An Integrative Framework for the Model-Driven Development of Traffic Simulations

Alberto Fernández-Isabel and Rubén Fuentes-Fernández

GRASIA**, Facultad de Informática, Universidad Complutense de Madrid
c/ Profesor José García Santesmases 9. 28040 Madrid, Spain
afernandezisabel@estumail.ucm.es, ruben@fdi.ucm.es

Abstract. Road traffic is a daily component of life in our societies. Researchers have made of simulations a key tool to study it. Their development still has to deal with the difficulties arising from the heterogeneity of the underlying theories, contexts, and goals. This frequently demands multiple profiles in development teams, which causes communication problems. This work proposes a model-driven development framework to address these issues. It organises tasks and roles around the work on models and transformations, making explicit all the information used in the development. The core modelling language is focused on the behaviour of participants in road traffic and their means of transport. Researchers use tailored tools to specify models compliant with that language and produce source code from them. A development process gathers the experience in these tasks to guide researchers. A case study on a simulation that incorporates a model of factors that produce driving mistakes illustrates the approach.

Keywords: Traffic modelling, road behaviour, development process, model-driven engineering, agent-based modelling

1 Introduction

Road traffic plays an important role in modern societies. People have to address different situations related to it as soon as they leave their homes: driving a vehicle, being a passenger, or walking down the street. These situations involve multiple interactions, depending on people features, knowledge, and means of transport. Their study has become an important research field related to goals such as improving vehicle and people flows, reducing dangerous situations, or pollution control. Real experiments regarding traffic are strongly limited given the involvement of living beings, and the complexity of its settings (i.e. number of elements, variables, and operations to consider). For this reason, the use of simulations has become common in these studies [9].

The development of a traffic simulation is a complex project. It needs to consider multiple theories about road traffic to explain the phenomenon from

** Group of Agent-based, Social, and Interdisciplinary Applications

different perspectives. Moreover, there are different levels of abstraction to consider, e.g. those of requirements, design, and code. These theories and levels frequently correspond to experts with different backgrounds. Integrating these aspects requires a precise understanding of all the information involved in the development of the simulation. Model-Driven Engineering (MDE) has been proposed to deal with these issues [3].

MDE [1] organises development around *models*. These models are compliant with Modelling Languages (MLs). *Transformations* automate some modifications of models (e.g. adding design or platform dependent information) and part of the generation of artefacts (e.g. code generation and reverse engineering). Processes provide guidelines for these tasks. MDE has the advantages of the high reusability of models and transformations, and the explicit specification of all the information involved in development. The main obstacle to adopt it in a given context is the high initial effort to set it up, e.g. to define MLs and usual transformations, or develop tailored tools.

This work proposes a MDE process to develop traffic simulations. It is based on a domain-specific Traffic ML (TML) and support tools. The design of the ML pursues facilitating the integration of different theories and the modelling of the multiple roles involved in traffic (e.g. drivers, passengers, and pedestrians). It is based on the Agent-Based Modelling (ABM) [5] and Driver-Vehicle-Environment (DVE) [13] approaches. ABM makes of *agents* its core modelling primitive. These are intentional and social entities. DVE organises traffic concepts in three categories corresponding to its acronym. It highlights the mutual influences of these elements in their behaviours. The framework uses Eclipse projects [6] to define the TML and two tools based on it, a model editor and a code generator.

The development process presented here describes the use of that infrastructure. It is specified with the Software Process Engineering Metamodel (SPEM) [7]. There are five phases: *Preliminary model evaluation*, *Model design*, *Source code generation*, *Simulation setup*, and *Simulation*. They cover the full development lifecycle, including the choice of theoretical models, their specification with the TML, the design of transformations, and the generation of code.

A case study models risks factors in traffic and develops code to use them in a simulation, illustrating the application of the process. It shows the involved tasks and tools, and the potential benefits of MDE for this kind of project.

The rest of the paper is organised as follows. Section 2 introduces the TML and development tools. Section 3 presents the process and Section 4 applies it to specify and simulate the case study. Section 5 compares the presented approach with related work. Finally, Section 6 discusses some conclusions and future work.

2 Traffic Modelling and Simulation Framework

The framework includes three main resources: the TML, a model editor, and a code generator. They are the basis of the proposed process.

The specification of the TML is made with a metamodel. It establishes its primitives and constraints. Metamodels are described using meta-modelling lan-

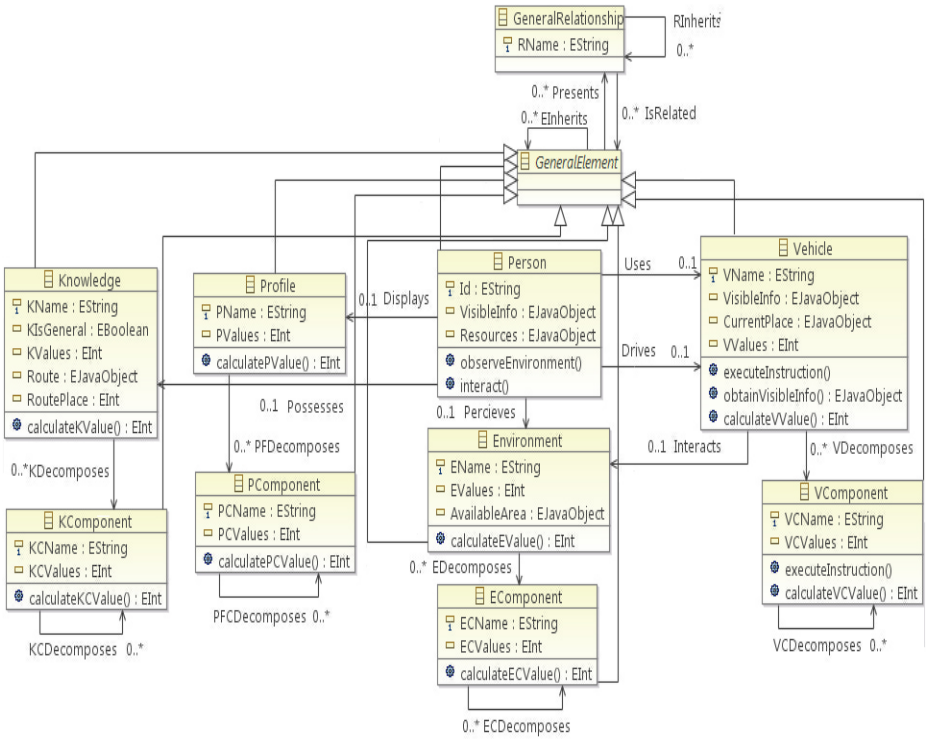

Fig. 1. Main elements of the TML metamodel.

guages [1]. In the case of Eclipse projects, this language is Ecore [6]. Figure 1 shows the core elements of the metamodel of the TML. They are organised using hierarchical structures based on inheritance and composition.

The inheritance hierarchies have two root meta-classes, the abstract *GeneralElement* for concepts and *GeneralRelationship* for relationships. The *EInherits* and *RInherits* references in these meta-classes allow introducing new inheritance relationships at the model level.

Those meta-classes which name includes *Component* (e.g. *KComponent* and *VComponent*) can be decomposed into others of their same type. For example, a *KComponent* into other instances of *KComponent*. This mechanism allows arbitrary composition hierarchies at the model level.

The metamodel introduces several constraints using the Object Constraint Language (OCL) [8]. For instance, these prevent inheritance in models among entities of different types using *EInherits*. In this way, in a model specification, a *Knowledge* class can only inherit from other *Knowledge* classes.

From the conceptual perspective, the TML is based on ABM [5] and DVE [13] approaches as said before. The *Person* meta-class is the core one. It represents people participating in traffic, regardless their means of transport (i.e. drivers, passengers, or pedestrians). This meta-class is related to the *Environment* meta-

class, which represents the information about the place where the action occurs. A *Person* can interact directly with it (i.e. pedestrians) or using an instance of the *Vehicle* meta-class (i.e. drivers or passengers). *Knowledge* and *Profile* meta-classes model characteristics of people, respectively their mental state (e.g. information on facts or how to maneuver a vehicle) and features (e.g. driving experience or emotional upsets). The *Knowledge* meta-class can represent global or individual information. The attribute *KIsGeneral* differentiates these uses.

Some information represented with the TML does not change in simulation time (e.g. age or vehicle type), but other does it (e.g. impatience or fuel consumption). The attributes which *Values* in their name (e.g. *PValues* or *VCValues*) and their related *calculate* methods are used to model these changes in terms of mutual influences among attributes from multiple entities. The details of these effects can be specified introducing code snippets.

Work with the previous metamodel is supported by two main tools. A graphical editor allows specifying models compliant with the TML, validating them, and applying OCL constraints. It is an Eclipse plug-in developed with the Graphical Editing Framework (GEF) [6]. The code generator supports the specification and execution of transformations to generate source code from models. It takes as basis the code templates generated for the metamodel elements by the Eclipse Modelling Framework (EMF) [6]. The tool provides graphical wizards to support the integration of platform-dependent libraries, the development of code snippets to attach to the model elements, and the final generation of transformations.

3 The Development Process

The development process for simulations starts when traffic experts study the problem and choose theories to model it, and finishes when the actual simulation is used in the platform. It considers the five phases that can be seen in Fig. 2. Though the diagram shows a linear workflow for the sake of clarity, the process is incremental and iterative, with feedback loops among the different phases and tasks. Regarding roles, *traffic experts* carry out most of the tasks, while *programmers* only collaborate in the *Source code generation* phase.

The process begins with the *Preliminary model evaluation* phase. Traffic experts select a theory suitable for the problem to study. This must be compatible with the TML (see Section 2), which implies that the theory must identify a set of characteristics of individuals or relationships among the elements considered that can be classified within the categories (i.e. concepts) in the TML. At this stage, experts must define a *modelling planning* with the main correspondences between the theory and TML concepts.

The *Model design* phase comprehends two internal phases. In *Build model*, traffic experts use the *modelling planning* to create a TML model with the types that represent the concepts from the chosen traffic theory. OCL [8] constraints can be used to set limitations beyond the TML in the use of the model types, e.g. limiting allowed inheritance and composition relationships. The result is a *theory model* with the base types and relationships for the simulation. In the

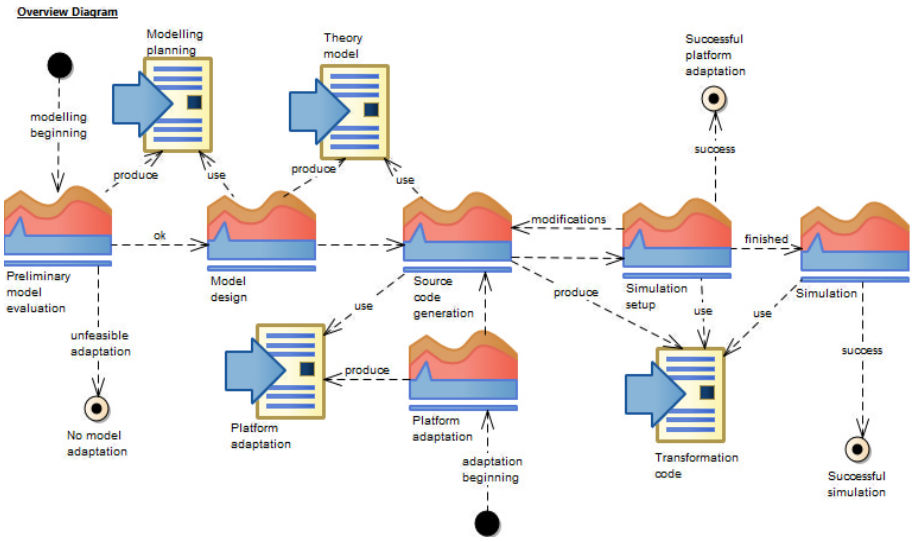

Fig. 2. Overview of the development process.

Validate model internal phase, traffic experts check the resulting *theory model* for assessing its semantic compliance with the TML and theory. Some semantic aspects are not represented at the syntactic level (i.e. with the metamodel and constraints), so experts manually check and document them. If problems are detected, the process must return to a previous phase to introduce modifications.

Next, the *Source code generation* phase produces from the *theory model* the basic source *result code* for the simulation. This code implements the concepts in the *theory model*. This phase is decomposed into three internal phases (see Fig. 3): *Get initial assets*, *Redefine source code*, and *Test result code*. All of them are supported by the code generator tool. In this phase, traffic experts can need in some tasks the help of programmers.

The *Get initial assets* internal phase prepares the code generator to start working. Experts load the *theory model* from the previous phase and external libraries, mainly describing the target simulation platform. They can also reuse a previous state of the workspace in order to bring code fragments from other projects.

The *Redefine source code* and *Test result code* internal phases are tightly linked in an explicit cycle of *generate & review*. The first one links model entities to code snippets, either reused or developed from scratch. The second one tests those fragments according to the simulation goals. Traffic experts and programmers work together here, respectively establishing the expected functionalities of the resulting code in the target platform, and developing that code when reusing is not possible or complex modifications are required. The code generator supports here exploring models, extending the classes defined in external libraries, code edition, and packaging and deployment. For instance, extending

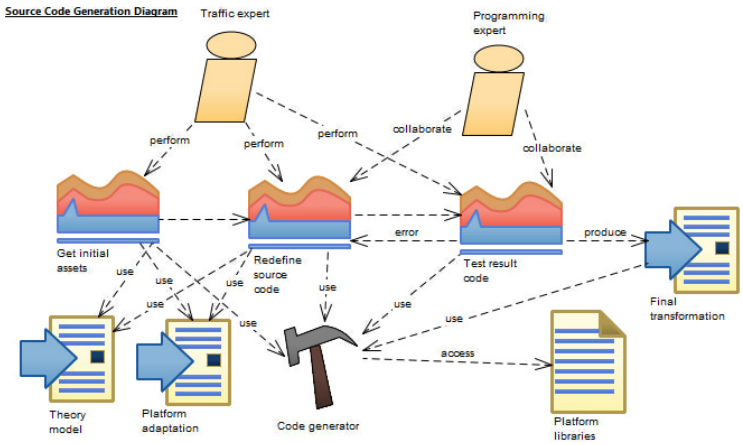

Fig. 3. *Source code generation* phase.

the external classes that implement the decision-making in an agent-based simulation platform with those that implement the model specification would allow modifying the choice of tasks. This would change the behaviour of agents in the platform, and support the specialisation of the model to a given platform.

The *Simulation setup* allows configuring the input parameters. A specific wizard in the code generator tool reads the *theory model* and *result code*. Then experts can create instances of types in the model and set their attributes. The resulting files are packed with the code from previous phases.

The *Simulation* phase is the last one. In it, traffic experts configure the simulation in the target platform (if needed),and run it for their studies.

4 Case Study

The case study illustrates how the development process produces source code for a test simulation platform starting from an initial traffic theory. This theory is a study of risk factors in driving [11, 12]. It classifies the main problems or situations drivers can find or feel during a trip (see Table 1), and how they affect their proneness to make mistakes.

The *Preliminary model evaluation* phase selects a theory and evaluates its adaptability to the proposed TML (see Section 2). In this case, the theory [12] is a taxonomy of concepts with a hierarchical structure. They can be classified following the categories of the DVE model [13], which is the main criteria of acceptance here. The *modelling planning* describes informally how the main theory elements can be related to those of the ML. In this case, categories I.A-C correspond to parts of the profile and II.A-C to knowledge of driver persons, III.A-B to the environment, and IV to the vehicle.

The *Model design* phase uses the graphical editor to translate the previous *modelling planning* to a base *theory model*. The first step is *Build model*. It

Table 1. Overview of driver error and incident causation factors.

I. Human Conditions and States:	**II. Human Direct Causes:**	**III. Environment Factors:**
A)PHYSICAL/PSYCHOLOGICAL	*A)RECOGNITION ERRORS*	*A)HIGHWAY-RELATED*
Alcohol impairment	Failure to observe	Control hindrance
Other drug impairment	Inattention	Inadequate signs and signals
Reduced vision	Internal distraction	View obstructions
Critical non-performance	External distraction	Design problems
	Improper lookout	Maintenance problems
	Delay in recognition for other or unknown reasons	
B)MENTAL/EMOTIONAL	*B)DECISION ERRORS*	*B)AMBIENT CONDITION*
Emotionally upset	Misjudgment	Slick roads
Preasure or strain	False assumption	Special/transient hazards
In hurry	Improper maneuver	Ambient vision limitations
	Improper driving technique or practice	Rapid weather change
	Inadequately defensive driving technique	
	Excessive speed	
	Tailgating	
	Excesive acceleration	**IV. Vehicular factors:**
	Pedestrian ran into traffic	Tire and wheel problems
C) EXPERIENCE/EXPOSURE	*C)PERFORMANCE ERRORS*	Brake problems
Driver inexperience	Panic or freezing	Engine system failures
Vehile unfamiliarity	Inadequate directional control	Vision obscured
Road over-familiarity		Vehicle lighting problems
Road/area unfamiliarity		Total steering failure

starts generating an initial diagram with the main container components, i.e. instances of the *Person*, *Environment*, *Profile*, and *Knowledge* meta-classes. The relationships specified in the metamodel among them must be inserted in order to get an appropriate validation of the model. Then, the elements from the target taxonomy [12] are added and related to the previous classes. For instance, the *Environment* instance is decomposed into *EComponent* instances for *III.A. Highway related* and *III.B. Ambient conditions*, which in turn are decomposed into their children from the original classification as additional *EComponent* instances. In *Validate model*, experts check the resulting model with the TML and the proposed theory. If no error was made, the process continues.

The *Source code generation* phase begins with *Get initial assets*. This uses the code generator tool to load the *theory model* from the previous phase. Then, it adds platform-specific libraries. In this case, a test traffic platform was developed to simulate environments with drivers, passengers, and pedestrians from real data on traffic. Its libraries are stored as a compressed file. Once these initial tasks are completed, the *Redefine source code* phase is focused on tailoring the original code EMF generates for the TML for its use with the *theory model*. Traffic experts select graphically the needed classes from the model (here all of them). These classes may contain empty or incomplete method bodies. Experts and programmers fill these methods with code snippets with the instructions to execute in the simulation.

An example of method description is specifying the *calculateXValue* methods included in several elements of the TML. For instance, code snippets similar to the default implementation with fuzzy logic provided in the code generator can be used. This is useful to consider the influence through relationships of some attributes over others. Customisation here is reduced to changing thresholds in rules. Other snippets require more coding. For instance, as this is the first use of the testing platform, to read values from it (e.g *CurrentPlace* in *Vehicle*).

The *Source code generation* phase ends with *Test result code*. Experts and programmers check if the produced code works as expected in the target platform. This phase also produces the result file with the generated code and the compressed file of the test platform.

Finally, the *Simulation setup* phase for the target platform can create files to configure the simulation. The target platform does not have configuration files, so classes with methods to create objects must be provided. These objects are instances of the classes in the code of the previous phase. The wizard reads the code of these classes, allows experts to specify their objects graphically, and generates the related source code.

This case study does not include behavioural components, which are not considered in this paper but are part of the TML. As the example simulation platform provides a default behaviour of path following adjusted with parameters as speed or reaction time, these can be linked to the generated *result code* here. For instance, this code can increase the speed for a vehicle in the platform when the related person has high values in its *PComponent* of *Driver inexperience*.

Two tasks appear as the most challenging for traffic experts and programmers. The specification of an appropriate *modelling planning* has to describe how to adapt and model the selected traffic theory with the TML. The example here is reduced, but more complex theories require a higher number of interconnected concepts. The second one is the code adaptation to the target simulation platform. It requires a cycle of generating specialised source code and testing it, repeated until the result is suitable for the goals of the study.

Despite these difficulties, the productivity of the process, considering the currently limited experimentation, is higher than in traditional development processes. These make a limited use of models (mainly for documentation purposes), guidelines (as few are documented in literature), and tools (mainly wizards) to guide experts. In our case, the application of these resources facilitates development, moving effort from coding to modelling.

5 Related work

Road traffic simulation is a broad field of research, that includes different areas depending on the specific problems simulated. This work mainly considers traffic aspects to model and the development process.

The proposed MDE process considers a full and flexible development process, while other works only offer partial constrained solutions [9]. For instance, the work in [4] specifies a model with an implicit specialisation relationship among available components. This can cause difficulties if the model must be extended to integrate other theories or elements. This is also the case of the taxonomy in [11]. To avoid this situation, our metamodel just specifies a general and open TML. This mitigates the problems related to integrate new elements from additional traffic theories. On the other side, there are also MLs too abstract, that can only produce a general specification of traffic. The work in [2] presents a model based on Activity Theory. It is designed to embrace multiple possibilities

regarding the elements present in traffic. However, it does not consider the different features of individuals, and how they can be related among them and with the environment in different ways. Thus, it provides little guidance to experts when modelling these aspects.

The adoption of tailored tools facilitates the application of the process. In particular, the code generator provides a visual interface for automating most of tasks, which facilitates the code transformation and the integration of the resulting source code in the target platform. The specialisation of model classes with code is isolated in specific tasks, which improves the autonomy of traffic experts. Programmers can perform these tasks using a well-known programming language (in this case Java), and avoiding the need of learning less-used specific transformation languages (e.g. MOFScript [10]).

6 Conclusions

This paper has introduced a MDE approach to develop road traffic simulations. It has described the phases of its process, going from the model design to the generation of source code.

The process is based on the TML. This is focused on the behaviour of participants in traffic (following ABM [5]) and their interactions (adopting the DVE model [13]). Its design pursues facilitating the integration of existing traffic theories, allowing representing multiple interactions of individuals with the environment. The ML also supports hierarchical structures of inheritance and composition to produce specialised elements.

Two main tools support the process. A graphical editor is used to specify models. The code generator allows transformations from models to source code, and code adaptation to the target platform. It also facilitates reusing artefacts from other projects.

The development process is conceived as iterative and incremental. It refines models and code, starting with those related to abstract concepts (those of the TML in the *theory model* and the code EMF generates for the TML) and moving to design concepts (in the *result code*). The process presents two difficult aspects: the generation of a *modelling planning*, where traffic experts determine how the elements from a theory fit into the TML; and the use of code snippets, where errors can appear before the target functionality is achieved.

The case study shows how the process meets its goals, producing appropriate code for the simulation in the target platform. The TML is also suitable to integrate the selected theoretical model.

More experiments support the approach and its guidelines, but it is still ongoing work with open issues. The TML needs more testing with additional theories to identify new potential modelling primitives. The process could need to redefine its phase structure, grouping some internal phases when tools automate them more, or introducing new steps with additional guidelines. For instance, the *Simulation setup* phase could be integrated in the previous one as an internal phase when it is simplified.

Acknowledgments

This work has been done in the context of the project "Social Ambient Assisting Living - Methods (SociAAL)" (grant TIN2011-28335-C02-01) supported by the Spanish Ministry for Economy and Competitiveness, the research programme MOSI-AGIL-CM (grant S2013/ICE-3019) supported by the Autonomous Region of Madrid and co-funded by EU Structural Funds FSE and FEDER, and the "Programa de Creación y Consolidación de Grupos de Investigación" (UCM-BSCH GR35/10-A).

References

1. Bézivin, J.: Model driven engineering: An emerging technical space. In: R. Lämmel, J.a. Saraiva, J. Visser (eds.) Generative and Transformational Techniques in Software Engineering, *LNCS*, vol. 4143, pp. 36–64. Springer, Heidelberg (2006)
2. Darbari, M., Asthana, R., Singh, V.K.: Integrating Fuzzy Mde-AT Framework for urban traffic simulation. International Journal of Software Engineering **1**(2), 24–31 (2010)
3. Fuentes-Fernández, R., Galán, J.M., Hassan, S., Villafañez, F.A.: Metamodelling for agent-based modelling: an application for posted pricing institutions. Studies in Informatics and Control **20**(1), 55–66 (2011)
4. Hidas, P.: Modelling lane changing and merging in microscopic traffic simulation. Transportation Research Part C: Emerging Technologies **10**(5), 351–371 (2002)
5. Janssen, M.A.: Agent-based modelling. In: J. Proops, P. Safonov (eds.) Modeling in Ecological Economics, pp. 155–172. Edward Elgar Publishing (2005)
6. Moore, B., Dean, D., Gerber, A., Wagenknecht, G., Vanderheyden, P.: Eclipse Development - using the Graphical Editing Framework and the Eclipse Modeling Framework, vol. 379. IBM RedBooks (2004)
7. Object Management Group: Software & Systems Process Engineering Meta-Model Specification, v2.0. `http://www.omg.org/` (2008). [Online: accessed 20-May-2015]
8. Object Management Group: Object Constraint Language, v2.4. `http://www.omg.org/` (2014). [Online: accessed 20-May-2015]
9. Pursula, M.: Simulation of traffic systems – an overview. Journal of Geographic Information and Decision Analysis **3**(1), 1–8 (1999)
10. The Eclipse Foundation: MOFScript Home page. `http://www.eclipse.org/gmt/mofscript/` (2015). [Online: accessed 1-September-2015]
11. Treat, J.R., Tumbas, N.S., McDonald, S.T., Shinar, D., Hume, R.D., Mayer, R.E., Stansifer, R.L., Castellan, N.J.: Tri-level study of the causes of traffic accidents: final report. Executive summary. Indiana University, Bloomington, Institute for Research in Public Safety (1979)
12. Wierwille, W.W., Hanowski, R.J., Hankey, J.M., Kieliszewski, C.A., Lee, S.E., Medina, A., Keisler, A.S., Dingus, T.A.: Identification and evaluation of driver errors: Overview and recommendations. Federal Highway Administration, USA (2002)
13. Xi, G., Qun, Y.: Driver-vehicle-environment closed-loop simulation of handling and stability using fuzzy control theory. Vehicle System Dynamics: International Journal of Vehicle Mechanics and Mobility **23**(1), 172–181 (1994)