scenario is validated according to the requirements of the driving simulation experiment. Once the scenario fits the requirements of the researcher, he conducts the experiment.

End-users are totally dependent on the technical persons for the experiments to be implemented. Conducting a driving simulator study is a 3 steps process which includes a designing step, a implementation step and an experiment execution step as shown in Figure 1.4.

During step 1, end-users design their study (i.e. environment, scenarios and data to be collected), mainly without software support. During the second step the protocol is implemented on the driving simulator. During the third step, the participants participate in the study and data is collected from the experiment which is analyzed in order to answer the research questions.

As shown in Figure 1.3, the end-users (researchers) are dependent on technical persons for the implementation of scenarios, as the scenario creation requires technical and program-



Figure 1.2: Low-fidelity driving simulator (credit: Ludoscience.com).

ming skills for which these researchers typically do not have enough training. Thus, they depend on the technical persons or on the scenario developers in their respective organizations who program scenarios for them. This can be a time-consuming task because of the dependency on the other people and the need for the researchers to create precise specifications of the exact scenarios they wish to be created (sometimes with a limited awareness of the capabilities of the simulator). Interpreting and translating these specifications into a simulator scenario can therefore be a time consuming task for technical persons as well. So there is a need of a scenario development environment using which end-users can develop scenarios without depending on technical persons. In other words, the direct involvement of technical persons in the development of scenarios should be minimized. Figure 1.5 and 1.6 shows the interface of two existing and popular driving simulators.

SCANer has a Graphical User Interface (GUI) while ARCHISIM has a textual interface to model scenarios. But it is still difficult to develop scenarios using these interfaces. If the scenario is long and complex then it is difficult to manage or debug.

1.3 Driving simulator scenarios

The term "scenario" in general means a sequence of actions or events. But this term is used with variance in the driving simulation literature. A driving simulator scenario may be the specification of road and traffic situations along the road in the driving simulator Olstam and Espié [2007]. According to Papelis et al. [2003], scenarios are termed as "everything that happens in the driving simulator", which include specifying and controlling ambient traffic and its attributes, ambient environment and simulation conditions, route of the participants and their position, and series of the real world situations which are to be ma-

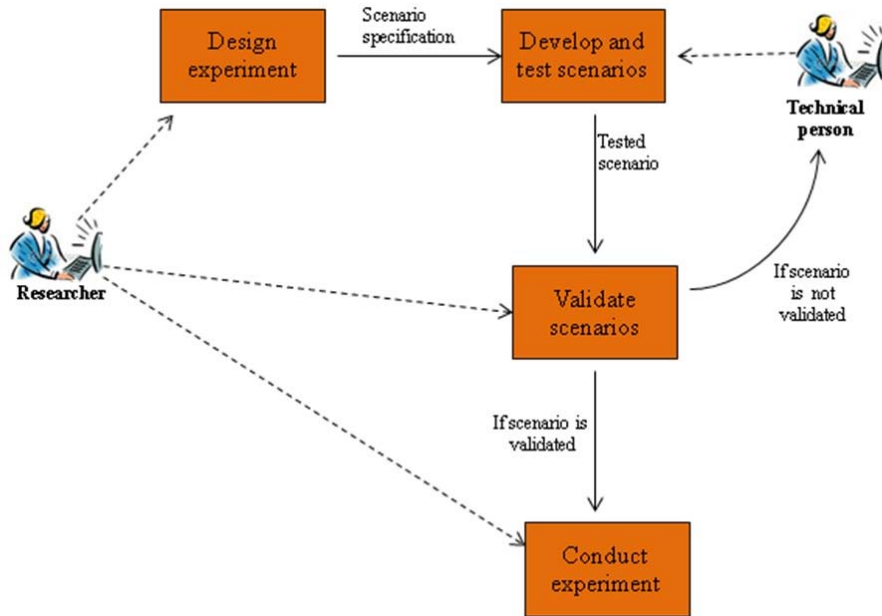


Figure 1.3: Scenario modeling process followed by researchers.



Figure 1.4: Basic steps for driving simulation study.

nipulated on driving simulators. Wolffelaar defines the scenario as a constellation of consecutive traffic situations [Wolffelaar, 1999]. It is sometimes used to specify both the layout and the activities during the experimental trial. Other authors use the term "Scene" to specify the layout (road network, terrain, driving environment, etc.) and the term "scenario" to specify what is going to happen during the simulation, i.e. critical situations and events Kearney and Timofey F [2011]. So the scenario is usually regarded as the sequence of activities that happens during the experimental trial. These activities include different situations and maneuvers to be tested, for example, crash situations and traffic-jam, etc. In the present document scenario modelling includes specifying the ambient traffic, environment, simulation conditions and the manipulation of the real-world traffic situations.

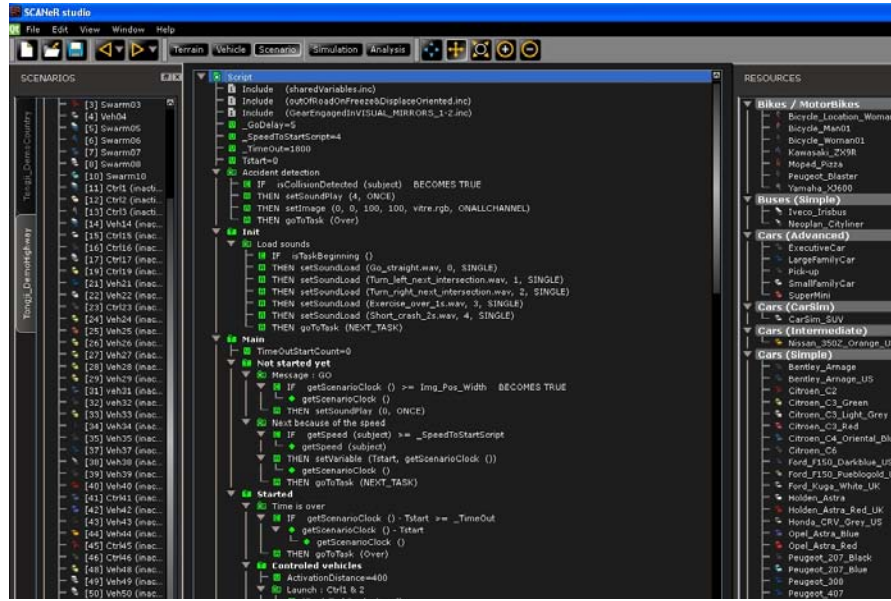


Figure 1.5: SCANet Interface (OKTAL SA).

1.4 Challenges in modeling scenarios

In the case of scenario programming on driving simulators, there are two global objectives shared by all driving scenarios: realism and reproducibility, which are also the key advantages of using the driving simulators. The researchers want the scenario to be realistic when participants encounter the scenario during the experimental study. Also they want all the participants to encounter the same situations, or at least to control the encountered situations.

There are two main factors which present the key challenges to making events happen at the right time and at the right place in a natural way [Kearney and Timofey F, 2011]. First, driving behaviour is complicated and not all aspects are well-understood (this is why so many studies experimental are needed!); so it is difficult to create realistic as well as controllable traffic. The second factor is the variability of driving behaviour, as people change their speed, lane position and tactical decisions with time during the simulation trials, which leads to variance in the drivers' behaviour. For example, imagine you want to create an accident situation, where a car overtakes the participant car and stops in front of his vehicle. To implement this situation using a modeling language, there are many parameters to control and consider including speed and position of the overtaking vehicle, speed of the participant's vehicle and the distance to stop from the participant's vehicle. Every driving simulator provides a scripting environment, where behavioural researchers can model scenarios, by using triggers and actions or some event-driven mechanism. Besides these issues, the interaction environment provided by driving simulators does not help to fill the gap between the user's skills and their goals.

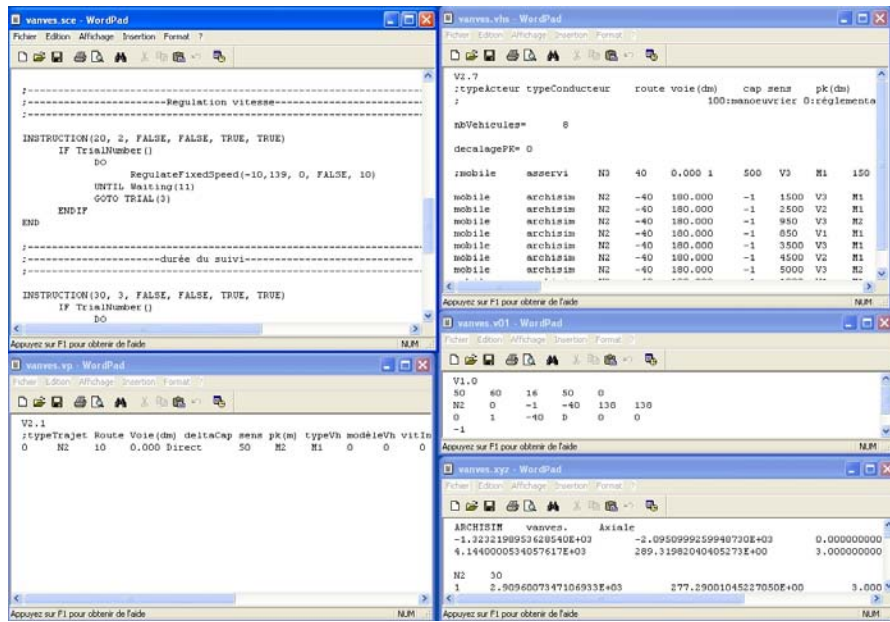


Figure 1.6: ARCHISIM Interface (IFSTTAR).

Driving simulators languages provide a broad range of support to manipulate and develop almost all kind of traffic situations but the scenario modeling systems offered by these driving simulators are at low-level. They do not correspond to the skill level of the behavioral researchers as they do not have any formal training for that. Also there is no standardized or common procedure to discuss the requirements and implementation of the scenarios between researchers and technical persons. The communication and the interaction between the researchers and technical persons are not standardized, so that researchers cannot easily communicate their requirements to the technical persons, and technical persons cannot easily make them understand the capabilities of the driving simulators. So there is a need for a standardized mechanism which could explain the role of all technical persons and researchers. The role of technical persons should be minimized and the researchers should be empowered, so that they could develop the scenarios with minimum or no help from the technical persons. Significant work has been done in order to develop the scenario authoring tools for driving simulator but little effort has been done to empower the end-user to use these tools with the current set of skills. One of the reasons why programming is difficult, is that programs are abstract [Cypher and Smith, 1995]. It is difficult for the people who do not have a solid programming background to think in an abstract manner to implement a given situation, compared to thinking about the same situation in the real world. So abstraction is one of the barriers, especially for novice users. Also it is a common observation that, the more complex the situation, the more abstraction is required in the program. Also, programming languages have been designed without careful attention to human-computer interaction issues [Newell and Card, 1985], which makes programming more difficult than necessary [Pane et al., 2001]. So there is a need to develop a user-centered interaction envi-

ronment which could support the users to achieve their goals with the driving simulator. As researchers are the primary users of driving simulators, so the needs, requirements and profile of the researchers should be kept in mind while designing the scenario authoring tools for the driving simulators. It would also reduce the workflow of the technical persons and make the scenario modeling process efficient and effective for both the researchers and technical persons. So one of the goals of this thesis is to provide an intuitive methodology for end-users to program scenarios, who do not have technical and programming skills.

1.5 Driving simulation platforms

Driving simulation platforms are quite heterogeneous, and it is not possible, given the current state of the art, to port scenarios from one driving simulation platform to another. Also, there is no common and standard way of describing the scenarios on driving simulators, though it is a common practice to execute the same scenarios on different driving simulators, especially in European projects, where similar scenarios are executed on different partner sites for different purposes. So there is a need for a common methodology and a framework that could provide interoperability of driving scenarios on different driving simulators. The second goal of this thesis is to propose an interoperability framework so that standardized scenarios could be executed on multiple driving simulation platforms.

1.6 Overview of the thesis

There is a need to develop environments for end-users corresponding to their skills to model scenarios on driving simulators. Almost all the driving simulators reviewed in this document are not developed with in mind the profile of the primary users of the driving simulators. So in this thesis a user-centered design approach is presented which takes into account the requirements and skills of the end-users.

In the second chapter, a review of existing driving simulators is presented. The existing scenario modeling languages, scenario modeling systems and the approach they follow to model scenario is discussed in detail. This chapter also presents the different end-user development paradigms which have been proposed. Then there is some literature review about Human-Computer Interaction (HCI) issues and the user-centered design approach. In the end, we discuss different interoperability frameworks and meta-languages which are developed by keeping the platform independence in mind.

In chapter 3, a study is presented which was conducted to get to know about the needs and requirements of the users. As user-centered design approach was used, so users were interviewed before the design phase, and were involved during all the phases of the development of our proposed solution.

Chapter 4 discusses the new multi-layer multi-user programming solution developed us-

Chapter 1. Introduction

ing the user-centered design approach. The user interface for end-users is split into 3 sub-interfaces corresponding to the skills of different users interacting with the driving simulators. This chapter also discusses the details of the prototype developed based on the proposed solution. Then the proposed interface is compared with existing driving simulator interfaces.

Chapter 5 discusses the details of all the testing of the proposed solution. The prototype was first tested with the end-users during a small study, where users gave some feedback at the early stage of the design. After this preliminary study the solution was improved and a new solution was developed and then was tested by the users in the real environment where users developed a real experiment. The user's activity was logged and their performance was measured using subjective as well as objective data. The users rated the solution using standardized usability questionnaires and questionnaires specific to the tasks they performed. Users were interviewed in detail as well to get detailed information about their subjective experience with the developed solution. In order to make our solution interoperable, and to be executed on most of the driving simulation platforms, an interoperability framework and a scenario-meta language is proposed. Chapter 6 discusses the interoperability framework and the schema of scenario-meta language in detail, which is followed by the implementation of a tool to execute the scenario on SCANeR software using the developed interface. Finally, the conclusion of the thesis is presented, followed by the perspective and the contribution of this research work.

Chapter 2

State of the art

This chapter discusses previous work regarding the end-user development and scenario authoring in driving simulators. The chapter is divided into 4 sections. Section 1 discusses the scenario authoring in driving simulators, section 2 discusses the different scenario languages in the domain of driving simulators section 3 is a literature review on end-user programming systems, and the last section discusses the issues encountered while dealing with user interfaces. The chapter overall focuses on the scenario modeling in the domain of driving simulator.

2.1 Scenario authoring in driving simulators

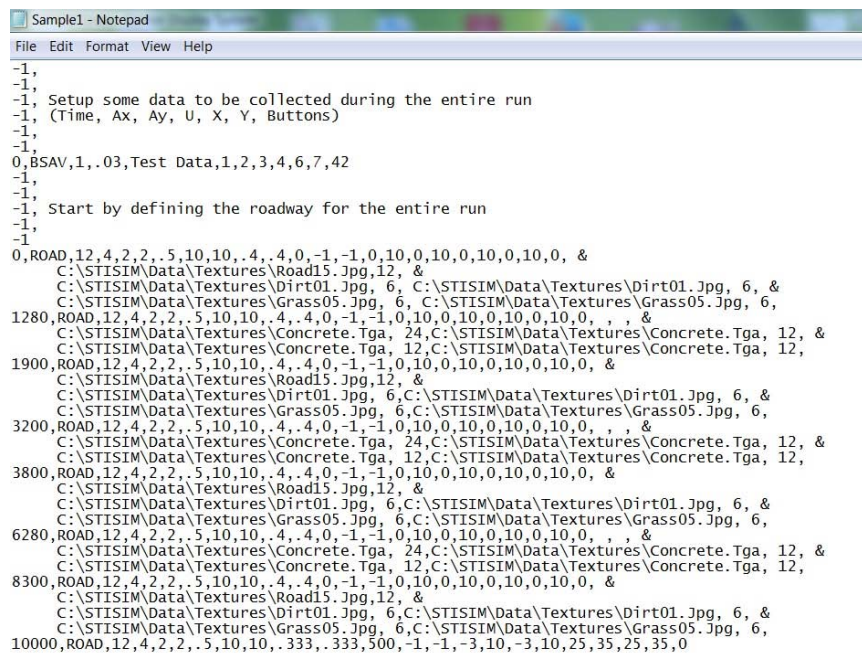
In the past decades, significant work has been done in order to make scenario modeling realistic for end-users. STI SIM Simulator developed by System Technology has done considerable work on scenario authoring on driving simulators. Allen et al. [2003, 2004] have identified the steps which are independent in a scenario modeling tool during the process of modeling the scenarios. They have also provided a procedural method to model scenarios. These steps include:

1. Identification of goals and objectives
2. Scenario definition (Sequence length of the scenario).
3. Preparation of models for specific components (vehicles, pedestrian, traffic control devices, etc.).
4. Scenario Programming (Program the scenarios using a modeling language).
5. Testing upgrading and validation of the whole scenario.

Chapter 2. State of the art

These authors use a text-based Scenario Definition Language (SDL) [Allen et al., 2001]. The scenario events are described in a framework which is tied to the driver's progress through the simulation. These events cover scenario layout, road network configuration, placement of objects and critical events. It has a Graphical user Interface (GUI) for the configuration management system of the driving simulator which includes specifying visual display system, resolution, aspect ratio, field of view, adjust lighting and atmospheric conditions [Rosenthal et al., 2003]. STI SIM Drive provides a mechanism to use the set of events as a reusable component called "Previously Defined Events" (PDE), and these PDE's can be specified of the traveling path of the driver.

This approach has advantages as well as limitations. By organizing everything with respect to the driver's path, it ensures the consistency from one run to another, and the roadway is built on the fly in order to match the specification of the scenario script. The limitation of this approach is topographical. If a driver takes a 90 degree turn they will be expecting to be on the road they crossed. Since a globally consistent road network is not constructed, the driver will never cross a road previously crossed. So this approach is not suitable for the experiments where map learning is important. An example of STI SIM SDL and PDE is shown in the Figure 2.1 and Figure 2.2 respectively.



```
Sample1 - Notepad
File Edit Format View Help
-1,
-1,
-1, Setup some data to be collected during the entire run
-1, (Time, Ax, Ay, U, X, Y, Buttons)
-1,
-1,
-1, BSAV,1,.03,Test Data,1,2,3,4,6,7,42
-1,
-1,
-1, Start by defining the roadway for the entire run
-1,
-1,
0,ROAD,12,4,2,2,.5,10,10,.4,.4,0,-1,-1,0,10,0,10,0,10,0, &
C:\STISIM\Data\Textures\Road15.Jpg,12, &
C:\STISIM\Data\Textures\Dirt01.Jpg, 6, C:\STISIM\Data\Textures\Dirt01.Jpg, 6, &
C:\STISIM\Data\Textures\Grass05.Jpg, 6, C:\STISIM\Data\Textures\Grass05.Jpg, 6,
1280,ROAD,12,4,2,2,.5,10,10,.4,.4,0,-1,-1,0,10,0,10,0,10,0, , , &
C:\STISIM\Data\Textures\Concrete.Tga, 24,C:\STISIM\Data\Textures\Concrete.Tga, 12, &
C:\STISIM\Data\Textures\Concrete.Tga, 12,C:\STISIM\Data\Textures\Concrete.Tga, 12,
1900,ROAD,12,4,2,2,.5,10,10,.4,.4,0,-1,-1,0,10,0,10,0,10,0, &
C:\STISIM\Data\Textures\Road15.Jpg,12, &
C:\STISIM\Data\Textures\Dirt01.Jpg, 6,C:\STISIM\Data\Textures\Dirt01.Jpg, 6, &
C:\STISIM\Data\Textures\Grass05.Jpg, 6,C:\STISIM\Data\Textures\Grass05.Jpg, 6,
3200,ROAD,12,4,2,2,.5,10,10,.4,.4,0,-1,-1,0,10,0,10,0,10,0, , , &
C:\STISIM\Data\Textures\Concrete.Tga, 24,C:\STISIM\Data\Textures\Concrete.Tga, 12, &
C:\STISIM\Data\Textures\Concrete.Tga, 12,C:\STISIM\Data\Textures\Concrete.Tga, 12,
3800,ROAD,12,4,2,2,.5,10,10,.4,.4,0,-1,-1,0,10,0,10,0,10,0, &
C:\STISIM\Data\Textures\Road15.Jpg,12, &
C:\STISIM\Data\Textures\Dirt01.Jpg, 6,C:\STISIM\Data\Textures\Dirt01.Jpg, 6, &
C:\STISIM\Data\Textures\Grass05.Jpg, 6,C:\STISIM\Data\Textures\Grass05.Jpg, 6,
6280,ROAD,12,4,2,2,.5,10,10,.4,.4,0,-1,-1,0,10,0,10,0,10,0, , , &
C:\STISIM\Data\Textures\Concrete.Tga, 24,C:\STISIM\Data\Textures\Concrete.Tga, 12, &
C:\STISIM\Data\Textures\Concrete.Tga, 12,C:\STISIM\Data\Textures\Concrete.Tga, 12,
8300,ROAD,12,4,2,2,.5,10,10,.4,.4,0,-1,-1,0,10,0,10,0,10,0, &
C:\STISIM\Data\Textures\Road15.Jpg,12, &
C:\STISIM\Data\Textures\Dirt01.Jpg, 6,C:\STISIM\Data\Textures\Dirt01.Jpg, 6, &
C:\STISIM\Data\Textures\Grass05.Jpg, 6,C:\STISIM\Data\Textures\Grass05.Jpg, 6,
10000,ROAD,12,4,2,2,.5,10,10,.333,.333,500,-1,-1,-3,10,-3,10,25,35,25,35,0
```

Figure 2.1: SDL in STI SIM.

The NADS (National Advance Driving simulator) developed by the National Highway traffic Safety Administration is located at the University of Iowa, USA. It uses a tool called ISAT (Interactive Scenario Authoring Tool) for scenario authoring and the Tile Mosaic Tool (TMT) for the creation of road networks Papelis et al. [2001]. ISAT is a front-end tool which uses

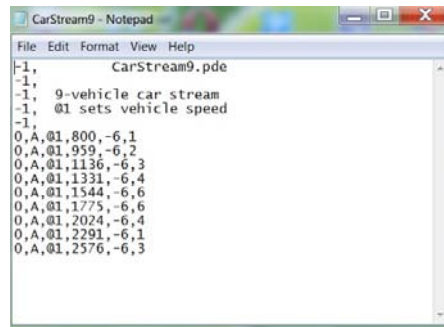


Figure 2.2: STIM SIM PDE.

a Scenario Definition and Control (SDC) software component. The SDC uses different scenario elements during the scenario modeling process. It has a Scenario Object Library (SOL) which includes deterministic Dynamic objects (objects with no autonomous behaviour), autonomous dynamic objects (objects with autonomous behaviors), traffic manager, traffic light manager, triggers and actions. An example of the ISAT is shown in Figure 2.3.

Besides the development of scenario modeling tools, some work has been done on the performance and efficiency of using scenario development approaches and tools, which affect the efficiency of using these tools in terms of acquiring good data. Papelis et al. [2005] has addressed some issues which affect the performance measures of the study as an outcome. Lot of efforts are being made in order to address issues to reuse or replicate the scenarios, which includes software-engineering, visual database, frame rate audio/vibration feedback, simulator disorientation and scenario implementation details. Unfortunately performance measures are not addressed. The point is to assess how different driving simulators render the cues necessary to implement a virtual environment.

Simulator fidelity has an effect on performance measures, which are quite sensitive to the hypothesis under study by the researcher who designed the experiment. Performance measures vary with the experiment and the hypothesis: for instance, in the study of drowsiness/drug impairment, during a car following event, the performance measures may be the velocity variance, the lane deviation, or lane deviation variance, etc.

Researchers usually replicate and re-use scenario events in order to compare the result during the studies. Ahmad [2005] has highlighted this issue as time-consuming and error prone. They have proposed a technique to solve this problem: they propose to specify the general topology of the scenario event, and to add some logical anchor points (position), to add scenario elements relative to these anchor points and finally apply the general specified event to the real environment by mapping the anchor points.

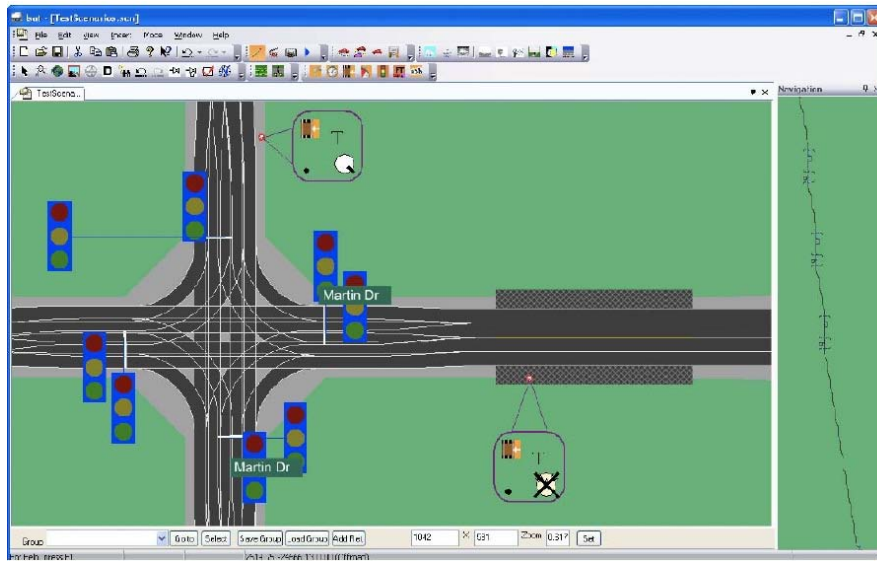


Figure 2.3: ISAT Scene Authoring GUI [Kearney and Timofey F, 2011].

2.2 Scenario modeling approaches

This section briefly discusses different flavors of interaction environments for scenario modeling approaches and the methodology used by the driving simulators for modeling scenarios. It briefly discusses the User Interface (UI) of some of these simulators, i.e. how to place scenario objects in the scenario, and how to manipulate the scenario events.

The SCANeR software [Reymond et al., 2000], developed by OKTAL SA and Renault, has a GUI for implementing the experimental protocols and to model the scenarios. The scenario objects (vehicles, pedestrians, traffic signals) are placed on the map using the mouse. The ambient traffic is generated by placing the "sources" on the map or by placing individual vehicle in the map. To build critical situations, it uses a GUI for defining condition-action pairs (If-else statements) (See Figure 2.4, top.).

ARCHISIM [Espie et al., 1994] developed by IFSTTAR has a textual interface to define an experimental protocol and to model scenarios. The scenario objects are specified in the text files by specifying the location on the road. Ambient traffic is created manually using text files. To construct critical situations, it uses a simple text editor like notepad. In the notepad, the user can specify the condition-action pairs for different events (See Figure 2.4, bottom.).

STISIM, as explained above, has a textual Interface for scenario modeling process. SDL (Scenario Definition Language) is a scripting language developed to define the scenario events [Park et al., 2011]. The scenario objects are placed in the scenario by the route travelled by the driver using the following statement of SDL:

```
on distance, Event, Appear Distance, Parameter1, parameter2... Parameter (n)
```

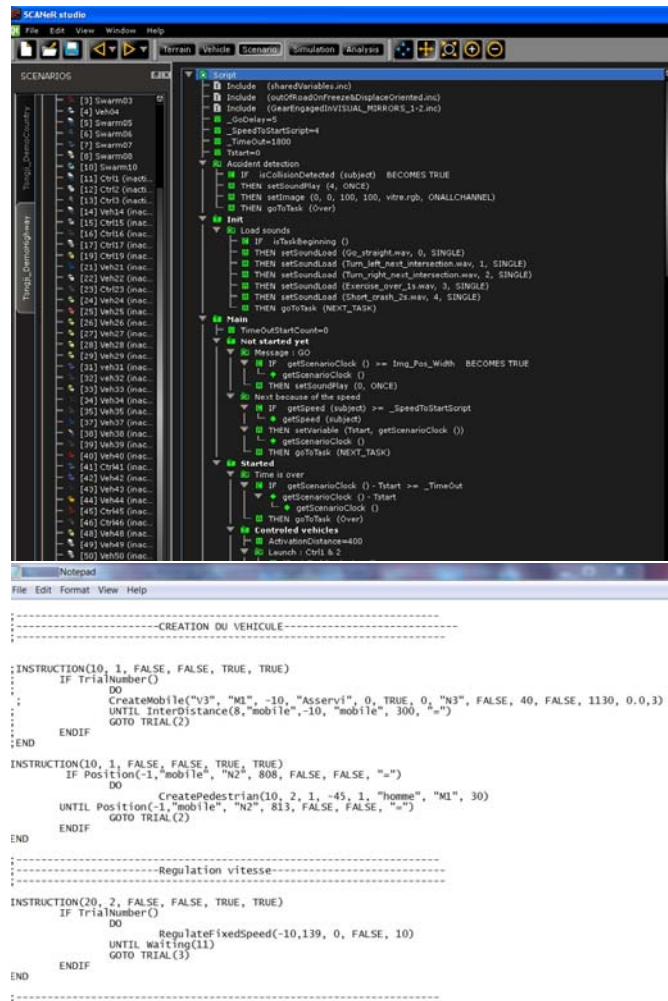


Figure 2.4: SCANer (top) and ARCHISIM (bottom) GUI for scenario authoring.

Where "on distance" is the distance travelled by the driver during the simulation. "Event" refers to the specific object (buildings, vehicle, pedestrian, etc.). "Appear Distance" is the longitudinal distance relative to "On Distance", when the event will appear in the scene. Ambient traffic is generated by pre-defined events in the text file using textual statement described above. (See Figure 2.1).

Three types of interfaces to model scenarios on driving simulators have been presented: placing objects directly on a map as in SCANer, specifying the position of the objects using textual statements as in ARCHISIM, and using the route traveled by the driver during the simulation trial.

Sometimes it is required to dynamically generate a script during the simulation. Some driving simulators provide this functionality, and some use other means to fulfill this requirement. Wassink et al. [2005] have proposed a movie-set metaphor to generate scenarios dy-

namically based on Green Dino Virtual Realities' Dutch Driving Simulator. They have proposed the movie-set as a driving simulator, where actors (vehicles, pedestrians etc) come at the scene and play a certain set of roles, which are assigned to them in the script. They have also emphasized on the user's problem of to model scenarios using a scripting language. A tile-based approach is also used to specify scene and scenario elements in driving simulators. The whole scene is divided into different tiles, which are configured, assembled and loaded into the driving simulator during the experimental trial. A 'Tile' is a section of the route which contains elements such as roads, traffic signals, buildings, trees and scenario objects. These tiles are then grouped together and loaded using an interface or by specifying the sequence of tiles. The scenario events are then created on the map. In some systems, tiles are static and may not be altered or moved during the experiment run [Papelis et al., 2003, Krueger et al., 2005], while there are some systems in which tiles as well as data on the tiles (scene objects, scenario objects) can be altered dynamically during the experiment run. This is the case with SimCreator (SimVista) [Romano, 2003] where tiles are assembled dynamically with the scenario data. They use sensors and markers as triggering objects, and traffic maneuvers objects are placed to perform different scenario operations.

Many scenario authoring tools use a tile-based approach for modeling scenarios have been proposed. It is the case with the "Tile Manager Interface" Software, which uses XML to define scenarios in the tiles [Suresh and Mourant, 2005], "SILAB", which claims to be a task-oriented simulation where the database is dynamically linked depending on the task the driver is currently performing on the driving simulator [Krueger et al., 2005]. Another similar approach is used at the Federal Highway Administration (FHWA) in their Event Driven Dynamic Interactive Experiments (EDDIE) scenario control software. There are some shortcomings in the tile-based approach: for instance, there is a restricted visibility for the drivers, and specifying traffic density especially at the edges of the tiles can be challenging. An example of scene authoring using tile-based approach for SimVista is shown in the Figure 2.5.

To sum up, almost all the scenario modeling approaches for driving simulators use condition-action approaches to specify the events and maneuvers encountered by the drivers during the simulation. These tools and approaches have not considered how end-users (e.g. researchers) interact with these tools or how these tools assist them, in such a way that they can specify the scenarios easily and in an intuitive manner.

2.3 Scenario authoring language

Scenario authoring languages determine and manipulate objects (Vehicles for example) and their behavior in order to achieve realism during the driving experience on the driving simulators. Kearney et al. [1999] have highlighted issues in the design and requirement for developing scenario languages for driving simulation. The scenario authoring languages should be able to address the following operations:

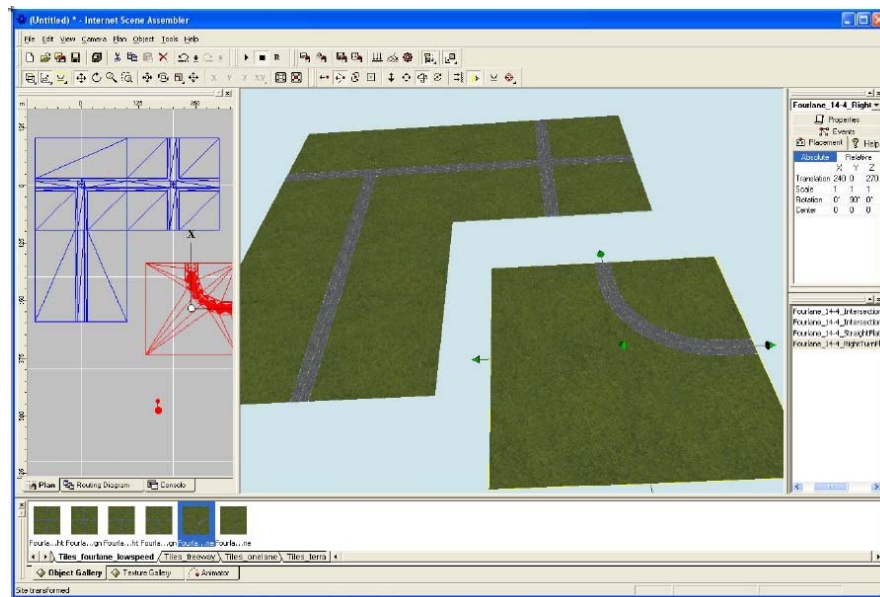


Figure 2.5: Assembling scene from tiles in Simvista [Kearney and Timofey F, 2011].

- Event management: How events are managed during the scenario, based on a continuous monitoring or by sensor on the events.
- Abstraction: To encapsulate the scenarios into modules which can be later reused during the same scenario or for other scenarios using different parameters.
- Semantics for scenario invocation: How different scenarios will be invoked during the scenario execution.
- Scheduling the scenarios: The order of the scenario events is often very critical, so one should be able to schedule the scenarios based on temporal and spatial relationships between the scenario events.

Based on these requirements and discussions, HPTS (Hierarchical Parallel transition Systems) and HCSM (Hierarchical Concurrent State Machines) [Cremer et al., 1995] have been designed at IRISA, Rennes and at IOWA state University, USA respectively.

Leitao et al. [1999] have developed a scripting language to control the ambient traffic in scenarios on driving simulators. The language is based on the Grafcet or Sequential functional charts (English version of Grafcet). Grafcet is a graphical language which is commonly used in industrial computing for programming the controllers [David, 1995]. The common elements of Grafcet are Steps (actions), transitions (based on conditions), and the connection (between different steps).

Devillers and Donikian [2003] developed a generic scenario language to author virtual worlds. The proposed language uses state machine (FSM) to specify the instructions. In the state machine approach, we have states and transitions. A transition from one state to another depends on the conditions or actions performed on the states. It discusses the grammar for the scenarios language fulfilling the requirements for scenario language, i.e. actor management, scheduling and abstractions. The use of state machines and transition systems can be used in an efficient way for authoring the scenarios but these states and transitions are made at low-level, which makes them difficult and complicate to specify the scenarios at high-level.

Wolffelaar et al. [1999] has proposed a NSL (Network Specification Language) and a SSL (Scenario Specification Language). The network specification language is used to define the scene of the scenario, for example the road network profile and the objects associated with the network. Like the condition actions features of all the languages described above, they also use the same architecture, but besides this condition action paradigm, they use the feature of high-level programming languages such as variables, functions.

Gajananan et al. [2011] have proposed a Scenario Markup Language (SML) to author scenarios in the domain of driving simulation. As the name specifies, it is a markup language based on XML and developed for multi-user driving simulations where multiple drivers are involved during one simulation trial. SML also claims to be at high-level. But the structure of this language is tightly bound to their simulation engine OpenenergySim [Nakasone et al., 2011]. The scenarios specified using SML are at high-level, but they use a specific structure which can only be executed on their specific simulation platform, and cannot be used on other platforms developed around the globe. So, one of the main objectives of our work is to develop a high-level scenario specification language which could be executed on most of the driving simulation platforms.

So far, not enough work has been done to develop a framework or a specification language which could be used on different driving simulation platforms. Some work has been done however in defining military scenarios for the purpose of interoperability. The OneSAF Objective System (OOS) program has created an XML-based mechanism for specifying scenarios: the Military Scenario Definition Language (MSDL) [Franceschini et al., 2004]. It is an open specification and is designed to allow the decoupling of planning and execution systems to enable a broad range of interoperability and reuse during the simulation. MSDL has four important characteristics [Wittman and Abbott, 2006]:

- Separation of code from data. MSDL contains all the data necessary to define the initial condition of a military scenario.
- Using Industry scenario for data format and content definition. MSDL is defined using the World-Wide Web Consortium (WC3) extensible Mark-up Language (XML).
- MSDL documents are application-independent and are explained by standardized XML schemas.

- Separation of concerns. MSDL contains information related to military scenario and does not include any application specific or device specific information.

Like the concept of MSDL, an attempt to create an open source Battle Management Language (BML) has also been carried out. The BML framework is used to:

- Command and control forces and equipment conducting military operations;
- Provide the situational awareness and a shared common operational picture. It can be viewed as a digital commander issuing commands to the troops [Hieb et al., 2004].

Military scenarios have a specific context different from the context of driving scenarios, so the architecture of the development of this specification language does not correspond totally to the driving simulations scenarios. Another purpose of these languages was to be able to provide an open framework to allow the simulation systems to communicate with command and control devices from different vendors involved in the military scenarios. The main idea in MSDL is the decoupling of the planning system and the execution system, which is related to our problem, in which we are decoupling the experiment building platform from their respective execution platform.

2.4 Driving simulators and other domains

All the authoring languages discussed above belong to the specific domain of driving simulation. We call Domain-Specific Languages the language specifically designed for the domain of driving simulation. "A domain-specific language (DSL) is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain" [Deursen et al., 2000]. The benefits of DSL over general purpose languages include an increased productivity, conciseness, reuse of domain knowledge and validation at the domain level. DSLs are used in almost every domain for productivity and efficiency of the systems developed in these domains for example finance, multimedia technologies, telecommunication, modeling and simulation, etc. Some well-known examples of DSL are SQL, spreadsheet and HTML. DSLs are also used in virtual reality systems. Driving simulation is an application of virtual reality systems and can be regarded in some cases as a serious game. Serious games are the games whose primary purpose is not entertainment but to educate end-users. As explained before, driving simulators are also used for the driver's training, so methodologies used in the domain of virtual reality systems and serious games can be used in the development of driving simulation systems.

But what makes driving simulation different from serious games and other virtual reality systems is that the primary users of these systems are non-technical users who have to manipulate the real-world traffic situations on the driving simulators in order to achieve their study

or training objectives. Significant amount of work has been done to ease the process of learning programming. But in the domain of driving simulation, the purpose of researchers is not to learn the scenario languages but to use these languages to develop their experimental protocol.

2.4.1 End-user programming Systems

In this section, we discuss different end-user programming approaches and techniques which have been proposed before. A significant amount of work has been done on end-user programming.

Ko et al. [2011] has conducted a literature review for end-user software engineering from the prospect of software engineering. They have analyzed the different phases of software engineering with a comparison to end-user development, from the requirement analysis to the design, debugging and testing phases of development. Paterno [2013] has also conducted a detailed and an interesting survey on end-user development with the prospect of empowering the end-user to develop applications. In contrast to Andrew's survey who addressed only desktop applications, Paterno also conducted the survey for mobile and web applications. Myers et al. [2006] have given an overview of the research being conducted in the domain of end-user programming systems. Many programming paradigms have been proposed for end-users in different domains, to make the process of programming easier for the end-users (Cypher and Smith 1995). One of the famous paradigms in end-user programming is Programming By Demonstration (PBD) or sometimes called programming by example. In PBD, end-users develop their application by carrying out the steps as an example. The programming system learns what the user is actually doing, then perform these steps automatically. So the end-user actually demonstrates an example by executing the sequence of actions, and the system infers a general application supporting the desired behavior. The earlier work on PBD was performed by Myers [1986], which was further extended by [Cypher, 1991, Cypher and Smith, 1995], who have used it for the simulation tool Eager for hyper card environments. Eager monitors the user-interaction history of the events, and detects the looping pattern. PBD is an easy approach to use, but generalizing the behaviour from a specific one is a challenging task for end-users, and they are not very expressive to build complex applications. Programming By Analogy (PBA) is a programming paradigm in which a problem is solved by using the concepts and techniques used to solve similar problems in the same or in different domains. Using analogies ease the process of understanding a problem, and can help the learners to understand the problems by linking the abstract concepts with specific situations [Anderson and Thompson, 1989].

Different notions regarding PBA have been proposed in the past [Dershowitz, 1986, Schmid et al., 1998, Dunican, 2002]. Repenning and Perrone [2000] have proposed an alternate version of PBA called programming by analogous examples. It is the combination of programming by example and programming by analogy. In this approach, the idea of programming by ex-

ample is preserved while increasing the reusability of the program by combining it with programming by analogy. But PBA is not free from challenges; one of the issues is to deal with the exceptions to one domain, which cannot be used in others. A similar to PBA is programming using metaphor. The difference between programming using metaphor and PBA is that PBA uses an abstract concept to solve a problem at a specific level, while programming using metaphor is a technique in which familiar situations are used during the process of programming. Programming using metaphor has been used in different systems. For example, Logo programming environment [Papert, 1980] was the pioneer to use the programming using metaphor. It uses the turtle physical movements as a metaphor to manipulate the programming activities as turtle graphics. Another systems called HANDS (Human-centered advances for Novice development of software) was developed in the framework of a natural programming project [Myers et al., 2004]. It is an event-based system featuring a concrete model for computation based on concepts which are quite known to non-programmers [Pane, 2002]. In HANDS, all the programming activities are responded by a character sitting on a table like people sit on the table while playing cards. The character reacts to the events and manipulates different cards that contain the data for the program. Natural programming is a programming through a language which is close to the way people who do not have programming experience would expect.

Another approach which has been widely used in many EUD systems is visual programming for example (LabView, Matlab Simulink, Prograph etc.). One of the goal of End-user development is to reach closeness of mapping as Green and Petre [1996] stated: "The close the programming world is to the problem world, the easier the problem-solving ought to be. Conventional textual languages are a long way from that goal." Even graphical languages often fail to furnish immediately understandable representation for the developers. Visual programming have been able to achieve this closeness of mapping and also helped to overcome the syntax problems, but they still have some weak points. As control flow is always a problem in visual languages, graphic representation of the programs are usually not abstract enough [Myers, 1986]. Some interesting surveys have already been conducted on visual programming [Glinert, 1990, Boshernitsan and Downes, 2004].

The main goal of End-user development is to empower the end-user so that they can develop the systems by themselves adapt it easily. According to Lieberman et al. [2006], from the user-centered design perspective, end-user activities can be categorized as belonging to one of two different types.

Parametrization or customization

These activities involve the change of behaviour of a programming entity by end-users, by customizing this entity. Or they can choose among different alternate behaviour available in the system. In adaptive systems, this customization happens automatically using the user activity on those systems. For example one may configure the email client by customizing it according to the user's choice. In programming by customization, the programming entity

is a kind of template which can be customized according to the end-user requirements. But they are not very expressive and it can be difficult to program the complex entities by just using customization.

Program creation and modification and reuse

These activities involve modifying the existing computational entities to modify the existing computing artifact or developing it from scratch. These computational artifacts can also be reused by modifying them, and used for the current programming task. The modification and reuse can be at a conceptual level, at a design level or at low-level.

Programmorphism [Ioannidou, 2003] is an end-user programming approach which enables end-user programmers (novice) to define the behaviour of the interacting agents in a high-level abstract language. Behaviour templates are used to structure domain concepts. It uses modification and customization in the process of programming and replaces high-level specification with low-level representation in the program. By definition, there are two transformations in programmorphism:

1. When the programming process is transformed from a synthesis task to a customization and modification task;
2. When the program itself is transformed from the high-level to low-level language specifications.

The main idea is the representation of the knowledge base in templates which can be customized by the users during the process of programming. But they are still strict to the rule specification. Evaluations of this technique have found that the major challenges to the end-users are the translation of their ideas into workable computer programs.

2.5 Dealing with designing interfaces for end-users

Human computer Interaction (HCI) also known as man-machine Interaction (MMI) or Computer-Human Interaction (CHI) involves the study, planning and design of the interaction between people and computers. It is often regarded as the intersection of computer science, behavioural sciences, design and several other fields.

According to [Pane et al., 2002], a programming system is the user interface between the programmers and the computer. Programming is a notoriously difficult activity, and some of this difficulty can be attributed to the user interface as opposed to other factors. Programming languages have been designed without giving careful attention to the HCI issues [Newell and Card, 1985]. This is why the purpose of interfaces is to minimize the gap between end-users and the programming system. The only way to allow the user to use these systems

effectively is to develop approaches close to the real-world and to minimize the gap between end-users and programming system by developing user-friendly and intuitive interfaces.

Task modeling technique is used to determine the user activities to reach their goals. Some work has been done about how to use this effectively to design user interfaces and interaction patterns [Hallvard, 2008]. Understanding and analyzing the user-tasks gives the basis for designing usable interfaces for end-users [van Welie, 2001].

One popular design methodology of HCI is the User-Centered Design (UCD) approach, in which user is in the centre of design for any computer system. In this approach users are involved at the early stage of the design, and they work together with designers and technical experts involved in the design. [Vredenburg et al., 2002] have conducted a survey of user-centered design practices and evaluated each practice practically. UCS [2002] has specified pros and cons of different UCD methods. Some work has been done on using two methods together in order to evaluate the UCD [Rieman et al., 1995].

The design and evaluation of a user interface prototype is central to the UCD process. Designers usually use a variety of prototyping tools and technologies, ranging from simple art supplies to custom prototyping languages and environments. Studies have been conducted in order to survey how designer program interactive behavior and how the prototype evaluation of the design is done in practice [Myers et al., 2008, Carter and Hundhausen, 2010]. There are tools and techniques which can be used to evaluate the design and model programming tasks in order to evaluate the user performance on the new interfaces (of programming systems), for example CogTool [Robin et al., 1991, Bellamy et al., 2010].

Chapter 3

End-user requirements

In order to find the solution of the problem as mentioned in the previous chapter, user-centered design approach is used. As the name suggests this is the design approach where user is focused during the design process.

3.1 User-Centered Design

As stated by Mao et al. [2005], "User-Centered Design (UCD) is a multidisciplinary design approach based on the active involvement of users to improve the understanding of user and task requirement, and the iteration of design and evaluation". As the name UCD suggests, the outcome of the product using UCD approach is developed by placing the user as the centre of the design. The international standard ISO 9241-210 (formerly known as ISO-13407) provides the basis for UCD processes. UCD is a common practice in the development of software which follows different phases like analysis (user and task analysis), design, prototype and evaluation (design iteration), implementation of the design and finally the product deployment. UCD methods claim to provide an end-product, which satisfies the users and enables them to achieve their goal while taking their profile into account. However many claims have been found in the literature about the UCD methods to be impractical and inefficient for different reasons [Gulliksen et al., 1999, Mao et al., 2005]. But many developers don't use basic usability principles, because they find them very expensive, time-consuming and intimidating in their complexity [Bellotti, 1988, Nielsen, 1994].

There are many UCD methods (e.g. user analysis, prototyping, heuristic evaluation, cognitive walkthroughs, etc) which are practiced by the Usability practitioners. Many surveys [Vredenburg et al., 2002, Mao et al., 2005] have been conducted to find out which UCD methods are being practiced in the industry, or which methods and measures are found to be effective in the UCD.

Though recent scenario modelling systems are powerful enough to manipulate almost all

kind of traffic maneuvers and situations, they do not take into account the profile of the end-users (researchers), who are the primary users of the driving simulators. Also, there is no such system, which is being designed by considering the user as the centre of the design. So we think that the needs and profile of the end-users of driving simulators should be reflected in the design of scenario authoring tools for driving simulators. That is why UCD approach has been used to model scenarios on driving simulator. The next section explains the details.

3.2 User Interviews

We have used Interview technique of the user-centred design practice. Interview technique has many advantages over other techniques. It is an effective way of gathering the user needs and requirements by asking open questions, and also to understand the user's profiles and the context of their use. So involvement of the users at the start of the designing process was necessary. The needs of the users' requirements was explored in 4 different themes, which includes

1. What are their general problems (Problems that they face when they are operating the scenario authoring tools).
2. Concept to script translation problem (If they think about a traffic situation in the real world, and they have to develop this situation using a scripting language).
3. What is an ideal method or approach for them to model scenarios? (With current set of skills, what are their expectations from the scenario authoring tools, and how can they account for the skills, which are required to model scenarios).
4. Their ideas and the feedback for the improvement of driving simulator in their use.

3.3 Method

This subsection explains the participants and the method to conduct the interviews of the end-users.

3.3.1 Participants

19 Driving simulator users (7 females, 12 males) with various backgrounds, profiles and experience on driving simulator were informally interviewed, as shown in Tables 3.1 and 3.2. All users except two software engineers had no programming background. Most of the users had used ARCHISIM or SCANeR software for their work. But users from other driving simulator were interviewed as well.

Driving Simulator	Number of User(s)
ARCHISIM	7
SCANeR	6
ST SIM	2
STI SIM Drive	1
VTEC Driving Simulator	1
Honda Simulator	1
DriS	1

Table 3.1: Driving Simulators.

Users	Number of User(s)
Ph.D. Researchers	7
Behavioural Researchers	8
Software Engineer	2
Research Engineer	1
Post. Doctorate	1

Table 3.2: User Profile.

3.3.2 Procedure

Six users were interviewed in person, as they were from IFSTTAR. The rest of the users were interviewed on phone, except 1 user who was interviewed on Skype. The average duration of the interviews in person was 40 to 45 minutes, the average duration of telephonic interview was 50 to 60 minutes, and the Skype interview took 83 minutes. The participants were not given any incentive to participate in the interviews.

The interview was qualitative, semi-standardized, and open-ended and same questions were asked to all the users. But based on the user feedback, some other questions were also asked to them.

The interviews were not recorded, but it was made sure that any relevant details were not missed by preparing some formats on note-taking sheet.

3.4 Results

These qualitative interviews have provided a very good platform to explore the user needs by discussing in depth about their experience of modelling scenarios on driving simulators. The details of the participants programming language experience is shown in Table 3.3.

The expertise level of the users for these programming languages was basic for all languages except software engineers, one researcher and one research engineer, who had good exper-

Programming Language	Number of User(s)
C++	3
Visual Basic	3
Matlab Simulink	3
HTML	2
Java	1
Action Script	1
Python	1
No experience	7

Table 3.3: Programming Language Knowledge of the Users.

tise in C++ and Matlab Simulink. The software engineers model scenarios for the behavioural researchers. 9 of the 19 users do not model scenarios at all, so technical persons or software engineers model scenarios for them. The remaining 8 users model scenarios with the help (major or minor) of the software engineers or technical persons in their institutions. Seven users had no knowledge of programming at all.

As explained above the interview has been categorised into 4 different themes. The subsections "General problems" and "Concept-to-script translation problems" describe the users problems, while the subsections "Ideal approach to model scenarios" and "Improvement of the driving Simulator in use" describe the users ideas and feedback for the improvement of the driving simulators.

In the remaining section, there is a discussion about the users' problems, their needs and requirements which is identified by users. The subsections "General problems" and "Concept to script translation problems" describe the users problems, while the subsections "Ideal approach to model scenarios" and "Improvement of the driving Simulator in use" describe the users ideas and feedback for the improvement of the driving simulators. The problems and the ideas mentioned below are mentioned by most of the users, while some problems are platform specific. Our focus is to identify the problems in general, which the user faces due to lack of programming skills.

3.4.1 General problems

1. Finding/Selecting the relevant functions in order to perform a specific task. It is difficult for the users to find the relevant low-level function/procedure to perform a certain task, and also to find out that which functions can fulfill their task.
2. Sometimes, it is difficult to test the rules before running the simulation. If a situation has to occur after 25 minute of simulation time, they can only test the situation after 25 minutes of simulation and not before.
3. Debugging the Script in order to find the error is quite difficult and time consuming. It

is difficult for the end-users to debug a scenario, to fix a scenario which is not working; it is even more difficult, when the script is long and complex.

4. Syntax of the scenario modelling language is difficult for the users to follow. Usually all the scenario modelling languages are at low-level, and require programming skills, knowledge and experience.
5. The users have to control and program the scenarios by doing programming at a low level, which is quite difficult for them. For example, the situations which require a precise tuning, for example, when a car approaches an intersection and another car have to cross the vehicle.
5. A scenario is composed of different testing conditions. When the script is long and complex, it is difficult to go back and forth between those conditions (mainly the IF-ELSE condition).
6. Scenario modeling authoring process is not transparent to the end-users, as they have to develop their scenarios using the terrain. They can't figure out that when and how a situation will happen unless they run the simulation.
7. In most cases, the scenarios are developed for researchers by scenario developers. Sometimes it is difficult for scenario developers to interpret the situations wanted by the researchers. For example, a specific maneuver requires that, a car should overtake another car suddenly. It is difficult to interpret the word suddenly. As for scenario developer, suddenly is a parameter that has to be set.

3.4.2 Concept-to-script translation problems

The "Concept to script" problem refers to the problems that users face, when they want to manipulate the real-world situation on driving simulator using the programming language of driving simulator. According to the users, this is a difficult step for them as they can't implement the traffic situation, the way they actually think about the situation they think. The main problems identified by users are as below.

1. Controlling the ambient traffic using the script is very difficult for the users. The ambient traffic is used to create some level of realism in the experiment. Each vehicle has to be created individually, which is a time taking and cumbersome task. In order to have realism and reproducibility, it has to be controlled by script, which is quite difficult for unskilled end-users.
2. Tuning the traffic situations is uneasy. In order to make the situation happen at the right place and right time, different variables have to be tuned. A change in one variable may impact the other variable, which is difficult to interpret for the user in most cases, if they are not aware of the internal working of the function (API) used to model a specific situation. Also end-users usually can't figure out which variable to control for a specific situation. Consider a situation that as a participant vehicle approaches

an intersection, a car has to cross the intersection violating the traffic rule, and crosses the road making the participant to either stop or encounter an intersection accident. For such situation, we cannot control the participant's speed or position, but we can control the speed or the position of the vehicle crossing the intersection.

3. Selection of triggers (speed, distance etc.) in order to tune the events. Sometimes it is confusing for the end-users which trigger (position of a vehicle, simulation time) to use for the situation.
4. Reproducibility of the situations needs extensive scripting, and is difficult to manage. For realism and reproducibility of the scenarios, low-level scripting is required, which is a difficult task for the end-users.

3.4.3 Ideal approach to model scenarios

After the users described their problems, they were asked about their ideal approach to model scenarios, and how their problems can be solved. These ideas are described as below:

1. Drag-and-Drop a situation for a scenario. Ten users out of 19 proposed the notion of Dragging and dropping the scenario situations. They think that, this way, they would just have to specify the specific parameter for a specific situation, and not the complete code.
2. High-level scripts for scenarios. This idea is the same as 1, but when users were told that, every user has his own requirements, so it will be difficult to manage long list of libraries for dragging and dropping, then they specified that, in order to use High-Level script for complex scenarios, there should be a functionality of editing the high-level script at low-level.
3. Interaction with the map. The users want to interact with the world map while modelling the scenarios.
4. There should be an image/preview of the work that the user is doing. They want to have the overview of their work.
5. It will be more comfortable for the users, if they could click on autonomous vehicle and set their rules and behaviours in the scenario.
6. It would be easier to specify and follow the tasks/events based on a time-line or distance-line of the experiment protocol.

3.4.4 Improvement of the driving Simulator in use

Users were asked how the driving simulator software in their use can be improved to model scenarios. Users' responses to these questions are described below.

1. Searching of the relevant functions should be made easier. A good search facility for the functions should be provided which would help user to find the relevant function for their task.
2. Interface of the scenario modelling software. All users have used textual interface for scenario modelling, except the users of SCANeR software, which has Graphical User Interface (GUI). So most of the users who are using textual user interface would prefer using a GUI for the scenario modelling.
3. There should be a very user-friendly documentation for how to use the relevant functions, with some examples.
4. There should be only one interface in order to configure different files on the driving simulators. One of to configure different files in some driving simulators in order to run a simulation. For example in ARCHISIM and STISIM.

3.5 Discussion and conclusion

The user problems and ideas given above have a minimum frequency of 3 occurrences among all the participants in the experiment. During the interviews, users were also asked about their general experience about driving simulators and what is the formal/informal process they follow to use a driving simulator for their experiment. Users were also asked about the different steps they follow to design an experimental protocol, which is explained in the next chapter.

Problems faced by the users and also their expectations from the driving simulators to model scenarios have been discussed. For some problems the solutions are already proposed by the end users while proposing the ideas for driving simulators. The user's problems matching with user ideas are shown in the Table 4.

As behavioral researchers are the primary users of the driving simulators, their requirements and ideas should be reflected in the design of authoring tools for driving simulators. Also the interaction environment of driving simulators should account for the skills they lack and fill the gap between the user skills and the goals the end-user wants to achieve using a driving simulator.

Chapter 3. End-user requirements

Ideas/feedback	Description	Problems addressed
Drag & Drop Situation for a scenario.	If there are some high-level packages of situations and maneuvers e.g. traffic jam. Traffic jam should be created based on drag and drop, providing number of vehicles. Or some kind of template for specific situations to be used just on the few clicks.	3, 4, 5, 8 9 and 10.
High-level scripts	If software takes most of the scripting responsibility and users just have to click on few things to generate the script. This script should be editable at the low-level, if users have skills to develop the script by themselves.	3, 4, 5, 8 9 and 10.
Interaction with map	It would be quite useful if we interact with map while modeling the scenarios.	7
Geographical user Interface (GUI)	A GUI would be useful in order to model the scenarios.	7
List of variables in GUI	A GUI should provide a list of variables to be studied that users could select.	7
Preview or Image of what users are doing	There should be an image or preview of the work, which users are trying to do Which means they should be able to see, what they are trying to do.	7

Table 3.4: Solution to problem mapping.

Chapter 4

Proposed Multi-Layer programming approach

This chapter discusses a new approach to model scenarios on driving simulator. First the key challenges are discussed, then the different users who interact with the driving simulators will be considered, which is followed by a discussion on the steps followed by the researchers, and finally our new approach is given in details.

As UCD approach has been used to find the solution of the problem, so users are involved at the start of the design and their profile is studied by conducting interviews and their requirements need and preferences are identified as explained in Chapter 3. The UCD process enables the users to get involved at the early stage of the design, which makes designers understand what end-users expect from the system and how they perceive their work, and also enables the designers to understand the user requirements and needs in the context of their work as well as their preference to interact with the systems in order to achieve their goal.

4.1 Key challenges for researchers

As discussed before, there are many key challenges which makes scenario modeling process a difficult task for the end-users.

1. It is difficult for the researchers in most of the cases to master the skills to develop scenarios. So they want to have a tool or approach which can fill the gap between their skills and the objectives they want to achieve using the driving simulators.
2. Every driving simulator language has its own syntax and method, so if researchers have to do collaborative work in other laboratories or at other jobs, they have to learn new tools.

3. It is a time-consuming task for researcher because of their dependency on technical persons. It is also a time-consuming task for technical people, because they need to have frequent meetings with the researcher in order to finalize and validate the scenario before the experiment.
4. Sometimes, it is difficult to interpret the situation between researchers and technical persons.

4.1.1 Technical challenges

Besides key challenges for researchers, there are some technical challenges that need to be considered while proposing a solution for end users.

1. First challenge that arises is, whether the proposed solution will be just another approach like many other approaches and methods offered by the scenario authoring tools of the other driving simulators. And will the existing system need an overhaul.
2. How different driving simulator users can reap the benefits from the proposed approach, as scenario authoring environment of all simulators is tightly bound to the execution platforms.
3. Transformation of researcher experimental development process into meaningful interface that reflects their way of thinking to develop an experiment.
4. There is no standard method of reusing the scenarios which are developed elsewhere even on the same driving simulators. As the probability of using the same scenarios with minimal changes is high, so if these scenarios can be reused by researchers without developing them from scratch, it will save time for both researchers and technical persons.

4.2 User roles

Three different kinds of users have been identified, who interact with the driving simulators based on their skills and roles assigned to them.

1. Researchers: Researchers design experiments and conduct studies in order to meet their research objectives. Researchers, in most of the cases are not equipped with technical and programming skills required to implement an experimental protocol.
2. Technical persons: Technical persons implement experiment for researchers. They have formal training for programming and develop scenarios for the researchers, in case researchers cannot implement the experiment by themselves.

3. Operators: Operators can be a researchers or technical persons depending on the organization culture. In small organizations, the researcher has to operate the experiment and collect the data, while in some big organization, they have specific people to conduct an experiment but under the instruction of the researcher.

The roles explained above may or may not be three different persons depending on the organization culture. But in most organization, researchers adopt the role of operator as well. And in some cases, if the researchers have technical skills, then they design the experiments, develop them, and then also adopt the role of an operator. It means, one person adopts more than one role, but theoretically, there are 3 different roles.

4.3 Experimental Protocol development Steps

From the user-survey, different steps have been identified that are usually followed while designing an experimental protocol. These steps are given below in the order of occurrence.

1. Terrain selection (Users design the terrain for their experiment i.e. Highway, city, suburbs, number of lanes on the road, etc)
2. Configure participant/subject vehicle. Participant or subject vehicle is the cockpit vehicle, which is driven by the subject during the experiment. At this step the users think about initial position, ADAS (if it is the requirement of the project), maximum speed, and other parameters.
3. Specifying the ambient traffic (At this step, end-users think about the ambient traffic around the participant vehicle at different positions in the simulation environment. They think about number of vehicles per hour or traffic density and their speed, itinerary and behaviour) and traffic signals.
4. Configure the environment (At this step, end-users think about the environment of the experiment for example in snow, fog, rain or dark, etc.).
5. Construct critical events using scripting. (At this step, end-users design different traffic situations involving different traffic maneuvers during the experimental trial. They precisely design all the necessary situations for the experiment, for example Traffic Jam, Pedestrian crossing etc.).
6. Set the dependent variables (At this step, end users think about the dependent variables, they want to study during their experiment)
7. Experiment execution and data collection (At this step, end users conduct the experiment and collect the data recorded during the experiment trial).

4.4 Proposed solution

Based on the above-mentioned challenges, the user's roles and the procedure to develop an experimental protocol, a new end-user multi-layer programming solution is proposed in this thesis.

Traditionally, in order to implement an experimental protocol, a user (technical person or researcher) uses the same interface for implementing an experiment protocol, regardless of level of his technical and programming skills. A multi-layer programming approach is proposed. As the name suggests, there will be different layers of programming and researchers and technical persons will be interacting at different layers corresponding to their skills. The end-user (researchers) is then empowered by exploiting the typical programming process, and replaces it with customization and configuration process.

4.4.1 Multi-layer Programming

In our proposed multi-layer programming solution, end-users (researchers) interact with the higher-layer, which is close to the human language or real world, and the technical persons interact with the middle layer, which are the Application Programming Interfaces (API's) or the high-level functions of the driving simulator software. API is the set of programming instructions to access software applications. In the case of driving simulators, the API's can be for example Change Lane function (Instruction to change the lane of the specific car), Change Speed function (Instruction to change the speed of the car) etc. Technical persons develop Templates for the researchers using the API of the driving simulator. Then, there is the lower-layer, which is a platform specific layer and close to the machine language for example (C++, VBScript and JavaScript etc) depending on the software architecture of the driving simulator. It is used by driving simulator manufacturer to build the platform for the execution of driving scenarios. The multi-layer programming architecture is shown in the Figure 4.1. The middle layer functions (APIs) are usually developed by driving simulators developers (for example SCANeR by OKTAL), or it can be developed by the Technical persons of the driving simulators, if the driving simulator is built the research institution (for example ARCHISIM by IFSTTAR). So in principle, the researcher will interact with the higher-layer, and the technical person will interact with the middle-layer, and driving simulator manufacturers with the lower-layer. Traditionally, Researchers and Technical persons interact with the API layer to develop an experiment, where they have to follow the programming rules and syntax. We add an extra layer (Higher-layer) for the researcher, where they can develop the experimental protocol by exploiting the programming primitives. Using this layer, end-users can develop scenarios using high-level (necessary) information using the graphical programming.

4.4.2 Empowering the end-users

Although in some driving simulators (for example SCANeR) provide customization of the traffic maneuvers, it is at low-level, and end-users still find it difficult because the APIs are still at an abstract level and do not correspond to real-world situations. So the end-users are empowered by adding an extra "Higher layer" as shown in the Figure 4.1.

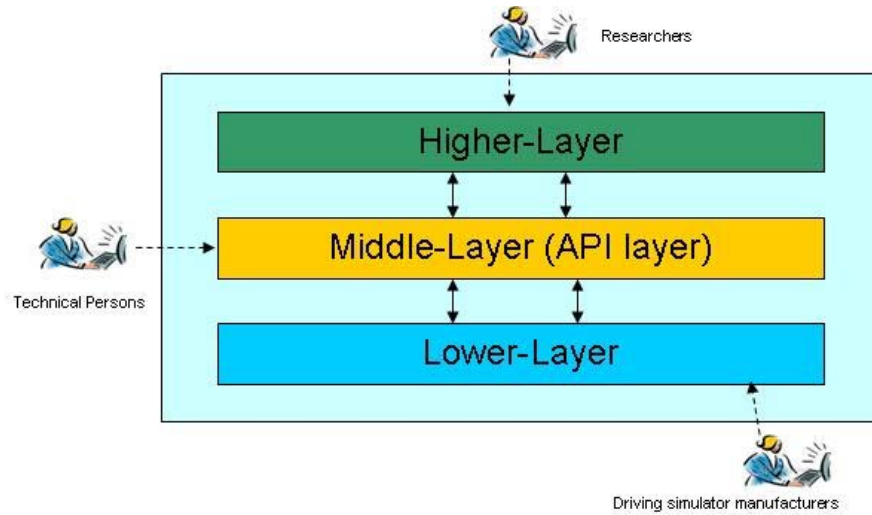


Figure 4.1: Multi-layer Programming Architecture.

The role of technical persons is not omitted, but they facilitate end-users (researchers) at the higher layer. The goal is to minimize the user dependency on the technical persons and empower them to develop experiments with no or minimal support from the technical persons.

Besides the empowerment of end-users, it will save time for technical persons as well. As they don't have to develop scenarios for each individual, they can develop situations for them, which will be used by the end-users. It is explained in detail in the rest of the chapter.

4.4.3 Scenario modeling process using Multi-layered approach

As explained above, 3 different kinds of users have been identified who actively interact with the driving simulators. So scenario modeling activity is divided into 3 sub-interfaces depending on the roles they have to perform while modeling scenarios and the set of skills they have. The 3 roles are "Technical Person" as "R1" (Good programming/technical skills), "Researcher" as "R2" (Low or no programming skills), and Operator as "R3" (No technical skills required). Based on the user roles, we split the scenario modeling activity into 3 sub-interfaces which are "Template builder", "Experiment Builder" and "Experiment Interface". The technical persons will interact with the Middle layer of our Multi-layer programming ar-

chitecture (see Figure 4.1) via "Template Builder" performing role "R1", the researcher will interact with the higher layer via "Experiment Builder" performing role "R2", and Operator will also interact with the higher-layer via "Experiment Interface" performing role "R3". The proposed approach is explained with the help of an example. The example scenario contains two events.

- *Accident Event:* A vehicle overtakes the participant's vehicle by increasing its speed, changes its lane to the lane of the participant's vehicle and then brakes.
- *Pedestrian crossing event:* A pedestrian walks and then crosses the road as the participant vehicle approaches.

Template Builder

This sub-interface will be used by technical persons performing role R1, and having good programming and technical skills. R1 will design the GUI-based templates of the scenario events. The template builder will let R1 use existing functions offered by the scenario-modeling environments of the driving simulator to model scenario events. In our example, at the back-end, for the template "Accident", the "Template Builder" will let R1 program a vehicle around the participant vehicle to accelerate, change position, and apply brakes at some distance from the subject vehicle. For the event "Pedestrian crossing", the "Template Builder" will let R1 program a pedestrian to walk and cross the road as the participant vehicle approaches the intersection. At the front end of the template, there would be different text fields to specify the parameters for the events "Accident" and "Pedestrian crossing". These parameters will be filled by the Researcher (R2) in the "Experiment Builder" sub-interface, if he or she wants to modify the default values. The templates developed by R1 will be stored in a template library, so that researchers could access the templates in the "Experiment Builder" interface as described in Figure 4.2.

Experiment Builder

This sub-interface will be used by researchers/trainers performing role R2 and possibly have low or no programming skills. R2 will define the whole experiment using a user-friendly and intuitive GUI which includes specifying the experiment conditions, environment, ambient traffic, and data to be collected (the usual steps of an experiment). To specify the critical events or situations to be studied, R2 will access the template library developed by technical persons in the 'Template Builder' and will place them in the scenario editor using drag and drop proceeded by a user-defined trigger. If needed, R2 will fill the parameters of the template.

In this example, the end-user will specify the position and actors involved in the templates "Accident" and "Pedestrian crossing" besides the template parameters, if needed. There will

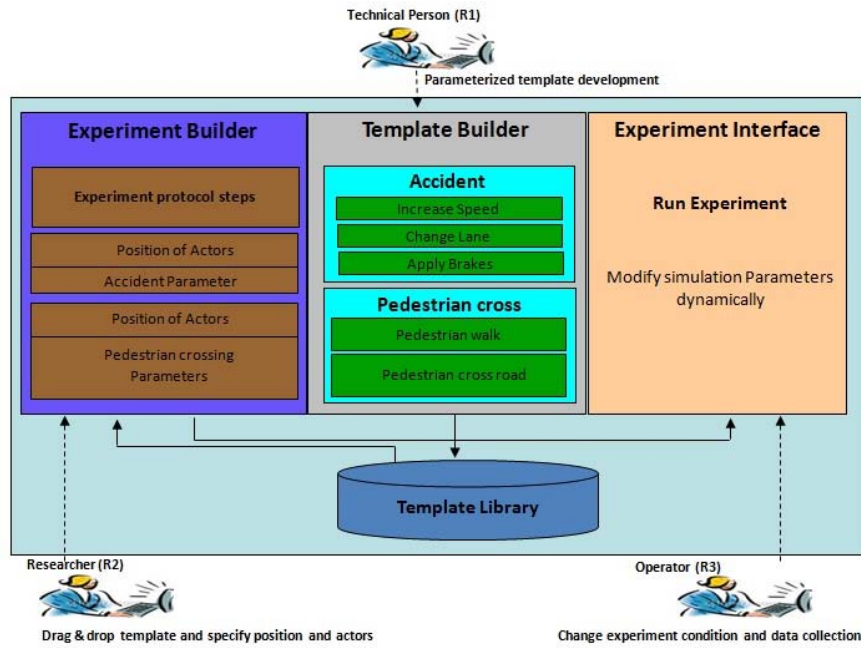


Figure 4.2: Experiment Building Environment.

always be appropriate default values for all the parameters of the template.

Traditionally, end-users have to configure different files to create an experimental protocol for example for autonomous vehicles, environment, variables to study, and critical events. In some tools, they have to configure everything in the same scenario editor or different text files, which are sometimes difficult to understand, when the experimental design is long and complex. So for researchers, a wizard-based interface is proposed, in which they can develop the experimental protocol in the same way they design it.

Experiment Interface

The "Experiment Interface" will be used by the user (researcher or the person who will execute the experiment, depending on the organization culture) performing role R3. Using this sub-interface, R3 will load and execute the scenarios in the driving simulator, developed with the "Experiment Builder". R3 can change the parameters of the scenario or template (if needed), during the experiment trial and finally collect the data.

4.4.4 Discussion on user roles

As explained in the previous chapters a scenario consists of different traffic situations (for example, overtaking of a car). The traffic situation itself consists of different low-level traffic or vehicle maneuvers. So the technical persons can develop traffic situations using API, which

will be used by researchers to develop scenarios as shown in the Figure 4.3.

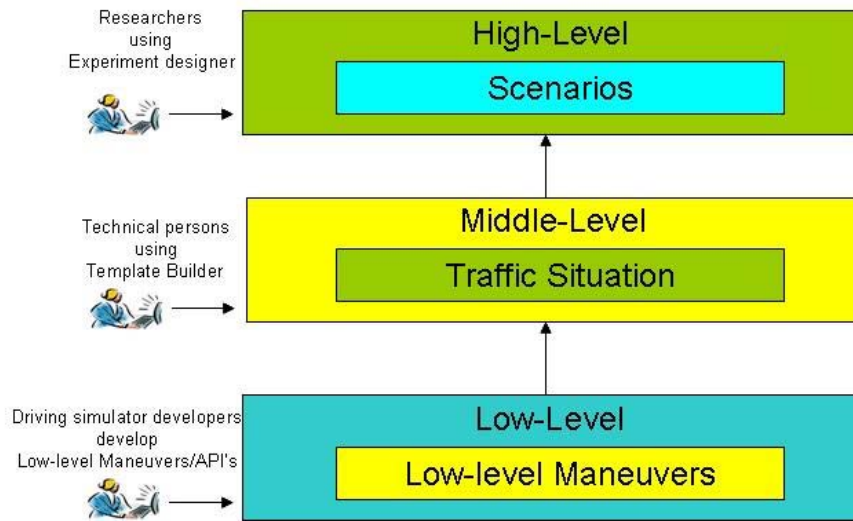


Figure 4.3: User roles corresponding to scenario levels.

4.4.5 Movie theater metaphor of the approach

The end-users' roles in the proposed approach can be explained using the theater play metaphor. Theater play metaphor has been previously used to describe traffic scenarios [Alloyer et al., 1997, Wassink et al., 2005, 2006, Olstam and Espié, 2007]. According to them, Theater play can be regarded as a driving simulation scenario. Where actors (vehicles, pedestrians) come on the stage (road environment) and adopt a role (vehicle type, color, model etc.) to perform acting according to the script (maneuvers or manipulation of the traffic scenarios). As in theater plays, the script for the play is written by the writers, which in most of the cases are technical persons, as script needs to be written in a specific scripting language used by the driving simulators. So many authors have used theater metaphor to tackle and explain the problems in the above-mentioned references.

Traditionally, researchers who have no programming skills usually have no active role in writing the script and assigning the script to the actors involved in any traffic situation. Technical persons act as the writer for this script and assigning roles to the actors in the play. The technical persons write this script exclusively for the specific play and researchers, which is not a very efficient process, as researchers totally depend on the technical persons who write that script, and this script cannot be used in an efficient way.

In order to make this process efficient for end-users, we specify the roles of researchers and technical persons. Technical persons will write the script for all the situations acting as a scriptwriter, as they have the expertise to write the script but they will just write the script for the situations without specifying the roles of a specific actor who is going to perform that

script. Role of the researcher is to pick up the specific actor and cast him for the specific role and assign a script from the already available scripts. It will reduce the total dependency of the end-users on the technical persons, and will make this process efficient and effective for the end-users.

4.5 Prototype Building

As discussed in previous chapter, UCD is used. One of the key practices in UCD design is the development of the prototype of the design and to get the feedback from the end-users at the early stage of the design.

A prototype based on the proposed approach is developed. This chapter focuses on the Experiment builder sub-interface will be focused, which will be used by researchers. The prototype of the proposed solution is based on the problems identified, user suggestions and the steps that were identified during the user survey. The prototype is built using "JustinMind Prototyper" [JustinMind, 2013], a tool to develop prototypes of any kind of application. We briefly explain all the steps which end-users will have to undertake in order to develop an experimental protocol. These steps correspond to the skills of the end-users. The user is guided using the Breadcrumb navigation [Nielsen, 2007] during the experiment development process. The details of all steps are explained as follows.

4.5.1 Step 1: Experiment Description

In this step, end-users provide the name and description of the experiment as shown Figure 4.4.

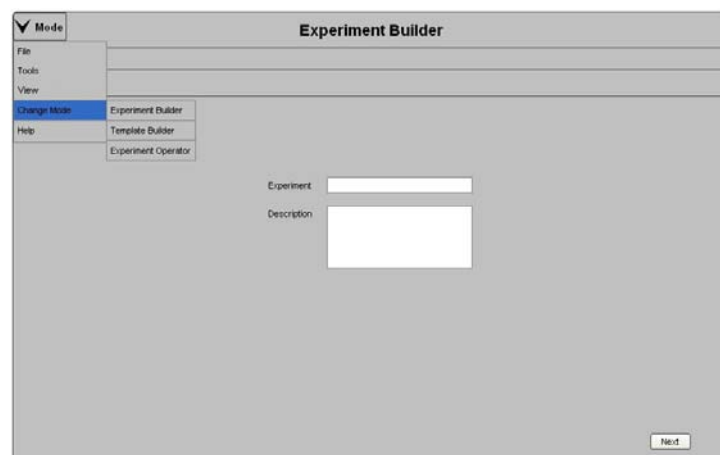


Figure 4.4: Experiment description (Step 1).

4.5.2 Step 2: Terrain selection

In this step, end-users select the terrain/map for the experiment from a library as shown Figure 4.5.



Figure 4.5: Terrain selection (Step 2).

4.5.3 Step 3: Configure subject vehicle

In this step, end-users configure the subject's vehicle. They specify the vehicle type, model, its initial position, heading maximum speed, etc. on the road. They can also specify the Itinerary for the vehicle, if it has to travel on a specific path, and they can also specify the ADAS and other parameters in the 'Advance' tab as shown Figure 4.6. There are default values for all the parameters, so that the user does not need to fix every parameter.

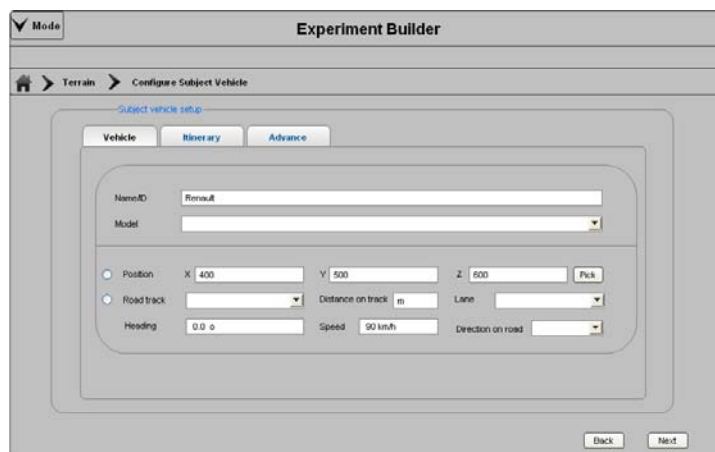


Figure 4.6: Subject Vehicle Configuration (Step 3).

4.5.4 Step 4: Configure Autonomous Traffic

In order to specify the autonomous traffic, the users can define zones for the traffic according to the design of the experiment. A Traffic zone has a start and an end position in the terrain. End-users can specify either the traffic density or the traffic flow in the traffic zone. They can specify the density or they can provide the custom density value for the traffic. They can also specify the traffic distribution and traffic's behaviour distribution in the zone as shown Figure 4.7.

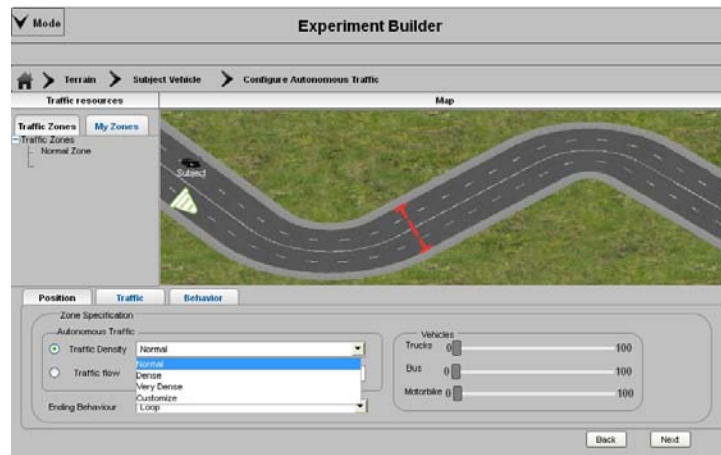


Figure 4.7: Autonomous traffic configuration (Step 4).

4.5.5 Step 5: Configure environment

In this step, end-users specify the environment (light, rain, fog, snow etc). They can also specify zones for the environment as well as for traffic. A zone has a start and an end position. One can configure various zone parameters e.g. rain, fog, snow etc, as shown Figure 4.8.

4.5.6 Step 6: Select dependent variables

In this step, end-users can select the dependent variables explicitly, that is, the variables they want to save in the output file for further analysis, as shown in the Figure 4.9.

4.5.7 Step 7a: Construct critical events and scenarios

This is a critical step for the end-users. In this step, end-users have to manipulate the traffic situations and vehicle maneuvers, which they want to study during the experimental trials. End-users drag the template of the traffic situation or the vehicle maneuvers in the scripting area and configure the parameters of the template. End-users will drag the template on the map, followed by configuring the condition (trigger) for this template to execute, as shown

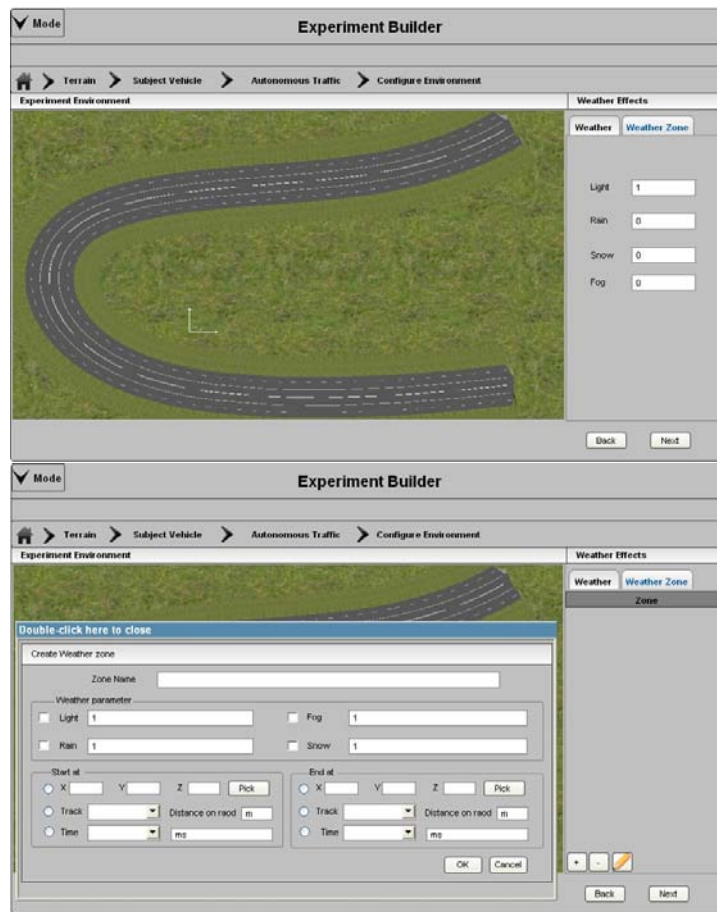


Figure 4.8: Top: Configure environment. Bottom: Weather zone configuration (Step 5).

in the Figure 4.10. The templates that end-users will drag have been built previously by the Technical persons using the Template Builder.

4.5.8 Step 7b: Specify Template parameters

After the selection of the trigger type and value for the template, end-users specify the parameters for the template as shown Figure 4.11.

4.5.9 Step 7c: Overview of the experiment

After the user has developed the experiment, he/she can visualize the whole experiment in a spatio-temporal representation. The templates which will be triggered based on the position, can be viewed on the map and also on the distance line. The templates which will be triggered based on the time, can be viewed on the time line and their description appears in the tool tip, as shown Figure 4.12.

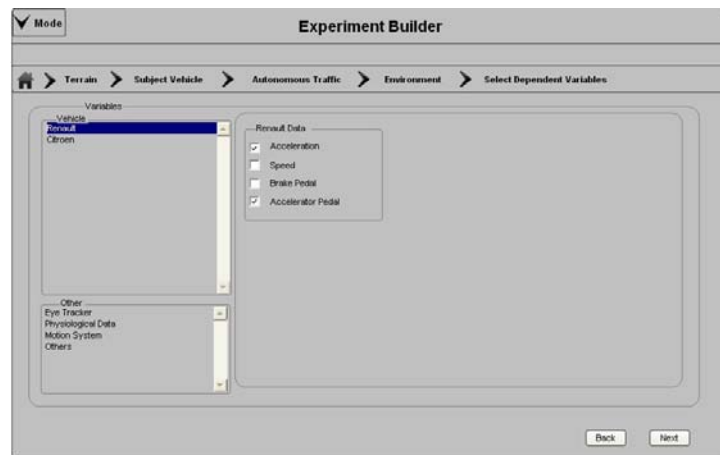


Figure 4.9: Variable Selection (Step 6).

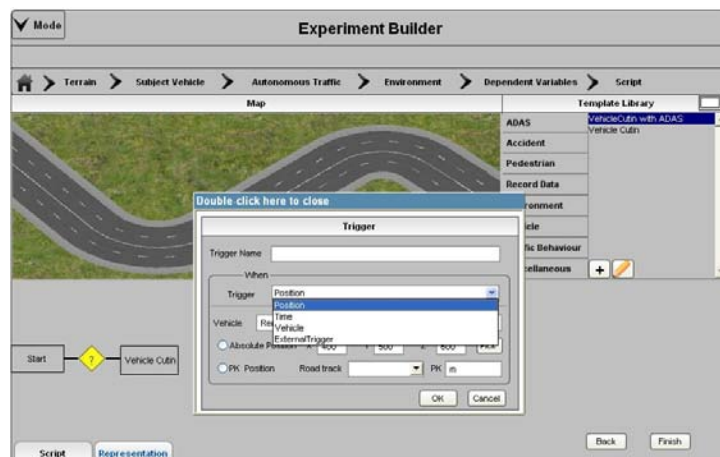


Figure 4.10: Trigger for the template (Step 7a).

The support is provided to the end-users during each step using different Interaction techniques in the experiment development process. End-users are kept away from the technical part of the development process, in which they previously had to write the low-level scripts. In fact, end-users are empowered by providing them the interface, with which they can develop scenarios with the skills they have and minimal help from the technical persons.

Traditionally users have to configure and set up each autonomous vehicle, whether it is interacting with the subject is vehicle or not. In this way they have to configure many vehicles, which is quite complex sometimes, when the traffic is dense. And most of the time the user's requirement is to specify the traffic at a high level, for example the terminologies they use is "normal traffic", "dense traffic", or "specific number of vehicles" in the specific area (Zone). So end-users are enabled to specify the traffic at high level. If the users want a vehicle with specific characteristics or a vehicle which will be involved in the scenario, they can configure



Figure 4.11: End-user specifying template parameters (Step 7b).

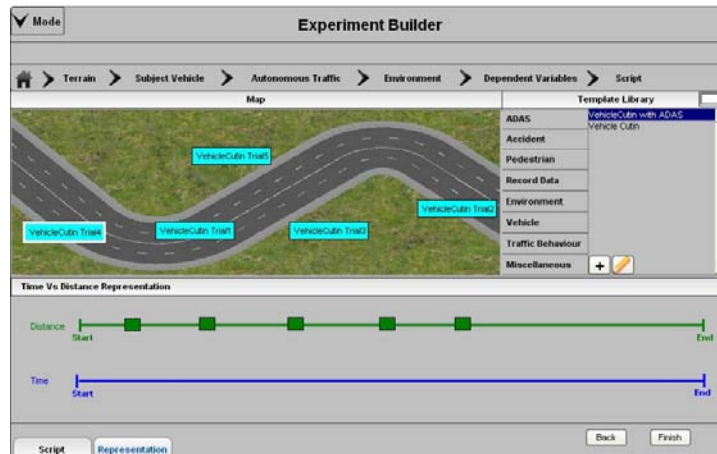


Figure 4.12: Visualization of the experiment on the Time line and distance line (Step 7c).

it in Step 7, during the construction of critical events.

In order to reduce the complexity, the environment configuration is separated from the scenario specification. The users can change the environment (fog, rain, light, snow) by interacting with the map (Figure 14.7 and 14.8). They can set the environment based on time or the vehicle position. If needed, users can also configure the environment based on some other condition using the template for the environment in step 7.

The end-users develop critical events and situations using the drag and drop. They can interact with the map, while specifying the triggers for the templates, which is very important for them. They said in the user survey that without interacting with the map, the process is not clear to them. Time/distance representation of the experiment will enable them to have an overview of the experiment, so that they can anticipate what is going to happen.

4.6 Significance of the approach

Programming requires formal training, which end-users usually lack. Thus, the programming environments should account for their skills. Also researchers develop scenarios in order to achieve their primary goals (conduct an experiment to answer their research questions). So the scenario authoring approaches should allow them to focus on the domain problems of their goals instead of limiting them by low-level programming.

Researchers do not have time or motivation to improve their technical skills so they depend on technical persons, which is a time consuming task for them. Traffic situations are numerous but not unlimited in number, so sometimes technical persons have to develop approximately the same scenarios; if it were already available to the technical person, he/she would be able to develop quickly. In the survey, end-users also specified that, it is easier for them to edit existing scenarios in order to achieve their goals rather than developing them from scratch.

4.6.1 Overcoming barriers

End-users usually have problems with the programming systems, because they cannot overcome the barriers to learn the programming system. [Ko et al., 2004] have identified six different barriers that end-users face while learning programming systems. These 6 barriers are Design Barriers, Selection Barriers, Use Barriers, Coordination Barriers, Understanding Barriers and Information Barriers.

Design Barriers correspond to how to transform a problem into solution, people can think of a solution, but it is difficult to transform it using computer artifacts. The challenge to overcome this problem is to make those solutions usable and customizable corresponding to the user's skills. The proposed solution overcomes this barrier by presenting the customizable templates of the situation at high-level, so users will just have to customize the traffic situations, rather than spending time on how to create the situations from scratch.

Selection Barriers correspond to selecting the relevant solution for the problem. What functions or API to use in order to solve a problem? Does a specific function can solve the problem? This is also an issue, the users specified during the user survey. The solution can be example solutions or code for a specific problem. Existing scenario authoring systems have low-level APIs, which end-users have to use in order to design traffic situations. In the proposed approach, end-users have to select the situation, which is specified at high-level in the form of template, and they just have to customize it.

Use Barriers correspond to the usage of specific actions in order to solve the problems. For example what can be the possible effect of these actions, or how can I perform a specific action using a particular function. In the proposed solution, end-users don't have to use low-level functions; they use templates and customize them according to their requirement. As

a situation template is composed of different low-level traffic maneuvers, so the end-users don't have to come across any use barriers, they can just drag the template and customize it.

Coordination Barriers correspond to how two programming interfaces can be combined together in order to solve a problem. In driving simulator scenarios, end-users have to use different API's in order to develop a specific traffic situation, Sometimes syntax and logical mistakes are made by the end-users. As scenario authoring languages use rule based method (if-else conditions), sometimes it is difficult to combine two low-level functions to create a specific situation. The proposed solution enables the users to customize the already developed situations. End users will use the same rule-based language, but the conditions of a specific situation are followed by the complete template for that situation rather than a low-level function.

Understanding Barriers refer to what causes the specific behavior in response to some specific action. There can be logical errors performed by the end-users, and when end-users have to use more than one functions for a situation, then it is difficult for them to understand why it is not behaving as it should. In the proposed solution, if end-users are unable to achieve the desired output behavior of a traffic situation, they just have to go through the editing of customization process rather than going through low-level function in order to fix the problem.

Information Barriers refer to the information about the internal working of the program. In scenario authoring tools, end-users usually need to follow the internal working of the specific functions to develop a specific situation. Usually there is no debugging tool to get the information about the internal function or variables used during the development of a traffic situation. As the templates will be developed and tested completely by the technical persons, end-users will not have to go through the internal working of the functions used to develop a traffic situation, and the documentation for the working of template will be available for the end-users, using which they can customize the templates according to their requirements.

4.6.2 Support higher-level goals

According to Pane et al. [2001], programming is the process of transformation of high-level mental plans into plans compatible with computers. The end-user programming systems should help the end-users for the transformation of these plans, but user's skills and experience resist this transformation. So end-user programming systems should minimize these difficulties by providing high level computational models which are closer to the real world, which is called "closeness of mapping" by Green [1989]. Even in the programming process, the end-user starts thinking about the problem at higher-level and start breaking down the problem into different meaningful subtasks and then goes to low-level computational models. In the proposed approach, templates are developed at a level closer to the real world, so that they can customize the templates and configure the situations according to their requirements, as end-users usually don't want to spend too much time in developing situations

using low-level programming languages.

4.6.3 Reusability

Technical persons who develop scenarios often use previously developed ones to develop the new scenarios in order to save time and efficiency as previously tested scenarios have already been tested. Also end-users who sometimes develop scenarios by themselves said during the user survey that if they have some scenarios, which can be used after major or minor modification, then the process is easier as compared to developing from scratch. Unfortunately, the process of using previous scenarios is not very efficient. In the proposed approach, a template library for traffic situation is used to develop scenarios. Using it, end-users will have access to all the situation templates which has been developed by technical persons, and they can just use them to develop their scenarios. Also, if they have good skills for programming then they can develop their own templates of the situations. It will save time and effort for both researchers and technical persons. Technical persons will not be overloaded with the work of developing scenarios for researchers. He just needs to develop the templates, and researchers can reuse them for their scenarios, or if a specific template does not exist in the library, then the Technical person can develop one for the end-users.

4.6.4 Is it another new System?

Execution platforms differ from one system to another. In order to model scenarios, they use different modeling languages with different syntax, and different methods followed by the scenario authoring tools. The proposed approach is not a new system which will require its own execution platform. In fact an extra higher layer has been added for researchers, which enables them to use their existing skills to model scenarios as explained above. So any system adopting this approach will not need a complete overhaul. This extra layer can be added to the existing system and researchers can interact with the system using this layer. So it is a generic system for most of the driving simulators. The detail about the interoperability framework is explained in Chapter 6.

4.7 Comparison with the existing system

This section explains the comparison of our approach towards modeling scenarios with existing approaches used in driving simulators. First it addresses the different steps followed by the researchers to model scenarios; then it compares that how researchers will model the scenarios using the proposed approach, or with two systems (SCANeR Studio and ARCHISIM) which are market leaders in the domain of driving simulation and have been used for the last two decades.

As explained above the different steps that researchers follow to design an experimental pro-

Chapter 4. Proposed Multi-Layer programming approach

to include Terrain development, Configuration of a participant vehicle, Configuration of autonomous traffic, Configuration of the environment, Critical situations, and choosing the dependent variables to be studied.

In almost all the systems under study, there is no usable interface which guides researchers during the experimental protocol development process. There is no standard way to develop the protocol. They are usually designed using the format and the method provided by different driving simulators, but not in the way researchers think about designing them.

4.7.1 Interface

In the proposed prototype researchers are guided (using breadcrumb navigation) to develop their experimental protocol. Now we explain each step with a comparison of two different platforms (SCANeR and ARCHISIM).

Step 1 (Terrain selection)

Terrain development addresses the simulated road environment, where the participant drives the car. The comparison of this step is explained in the Table 4.1.

SCANeR	In SCANeR studio, one can develop a terrain using the 'Terrain' module and select it to develop scenarios.
ARCHISIM	In ARCHISIM, one can develop a terrain using the tool 'WR2' and select it to develop scenarios.
Proposed Approach	Development of the terrain is not considered, as it is not the focus of the work. Usually every system provides a tool to develop a terrain. Also we are not developing a new system, but trying to improve existing tools. The proposed approach is compatible with the existing ones. So users are just provided with the interface to select the already developed terrain.

Table 4.1: Terrain Selection comparison.

Step 2 (Configure Participant Vehicle)

The comparison of how different software configure the participant vehicle is explained in Table 4.2.

4.7. Comparison with the existing system

SCANeR	Studio In SCANeR studio, one has to select the vehicle from the list of vehicles using the mouse, and configure lots of parameter of this vehicle, some of which are not very important for most of the users then, one has to set this vehicle as the vehicle driven by the participant as shown in the figure 4.13. There is an interaction with the map, but one can place the vehicles on the map using the co-ordinates. It is difficult to specify them using the distance.
ARCHISIM	In ARCHISIM, one has to configure a file with extension "xxx.vp". The file contains different parameters to configure, such as Initial position, maximum speed, equipped with ADAS (Advance Driving Assistance Systems), etc.
Proposed Approach	In the proposed approach, one can provide the important and relevant information to configure a participant vehicle. There is also a GUI to configure an ADAS. The ADAS and its parameters have been selected after the extensive literature review and studying the different experimental protocol developed for the researchers. One can interact with the map to place the participant's vehicle on the map, as shown in figure 4.13.

Table 4.2: Participant vehicle comparison.

Chapter 4. Proposed Multi-Layer programming approach

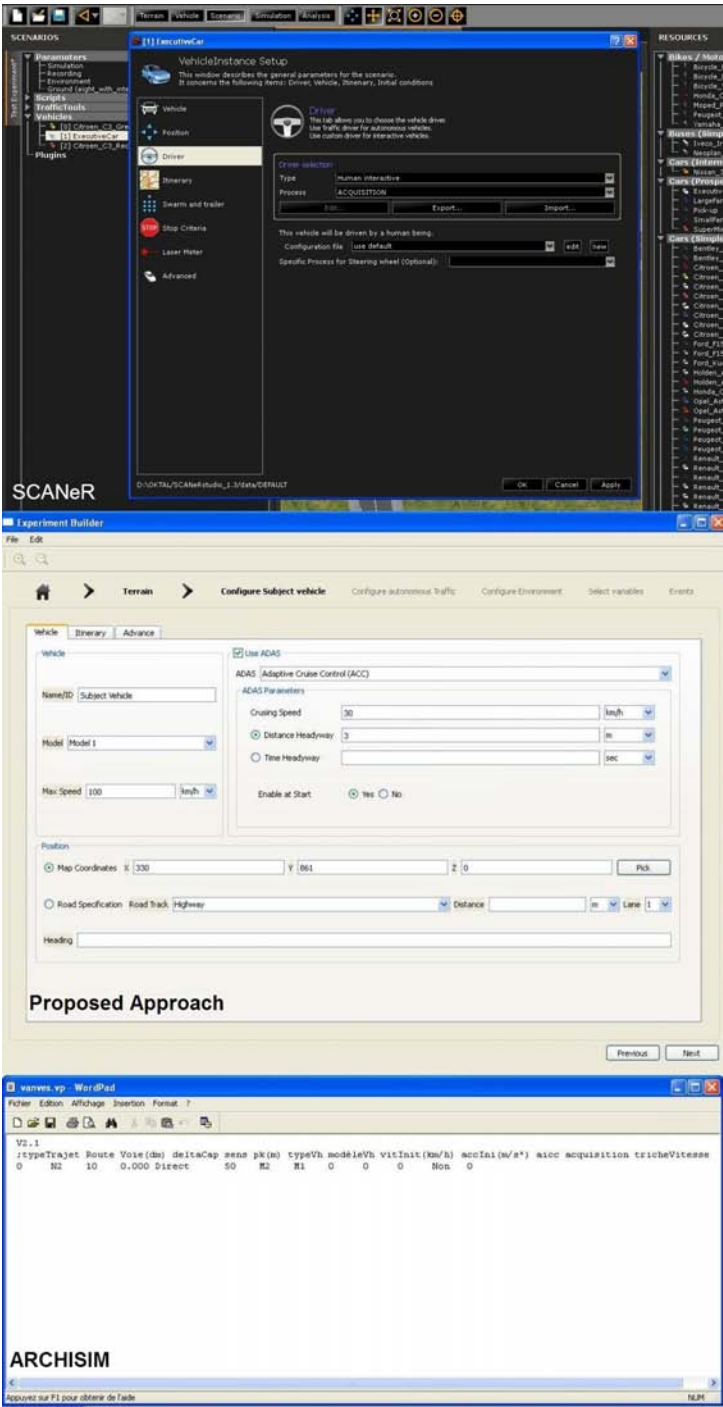


Figure 4.13: Comparison of Participant vehicle configuration.

Step 3 (Configure Autonomous Traffic)

Configuration of the autonomous traffic is a very tricky step during the development of an experimental protocol. One has to make a trade-off between the reproducibility and the realism of the autonomous vehicles. Every driving simulator provides a means to configure the autonomous traffic, by using textual scripting or using GUI, but still it is a complex task, as one has to configure the behavior of each vehicle. The comparison of traffic configurations is explained in the Table 4.3 and displayed in Fig. 4.14.

SCANeR	Studio In SCANeR studio, one can place each vehicle by dragging and dropping the vehicles on the map and configure them (Initial and maximum speed, behavior etc). If one has to generate traffic, then "Source" (from where vehicles start emitting) and "Sink" (where vehicles vanish) can be used to generate the traffic by specifying the traffic flow. But "Sources and "Sinks" can only be created per lane per direction. This does not fully fulfill the goals of the researchers, as they work with the traffic density with concepts such as "normal", "dense", "very dense" traffic. In order to keep or create the desired traffic in a specific area, scenario developers have to do scripting for that and also for complex scenarios as shown in the figure 4.14.
ARCHISIM	In ARCHISIM, one has to configure textual files to create the autonomous traffic. One has to configure each vehicle in more than one file with the relevant information. In ARCHISIM, there are two kinds of vehicles: "archisim" (intelligent vehicle) and "asservi" (unintelligent vehicles), and they are configured according to the requirement of the experiment as shown in the figure 4.14. It is really difficult to manage and control these vehicles, they are greater in number, and if the scenario is complex, it becomes a very time-consuming task for the scenario developers.
Proposed Approach	Neither ARCHISIM nor SCANeR follow the way researchers usually design or think about the autonomous traffic in an experiment. Configuring and managing vehicles at the individual level can be difficult, complex, time consuming and unintuitive as well. We use a user-centered approach for the specification of autonomous traffic is used. Researchers can specify the traffic density they want in a specific road section rather than specifying each vehicle, and this is how an average researcher thinks. End-user can specify the traffic flow or the traffic density, which will be maintained by the system as shown in the figure 4.14. It is more efficient, as no extensive scripting is involved, which will take less system resources.

Table 4.3: Comparison of configuration of autonomous traffic.

Chapter 4. Proposed Multi-Layer programming approach

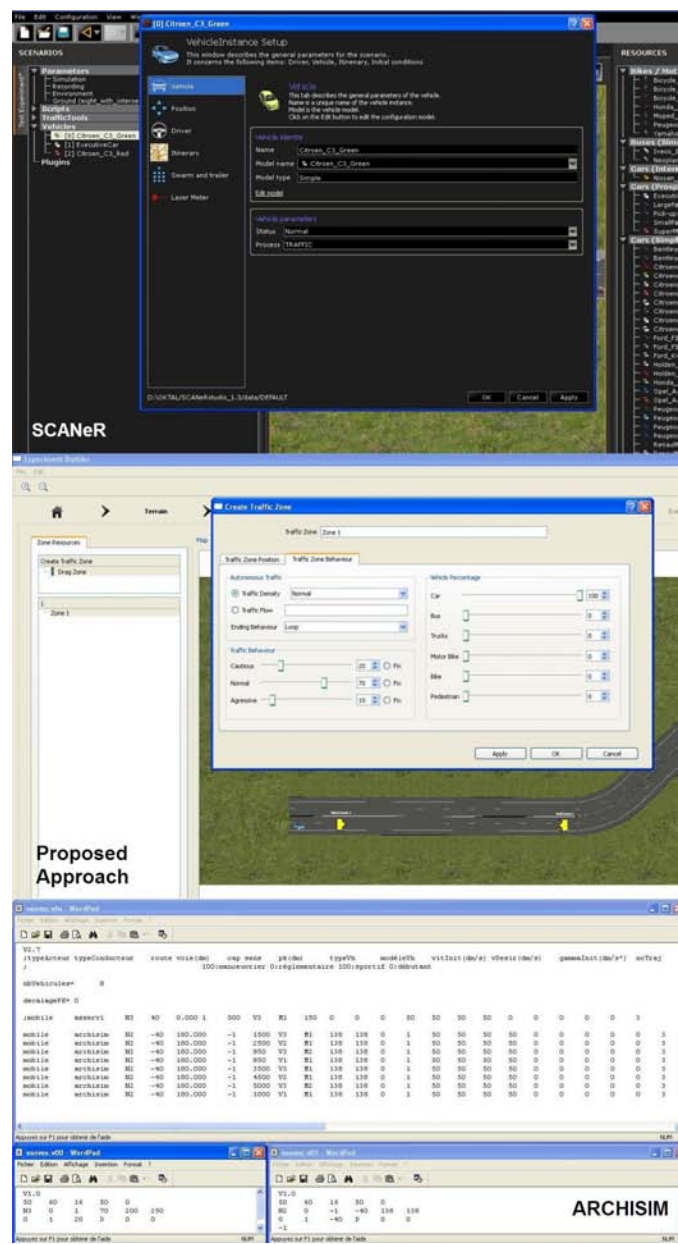


Figure 4.14: Comparison of Autonomous traffic configuration.

Step 4 (Configure the Environment)

Configuring the environment includes, setting the default environment for the experiment, and changing it dynamically during the trials. The comparison of environment configuration is explained in Table 4.4; the interfaces are displayed in Fig. 4.15.

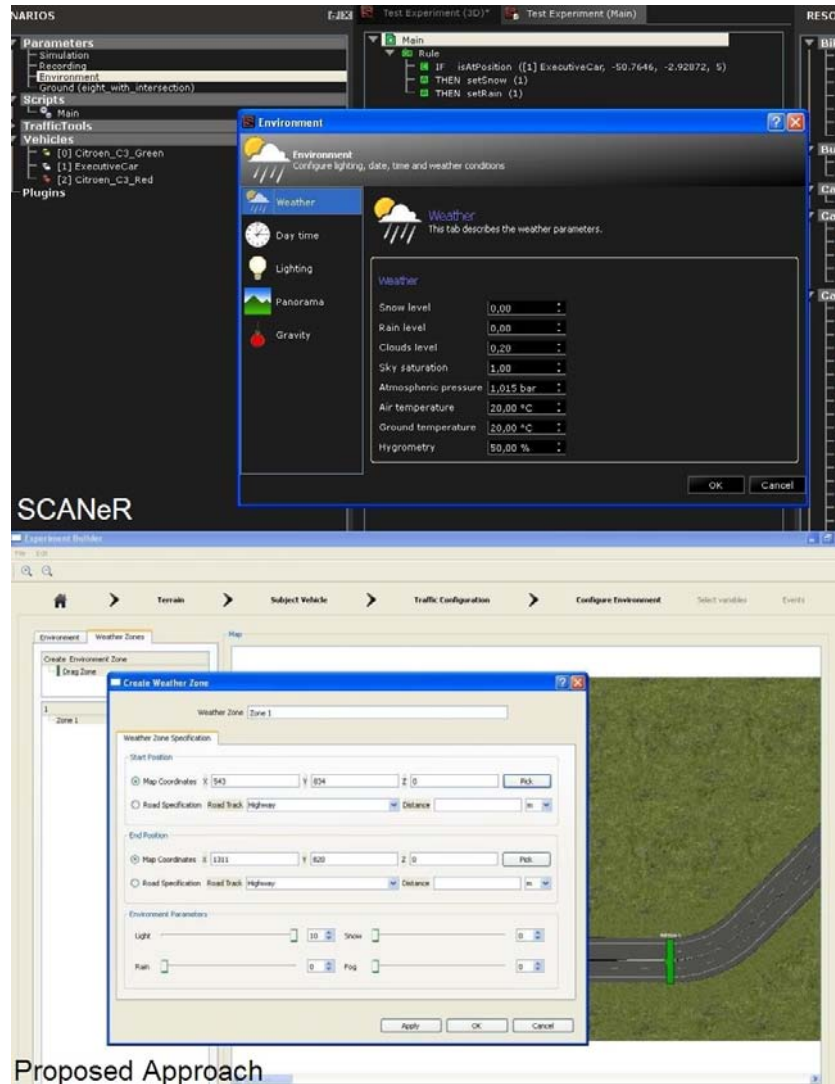


Figure 4.15: Comparison of environment configuration.

Chapter 4. Proposed Multi-Layer programming approach

SCANeR	In SCANeR studio, one can set the default environment of the experiment, and it can be changed using the software during the simulation trials, or it can be changed using the scripting language as shown in the Figure 4.15.
ARCHISIM	In ARCHISIM, one has to configure the environment in the scenario file using the rules based language as shown in the Figure 4.15.
Proposed Approach	As scripting is already a complicated task for end-users, so in order to simplify the scenario, dynamic changes of the environment have been shifted from the scripting step, and one can specify the environment using the zones as for the traffic. Users can specify at which specific road section they want to change the environment shown (see Figure 4.15).

Table 4.4: Comparison of configuration of Environment.

Step 5 (Selecting the dependent variables)

The selection of the dependent variables includes the output file, which includes all the variables which are to be studied by the researchers. The comparison of variable selection is explained in Table 4.5; the interfaces are displayed in Fig. 4.16.

SCANeR	SCANeR studio records all the data of the experiment, and after the experiment an analyzing tool is used to extract the required data.
ARCHISIM	In ARCHISIM, One has to configure a textual file where one can specify the variables that researchers want to study after the experiment as shown in Figure 4.16.
Proposed Approach	In order to provide the facility of on-the-fly analysis to the researchers after the experiment, researchers are provided with the facility to specify the relevant data; the end-user can analyze the selected data soon after the experiment. Also there is an interface. In other simulators, one has to select the variables for all the external devices attached to the driving simulators (e.g. Eye trackers). In the developed interface, all the data can be configured as shown in the Figure 4.16.

Table 4.5: Comparison of the process of selecting the dependent variables.

4.7. Comparison with the existing system

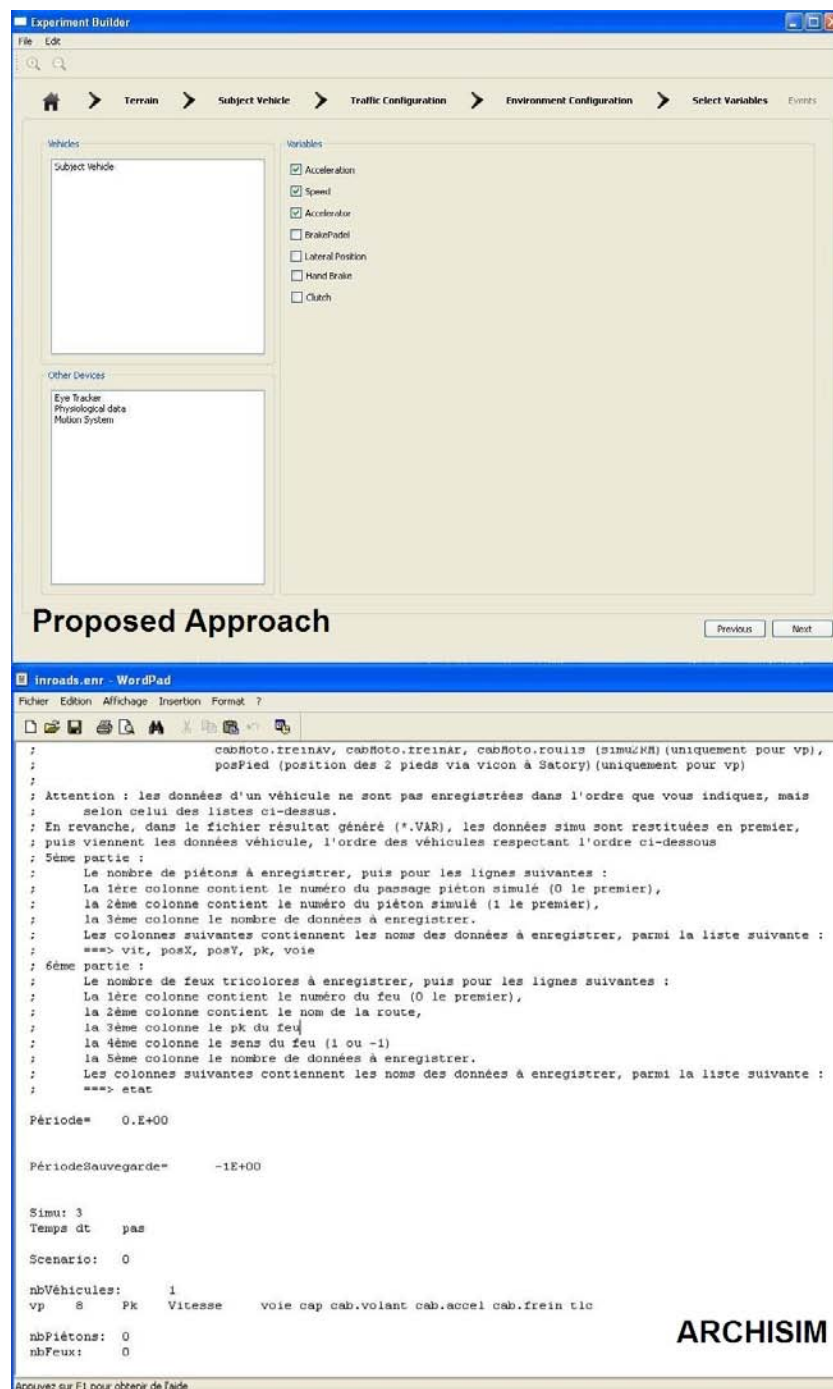


Figure 4.16: Comparison of selecting the Variables.

Step 6 (Construction of critical situations)

The complexity of constructing critical events and how different software use different techniques and metaphors in order to create critical events are already explained in the previous chapters. The comparison of SCANeR and ARCHISIM with our approach is explained in Table 4.6; the interfaces are displayed in Fig. 4.17.

SCANeR	SCANeR studio provides a low-level but GUI based interface to construct critical events. It provides different API's and functions to construct different events using a rule-based (if else) approach and programming techniques like variables as shown in Figure 4.17. The tool for modeling the scenarios on SCANeR works on Middle layer of the Multilayer programming architecture as shown in the Figure 4.1. It is difficult to debug the script, if the script is not working accordingly.
ARCHISIM	In ARCHISIM, One has to write the scenarios using a specific syntax in the textual format which also use programming techniques variables as shown in Figure 4.17. It also works on the Middle layer of multilayer programming architecture as shown in Figure 4.2. It is also difficult and time-consuming to debug the scenarios in ARCHISIM.
Proposed Approach	It is difficult for end-users to develop scenarios at Middle layer of the multilayer programming architecture. So a new layer is introduced, which will help user to exploit the programming primitives, and to focus only on their goals. As explained before, the end-user will not have to use traditional rule-based scripting. They will just do the customization of the templates. Debugging is also easy, as end-users will not have to take care of the syntax; they will just specify the parameters, so that the situation could be simulated according to their requirements. There is also a summary or the bird eye-view of the whole experiment, and end-users are given the opportunity to interact directly with the terrain for every step of the experiment development process, as shown in Figure 4.17.

Table 4.6: Comparison of construction of critical events.

4.7. Comparison with the existing system

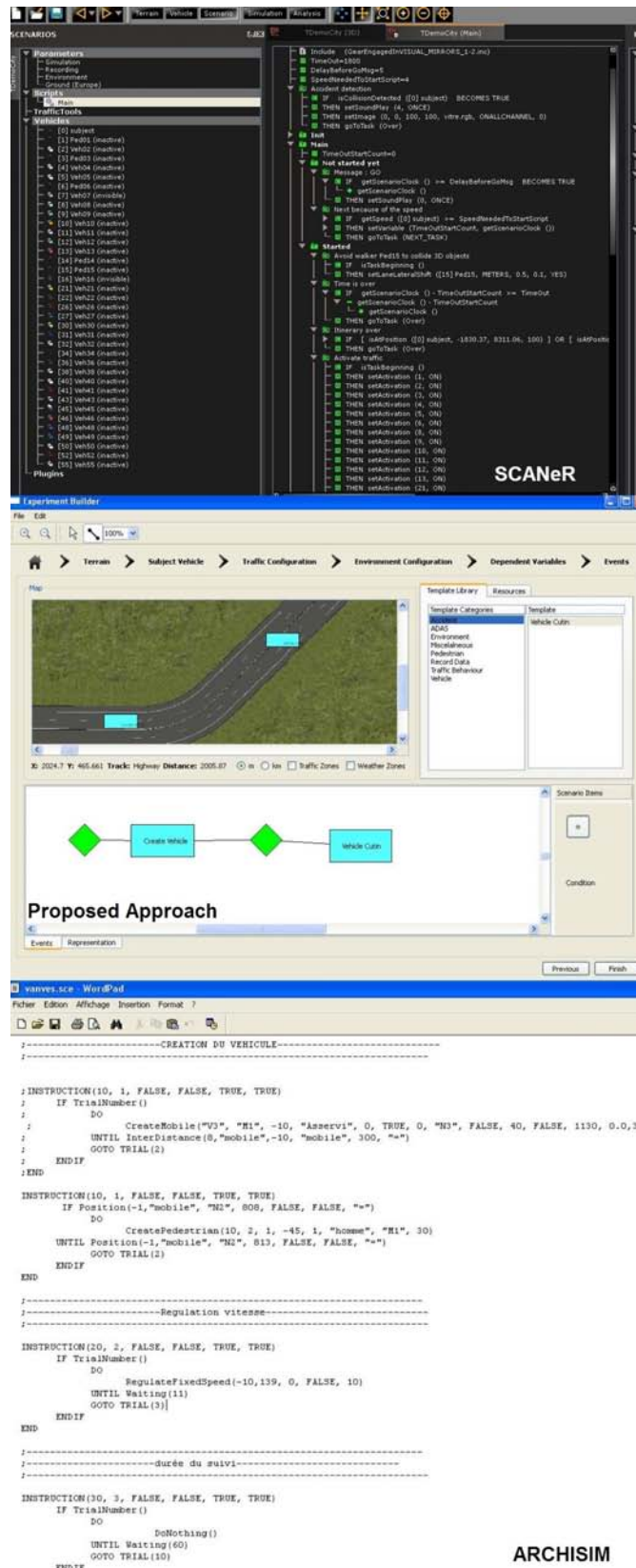


Figure 4.17: Comparison of constructing the critical events.

Chapter 5

User Experience

Evaluation of the system gives an insight on system's behaviour, when users interact with the system. As the UCD approach is being followed, so the users were involved during all the phases.

The main purpose of software prototype is to evaluate the system at the early stage of the design. So a user evaluation was conducted on the developed prototype. After the user evaluation on the prototype, the system was improved based on the user feedback. It was implemented as a fully functional system, and was tested again by the end-users in a controlled environment.

5.1 Preliminary study

The first iteration of the design was conducted to get a general overview of the proposed approach. It was a qualitative study, so that more information can be gathered from the end-users.

5.1.1 Procedure

A total of 9 users participated in the study. These users were behavioral researchers who used driving simulators for their research but had no technical or programming skills to program scenario. The approach was explained to the users and then they performed a small exercise using the prototype. They were observed during the exercise and interviewed later on. In the exercise, the users created a small scenario, in which first, they had to create a lead vehicle the driver should follow. After this, there was another event, in which the vehicle being followed should brake as shown in the Figure 5.1. The end-users had to create a traffic and a weather zone. The interviews were not recorded. The average time for a user was 30-40 minutes with 5-7 minutes for explanation, 15-20 minutes for the exercise and 10-20 minutes for the

interviews.

5.1.2 Interviews

In the interviews, users were asked about their feedback with respect to the new approach, division of the scenario into experiment protocol steps, the difficulties during the exercise (i.e. which step was difficult to comprehend), and they were also asked, whether there anything that needs to be improved. The questions were not limited to what was specified or prepared. It was discussed in detail if user raised any issue.

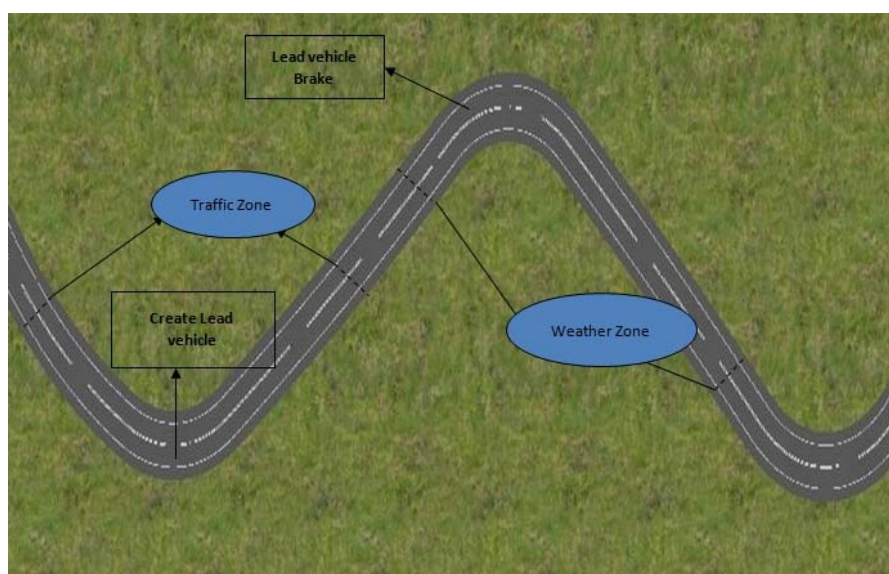


Figure 5.1: Exercise to model scenario.

5.1.3 Results

The structure of the UI and the steps to create the experimental protocol were quite clear to all the users. They all gave a positive feedback about the approach for creating scenarios/events by using templates. There were some minor problems, for example, 4 users told that creating a zone for autonomous traffic was not very intuitive and a bit uncomfortable, which was observed for 2 more users. Three users also suggested that, in order to create events based on position, they would have preferred to be able to drop the template on the map. Previously they had to drop the template in the scripting area and later they were able to change the position of the template by dragging it on the map. Five users attempted to do that as well.

5.1.4 Discussion

Users were uncomfortable to create the traffic zone as when they tried to create another zone they clicked "next" and went to the next step, although they tried to create another zone. During the interviews, they felt the prototype easy to use and it was intuitive for them to configure the templates, rather than doing the low-level coding, and they were also interacting with the map, so they were aware of the activity where and what is going to happen. Users also suggested that the length of the zone can be helpful in creating the experiment. Users appreciated the guidance by navigation, that there is a separate sub-interface for each step minimizing the complexity to create the experiment. After the user feedback, the prototype is being improved and implemented based on the suggestions by the end-users.

5.2 Main study

The first iteration of our design was conducted to get the general overview of the proposed approach. After the first iteration, a fully functional tool was developed using which participants were able to develop a complete experiment. The development of the tool included the improvements as suggested by the first iteration. It was a qualitative study, so that more information can still be gathered from the end-users. The goal of this study was to evaluate the approach and interface in detail by the end-users. Three hypotheses were explored during the experiment.

- H1: The proposed method/tool will empower the user to develop an experimental protocol.
- H2: End-User will focus on domain problems rather than programming issues
- H3: End-User will need lesser or no help using the proposed method/tool.

5.2.1 Participants' selection and participation

Eighteen participants of two types were recruited for the study. First, were the ones who had no programming skills and experience of developing scenarios on driving simulators, and second those who had experience of developing experiment on driving simulators but no programming skills or formal training in programming the scenarios. 10 inexperienced and 8 experienced users participated in the experiment. These participants were selected from IFSTTAR (from different sites of IFSTTAR) and from ADAPTATION project (www.adaptation-itn.eu). 14 users were using ARCHISIM for their research, 2 users were using SCANer software and rest of the 2 users had used STI Simulator. Out of 14 ARCHISIM uses, there were 8 experienced users and 6 inexperienced users. For SCANer and STI Simulator, there was 1 experienced and 1 inexperienced user for each. Nine participants participated in the experiment on site while the remaining participated online. All participants were behavioral

researchers who use driving simulators for their research and had an average of 4 to 5 years experience of working on driving simulator.

5.2.2 Experiment Setup and Procedure

The participants who participated on-site used the tool on the provided system. The system was equipped with Atomi [2013]. ActivePresenter is a tool to record screen activity. The participants who participated online installed the tool and Skype or TeamViewer [2013] on their system. TeamViewer is a software to host online meetings. And the screen of the participants was shared using TeamViewer or Skype.

Each participant was given the same instructions. They were given a tutorial on how to use the tool and they were allowed to ask any question about the tool. After the tutorial, participants developed an experiment using the tool. The experiment was based on true experiment which was already conducted in the ADAPTATION project. After the development of the protocol, participants filled two questionnaires. These questionnaires included the System usability Scale (SUS) [Brooke, 1996], the Single-Ease Question (SEQ) [Sauro, 2013] and a questionnaire related to the tasks performed during the experiment. After the questionnaires, end-users were interviewed and asked about their experiment. They were also told to think aloud [Van Someren et al., 1994], if they feel any difficulty during the experiment development process.

5.2.3 Experiment Task

The purpose of this experiment was to test the new approach and tool in the real environment. Also we wanted to use an experiment which is neither very complex nor not very easy, and which could cover all the features of the tool. So an experiment was selected from the ADAPTATION project as mentioned above. The reason to select this experiment was that, there was direct access to the author in order to design the experimental protocol for the participants who had to develop the scenario with the new tool. Also this experiment included all the necessary information (ADAS selection, Traffic Zones, weather zones, traffic scenarios) that we wanted to test during the experiment. Also we wanted to select both simple and a bit, but not very complex scenarios, because of the time constraint. The results of the study conducted on this experiment is already published [Beggiato and Krems, 2013]. The documentation for templates was provided to the end-users so that if they want to take some help regarding the templates they can study the documentation. The details description of the task was written as follows:

Description of the scenario

The simulator track is comprised of an approx. 23-km long two-lane highway with an average driving time of 17 min and 40 s. With the exception of a 100-km/h speed limit in construction

zones, simulator speed limit is set to 120 km/h. The route consists of five consecutive road sections of approximately 4 km each and approximately 2 km of straight road at the end. Every road section includes:

- Two left bends and one right bend.
- A cut-in situation.
- A construction zone of approximately 1 km, where only the right lane could be used and a lead car with a constant speed of 80 km/h set the driving pace.
- A situation involving queuing at the beginning of the construction zone.

However, the five base modules differ as follows:

- Weather conditions (good weather / light fog / heavy fog)
- Cut-in vehicle (normal car / motorbike / white car)
- Last car in the queue (normal car / white truck)
- Lead car in the construction zone (normal car / white truck)

Scenario Details

General settings

- Subject vehicle is equipped with ADAS (ACC)
- You can use the default values, if information is not provided for any field.
- You will select "Terrain 1" while selection experiment terrain in the tool.

1st Road section

- Good weather (Baseline)
- Cut-in by a yellow car at 1.7 Km (ACC is 'ON' before the situation)
- Normal Traffic in the module
- A car is stopping at the start of construction zone and starts moving when subject car approaches it at distance 15m from that car.

2nd Road section

- Light Fog
- Cut-in by a yellow car at 5.7 Km (ACC is 'ON' before the situation)
- Normal Traffic in the module
- A car is stopping at the start of construction zone and starts moving when subject car approaches it and at distance 15m from that car.

3rd Road section

- Good weather
- Cut-in by a Motorbike at 10.2 Km (ACC is 'OFF' before the situation)
- Normal Traffic in the module
- A white truck is stopping at the start of construction zone and starts moving when subject car approaches it and at distance 15m from that car.

So there are two situations in each zone. 1st is Cut-in situation, and 2nd is car-following situations.

5.2.4 Data Collection

Data was collected from different sources during the experiment.

Screen recording and audio data

As explained above, the off-site users participated using Skype or TeamViewer and their screen were recorded to log the user activity during the experiment. Users were asked to think-aloud if they encounter any problem or stuck somewhere. Think-aloud technique is used to test the user, that what are their thoughts, what he wants to do while performing a certain task [Lewis and Rieman, 1993]. Think-aloud moments and the user-activity were recorded using the audio recording facility provided by the ActivePresenter software.

Task performance data

The output of the experiment was an XML file to be executed on the specific simulation platform. The output file represents the user-performance: how correctly they have implemented the experimental protocol according to the instructions provided to the end-users. The time that each user spent on each sub-task was also recorded. Using this data, it could be identified, how correctly they have performed a specific task.

Questionnaires

A questionnaire is one of the methods to collect information from the users. They have many advantages in the usability research. If the questions are designed carefully and used correctly they can give a useful feedback of the system or the interface being tested. There are some standard usability questionnaires which are used in the industry for the usability research. There are different kinds of standardized questionnaires for example post-study questionnaires and post-task questionnaires. Some post-study questionnaires are:

1. QUIS (Questionnaire for User Interaction Satisfaction) [Tullis and Stetson, 2004].
2. SUMI (Software Usability Measurement Inventory) [Kirakowski and Corbett, 1993].
3. PSSUQ (Post-Study System Usability Questionnaire) [Lewis, 2002].
4. SUS (System Usability Scale) [Brooke, 1996].
5. UMUX and UMUX-LITE (Usability Metric for User Experience) [Lewis et al., 2013].

Some post-task questionnaires are:

1. ASQ (After Scenario Questionnaire) [Lewis, 1991].
2. SEQ (Single Ease Question) [Tedesco and Tullis, 2006].
3. SMEQ (Subjective Mental Effort Questionnaire) [Sauro and Dumas, 2009].
4. UME (Usability Magnitude Estimation) [Sauro and Dumas, 2009].

Using the standardized questionnaires has many advantages for instance Reliability and Validity. We used SUS was used as a post-study questionnaire. It is a reliable questionnaire and has been used in the in more than 600 studies (Sauro). It is a short questionnaire with 10 items and each item has 5 choices as shown in the Figure 5.2. The SUS can be used to test the usability of almost all kinds of software, hardware, Mobile phone and Smartphone applications. It is a reliable and valid questionnaire which can measure the usability of the system. The shortcoming of this questionnaire is that it cannot identify the usability problems encountered by the end-users.

Strongly disagree	Disagree	No idea	Agree	Strongly agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Figure 5.2: 5-point Likert scale.

SEQ was used as a post-task questionnaire. As its name says, it is a one-question questionnaire for each task. It can measure the satisfaction of the user while performing a task after the task has been performed and can diagnose the usability problems faced while performing the task. It does not measure the overall satisfaction of the system or interface, but for a specific task. It is a very small simple, short and reliable questions which is easy for users to respond and easy to administer (Sauro and Dumas 2009). The SUS and SEQ questionnaires can be found in Annex A.

5.2.5 Data Analysis

All the sources and hypotheses were explored using the data gathered from the experiment. The end-users were categorized into two groups of users: Experienced and Inexperienced users. Experienced users are those users who do not have a technical background but are involved in developing an experimental protocol on driving simulators by themselves or by partial help from the technical persons. Inexperienced users are those users, who do not have any technical background and who have never been involved in developing an experimental protocol. First there is an analysis of all the users, and then we go deeper in detail to analyze the difference between the two types of users. The data of two inexperienced was corrupted, so we excluded those two inexperienced user from the analysis, and we processed the data from 8 inexperienced and 8 experienced users.

Time

The time that the end-user spent on each trial was recorded at each trial. In a driving simulator experiment, end-users normally test more than 1 trial of the same situation in order to test the same situation in baseline, and in a specific environment. The time at each trial was recorded so that the learning behavior of the end-users could be identified. Then the time was compared across the participants as well.

Task performance/completion data

The task performance data was gathered from the output XML file that user had generated after developing the experiment. The output file contains all the configuration of the experiment including the parameter values for the traffic situation. The task performance data was computed using a formula for each step of experiment protocol. For description, terrain selection, participant vehicle and variable selection steps, the task was easy, and if users had completed that step, It was regarded as 100% completed tasks.

For environment zones, if end-users created two zones successfully, it was considered as 50% (25% for each task) completed task, and If end-users successfully configured the environment zones, then it was considered as remaining 50% (25% for each task) completed task. For traffic zones, if end-users created 3 zones successfully, it was considered as 50% (16.6% for each

zone) completed task. If end users successfully configured the traffic zones, it was considered as remaining 50% (16.6% for each zone) completed task.

There were 2 critical situations (vehicle cut-in and car-following) in every zone, so in 3 zones there were 6 critical situations. If end-users successfully configured all the Cut-in situations, it was considered as 50% completed task. If end-users had successfully configured all the car-following situations it was considered as the remaining 50% completed task. Computation of task performance for both situations is written as follows:

- *Cut-in vehicle*: There were three cut-in situations in the experiment with varied parameters. In the third situation, end-user had to turn-off the ADAS. So they had to add one more template. Thus, there were 4 templates in total to configure. If end-users had successfully configured the condition of the template it was considered as 25% completed task (6.25% for each template). If end-users had successfully configured the templates of the cut-in situation it was considered as the remaining 25% completed task (6.25% for each template).
- *Car-following*: There were three car-following situations in the experiment with variable parameters. In each situation, end-users had to create a lead vehicle, allow it to move, so that subject vehicle could follow this lead vehicle. So there were 6 templates in total to consider. If end-users had successfully configured the condition of the template it was considered as 25% completed task (4.1% for each template). If end-users had successfully configured the templates of car-following situations, it was considered as 25% completed task (4.1% for each template).

Usability Data

The audio and video (data from the screen) data was analyzed to gather the usability problems faced by the user during the evaluation. The "ActivePresenter" software makes it easy to capture the event when user thinks-aloud for any problem. The user activity was analyzed at that time on the screen. The SUS and SEQ questionnaires were studied and compared against the interviews conducted with the end-users after the evaluation. The SEQ data was compared with the user performance data and if the user could not complete a specific task; it was compared with the user response during the interview regarding this task.

5.2.6 Results

In this section, the results of all the users are presented, and then we compare the user activity of both experienced and inexperienced users. After that, our hypotheses are explored in detail. There was problem with the data of 2 users, so actually the analysis was done on 16 users.

Task Completion

The task performance data can be found in Figure 5.3. All the users completed all the tasks except configuring the environment and configuring the traffic situations. Only 1 user could not complete the task of Environment configuration. To create critical situations, 4 users completed the task, 10 users completed more than 75% of this task. One user completed 50% and one user could not do this task at all.

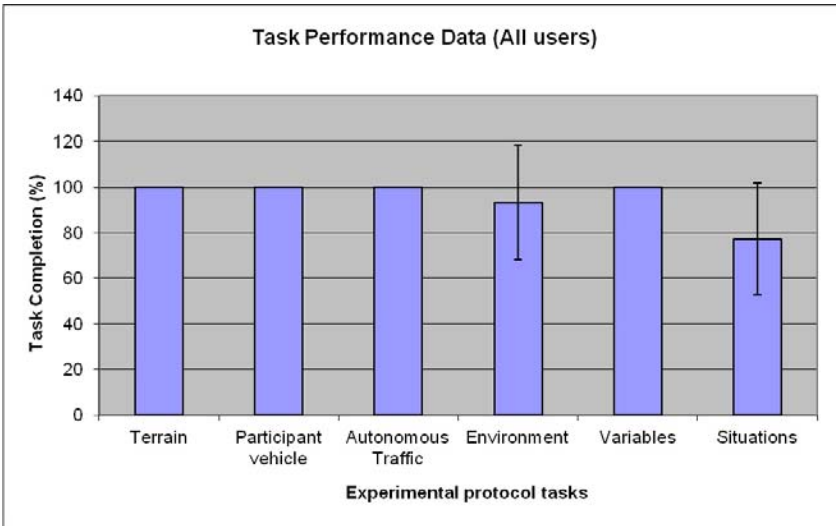


Figure 5.3: Task performance data.

Learning Experience

Configuring the autonomous traffic and creating the traffic situations are difficult tasks for the end-users. As there were different trials for traffic situations and autonomous traffic, the time that end-users spent on each task and their learning experience is displayed in figure 5.4. As we can see, there is a learning behavior, as the time spent in 2nd and 3rd trial is decreased compared to the first trial. There is a slight increase in the value for Situation 1 in the 3rd trial because end-user had to create another subtask which lead to this slight increase.

Usability of the system

In order to get the usability data of the tool, the SUS questionnaire was used. The result of the SUS is shown in the Table 5.1.

Table 5.1 shows that, 78% of the users have rated the tool with a score above 70%, and 22% of the users have rated the tool as less than 60. The SUS has been used in many studies so it was easy to compare the result with other studies. Bangor et al. [2009] has presented a table using which an adjective rating to the SUS score can be given, it can be found in the Figure 5.5.

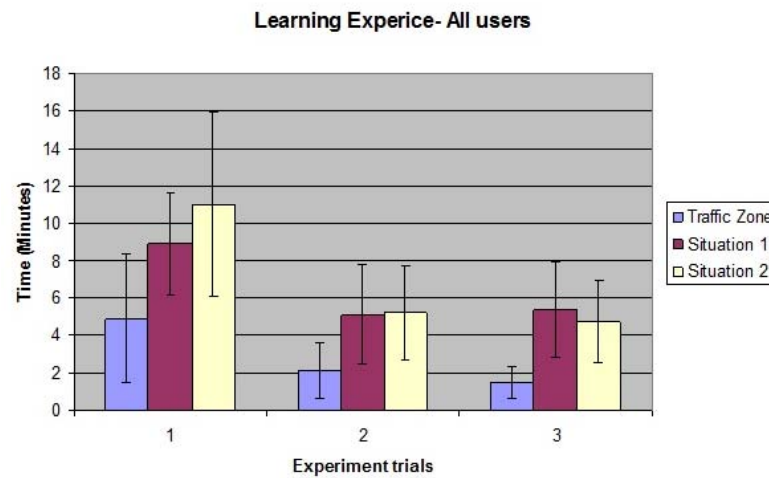


Figure 5.4: Learning Experience of all users.

Score	% of users
80-100	6%
70-79	72%
60-69	0%
50-59	11%
lower than 50	11%

Table 5.1: SUS score of all users.

According to figure 5.5, 6% of the users have rated the tool as excellent, 72% of the users have rated the tool as good, 11% of the users have rated the tool as OK, and 11% of the users have rated the tool as poor

Task Easiness rating

The user also filled the SEQ questionnaire in which they specified the easiness rating for every experiment protocol task. User rating for these tasks can be found in table 5.2.

Table 5.2 shows that 12% of the users find the autonomous traffic task difficult as well as 6% of the users for the environment and critical event part. While some users said that they have no idea whether it was difficult or not, according to the users, it means that it was neither very difficult nor very easy, as it requires some practice to learn.

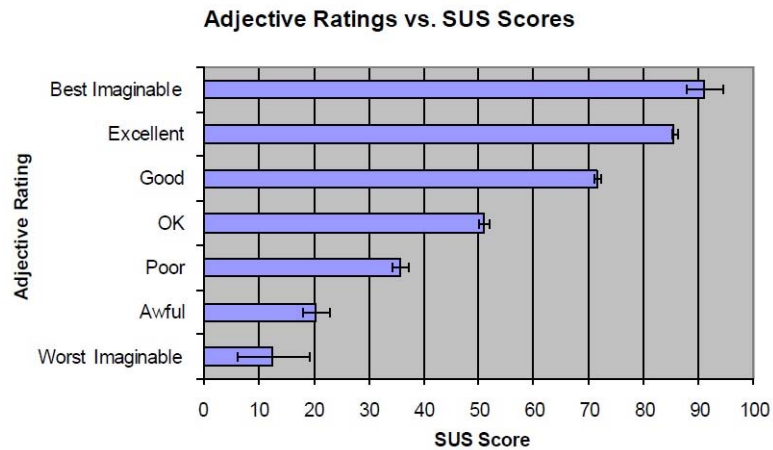


Figure 5.5: Mean SUS score rating [Bangor et al., 2009].

Task	Easy	Difficult	No idea
Terrain	100%	0%	0%
Participant Vehicle	94%	0%	6%
Autonomous Traffic	89%	12%	0%
Environment	88%	6%	6%
Variables	100%	0%	0%
Critical Situations	72%	6%	22%

Table 5.2: Task easiness rating (All users).

Experienced vs. inexperienced users' data analysis

In this section, we compare the performance of experienced and inexperienced users. Experienced users have already tried to develop experimental protocol by themselves using the help from technical persons.

Task Completion (Experienced vs Inexperienced) Figure 5.6 shows the comparison of task completion data of Experienced and Inexperienced user. One can see from the "environment" and "templates" items that, experienced users' performance was better than inexperienced user.

Inexperienced user could not complete the critical situation task at all, and one inexperienced user could not configure the environment zones. Also, the average of task completion of experienced user is better than for inexperienced users. One inexperienced user could not perform the environment configuration task, because he could not understand that he has to create the traffic zones, so every time he wanted to configure the environment in the zone, he changed the default configuration environment, which was wrong.

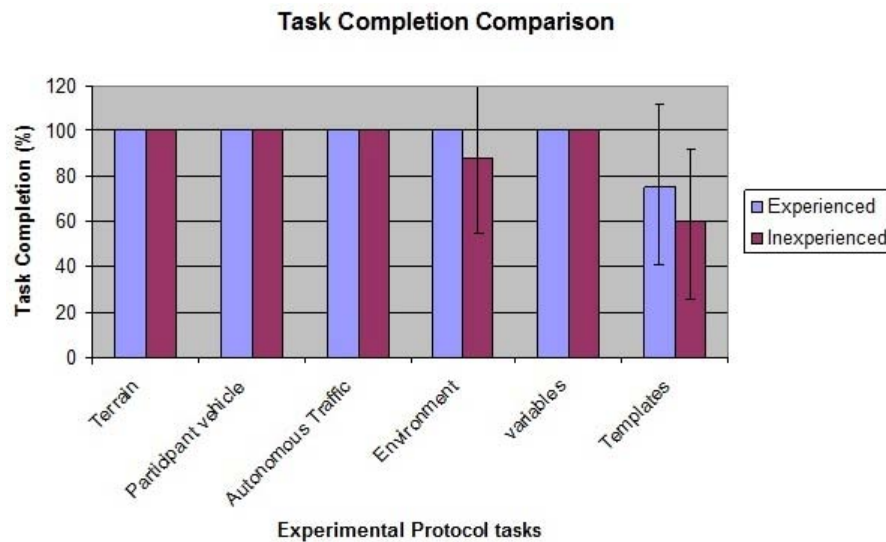


Figure 5.6: Task performance data (Experienced vs inexperienced users).

Learning experience (Experienced vs inexperienced) The figure 5.7, 5.8 and 5.9 show the comparison of learning experience between the experienced and inexperienced users for the Traffic zone configuration, Situation 1 and Situation 2. These figures show that the learning happened for all the users as the time of the 2nd and 3rd trial decreased for both groups. It can also be seen that, both groups have roughly the same performance during the 3rd trial.

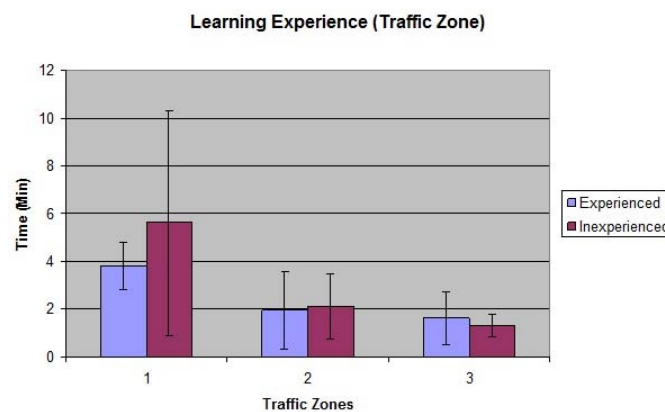


Figure 5.7: Learning experience of Traffic zone (Experienced vs inexperienced).

Usability rating (Experienced vs inexperienced) There was not much difference between the usability score of the experienced and inexperienced user as shown in the figure 5.10.

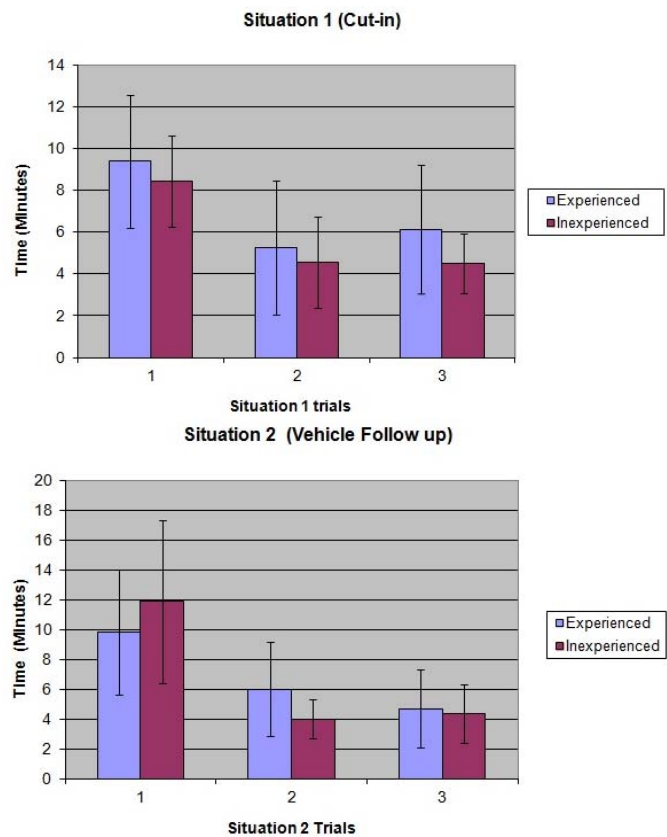


Figure 5.8: Learning experience of Situation 1 (Top) and 2 (bottom).

Task easiness rating Task easiness rating for experienced and inexperienced users is shown in the table 5.3 and table 5.4 respectively. None of the inexperienced user found the critical situation task difficult, but 30% said that they have no idea whether it was difficult or not. 10% of the inexperienced users found the autonomous traffic and environment task difficult.

Interview data Users were asked about their experience while using the tool, if any information was missing which was necessary to develop during the experiment, or any suggestions or remarks: what was the difficulty, if any, while performing each task, and if they had any remarks or suggestions to improve the tool. The users did not find anything missing in the tool except 1 user who said that there was no information to specify the vehicle as right-hand or left-hand side. That's why he rated the task as "No idea".

As shown in Figure 5.6, some tasks were not completed by the users, When they were asked about that, some said that, they did not know that they had to use two templates to complete the situations, so they tried to complete the task using only one template.

Users were asked to explain their experience on the tool under evaluation. Almost all users

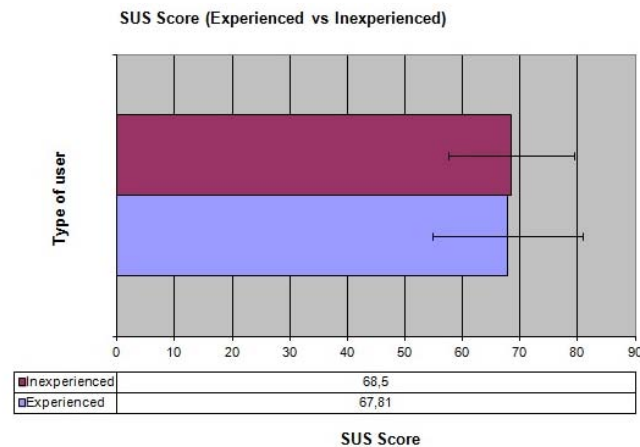


Figure 5.9: SUS score (Experienced vs inexperienced users).

Task	Easy	Difficult	No idea
Terrain	100%	0%	0%
Participant Vehicle	88%	0%	12%
Autonomous Traffic	88%	12%	0%
Environment	100%	0%	0%
Variables	100%	0%	0%
Critical Situations	76%	12%	12%

Table 5.3: Task easiness rating (Experienced User).

have raised the same points and given the same remarks. Experienced users raised the points that classification of the category for the templates is very important, as it is easy to look for a specific template according the situation category. For example all templates related to vehicle movement or overtaking should be in one section. They mentioned that at the start it is a bit difficult but after some practice, it is easy to work on that. So if a good tutorial or training is provided at the start, we can learn this tool and approach easily. They also specified the need of a good documentation of all the templates, as it will make them understand what the functionality of each specific template is. These users said that GUI stepwise development was intuitive, easy to understand and that no low-level scripting was required. But they also said that if there are some improvements like copy-paste the templates in the scripting area and when we change the environment, the environment change should reflect it on the map. They also wanted to preview the situation after it is configured. Some users said that since they have worked on other tools, it was difficult for them to adopt this new tool though it was helpful some practice. Also if possible, one should be able to trigger more than one template from one condition. Inexperienced users' views were not different from the experienced users, they also said that, it was difficult at start, but after some practice, they learnt to develop the experimental protocol. And it was really easy, intuitive and simple to develop,

Task	Easy	Difficult	No idea
Terrain	100%	0%	0%
Participant Vehicle	100%	0%	0%
Autonomous Traffic	90%	10%	0%
Environment	80%	10%	10%
Variables	100%	0%	0%
Critical Situations	70%	0%	30%

Table 5.4: Task easiness rating (Inexperienced User).

as they do not have to do any scripting or low-level programming, the guidance during the experiment development process is really helpful, so only a good documentation of the templates can help them to develop scenarios by themselves. They also mentioned that it would be good, if they were able to preview the situations after configuration.

Users were instructed to think-aloud, but most of the time they did not. They argued that they forget to speak. They were asked what kind of help they think they would need in order to develop critical situations. They said that they might need help at the start, but after this they can do that by themselves, if there is a good documentation of the templates, which should explain all the parameters. Also, if one situation is not available in the library then they will have to take help from the technical persons. These were the answers of both experienced and inexperienced users.

When users were asked if they still need help from the technical persons while developing the experimental protocol, 89% of the users said that they do not need any help anymore using this tool.

5.3 Discussion and hypotheses exploration

The results show that the end-users have significantly improved their skills to develop an experimental protocol on driving simulators.

End-users usually take total or partial help from technical persons, but using the new approach, end-users have developed the experimental protocol without any help from technical persons, as no help was provided to the end-users during the experiment. During interviews, they said during the interviews that they only need help at the start of learning the tool and a good documentation. Every driving simulator has documentation, so if documentation can be organized to help user to configure the templates, they can really do well. And the role of technical person is minimized during the experimental development process, which is a good point for them as well.

5.3.1 User selection

Most of the users were selected from ARCHISIM. For this experiment, such users were needed who have some or no experience and no technical background. Because most of the users who use driving simulators have no technical background, it probably could be a good idea to compare the performance of users from different simulators and compare their result. For inexperienced users, the type of the simulator is not relevant because they never program scenarios on driving simulators. For experienced user, there was 1 user from SCANeR and 1 user from STI simulator. But there was no difference between the performances of these users compared to ARCHISIM. Also, there are many driving simulators in the market so it would be difficult to compare results of users of all driving simulators. Since our proposed approach is independent of any driving simulation platform, it can be tested by the end-users of any simulators. During the user survey (as explained in chapter 3) end-users of most simulators have mentioned the same problems and requirements.

5.3.2 User performance

Task completion meant that end-users have successfully completed each step to execute it on a driving simulator. If end-users completed all the tasks, it was considered as a complete experimental protocol. But some users could not complete the critical situations. The completion rate for inexperienced user was around 60% and completion rate for experienced user was 75%. The difference is obvious as experienced user have developed experimental protocol before. But the improvement of inexperienced user from 0% to 60% is an interesting figure. Because even the inexperienced end-users have experience of working on driving simulators they were never able to develop an experimental protocol by themselves.

User learning experience shows in Fig 5.4 that there was decrease in the time during the trial 2 and 3, except for trial 3 of the situation 1 (vehicle cut-in) because end-users had to configure more than one template during that trial. There was a difference between the time to complete the task between experienced and inexperienced users at the start but in the third trial, the difference between two groups was minimized.

It would have been a good idea to compare the time the end-users took to complete this experimental protocol and the time they spent on other driving simulators, but there is no previous study that could be compared to the time taken by the users using the proposed approach. But using the time information, one can say that users have actually learnt to use the tool. Also, time is not an issue for the users when they have to do it by themselves but if they have to depend on the technical persons then it takes much more time, as technical persons are usually very busy developing scenarios for all researchers in the team.

5.3.3 Problems with the users

Some users could not complete the task critical situation task. One of the reasons is that they did not know that they had to use the two templates in the case of situations (car-following). So some users suggested that there should be only one template for one situation. Also, when a researcher designs an experiment, he has an in-depth knowledge of all the situations that he/she is going to test during the experiment. In the case under study, the experiment was designed by somebody else, and researchers have to design it using the explanation of the author who designed the experiment. Another point is that the users were given instructions and tutorial in English, and for some users, it was difficult to follow the tutorial and instructions in English. One user who could not complete the environment configuration task and some users who could not complete the task of creating the critical situations mentioned this problem. They said, if the instructions would have been given in French, it would have been easier to understand.

5.3.4 Hypotheses Exploration

In this section we go back to the hypotheses under study during the user experience.

H1 : The Proposed method/tool will empower the user to develop an experimental protocol. The main objective of this tool was to empower the end-users who do not have skills to develop experiments by themselves. The inexperienced users had never developed any experimental protocol. But they managed to complete the tasks to develop an experimental protocol and for critical situation, average performance of all the inexperienced users was 60%, which shows that end-users have managed to develop an experimental protocol, and it is also evident from the user learning experience that they started to adopt the tool.

H2 : End-User will focus on domain problems rather than programming issues. Syntax or the abstraction of the programming languages is one of the barriers/hurdles for end-users while using end-user programming systems. While talking to the users during the interviews, end-users did not specify any syntactical problem but they emphasized more on the documentation of the templates, how can they use a specific template in order to perform a task. They were comfortable with the drag and drop environment, and it is also a good point, when considering the usability of the tool, that users do not make any syntactical error.

H3 : End-User will need lesser or no help using the proposed method/tool. Another objective of the work was to minimize the role of technical persons so that dependency on technical persons could be reduced in order to make the experimental development process efficient for researchers and save time for technical persons. Users mentioned that they do not need any help while using this tool; also they completed all tasks by themselves, as no help

was provided.

5.4 Conclusion

An improved capacity for end-users with no programming skills has been demonstrated, which was the purpose of the new interface concepts. Following a User-Centered Design approach, a prototype interface has been designed in order to design scenarios for driving simulators. The skills we addressed were in terms of programming skills. We did not want to improve the end-users skills, because most end-users are behavioral researcher, with no background in computer science and do not want to improve these skills. Instead, an interface is proposed where these programming skills are not needed any more, so that the end-user may achieve his goal of building the scenario with his own skills. The prototype was evaluated on a panel of unskilled and low-skilled end-users, showing good results in terms of completion rate. Also, the subjective evaluation was encouraging, and we could show that the users could easily learn how to use the tool, so that after 3 trials, the performance of low skilled and unskilled users was roughly the same. The proposed approach is not platform dependent, and should now be implemented on several driving simulator platforms. We hope it would be beneficial for both the behavioral researchers and the technical teams working on driving simulators.

Chapter 6

Interoperability of the solution

This chapter explains how the proposed solutions in Chapter 4 can be made on different driving simulation platforms. For this an interoperability framework and a Scenario-meta language have been developed. The scenario meta-language is then integrated with the proposed approach using the interoperability framework.

6.1 Need for interoperability of scenarios

Executing scenarios from one platform to another is an idea that has not been discussed before in this manuscript. But it is sometimes important to make it possible to execute the same scenarios on more than one simulation platform. It is a common objective in different European projects in the domain of transport that for different research objectives, one scenario needs to be executed on different driving simulators (for example the TrainAll and SafeRider projects). In the TrainAll project, same curriculum was given to different partners in the project to train the people in their respective sites and so the same scenarios need to execute on different driving simulators. So partners are usually given the specification of the scenarios to be developed on the driving simulators.

When researchers have to work together in a team, and they have to work on different partner sites, or if they move to other organizations, they have to interact with the different driving simulator with different technical specification, which is a time-consuming tasks, as explained in previous chapters that in most of the cases researchers are not equipped with enough technical knowledge to learn it quickly and develop the scenarios without the help from technical persons. So there is a need of a framework that if they have learnt to develop the scenarios on one platform, they should be able to execute the same format other driving simulators.

6.2 Challenges to develop an interoperability framework

Every driving simulator has a different execution platform, and software architecture, and the language which is used to develop scenarios on a driving simulator is also different. Also driving simulators vary in providing different functionalities. Some functionality which is offered by one platform is not always provided by other platforms.

6.3 Scenario modelling process

Figure 6.1 shows the typical scenario modelling process across different platforms. All the interfaces to develop scenarios on driving simulators are tightly coupled with their respecting execution platforms.

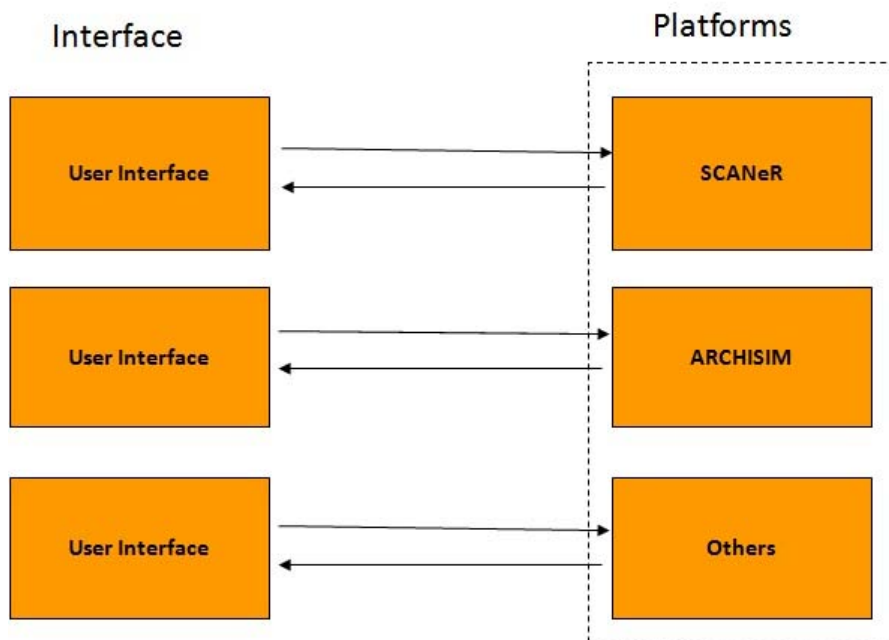


Figure 6.1: Typical scenario modeling process.

As the end-users develop scenarios using the interface of a specific driving simulator, it can only be executed on the specific platform and not on any other platform, as they are developed in the specific low-level programming language offered by the platform.

It is difficult to port the information at low-level, as all the simulators have different way of describing the scenarios as well. So there is a need to describe a general format, which could be understood by different driving simulators. So there is a need of meta-format, which could be executed on different driving simulators.

6.4 Scenario Meta-Language (SML)

A new meta-model is needed which could be translated to the platform specific languages for the driving simulators. There are some functional and technical requirements for this language. All the information about the experiment is introduced in order to define a standard format to port the scenarios from one platform to another.

6.4.1 Functional requirements

- It should describe the initial description of the environment, the actors, and the activities of the different actors involved in the experimental trial.
- It should also describe the list of all the critical events which will change the initial description as well as information about where, when (Event Trigger) and what (Event action e.g. vehicle maneuvers) will happen during the experimental trial.

6.4.2 General requirements

- It should allow the decoupling of experiment development from experiment execution to provide the interoperability and the reuse among different driving simulators.
- It should keep the data separate from the business logic
- It should be application independent
- It should be able to transform on almost all the execution platforms.
- It should be concise and clear.

Note that all these requirements are not specific to driving simulation, and would apply to any virtual reality simulators.

6.4.3 Description of the SML

XML (Extensible Markup Language) is used to define the format for SML. XML is a very commonly used mark-up language that defines sets of rules to encode a document in a format which is understandable by both humans and machines. XML is independent of any execution platform and widely used over the Internet as well as for desktop and simulation application.

The main purpose of SML is to describe a format which is understandable by different driving simulation platforms. The same steps are included in the SML which were used to describe the experimental protocol as explained in Chapter 4. So the language should be able to describe the following information in order to run an experimental trial.

- Experiment Information
- Terrain Information
- Participant vehicle Information
- Ambient Traffic
- Environment
- Critical situation/events
- Variables

Figure 6.2

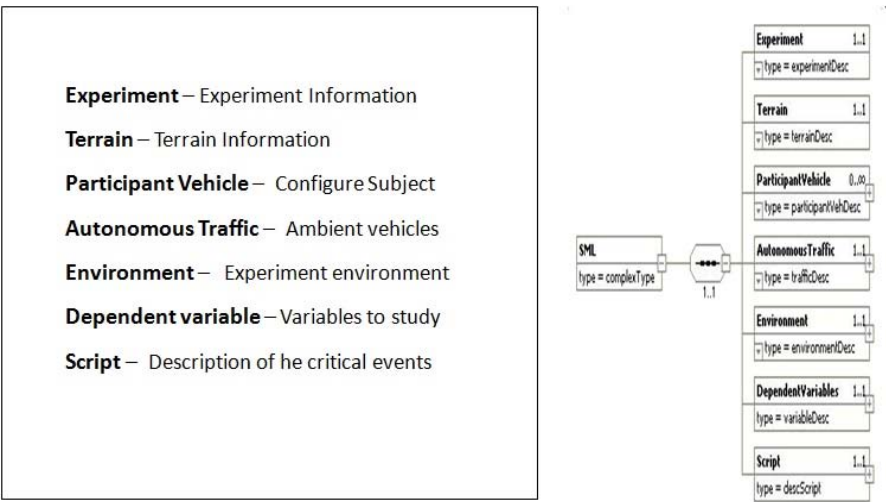


Figure 6.2: SML Schema.

The overview of schema of SML is shown in Fig 6.2. The detail of the schema can be found in Annex B. The details of the functional elements of the SML are shown below.

Experiment information

This step describes general information about the experiment. The description of an experiment in SML format is shown below.

```
1 <Experiment name="ACC highway" Description="Experiment on ADAS on Highway"/>
```

Terrain information

This step describes more specific information about the experiment. The example is shown below

```
1 <Terrain name=" highway"/>
```

At the moment, Terrain specification format is not taken into account. But we can specify the Cartesian coordinates of the terrain and also the general road database, which includes the information about the road name, kilometric position or the distance on a specific road and lane information.

Participant Vehicle Information

This step describes the initial description and the parameters of the participant vehicle, which includes position, speed, itinerary, and the ADAS system, they will use. The position can be either 2D based on the terrain co-ordinates or based on road database and the distance on the road. An example of the participant vehicle configuration is shown below.

```
1 <ParticipantVehicle name-ID="DrivingVehicle" model="Model1">
2     <Position Type="PK Position">
3         <Roadtrack>Highway\_2lane</Roadtrack>
4         <Distance unit="m">200</ Distance >
5         <Lane>1</ Lane>
6         <Heading>0.0</Heading>
7     </Position>
8     <MaxSpeed unit="Km/h">90</MaxSpeed>
9     <ADAS include="Yes">
10         <Type>ACC</ Type>
11         <CrusingSpeed>90</ CrusingSpeed>
12         <Headway type="Time">3</Headway>
13         <EnableAtStart>yes</ EnableAtStart>
14     </ ADAS>
15     <Itinerary>
16         <Intersection id="1">
17             <Direction>0</Direction>
18         </ Intersection>
19     </Itinerary>
20 </ParticipantVehicle>
```

Ambient Traffic

This step describes the autonomous traffic during an experimental trial. We define the autonomous traffic in zones and the whole experiment is divided into virtual traffic zones, and each zone contains the information about the description and behavior of traffic in that zone.

Zone description has a start and end position. These positions can be based on terrain coordinates or road database as described in the step 2. Each zone has a name and a type. The type of zone can be 'traffic density' or 'traffic flow'. When the user will specify the traffic density, it can be 'Normal', 'Dense' 'Very dense' and 'customize'. The value for 'Normal', 'Dense'

and 'Very dense' can be computed on the execution platform. We can also specify the value of density, if we select the 'customize' option.

Each zone has an ending behavior which can be 'Loop', 'Disappear' and 'Random'. The 'loop' selection will let the vehicles in the zone to re-initialize at the same position after reaching at the end of the zone. The 'Disappear' selection will let the vehicles to disappear at the end of the zone and 'random' selection re-initialize vehicles at any random position within the zone.

Traffic behavior is distributed at three different levels which are 'Cautious', Normal" 'Aggressive'. The percentage value for each level can be specified. The 'Cautious' behavior means that vehicles will be driven very carefully e.g. with large time-headways or distance headways and try to avoid risky overtakes. The 'Aggressive' behavior mean that, vehicles will be driven aggressively and undertaking risky over takes and keeping very small headways. The 'Normal' behavior means that vehicles will be driven with normal driving parameters e.g. overtakes and lane change maneuvers will be performed when it is necessary. The parameters for specifying the behavior depend on the parameters of the traffic model followed by the driving simulator. The end-user can also specify the types of vehicles required for each specific zone. In a zone, the percentage value of all kinds of vehicles can be specified.

```
1      <AutonomousTraffic trafficZones="2">
2          <TrafficZone name="Normal traffic" type="Traffic Density">
3              <StartPosition Type="PK Position">
4                  <Roadtrack>Highway\_2lane</Roadtrack>
5                  <Distance unit="m">400</ Distance >
6                  <Lane>1</Lane>
7              </StartPosition>
8              <EndPosition Type="PK Position">
9                  <Roadtrack>Highway\_2lane</Roadtrack>
10                 <Distance unit="m">900</ Distance >
11                 <Lane>1</Lane>
12             </EndPosition>
13             <TrafficDensity>
14                 <Traffic>Normal</Traffic>
15                 <EndingBehaviour>Loop</EndingBehaviour>
16             </TrafficDensity>
17             <ZoneDistribution>
18                 <VehicleDistribution>
19                     <Car>60</ Car >
20                     <Bus>30</ Bus >
21                     <MotorBike>10</ MotorBike >
22                     <VehicleDistribution/>
23                 <BehaviourDistribution>
24                     <Cautious>20</Cautious>
25                     <Normal>70</Normal>
26                     <Aggressive>10</Aggressive>
27                 </Behaviour>
28             </ZoneDistribution >
29         </TrafficZone>
30 </AutonomousTraffic>
```

Environment

This step describes the environment of the experimental trial. The end-user should specify the default environment for the whole experiment, which can be changed by dividing them into virtual zones as for Autonomous traffic in step 4.

Like zones for autonomous traffic, there is a start and end position of the zone, which can be based on terrain coordinates or road database. For each zone we specify the parameters to change in the environment, which are light condition, rain, fog, and snow. An example is shown below:

```

1 <Environment>
2     <EnvironmentSetting hasEnvironmentZone="Yes">
3         <Light>1</Light>
4         <Rain>0</Rain>
5         <Snow>0</Snow>
6         <Fog>0</Fog>
7     </EnvironmentSetting>
8     <EnvironmentZone noofZones="2">
9         <Zone name="Light Fog">
10            <ZoneStart startCondition="Position"
11                Type="Co-ordinates">
12                <X>100</X>
13                <Y>200</Y>
14            </ZoneStart>
15            <ZoneEnd EndCondition="Position"
16                Type="Co-ordinates">
17                <X>500</X>
18                <Y>600</Y>
19            </ZoneStart>
20            <ZoneParameters>
21                <Light>1</Light>
22                <Rain>0</Rain>
23                <Snow>0</Snow>
24                <Fog>3</Fog>
25            </ZoneParameters>
26        </Zone>
27 </Environment>

```

Dependent Variables

This step describes the variables that are going to be recorded during the experimental trial. There are different categories of data to be recorded, which include the vehicle data and also the equipment attached to the driving simulator, for example eye-trackers, equipment to record physiological data. A sample of vehicle data and the eye-tracker is shown below. An example is shown below.

```

1 <Dependentvariables>
2     <Categories>
3         <Vehicle name="/DrivingVehicle">
4             <Speed/>
5             <Acceleration/>
6             <BrakePedal/>
7         </Vehicle>

```

```
8           <EyeTrackerData>
9               <GazeDirection/>
10              <HeadDirection/>
11          </EyeTrackerData>
12      </Categories>
13  </Dependentvariables>
```

Critical Situations/events

This step describes the critical events to be studied during the experimental trial. There is a template of each event or situation to be studied. And each situation/event will be triggered by a condition, which could be based on simulation time, position of the vehicle or any external input etc. So each template will have a name and the parameters to be set for the critical event, variables to be studied during the critical event. The three section of a critical event: Template Condition, Template Action and Template Variables are discussed below.

Template Condition Template condition will trigger the situation/event followed by the trigger. The triggers can be the Position of the vehicle and based on terrain co-ordinates, or the road database as described in previous steps. It can also be Simulation time as we have described for the environment zones in Step 5. It can also be an external input by keyboard or from the cockpit of the driving simulator. Template Condition is followed by the Template Action.

Template Action Template Action contains the parameters of a specific situation. e.g. in order to set the speed of a vehicle, we specify vehicle ID and the speed. The parameters can be varied from template to template.

Template Variables Template variables contain the variables that will be collected during the specific situation. These variables are the same as we have discussed in step 6.

A sample of 2 events is defined below. The first event is 'Vehicle Cut-in', which will be triggered, based on the condition provided. In this event, we create the cut-in vehicle, based on the absolute or relative position to the participant vehicle. Then we specify the relative or absolute speed, with which the cut-in vehicle will overtake the participant's vehicle and at the cut-in distance provided by the end-user, will changes lane. Then, the speed after the cut-in is specified. In the 2nd event, the speed of a vehicle suddenly changes.

Examples of SML formats for creating critical events or situations are shown below.

```
1 <Events noOfEvents="2">
2     <Template name="VehicleCutin">
3         <TemplateCondition type="Time" multipleCondition="No">
4             <Time>
5                 <Condition>Equal </ Condition>
```



```

6             <Value unit="sec">500</Value>
7         </Time>
8     </TemplateCondition>
9     <TemplateAction nameOfParametersSet="Cutin Left">
10         <ChangeADASSettings Change="No"/>
11         <CutinVehicle selectionType="Create" name-ID="
            Yellow car">
12             <Type>Car</Type>
13             <Model>M1</Model>
14             <Color>Yellow</Color>
15             <Position type="relative">
16                 <Lane>1</Lane>
17                 <PositionOffset>10</PositionOffset>
18             </Position>
19             <InitSpee Type="Relative">20</InitSpeed>
20             <MaxSpeed>100</MaxSpeed>
21         </CutinVehicle>
22         <CutInParameters>
23             <CutInDistance>3</CutInDistance>
24             <OvertakingLane>-1</OvertakingLane>
25         </CutInParameters>
26         <TemplateVariables>
27             <Categories>
28                 <Vehicle name="DrivingVehicle">
29                     <Speed/>
30                     <Acceleration/>
31                     <BrakePedal/>
32                 </Vehicle>
33             </Categories>
34         </TemplateVariables>
35     </TemplateAction>
36 </Template>
37
38 <Template name="Change Speed">
39     <TemplateCondition type="Position" umultipleCondition="No">
40         <Position Type="PK Position">
41             <Roadtrack>Highway\_2lane</Roadtrack>
42             <PK unit="m">400</PK>
43         </Position>
44     </TemplateCondition>
45     <TemplateAction nameOfParametersSet="Vehicle yello speed">
46         <Vehicle name-ID=" Yellow car "/>
47         <Speed Unit="Km/h">0</Speed>
48         <TemplateVariables>
49             <Categories>
50                 <Vehicle>
51                     <Speed/>
52                     <BrakePedal/>
53                 </Vehicle>
54             </Categories>
55         </TemplateVariables>
56     </TemplateAction>
57 </Template>
58 </Events>

```

Discussion on SML

The SML is developed at high-level, because the purpose is to make it provide information for all the necessary information to execute a driving scenario on different simulation platforms and it is difficult to port all the information at low-level.

Terrain Every driving simulator uses a specific terrain format in order to build the terrain database for the simulation trial. Attempts are being made to develop standard format to build the terrain database. The most common formats used in the industry and research organizations are RoadXML developed by IFSTTAR, OKTAL, Renault, PSA Peugeot Citroen, Thales, and OpenDrive developed by BMW, Daimler, TNO, VTI and other industrial partners. The goal was to provide non-proprietary, consistent and managed and standardized format to build the road database. In driving simulators, there are usually two ways to specify the position. One is using Cartesian coordinates (for instance OKTAL's SCANeR Studio), and the other is by specifying the position of the object on the road (for instance IFSTTAR's ARCHISIM). There is another way in which objects are specified by the distance travelled by the driver during the simulation trial (for instance System technology's STISIM Drive). In the SML, Both cartesian coordinates based and position based positions can be used.

Participant vehicle Normally, the participant's vehicle is created and configured the same way as all other vehicles but with a very small change in a parameter to consider it as a vehicle driven by the subject. So in SML there is a tag to specify the characteristics of a participant's vehicle.

Autonomous Traffic Specifying autonomous traffic is a critical and difficult task. In most current architectures, the vehicles are specified individually and configured accordingly. So usually the traffic is specified at low-level. Unfortunately, end-users are usually not concerned with the traffic at individual level, but they have to create the traffic by following the procedure offered by the driving simulators. In the SML framework that we propose, the information is provided at the high-level (i.e. number of vehicles in a specific area, types of vehicles, and their behaviour in the specific zone). If the end-user wants to specify the specific behavior of a vehicle from the traffic, s/he can specify it in the critical event section.

Environment In existing driving simulation scenario languages, the environment parameters (Rain, Fog, Light etc) are usually handled in the script written by the scenario developers. Separating it from the script will not only simplify the script but will be corresponding to the user way of designing the experimental protocol.

Critical Situation/Events Most driving simulators follow a rule-based language to specify the critical situations. And every driving simulator has list of API's which are used to develop a certain situation. So the APIs in different driving simulators provide different functionalities, so it was difficult to provide the generality of the traffic situation parameters at the low-level. Instead, the information is provided at the higher-level where users (researchers) specify the parameters of the situations. For example for a simple vehicle-cut-in event, the parameters could be characteristics of the cut-in vehicle, and the distance when the vehicle will change its lane to the specific lane, and some other parameters depending on the situation to be studied. Any driving simulator could be equipped with the template library as specified in chapter 4, using which the end-user can manipulate the parameters at low-level using the low-level language provided by the parameters; it can be done using the Interoperability framework which is presented in the next section.

6.5 Interoperability framework

This section presents an interoperability framework, in order to execute the driving scenarios on different driving simulators. Figure 6.3 presents the scenario modelling process using the SML and the high level view of the interoperability framework. End-users can develop scenarios using the user interface developed in chapter 4 in the SML format, or they can just develop the SML format for the experiment. The SML format can be imported to the specific platform using the importer that will convert the SML information into the platform specified code for the driving simulator.

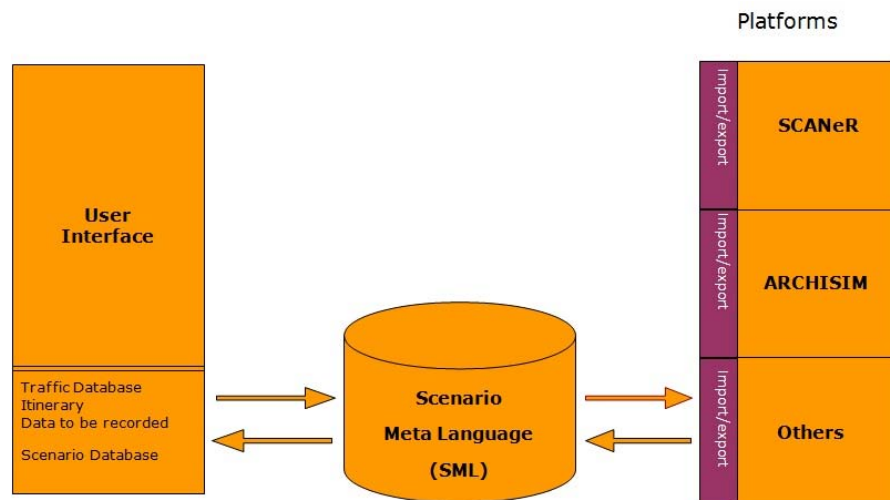


Figure 6.3: Typical scenario modeling process.

The proposed approach (interface in chapter 4) can be integrated with SML using the framework as shown Figure 6.4 and 6.5. The technical person will develop a template, which has two representations: XML representation and low-level specification.

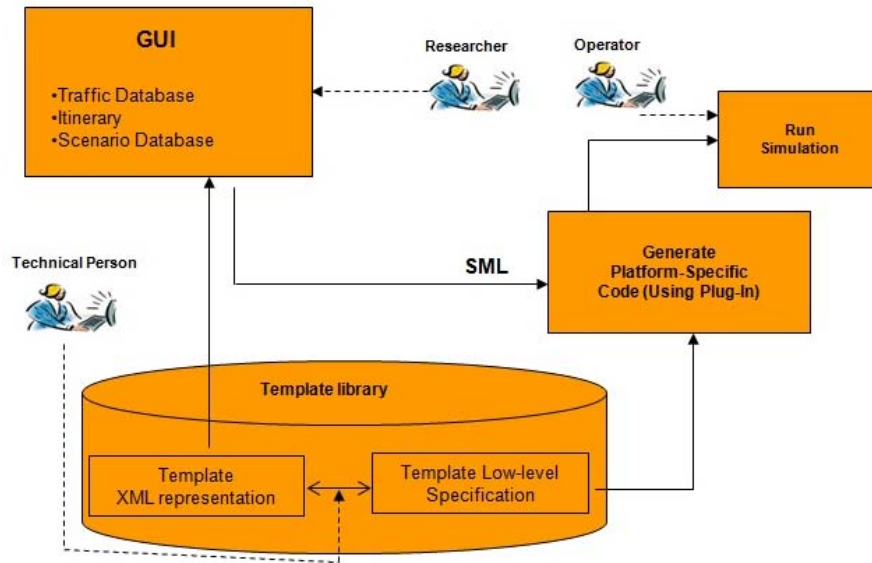


Figure 6.4: Interoperability Framework.

The end-user will fill the template parameters along with the other experiment steps in the interface and the whole scenario will be converted into SML format. The SML file will then be imported by the importer on a specific platform, which will use the low-level specification of the template and execute the scenario on the driving simulator. The whole framework can be regarded as a multi-layer programming environment, where end-users interact with the higher upper layer (High-level), i.e. the typical programming process is converted into customization process.

The technical-persons interact with the lower-layer and develop the templates to be used at upper layer, along with the low-level representation. There are two representation of a template, XML and low-level. The XML representation contains the parameters of the template and the low-level specification contains the logic and the programming representation of the template developed with the API of a specific driving simulator.

6.5.1 Implementation of the Interoperability Framework

In order to test the interoperability framework, an SML was implemented on the SCANeR software. An importer was developed to parse the SML file and generate the SCANeR specific scenario file to be executed on the SCANeR platform. The creation of participant vehicle is simple, and it can be created easily with the parameters specified by the user. As all driving simulators have different ways of specifying traffic vehicles, so in order to create the vehicles in a specific zone, the importer was able to generate vehicles that will be active in the specific zone. Implementation of the traffic zones can be a tricky part depending on the architecture of the driving simulator software. In some simulators it is not possible (at the moment) to

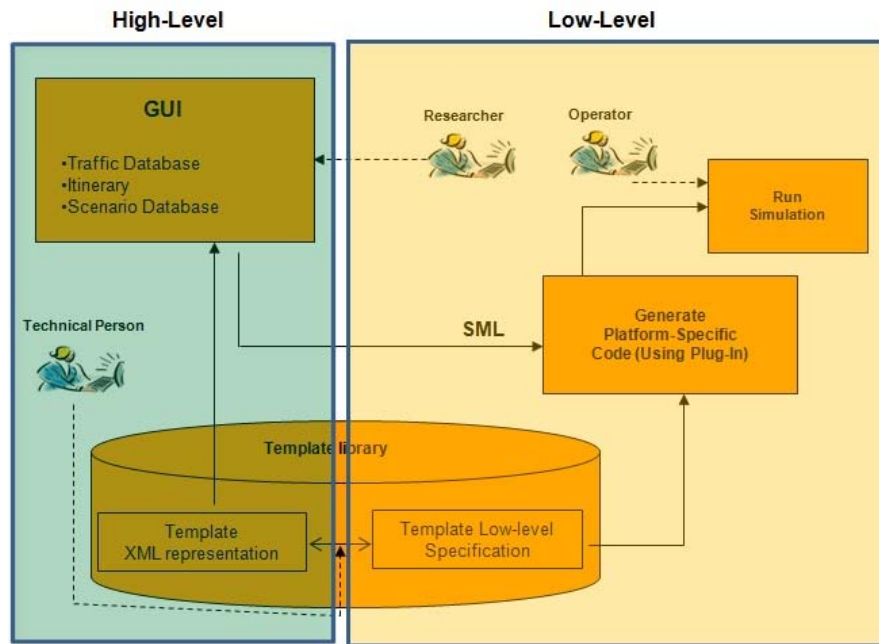


Figure 6.5: Interoperability Framework using multi-layer programming.

create vehicles dynamically, for example SCANeR. While in some simulators it is possible to create the vehicles dynamically. The formula applied during the implementation on SCANeR was that, create the maximum number of vehicles specified in the scenario, but activate or make them visible according to density specified by the user in a specific traffic zone. For example in one zone user has specified the traffic density as 10 vehicle/mile and in second zone the user has specified the density as 20 vehicle/mile. Then maximum 20 vehicles will be created, and 10 vehicles will be deactivated when the participant will be driving the car in the first zone, and all 20 vehicles will be activated while driving in the second zone.

The environment ones can be created the same way as the traffic zone, but they are easier to configure than traffic zones. The environment zones can be configured depending on the conditions specified by the user in the interface.

Similarly, template parameters specified by the end-users can be adapted to the low-level libraries developed for the simulators. As discussed earlier that, templates are composed of different low-level functions hidden to the researchers. So the configuration of the templates is adapted to the low-level functions, which are finally executed on the driving simulators. The traffic and environment zones were successfully tested on the SCANeR platform, but templates could not be completed implemented because of the shortage of time. When the end-users will import the SML file, it will actually generate the platform specific file (files) which is be executed on the driving simulator.

6.6 Conclusion

The Interoperability framework and SML help the scenario to be ported on more than one platform. The SML format is proposed based on the necessary information required to develop an experimental protocol. The interoperability framework separates the implementation of the scenarios at low-level from the high-level parameters entered by the end-user in order to configure a scenario. The High-level information is converted to Meta-format (SML) and finally to the platform specific format. So in order to execute an SML file, the platform needs to parse the SML file to execute the SML on the driving simulation platform. At the moment the conversion is one way. i.e. from proposed interface to the specific platform. The complexity of the scenarios is managed during the creation of the templates, when the template is created successfully it can be easily converted to be executed on the specific platform. The test case on SCANeR has been conducted in order to verify the format and the framework to execute it on different driving simulators.

Chapter 7

Conclusion

This chapter explains the concluding discussion followed by the contribution of the thesis and In the end future perspective of the presented work is explained.

7.1 Concluding discussion

End-user programming has been growing among users from different domains, and in the future it is expected that there will be an extensive need of end-user programming for end-user who can use such programming systems to fulfill their goals. In driving simulators, there is a need of a usable and intuitive end-user programming system using which researchers can develop experimental protocols to conduct experiments. Existing scenario authoring tools use low-level approaches to author scenarios on driving simulators, which make it difficult for the primary users of the simulators to model the scenario of their experiment. The reason is the gap between the end-user's skills and the goals they want to achieve using driving simulators. The user interfaces are expected to fulfill the gap between the end-users and the system they use. So one of the ways to fulfill this gap is to develop intuitive and user-friendly approaches and interfaces which could help end-users to develop scenarios using driving simulators. Many scenario modeling approaches and scenario modeling languages have been proposed, but none actually achieved the goal of filling the gap between end-users and scenario authoring systems. To sum up, no existing systems have the objective of allowing the users to develop scenarios with their current set of skills, so that they are currently dependant on technical persons. Many end-user programming techniques and methods have been proposed to support end-user programming activities. These techniques have been developed with different objectives in the mind. Also, there is no framework for the portability of scenarios among driving simulation platforms. Existing driving simulation platforms use different methodology and approach to model scenarios, so it is difficult to port scenarios at low-level, because all driving simulation platforms have their own specific low-level language. So there is a need for an interoperability framework and a meta-language which allow

the users to port scenarios from one platform to another for different reasons (e.g. different partners working on a same project, end-user need to learn a different technique or language syntax whenever they work on new driving simulations). So there is a need for an interface which could be used to develop scenarios on driving simulators and could be used on existing driving simulators.

In order to develop a system for end-users, the User-Centered Design (UCD) technique was used, in which end-users were involved at an early stage of the design process. Nineteen users with various levels of experience and background were interviewed. During the interviews they could explain their problems in detail while programming scenarios on driving simulators, and gave suggestions about the improvement to be made in driving simulators. The main problems shared by most users were 1) to control the ambient traffic around drivers during the critical situations; 2) to be able to tune or optimize the critical events; 3) to find the relevant functions in order to design an event; and 4) the availability of triggers to design critical events. The users also suggested some ideas in order to improve the scenario design: to drag and drop critical situations at high-level from a database, to be able to interact with the environment or to provide a preview of the successive steps needed in order to design an experimental protocol. User interviews gave a good insight of the user requirements corresponding to their needs to model scenarios on driving simulators.

Based on the user study and on the literature review, different users interacting with the driving simulators and their roles have been identified and a multi-user multilayer user centered design approach is proposed. Three sub-interfaces (Experiment Builder, Template Builder, and Experiment Interface) are proposed for each layer corresponding to the skill of end-users. Sub-interface The Experiment Builder (Higher-layer) is for the researcher who programs scenarios using the customization of the templates, the Template Builder (lower-layer) is for technical persons who develop the scenario templates. Researchers interact with the higher layer to customize the templates, exploiting the low-level programming practices while the technical persons interact with the lower-layer and develop scenarios using the low-level API's of the driving simulators. The sub-interface Experiment Interface is used by researchers or by the person who operates the experiment. Splitting the scenario development into sub-tasks not only empowers the end-users (researchers, psychologists, etc.) to develop scenarios by themselves using their own skills, it also reduces the workload of the technical persons, and both types of users can achieve their goals more efficiently. Technical persons build the high-level template library, while the researchers can use this library to customize the scenario templates, thus achieving their goals without being totally dependent on technical persons.

A prototype based on the proposed approach has been developed, and preliminary evaluation was conducted on 9 end-users. The users performed a small exercise on the prototype and were observed and interviewed later on. The early feedback was really encouraging. During the interviews, the users felt easy and it was intuitive for them to configure the templates, rather than low-level coding, and they were also interacting with the map. They appreciated

the guidance by navigation: there is a separate sub-interface for each step, minimizing the complexity of creating an experiment. After this preliminary study, the approach was improved and a new fully functional prototype was developed. Another study was conducted to get detailed feedback of the end-users. During this study qualitative as well as quantitative feedback were collected. Eighteen users participated in this study and they developed a real driving simulator experiment. In the exercise, there were two critical situations (cut-in and car-following). In driving simulator experiments, researchers usually have more than one trial for a given task with varying parameters.

This experiment was conducted with 3 hypotheses: 1) the proposed approach will empower the end-users, 2) The end-users will focus more on domain problems rather than programming issues; 3) The end-users will take less or no help using the proposed tool. Researchers of two categories were involved during the study. The first one had no programming experience at all and had never developed scenarios by themselves, while the other had some programming experience of developing scenarios on driving simulators with the help of technical persons. After performing the exercise, the end-users filled different questionnaires (Study-related, and SUS (System Usability Scale)). They were interviewed later on about their experience using the tool based on the proposed approach. The results of these two categories of users were compared.

All users completed all tasks, except configuring the environment and configuring the traffic situations. Only one user could not complete the Environment configuration task. To create critical situations, 4 users completed the task, and 10 users completed more than 75% of the task, one completed 50% and one could not do the task at all.

Configuring the autonomous traffic and creating the traffic situations has been found quite difficult, as expected. Along the situations, it was possible to see whether practice improved performance. Fig. 8 shows a learning behaviour, in terms of time spent to the task, for the configuration of the traffic zone and for both situations (cut-in and car-following). The learning effect only occurred between the first and second trials, which suggest that the tool was quite intuitive. From the SUS questionnaire, 6% of the users rated the tool as excellent, 72% rated it as good, 11% rated it as OK, and 11% rated it as poor. The user also filled a questionnaire in which they specified the easiness of each task. All tasks were considered easy by at least 88% of the users; 12% found the autonomous traffic task difficult, and 6% found the environment and critical event tasks difficult. We found that the performance of experienced user was better than for inexperienced user, but an interesting fact was that for each three tasks (traffic, first and second critical situations) the performance was nearly the same during the second and third trial, showing that inexperienced user adopted the tool and improved their performance with time. Some users, having worked on other tools, said it was uneasy in the beginning to adapt to this new tool. One encouraging result is that 89% of the users said they would not need help from technical persons anymore using the proposed interface.

Following a User-Centered Design approach, a prototype interface was designed in order to

design scenarios for driving simulators. This prototype followed some principles taken from a user needs study: for instance, in the proposed interface, the user's roles, interacting with driving simulators, have been separated. One of the difficulties in the programming is the division of a programming task into meaningful sub-tasks. The experimental protocol procedure was divided into different sub-tasks corresponding to the user's way of developing the experiment.

Support is provided at each step using different interaction techniques. The customization of the template should enable the end-user to develop his experiments without following a typical low-level programming process. The main objective of this work was to give unskilled users a tool to build their experiment by themselves, as much as possible, instead of asking the technical team. The prototype was evaluated on a panel of unskilled and low-skilled end-users, showing good results in terms of completion rate. Also, the subjective evaluation was encouraging, and we could show that the users could easily learn how to use the tool, so that after 3 trials, the performance of low skilled and unskilled users was roughly the same.

Another advantage of developing scenarios at high-level is that it is easy to port scenario from one simulation platform to another, which was another objective of this thesis work. In order to achieve this objective an interoperability framework and a meta-language were proposed and developed. The meta-language was successfully integrated on the SCANeR platform. It is practically possible to generate low-level platform specific code from the high-level information by developing an import module for most driving simulation platforms. Using this information the end-user can exchange scenarios within projects with common goals. It is usually a practice that simulation data are archived which can be further used by other people. Similarly, the high-level scenarios can also be archived and used by other people following the standards of the proposed meta-language. Although the role of the technical persons is not omitted completely, it is assigned as an assistant who develops the template which can be used by end-users in their absence. So end-users will not totally depend on the technical persons and they can develop a complete experimental protocol by themselves.

7.2 Contributions

In this section, we revisit the main contributions of this thesis work.

Behavioral researchers are the main users of the driving simulators, so they should be able to use these systems efficiently and effectively. But existing systems are not designed with the requirements and needs of behavioral researchers, and they also do not incorporate the profile of the researchers. We have proposed a new design for scenario development, using a user-centered design approach so that they could effectively and efficiently develop scenarios and use driving simulators to achieve their goals. End-users usually do not have programming and technical skills, which makes it difficult for them to program scenarios on driving simulators. A proof of concept of a new multi-layer programming approach is proposed, with which end-users can develop scenarios without programming and technical skills.

User-interfaces fill the gap between systems and the users to help them to achieve their goals from the system in use. We have proposed an example of usable and intuitive user-interface using which end-users developed experiment protocol, which proves that HCI techniques can effectively be used to fill the gap between systems and user skills. Technical persons are also among the users of driving simulators. They develop scenarios for researchers, but there is no standardized way with which they can help researchers to achieve their goals. The proposed multi-layer programming approach defines the role of the technical persons and empowers end-users so that they can develop scenarios by themselves without totally depending on the technical persons.

The need for an interoperability framework to port scenarios from one platform to another is discussed in detail in Chapter 6. Such an interoperability framework is proposed, where scenarios at high-level can be ported from one platform to another. In order to transfer scenarios from one platform to another, a meta-format is needed, which could be adapted by different platforms; an attempt to develop such a meta-language is proposed and can be used to integrate existing driving simulators with any interface developed with the proposed approach. This interoperability framework is validated on the SCANeR platform by developing an importer to integrate meta-language with the SCANeR software.

7.3 Prospective

A study was conducted with the tool developed during this thesis work, but there was no existing study with which we could compare the results, showing to what extent end-users have improved their performance in comparison to other tools. Thus, a within-group study should now be conducted where the performance of users (with the same profile) could be compared on existing tools vs. on the tool developed during this work.

A meta-framework has been proposed to port scenarios from one platform to another. The interoperability framework was tested on the SCANeR platform, which imported the meta-format (SML (Scenario Meta-Language) format) of the scenarios and generated SCANeR-specific low-level code. There is a need for improvement in the meta-framework during its integration with other tools, as it would be tricky for existing driving simulation platforms to adapt the high-level information and convert it into low-level platform specific information. So this interoperability framework still has a room to be improved while validating it for other driving simulation platforms. We hope that a standardized framework could be emerged.

The 'Template Builder' used by technical persons will be used to develop templates for future situations. It represents how different actors or simulation objects will behave during the simulation trial, and end-users will customize these templates in the 'Experiment Builder'. So Templates must reflect the end-users mental model of a specific situation. More information is needed to find out to what extent a template is actually expressive for the end-users. For instance, it was observed during the experiment that in the first traffic situation the end-users had to use one template while in the second situation, two templates were needed, which

Chapter 7. Conclusion

was sometimes confusing. So more information is needed while developing templates for the end-users. It can be identified by discussing with the end-users during the enrichment of the template library and by using the experience of technical persons. In order to support the users during the development of experiment protocols, additional features can also be provided to entertain the users, for example copying and pasting the scenario templates etc.

With the growing need of end-user software development, this thesis is an attempt to empower unskilled end-users of the driving simulators to develop scenarios by themselves. Users are the center of any system, so user-centered design technique has been used to design the scenario authoring tools for driving simulation end-users.

Publications

International Journal

- Ghasan BHATTI, Roland BREMOND, Jean-Pierre JESSEL, Nguyen-Thong DANG, Fabrice VIENNE, Guillaume MILLET. Conception and Evaluation of a User-Centered User Interface to Model Scenarios on Driving Simulators. *Transportation Research: Part C - Emerging Technologies* [IF=2.82], special issue on new technologies and emerging methodologies in road safety (available online)

Book chapter

- Ghasan BHATTI, Roland BREMOND, Jean-Pierre JESSEL, Nguyen-Thong DANG, Fabrice VIENNE, Guillaume MILLET. User-Centered Design (UCD) Approach to Model Scenarios on Driving Simulators. In "Driver Adaptation to Information and Assistance Systems" Edited by A. Stevens, J. Krems and C. Brusque (IET Publisher), pp. 275-299, 2013.

Conference papers

- Ghasan BHATTI, Roland BREMOND, Jean-Pierre JESSEL, Nguyen-Thong DANG, Fabrice VIENNE, Guillaume MILLET. A detailed description of a user-centered interface to model scenarios on driving simulator. In Proc. International Conference on Road Safety and Simulation (RSS 2013), Rome (Italy), October 2013.
- Ghasan BHATTI, Roland BREMOND, Jean-Pierre JESSEL, Nguyen-Thong DANG, Fabrice VIENNE, Guillaume MILLET. Filling the User Skill Gap Using HCI Techniques to Implement Experimental Protocol on Driving Simulators. In Proc. International Conference on Advances in Computer-Human Interactions (ACHI 2013), Nice (France), February 2013.
- Ghasan BHATTI, Roland BREMOND, Jean-Pierre JESSEL, Fabrice VIENNE, Guillaume MILLET. Towards the development of a user interface to model scenarios on driving simulators. In Proc. Driving Simulation Conference (DSC 2012) Paris, September 2012.

Chapter 7. Conclusion

- Ghasan BHATTI, Roland BREMOND, Jean-Pierre JESSEL, Fabrice VIENNE, Guillaume MILLET. User-requirements to model scenarios on driving simulators. In Proc. International Conference on Drivers Behaviour and Training (ICDBT 2011) Paris, November 2011.

Bibliography

- Ahmad, O. (2005). Issues related to the commonality and comparability of driving simulation scenarios. In *Proc. IMAGE*, Scottsdale, AZ.
- Allen, R. W., Park, G., Rosenthal, T. J., and Aponso, B. (2004). A process for developing scenarios for driving simulations. In *Proc. Image Conference*, Scottsdale, AZ.
- Allen, R. W., Rosenthal, T. J., Aponso, B., and park, G. (2003). Scenarios produced by procedural methods for driving research, assessment and training applications. In *Driving Simulation Conference North America*.
- Allen, W., Rosenthal, T., Aponso, B., Parseghian, Z., Cook, M., and Markham, S. (2001). A scenario definition language for developing driver simulator courses. In *Driving Simulation Conference*, pages 369–377, Sophia Antipolis, France.
- Alloyer, O., Bonakdarian, E., Cremer, J., Kearney, J., and Willemsen, P. (1997). Embedding scenarios in ambient traffic. In *Driving Simulation Conference*.
- Anderson, J. R. and Thompson, R. (1989). *Use of analogy in a production system architecture*, pages 267–297. Cambridge University Press, NY.
- Atomi (2013). Activepresenter.
- Bangor, A., Kortum, P., and Miller, J. (2009). Determining what individual sus scores mean: Adding an adjective rating scale. *Journal of usability studies*, 4(3):114–123.
- Beggiato, M. and Krems, J. F. (2013). The evolution of mental model, trust and acceptance of adaptive cruise control in relation to initial information. *Transportation research part F: traffic psychology and behaviour*, 18:47–57.
- Bellamy, R., John, B., Richards, J., and Thomas, J. (2010). Using cogtool to model programming tasks. In *Proc. of Evaluation and Usability of Programming Languages and Tools*, pages 1–6, New York, NY. ACM.
- Bellotti, V. (1988). Implications of current design practice for the use of hci techniques. In *Proceedings of the Fourth Conference of the British Computer Society on People and computers IV*, pages 13–34. Cambridge University Press.

Bibliography

- Blana, E. (1996). *Driving Simulator Validation Studies: A Literature Review*. Leeds University, UK.
- Boshernitsan, M. and Downes, M. S. (2004). Visual programming languages: A survey. Technical report, UC Berkeley, CA.
- Brooke, J. (1996). Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189:194.
- Carter, A. and Hundhausen, C. (2010). How is user interface prototyping really done in practice? a survey of user interface designers. In *IEEE VL/HCC*, pages 207–211.
- Cremer, J., Kearney, J., and Papelis, Y. (1995). Hcsm: a framework for behavior and scenario control in virtual environments. *ACM Transactions on Modeling and Computer Simulation*, 5(3):242–267.
- Cypher, A. (1991). Eager: Programming repetitive tasks by example. In *Proceedings of the SIGCHI conference on Human factors in computing systems: Reaching through technology*, pages 33–39. ACM.
- Cypher, A. and Smith, D. C. (1995). Kidsim: end user programming of simulations. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 27–34.
- David, R. (1995). Grafcet: A powerful tool for specification of logic controllers. *Control Systems Technology, IEEE Transactions on*, 3(3):253–268.
- Dershowitz, N. (1986). *Programming by analogy*, pages 395–423. Morgan Kaufmann.
- Deursen, A. V., Klint, P., and Visser, J. (2000). Domain-specific languages: an annotated bibliography. Technical report, Technical report, Centrum voor Wiskunde en Informatica, Amsterdam.
- Devillers, F. and Donikian, S. (2003). A scenario language to orchestrate virtual world evolution. In *Proc. Eurographics*, pages 265–275. Eurographics Association.
- Dunican, E. (2002). Making the analogy: Alternative delivery techniques for first year programming courses. In *Proceedings from the 14th workshop of the psychology of programming interest group*, pages 89–99, Brunel University, Belgium.
- Espie, S., Saad, F., Schnetzler, B., Bourlier, F., and Djemane, N. (1994). Microscopic traffic simulation and driver behaviour modelling: the archisim project. In *Proc. Road Safety in Europe and Strategic Highway Research Program*, pages 22–31. VTI.
- Franceschini, D., Franceschini, R., Burch, R., Sherrett, R., and Abbott, J. (2004). Specifying scenarios using the military scenario definition language. In *Proceedings of the 2004 Simulation Interoperability Workshop*.

- Gajananan, K., Nakasone, A., Prendinger, H., and Miska, M. (2011). Scenario markup language for authoring behavioral driver studies in 3d virtual worlds. In *Visual Languages and Human-Centric Computing (VL/HCC), 2011 IEEE Symposium on*, pages 43–46. IEEE.
- Glinert, E. P. (1990). *Visual programming environments: paradigms and systems*. IEEE Computer Society Press.
- Godley, S. T., Triggs, T. J., and Fildes, B. N. (2002). Driving simulator validation for speed research. *Accident Analysis and Prevention*, 34(5):589–600.
- Green, T. and Petre, M. (1996). Usability analysis of visual programming environments: a 'cognitive dimensions' framework. *Journal of Visual Language and Computing*, 7(2):131–174.
- Green, T. R. G. (1989). Cognitive dimensions of notations. *People and computers V*, pages 443–460.
- Gulliksen, J., Lantz, A., and Boivie, I. (1999). User centered design in practice - problems and possibilities. Technical report, KTH, Royal Institute of Technology, Stockholm (Sweden).
- Hallvard, T. (2008). Ui design without a task modeling language - using bpmn and diamodl for task modeling and dialog design.
- Hieb, M. R., Tolk, A., Sudnikovich, W. P., and Pullen, J. M. (2004). Developing extensible battle management language to enable coalition interoperability. In *European Simulation Interoperability Workshop*.
- Ioannidou, A. (2003). Programmorphosis: a knowledge-based approach to end-user programming.
- JustinMind (2013). Justinmind prototyper.
- Kaptein, N. A., Theeuwes, J., and Van Der Horst, R. (1996). Driving simulator validity: Some considerations. *Transportation Research Record: Journal of the Transportation Research Board*, 1550(1):30–36.
- Kearney, J., Willemsen, p., Donikian, S., and Devillers, F. (1999). Scenario languages for driving simulation. In *Driving Simulation Conference*.
- Kearney, J. K. and Timofey F. G. (2011). Scenario authoring. In *Handbook of Driving Simulation for Engineering, Medicine, and Psychology*, Boca raton, FL. CRC Press/Taylor & Francis.
- Kirakowski, J. and Corbett, M. (1993). Sumi: The software usability measurement inventory. *British journal of educational technology*, 24(3):210–212.
- Ko, A., Abraham, R., Beckwith, L., Blackwell, A., Burnett, M., Erwig, M., Scaffidi, C., Lawrance, J., Lieberman, H., Myers, B., Rosson, M., Rothermel, G., Shaw, M., and Wiedenbeck, S. (2011). The state of the art in end-user software engineering. *ACM Comput. Surv.*, 43(3):1–44.

Bibliography

- Ko, A. J., Myers, B. A., and Aung, H. H. (2004). Six learning barriers in end-user programming systems. In *IEEE Symposium on Visual Languages and Human Centric Computing*, pages 199–206. IEEE.
- Krueger, H.-P., Grein, M., Kaussner, A., and Mark, C. (2005). Silab - a task-oriented driving simulation.
- Leitao, M., Sousa, A. A., and Ferreira, F. N. (1999). A scripting language for multi-level control of autonomous agents in a driving simulator. In *Driving Simulation Conference*, volume 99, pages 339–351.
- Lewis, C. and Rieman, J. (1993). *Task-centered user interface design: A Practical Introduction*. Univ. Colorado, Dept. Computer Science.
- Lewis, J. R. (1991). Psychometric evaluation of an after-scenario questionnaire for computer usability studies: the asq. In *ACM SIGCHI Bulletin*, volume 23, pages 78–81.
- Lewis, J. R. (2002). Psychometric evaluation of the pssuq using data from five years of usability studies. *International Journal of Human-Computer Interaction*, 14(3-4):463–488.
- Lewis, J. R., Utesch, B. S., and Maher, D. E. (2013). Umux-lite: when there's no time for the sus. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2099–2102. ACM.
- Lieberman, H., Paterno, F., Klann, M., and Wulf, V. (2006). *End-User Development: An Emerging Paradigm End User Development*, volume 9 of *Human-Computer Interaction Series*, pages 1–8. Springer Netherlands.
- Mao, J.-Y., Vredenburg, K., Smith, P. W., and Carey, T. (2005). The state of user-centered design practice. *Communications of the ACM*, 48(3):105–109.
- Myers, B., Park, S. Y., Nakano, Y., Mueller, G., and Ko, A. (2008). How designers design and program interactive behaviors. In *Proc. IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 177–184. IEEE Computer Society.
- Myers, B. A. (1986). Visual programming, programming by example, and program visualization: a taxonomy. In *ACM SIGCHI Bulletin*, volume 17, pages 59–66.
- Myers, B. A., Ko, A. J., and Burnett, M. M. (2006). Invited research overview: end-user programming. In *CHI'06 extended abstracts on Human factors in computing systems*, pages 75–80. ACM.
- Myers, B. A., Pane, J. F., and Ko, A. (2004). Natural programming languages and environments. *Commun. ACM*, 47(9):47–52.

- Nakasone, A., Prendinger, H., Miska, M., Lindner, M., Horiguchi, R., and Kuwahara, M. (2011). Openenergysim: A 3d internet based experimental framework for integrating traffic simulation and multi-user immersive driving. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, pages 490–498. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).
- Newell, A. and Card, S. K. (1985). The prospects for psychological science in human-computer interaction. *Human-computer interaction*, 1(3):209–242.
- Nielsen, J. (1994). *Guerilla HCI: Using discount usability engineering to penetrate the intimidation barrier*, pages 245–272. Academic Press, Boston MA.
- Nielsen, J. (2007). Breadcrumb navigation increasingly useful. Jakob Nielsen's Alertbox.
- Olstam, J. and Espié, S. (2007). Combination of autonomous and controlled vehicles in driving simulator scenarios. In *International Conference on Road Safety and Simulation*, pages 23–32, Rome, Italy.
- Pane, J. F. (2002). *A Programming System for Children that is Designed for Usability*. PhD thesis, Carnegie Mellon University, Computer Science Department.
- Pane, J. F., Myers, B. A., and Miller, L. B. (2002). Using hci techniques to design a more usable programming system. In *IEEE Symposia on Human Centric Computing Languages and Environments*, pages 198–206. IEEE.
- Pane, J. F., Ratanamahatana, C., and Myers, B. A. (2001). Studying the language and structure in non-programmers' solutions to programming problems. *International Journal of Human-Computer Studies*, 54(2):237–264.
- Papelis, Y., Ahmad, O., and Schikore, M. (2001). Scenario definition and control for the national advance driving simulator. *SAE Technical Papers*.
- Papelis, Y., Ahmad, O., and Watson, G. (2003). Developing scenarios to determine effects of driver performance: Techniques for authoring and lessons learned. In *Driving Simulation Conference North America*.
- Papelis, Y., Ahmad, O., and Watson, G. (2005). Driving simulation scenario definition based on performance measures.
- Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc.
- Park, G. D., Allen, R. W., and Rosenthal, J. (2011). Flexible and real-time scenario building for experimental driving simulation studies. In *Proc. CHI*.
- Paterno, F. (2013). End user development: Survey of an emerging field for empowering people. *ISRN Software Engineering*, 2013:11.

Bibliography

- Repenning, A. and Ioannidou, A. (2006). *What Makes End-User Development Tick? 13 Design Guidelines*, pages 51–86. Springer, Dordrecht.
- Repenning, A. and Perrone, C. (2000). Programming by example: programming by analogous examples. *Communications of the ACM*, 43(3):90–97.
- Reymond, G., Heidet, A., Canry, M., , and Kemeny, A. (2000). Validation of renault's dynamic simulator for adaptive cruise control experiments. In *Driving Simulation Conference*, pages 181–191.
- Rieman, J., Franzke, M., and Redmiles, D. (1995). Usability evaluation with the cognitive walkthrough. In *Proceedings of CHI*, pages 387–388.
- Robin, J., James, R. M., Cathleen, W., and Kathy, U. (1991). User interface evaluation in the real world: a comparison of four techniques. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 119–124. ACM.
- Romano, R. (2003). Real-time multi-body vehicle dynamics using a modular modeling methodology. *SAE Technical Papers*, pages 207–212.
- Rosenthal, T. J., Allen, R. W., and Aponso, B. (2003). Configuration management and user's interface for a pc based driving simulator.
- Sauro, J. (2013). Measuring usability with the system usability scale (sus).
- Sauro, J. and Dumas, J. S. (2009). Comparison of three one-question, post-task usability questionnaires. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1599–1608. ACM.
- Schmid, U., Mercy, R., and Wysotzki, F. (1998). Programming by analogy: Retrieval, mapping, adaptation and generalization of recursive program schemes. In *Proc. of the Annual Meeting of the GI Machine Learning Group*, pages 140–147.
- Suresh, P. and Mourant, R. R. (2005). A tile manager for deploying scenarios in virtual driving environments. In *Driving Simulation Conference North America*, Orlando, FL.
- TeamViewer (2013). Teamviewer.
- Tedesco, D. and Tullis, T. (2006). A comparison of methods for eliciting post-task subjective ratings in usability testing. *Usability Professionals Association (UPA)*, 2006:1–9.
- Törnros, J. (1998). Driving behaviour in a real and a simulated road tunnel. A validation study. *Accident Analysis and Prevention*, 30(4):497–503.
- Tullis, T. S. and Stetson, J. N. (2004). A comparison of questionnaires for assessing website usability. In *Proc. Usability Professional Association Conference*, pages 1–12.
- UCS (2002). User centered design methods. Usability Consulting Service.

- Van Someren, M. W., Barnard, Y. F., and Sandberg, J. A. C. (1994). *The think aloud method: A practical guide to modelling cognitive processes*. Academic Press, London.
- van Welie, M. (2001). *Task-Based User Interface Design*. PhD thesis, Vrije Universiteit, Amsterdam.
- Vredenburg, K., Mao, J.-Y., Smith, P. W., and Carey, T. (2002). A survey of user-centered design practice.
- Wassink, I., van Dijk, B., Zwiers, J., Nijholt, A., Kuipers, J., and Brugman, A. (2006). In the Truman show: Generating dynamic scenarios in a driving simulator. *IEEE Intelligent Systems*, 21(5):28–32.
- Wassink, I. H. C., Dijk, E. M. A. G. v., Zwiers, J., Nijholt, A., Kuipers, J., and Brugman, A. O. (2005). Bringing hollywood to the driving school: dynamic scenario generation in simulations and games. In *Proceedings of the First international conference on Intelligent Technologies for Interactive Entertainment*, pages 288–292. Springer-Verlag.
- Wittman, R. and Abbott, J. (2006). Keeping up with the military scenario definition language (msdl).
- Wolffelaar, P., Bayarri, S., and Coma, I. (1999). Script-based definition of complex scenarios. In *4th Driving Simulation Conference*.
- Wolffelaar, P. C. v. (1999). Functional aspects of the driving simulator at the university of groningen. Report, University of Groningen, Groningen, Netherlands.

Appendix A

Appendix A

SUS Questionnaire

The 10 item questionnaire with 5 response options is show below.

1. I think that I would like to use this system frequently.

Strongly disagree	Disagree	No idea	Agree	Strongly agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

2. I found the system unnecessarily complex.

Strongly disagree	Disagree	No idea	Agree	Strongly agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

3. I thought the system was easy to use.

Strongly disagree	Disagree	No idea	Agree	Strongly agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

4. I think that I would need the support of a technical person to be able to use this system.

Strongly disagree	Disagree	No idea	Agree	Strongly agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

5. I found the various functions in this system were well integrated.

Strongly disagree	Disagree	No idea	Agree	Strongly agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

6. I thought there was too much inconsistency in this system.

Strongly disagree	Disagree	No idea	Agree	Strongly agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

7. I would imagine that most people would learn to use this system very quickly.

Strongly disagree	Disagree	No idea	Agree	Strongly agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

8. I found the system very cumbersome to use.

Strongly disagree	Disagree	No idea	Agree	Strongly agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

9. I felt very confident using the system.

Strongly disagree	Disagree	No idea	Agree	Strongly agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

10. I needed to learn a lot of things before I could get going with this system.

Strongly disagree	Disagree	No idea	Agree	Strongly agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

SEQ Questionnaires

1. Creating and Specifying Participant vehicle was

Strongly disagree	Disagree	No idea	Agree	Strongly agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

2. Overall, Traffic zone creation and configuration task was

Strongly disagree	Disagree	No idea	Agree	Strongly agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

3. Overall, Environment/Weather zone creation and configuration task was

Strongly disagree	Disagree	No idea	Agree	Strongly agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

4. Overall, Creation of traffic situation task was

Strongly disagree	Disagree	No idea	Agree	Strongly agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

5. It was easy to specify the triggers

Strongly disagree	Disagree	No idea	Agree	Strongly agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

6. It was easy to specify the template parameters while creating traffic situations

Strongly disagree	Disagree	No idea	Agree	Strongly agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Other questions

1. The proposed approach to construct events can help the non programmers to develop an experimental protocol without any help (or minimum help) from the end-users

Strongly disagree	Disagree	No idea	Agree	Strongly agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

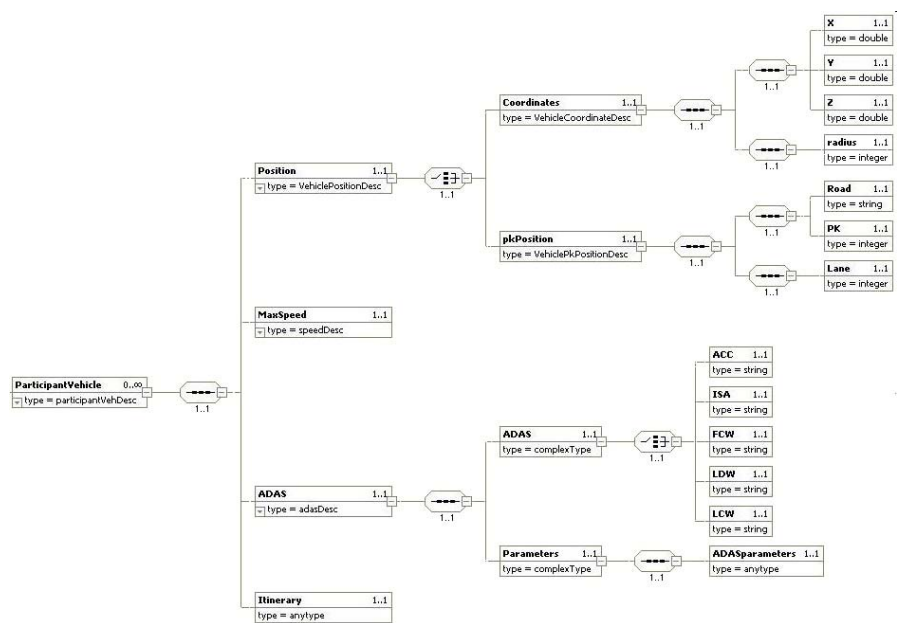
2. I spent less time on this tool while developing this protocol as compared to the tool that I use or have used for my research

Strongly disagree	Disagree	No idea	Agree	Strongly agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

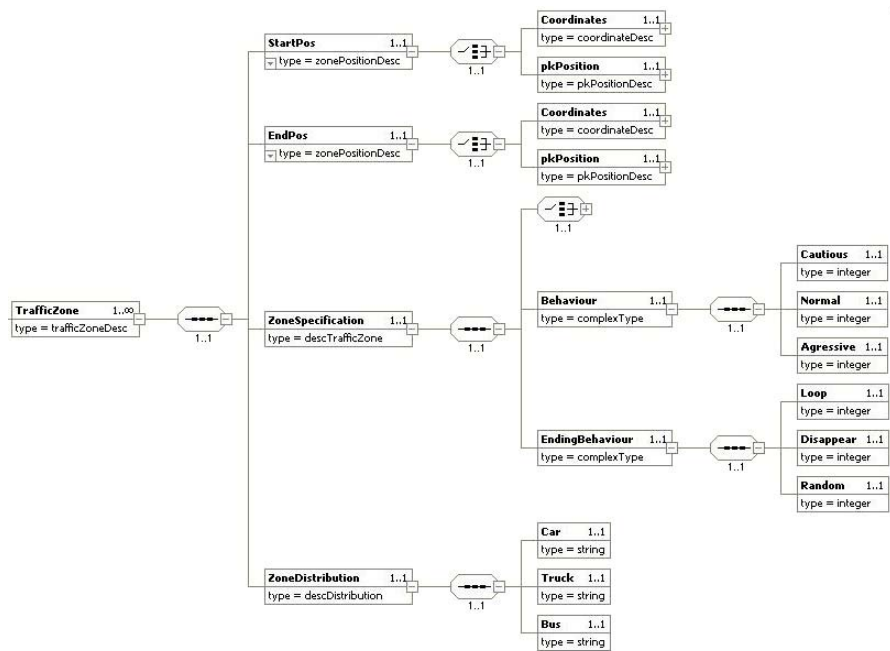
Appendix B

Appendix B

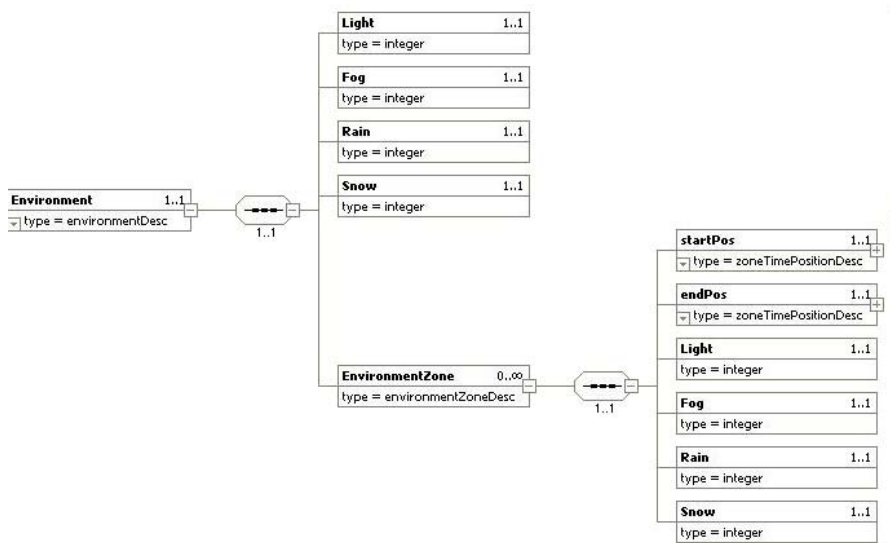
SML schema



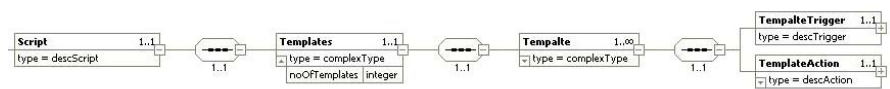
Participant Vehicle



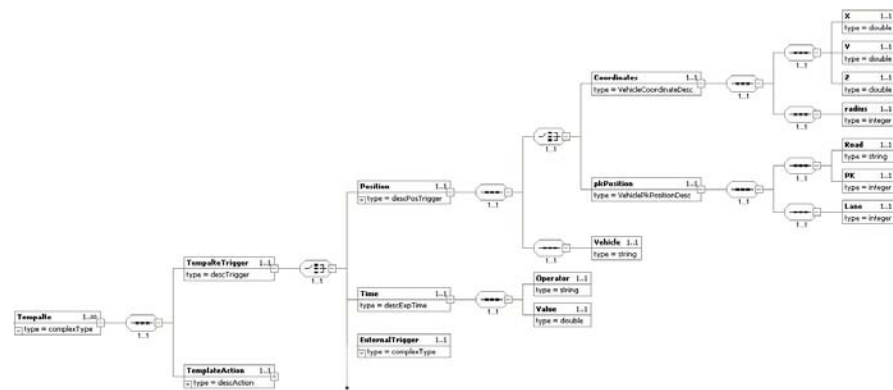
Traffic Zones



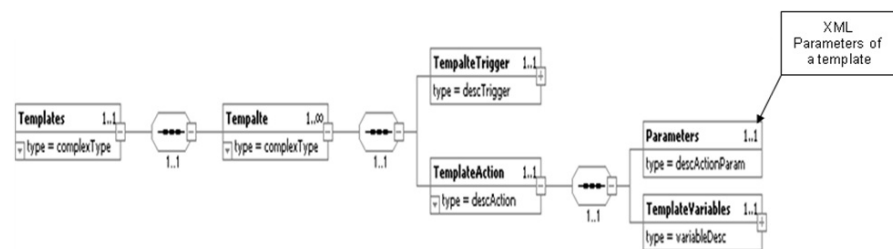
Environment Zones



Critical Situations



Triggers/Conditions for Critical situations



Template Parameters for Critical situations