Proceedings of the Federated Conference on
Computer Science and Information Systems pp. 1745–1756

DOI: 10.15439/2015F248
ACSIS, Vol. 5

Developing an Integrative Modelling Language for Enhancing Road Traffic Simulations

Alberto Fernández-Isabel
Rubén Fuentes-Fernández
{afernandezisabel@estumail, ruben@fdi}.ucm.es
Universidad Complutense de Madrid
Madrid, Spain

***Abstract*—Road traffic is a pervasive aspect in modern societies that affects millions people. The study of its multiple aspects is a very demanding task. Due to its complexity, traffic simulations become a key tool. Their development demands multidisciplinary teams, where communication problems are frequent. Model-driven engineering alleviates this situation providing graphical instruments for designing Modelling Languages (MLs) and semi-automatic transformations. This work presents a model-driven infrastructure composed by an integrative ML, a model editor, and a code generator. The ML is based on related literature and facilitates modelling different theories and simulations based on them. It considers the roles of individuals involved in road traffic, and partially adopts agent-based methodologies to model their decision-making. A case study shows how to produce a simulation specification adapting an existing traffic theory to the ML, and adjust this specification to a simulation platform for testing. It provides the basis for comparison with related work.**

I. Introduction

ROAD traffic is a complex phenomenon. Its study requires considering a large amount of variables, and it affects a variety of aspects, such as pollution, economical factors, leisure organisation, and health issues. The individuals involved play multiple roles in a broad range of scenarios, being able to establish complex relationships among them. These features make difficult studies based on experiments in real settings, which leads researchers to limit the variables considered and focus only on very specific aspects. In order to alleviate these restrictions, traffic simulations appear as a possible solution. Nevertheless, simulations present their own weak points [1]. Some of the most relevant are related to the difficulties to align (and check this alignment) the theories, goals, code, and results of the simulation, particularly because of the different backgrounds of people involved and the use of implicit information.

Approaches based on Model-Driven Engineering (MDE) [2] have been proposed to deal with these issues. They are organised around *models*, which are compliant with MLs, and generate other artefacts via semi-automatic *transformations*. MDE processes are usually incremental and iterative, allowing introducing improvements and modifications at any part of their workflows. MDE requires an initial effort for developing the elements of its infrastructure. This effort is higher than just implementing a simulation, but it compensates it with reusability (i.e. the resources can be used as a basis for other projects) and the explicit description of all the information (with MLs, models, and transformations).

Our approach provides a complete and integrative MDE infrastructure for road traffic simulations, focused on modelling individual behaviours. It introduces a Traffic Modelling Language (TML) and two development tools. A graphical editor supports describing the specifications, and a code generator helps to produce the semi-automatic transformations to generate the source code for a target simulation platform from those specifications.

The TML pursues being able to integrate (i.e. support the modelling and combination of) different theories related to road traffic. It considers the multiple viewpoints of the involved individuals (i.e. drivers, pedestrians, and passengers) and the roles in development teams (e.g. traffic expert and programmer). In this sense, it is a Domain-Specific ML (DSML) for this kind of problem. Faced to the traditional dichotomy in MLs between general and specific ones [3], our work chooses limiting the applicability of the approach to traffic simulations in order to provide better support in terms of guidance and tools to experts. The language is also intended to be platform-independent, so the details of the target platform can be considered just in late design tasks.

Following common practices in MDE, a metamodel describes the TML. It is organised using inheritance and composition hierarchies. The inheritances provide specialisation of concepts, while compositions are based on relationships of purpose, functional groups, physical links, or similarity. The metamodel is divided into three clusters: a *Mental cluster* where the different features of individuals are considered, an *Environmental cluster* to specify environment information, and an *Interactive cluster* to represent the interactions among individuals and the environment and the decision-making.

The *Mental cluster* considers the psychological features of individuals [4]. It plays a role similar to the mental state of the agent paradigm [5]. It adopts the BDI model [6], incorporating some of its knowledge concepts to model the current information that an individual or group possesses.

The *Environmental cluster* is based on the Driver-Vehicle-Environment (DVE) [7] approach. It considers that individuals can interact among them or with the environment, either directly or using their means of transport. These dynamic interactions influence the individual behaviours. This assumption

978-83-60810-66-8/$25.00

fits with Agent-Based Modelling (ABM) [8], where agents are intentional entities that can establish communications among them for different purposes (e.g. collaborate or interact).

The *Interactive cluster* models the decision-making of the individuals. It adapts this aspect from methodologies based on Agent-Oriented Software Engineering (AOSE) (e.g. INGENIAS [9] or Tropos [10]). *Goals* represent the people's objectives, and *tasks* are the instructions to execute in order those satisfy goals. These elements are considered in a *perceive-reason-act* cycle [11].

Regarding the development tools, the graphical editor allows designing the model instances compliant with the TML. The code generator takes as input these instances, and provides a set of functionalities to generate source code and adapt it to the target traffic simulation platform.

A case study shows the suitability of the MDE infrastructure to develop traffic simulations. It specifies the work in [12] using the ML. The graphical editor tool supports and guides this process, which produces a model specification and default source code templates for the primitives of the ML. These artefacts are the input of the code generator tool. It supports the completion of these templates with graphical wizards that assist users. This code is finally adapted to a specific traffic simulation platform, MATSim [13]. For this purpose, the code generator allows adding specific classes and code snippets to modify available code (e.g. using libraries or algorithms).

The rest of the paper is organised as follows. Section II presents the basic concepts of MDE and the related tools. The TML is introduced in Section III through its metamodel and clusters. Section IV presents the two development tools based on the TML, the model editor and the code generator. The case study in Section V illustrates the application of the approach. Then, Section VI compares this with related work. Finally, Section VII discusses some conclusions and future work.

II. Model-Driven Engineering

MDE [2] is a development methodology that is composed around *models*, in contrast to traditional approaches that are based on source code. The development process is focused on the production of iterative and incremental specifications of models going to abstract to accurate, where developers refine and add new elements to them at each step. During this process, *transformations* are introduced in order to automate repetitive modifications in models. For instance, generating patterns or concrete specialisations to target platform in order to produce models. Other related elements (e.g. source code or documentation) are compliant to these considerations, as they can be obtained from models using manual settings and transformations.

This development approach is based on modelling languages. In the case of graph-oriented languages, which are the most popular ones [14], the main instrument to achieve these definitions is the metamodel. Metamodels are commonly selected to describe their abstract syntax, but also they can be used to define their specific syntax or semantics [15]. Also, these metamodels are defined using meta-modelling languages. The Meta-Object Facility (MOF) [16] provided by the Object Management Group (OMG) is the standard in the domain. Nevertheless it presents some limitations. The absence of extensive tool support promotes that users frequently choose alternative languages or develop their own related tools. The Ecore meta-modelling language [17] is considered as an alternative as it is supported by multiple Eclipse modelling tools. These tools are organised around the Eclipse Modelling Framework (EMF) [17] and the Graphical Editing Framework (GEF) [18]. Also, Ecore adopts the Object Constraint Language (OCL) [19] to define model constraints and it is almost compliant to Essential MOF (EMOF). EMOF is a part of MOF focused on object oriented concepts and able to specify reflective operations. These features encourage this approach to select Ecore as its meta-modelling language in order to develop the TML.

Fig 1 shows the principal primitives of Ecore. An instance of *EClass* plays the role of its similar entities at the model level (i.e. classes). It clusters *EAttribute* and *EReference* elements. *EAttribute* instances provide features coming from *EDataTypes* (i.e. primitive types) to *EClass* instances. These primitive types include the most common (e.g. integer, char or string). An *EReference* instance symbolises a binary relationship in only one direction among two *EClass* instances. It allows creating containment and non-containment relationships. The *EReferenceType* of a specific *EReference* instance is indicated by its target *EClass*. Multiple *EClass* instances can be considered by *ESuperType* relationship in order to express inheritance among them. *EPackage* instances contemplate the possibility of grouping the structures of the metamodel.

Regarding the transformations, they are the other core instrument of MDE. They present different types of inputs and outputs, being able to be classified in [20]: Model-to-Text (M2T), Text-to-Model (T2M) and Model-to-Model (M2M) [21]. These transformations can be developed using general-purpose programming languages or transformation languages. In the first case, a module uses programming structures to manage its inputs and outputs. In the second case, the module is developed using a specific language for transformations and presents an engine that executes it in order to accomplish the process.

In this work, a module defined by a general-purpose programming language (i.e. Java) is adopted (see Section IV), as it can use techniques from reflection-based programming [20] and integrates several wizards that assist developers.

III. Traffic metamodel

Road traffic is a pervasive phenomenon that involves elements and situations. In order to study it, there are different theories that consider its aspects from multiple backgrounds and purposes. The same situation occurs with traffic simulations, where infrastructures differ on their modelling approach and goals.

In order to facilitate the integration and modification of elements in the TML and its study, this approach uses a metamodel [17] defined with Ecore.

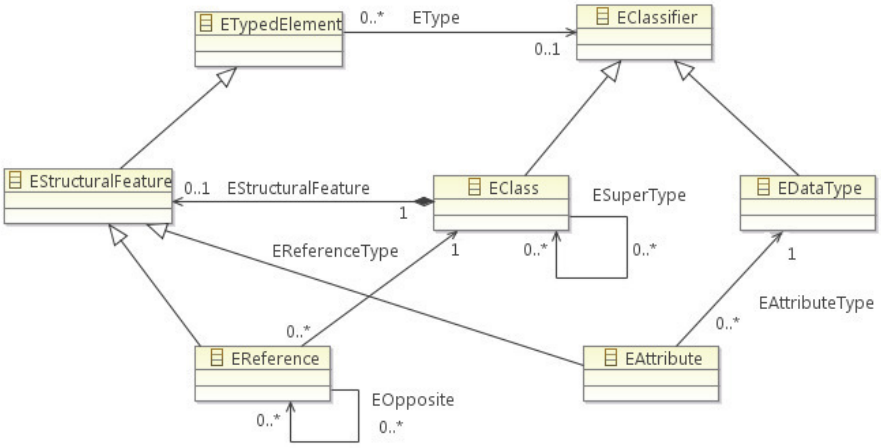

Fig. 1. Extract of the Ecore model selected from [17]

Regarding the concepts of the TML,they are mainly based on ABM [8] and structured into three clusters. The *Mental* and *Environmental* clusters gather the different concepts obtained from traffic literature. The *Mental cluster* represents the inner state of the participants in traffic [4]. The *Environmental cluster* includes the DVE approach [7]. These clusters have similar structures (see later in this section). The *Interactive cluster* is focused on representing the goals and actions of people involved in traffic. It is based on the guidelines of methodologies coming from Agent-Oriented Software Engineering (AOSE), integrating a *perceive-reason-act* cycle [11].

The core concept of the metamodel is the *Person* meta-class. It represents the types of people involved in traffic. According to their means of transport, they can be drivers, passengers, or pedestrians. These *Person* instances can interact with an *Environment* instance. This interaction is direct (in the case of pedestrians), or indirect when a *Vehicle* instance is used for it (for drivers and passengers). People' features are modelled with *Profile* instances, and the information they possess with *Knowledge* instances. Their acts are motivated by *Goal* instances, and the potential ways to achieve them are represented by *Task* instances. *Evaluator* instances determine how people have actually to act according to the circumstances, and *Actuator* instances execute the planned tasks.

The previous elements are arranged in inheritance hierarchies, adding the needed specialisation and structure to the metamodel. All concepts inherit from the *GeneralElement* meta-class (see Figs. 2 and 3). This meta-class provides the *EInherits* reference in order to represent inheritance among elements of the same type in model instances. The *GeneralRelationship* meta-class (see also Figs. 2 and 3) supports the introduction of relationships (e.g. affects or influences) among other elements. The *RInherits* reference allows its specialisation. Both types of references are constrained by expressions written with OCL [19]. For instance, constraints only allow inheritance among instances of the same type of meta-classes (e.g. a *Knowledge* instance only extends another *Knowledge* instance using a *EInherits* reference).

The internal structure of the *Mental* and *Environmental* clusters allows composition hierarchies using the *XComponent* (e.g. *KComponent* or *VComponent*) meta-classes. These meta-classes can be decomposed into others of their same types. All these compositions are constrained by OCL expressions. For instance, a *Profile* instance can be decomposed only into *PComponent* instances, while these *PComponent* instances can be only decomposed into others of the same type.

The meta-classes of the metamodel include attributes and predefined methods. Attributes can be specific for certain meta-classes or common (with similar name and meaning) to several meta-classes. An example of the first case is the *AvailableArea* attribute in the *Environment* meta-class; the *XValues* attributes (e.g. *PValues* or *EValues*) for storing the impact that an element has in the rest of the elements of a model instance are examples of the second group. Methods are placeholders for specifications that describe behaviour or attributes derived from others. For instance, code snippets can be attached to these methods in the model specification for code generation.

Next sub-sections discuss these aspects in detail. Subsection III-A describes the mental state and features of participants in traffic. Subsection III-B focuses on concepts to describe the traffic setting according to the DVE model [7], i.e. vehicles and the environment. Subsection III-C introduces the concepts to represent interactions among the previous elements and decision-making.

A. Mental cluster

The *Mental cluster* (see Fig. 2) represents the different concepts that can appear in the road traffic domain influencing the behaviour of individuals [4]. These concepts are classified as features of people or their current state.

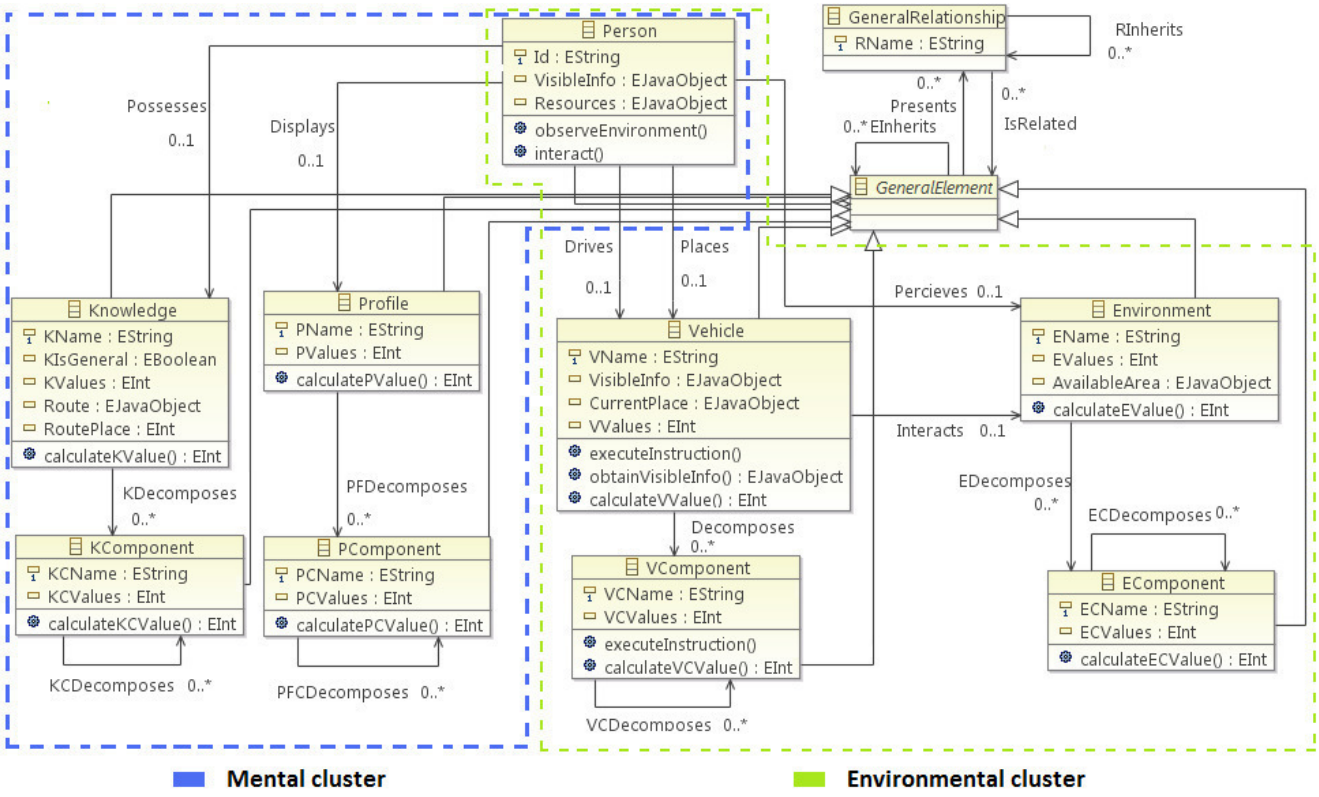

Fig. 2. Excerpt of the *Mental* and *Environmental* clusters of the metamodel.

The cluster includes three main meta-classes: *Person*, *Profile*, and *Knowledge*. *Profile* represents the different features of people in traffic. *Knowledge* considers the current mental state (but the goals) that a *Person* uses when dealing with traffic. It can be factual (e.g. traffic signs), procedural (e.g. how to overtake a vehicle), and normative (e.g. drivers should respect safe distances with other vehicles) knowledge. *Profile* instances describe people features (e.g. age or fatigue).

Both knowledge and features of people can specify information that does not change in simulation time (e.g. gender or meaning of signs), or does it (e.g. stress or mood). Proper *calculate* methods and their associated attributes must be specified to describe how to calculate them later.

The instances of the *Knowledge* meta-class and their composition meta-classes can represent information belonging to individuals (e.g. the current route), or global information available for every participant in the simulation (e.g. the speed limit in a specific type of road). The *KIsGeneral* attribute differentiates both uses.

This cluster is closely related to the agent paradigm [5]. For instance, the *Knowledge* meta-class can consider the *Beliefs* of people involved in road traffic, which are contemplated in the BDI model [6].

B. Environmental cluster

The *Environmental cluster* (see Fig. 2) adapts the concepts of the DVE model [7], as this is focused only on the driver role and the TML considers others. Thus, here it is considered that an individual can get information from the environment (any participating person) and the vehicle (only drivers and passengers). These elements can be extended to facilitate the potential accommodation of other theories.

The cluster considers how individuals relate to their means of transport and environment. It comprehends three main meta-classes: *Person*, *Environment*, and *Vehicle*. *Environment* represents the place where people (i.e. *Person* instances) interact, including the physical conditions that can occur (e.g. weather and road conditions). A model specification has a unique *Environment* instance shared by all the individuals. The *Vehicle* represents the means of transport, considering the different roles of people in road traffic (i.e. driver, passenger, and pedestrian). Drivers and their passengers relate to the *Environment* through their vehicles, but only drivers can use them to act on it. In the case of pedestrians, they have a direct relationship with the environment.

The mutual influences among *Person*, *Environment*, and *Vehicle* instances because of their relationships are partially represented in the metamodel with some attributes. The *Environment* meta-class has an *AvailableArea* attribute. It indicates the part of the environment that can be perceived. The *Person* and *Vehicle* meta-classes include the *VisibleInfo* attribute to specify which information from the *Environment* instance can be perceived in or through their instances.

C. Interactive cluster

The *Interactive cluster* (see Fig. 3) describes how *Person* instances act on the traffic situations considered by the *Mental* and *Environmental* clusters (see Sections III-A and III-B). Its components are organised into two groups. The first one describes the objectives of people and their capabilities to achieve them. The second one represents the elements that carry out the acting cycle.

The first group includes the *Goal* and *Task* elements. These two concepts come from Multi-Agent Systems (MAS) [22], and agent-based methodologies. These methodologies include a specific acting architecture where agents play multiple roles and try to meet the requirements of their different goals. These goals are enabled according to the agents' mental states and are directly related to task elements that can satisfy them. These goals can be decomposed into others, generating *OR* or *AND* compositions. Tasks can be decomposed in a similar way in order to describe complex jobs.

In our work, the *Mental cluster* represents the mental state of agents, and the *Environmental cluster* provides information from the environment and the vehicle (only in the case of drivers and passengers). The *Goal* meta-class represents a state of some traffic elements a person aspires to keep or reach, and the *Task* meta-class models person's capabilities. Both meta-classes hold specific attributes to characterise them. *Goals* have *Satisfaction* attributes that represent their satisfaction conditions. *Tasks* include *Instructions* attributes to specify the atomic actions that implement them.

These meta-classes can be decomposed into others of their type (constrained with OCL expressions), following the structure already seen for sub-components in the other two clusters. However, semantics are different. Here, they are related to satisfaction instead of determining the features of a component. The *Goal* and *Task* meta-classes present the *GType* and *TType* attributes in order to specify the type of compositions (e.g. *OR* or *AND*). The *GType* attribute represents the type of goal satisfaction compositions, while the *TTYpe* attribute indicates if the current task is accomplished by completing one or all its sub-tasks. These semantics are flexible, as both attributes could be modified to support different structures and classifications.

The second group represents those elements of a person that are in charge of evaluating the actual known state and executing actions. It follows a classical *perceive-reason-act* cycle [11] with evaluators and actuators (based on [23]). The information perceived from the environment is stored in the elements of the *Environmental cluster* (including the *Person* meta-class), the reasoning is carried out by *Evaluator* instances, and the acting is achieved with *Actuator* instances.

Evaluator instances can be decomposed into others using the *EVDecomposes* reference, being able to distribute the liabilities among them. In the case of *Actuator* instances, they cannot be decomposed into others. They can use inheritance through *EInherits* references, but each *Person* instance (i.e. a person type modelled) can only present one *Actuator* instance related to it (see the *Utilizes* reference in Fig 3).

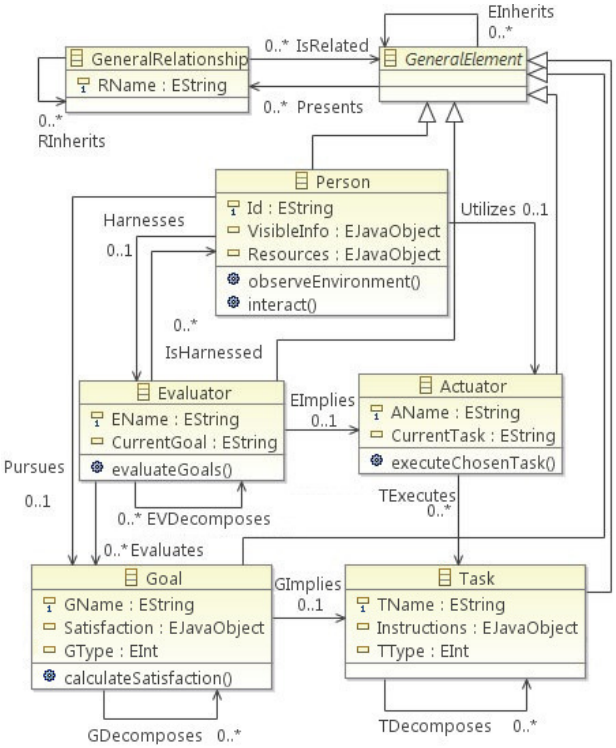

Fig. 3. Excerpt of the *Interactive cluster* of the metamodel.

Evaluator instances assess the information obtained from the *Environment* and *Vehicle* instances, the elements they are composed, and the available relationship instances linked to their *Person* instance. From that, they update the internal state of the *Person* instance. All this information determines the current state of goals. Once a candidate goal is selected, an *Actuator* instance picks its associated tasks. It executes these *Task* instances through its *Instructions* or subtasks.

IV. Development tools

The MDE approach presented in this paper is supported by two main tools: a graphical editor where the model specifications are developed, and a code generator where multiple operations to produce source code from models are achieved through semi-automatic transformations. Their implementation is based on the Eclipse Modelling Framework (EMF) [17] and the Graphical Editing Framework (GEF) [18].

The graphical editor is an Eclipse plug-in that guides users in the development of model specifications. It generates models compliant with the TML through a visual interface. This interface provides a canvas and a palette for displaying the model and the concepts of the metamodel. The model generated can be validated to ensure its compliance.

The code generator tool takes as input the model specifications produced by the graphical editor. In a first step, it associates to the classes in the model the source code EMF generates automatically for their meta-classes. Also, it provides options to integrate other external files, e.g. specific

libraries coming from the target platform or even the entire simulation with its associated dependencies. This can be used later to modify the preliminary code.

Over that input, the tool presents a graphical interface for displaying the information captured in the model specifications, allowing an intuitive navigation of them. The information about a selected element instance includes the methods associated (original and newly created), the decomposed elements it presents, the *GeneralRelationship* instances where it acts as the origin, and the elements from which it inherits (see Fig.4).

The code generator implements operations related to source code transformation and model adaptation, and operations related to specialisation to target simulation platform. Most of them are partially automated through wizards in order to provide guidance to users. A text editor, an internal graphical editor, and a compiler are integrated in the infrastructure in order to support these features.

Regarding the code transformation, main functionalities are: source code injection for platform adaptation, design and storage of self-contained *Interactive clusters* (see Section III-C), and cluster integration.

The injection of source code offers two alternatives. They are based on techniques from reflection-based programming [20] in order to modify and compile dynamically the default EMF implementation. The first one redefines only the body of the methods of the classes adding different instructions, using suitable code snippets for the target simulation platform. The second one is more complex, being able to complete the entire class or extending it from another one of the same type previously redefined. This allows adding new attributes and methods. As these operations require some programming skills, the graphical interface and the integrated modules (i.e. text editor and compiler) assist to examine the metamodel and model elements, and their code. This facilitates these tasks and produces a more intuitive development environment. There is also on-line help and examples to guide users in this point.

The design and storage of self-contained *Interactive clusters* uses the integrated graphical editor. It provides (in a similar way to the graphical editor plug-in explained above) a canvas and a palette to create the multiple elements (i.e. *Goal*, *Task*, *Evaluator*, *Actuator*, and *GeneralRelationship* instances) and a validation tool. After that, the generated cluster can be loaded into the tool in order to perform other tasks, such as code injection or storage of the development stage. In the last case, a wizard creates a compressed file including the current stage and the model designed, which allows continuing with the graphical design in the future.

The cluster integration functionality is linked to the previous one. It merges a model specification only with elements from the *Mental* and *Environmental* clusters previously loaded in the tool, and a stored self-contained *Interactive cluster*. A graphical wizard facilitates the process showing to users the available elements in each cluster in order to link them through references from the TML. Also, *GeneralRelationship* instances can be added or completed (i.e. relationships with origin in

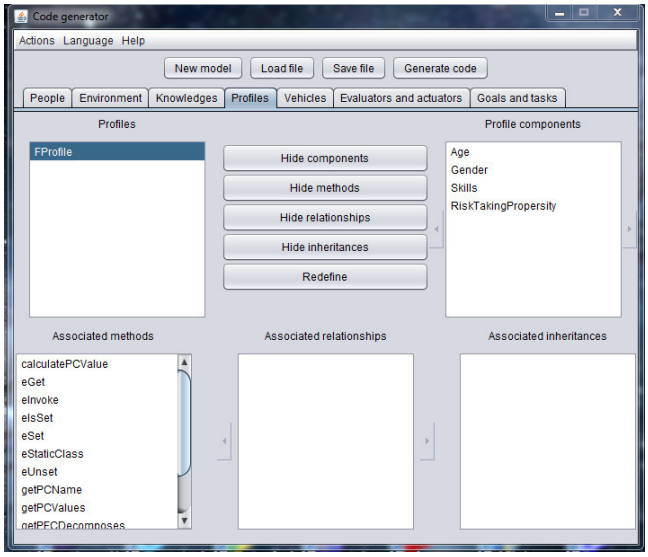

Fig. 4. Graphical interface of the code generator tool.

elements of *Mental* and *Environmental* clusters and destination in elements of *Interactive cluster* or vice versa). This functionality is particularly useful since models of the *Interactive cluster* can frequently be reused with different models of the *Mental* and *Environmental* clusters. Moreover, their attached code is the most depending on the target platform. Once the integration is completed, the rest of the tool functionalities can be used taking the new integrated model specification as the current one.

Regarding the platform adaptation, the code generator tool presents two main functionalities that promote the specialisation of the design and a better integration between the model specification and the target traffic platform: a dynamic compiler insertion and new classes generation.

The dynamic insertion assists in the attachment of libraries provided by the target platform (i.e. as external files) to the code generator, making them available to the compiler. Once the library or libraries are selected, the process is internally managed by the tool making it transparent to users. It allows producing platform-related elements in the source code of a model specification class.

The new class generation functionality supports the creation of new classes extending the original ones coming from these external libraries. It uses a wizard that eases the selection among the available classes. These classes are inserted dynamically in the path of the compiler and can be used as the others. This allows the creation of new objects of these classes in the model specification classes or vice versa.

Finally, when both the code transformation and platform adaptation are completed, the final file is produced. It can be generated using two different approaches: a specific plug-in and a new adjusted platform, being both supported by wizards to make the process more intuitive. The specific plug-in approach builds a compressed file packaging the model

specification, the dependencies, and the classes generated that could be used or inserted into the target platform. The new adjusted platform approach integrates the entire target simulation platform and their dependencies (if they are needed), using external libraries, with the model specification and the classes modified and generated. The result is a runnable compressed file that contains the target platform. This platform is able to develop simulations considering the model specification inserted.

In both cases, configuration files can be created by another wizard. These files can be added to the compressed files in order to indicate parameters related to the simulation that must be considered.

The functionalities of these tools support our MDE approach for traffic simulation. They facilitate the process allowing the graphical examination of elements and the integration of multiple artefacts (e.g. model specifications or code snippets). Also, they encourage reusability and incremental development, reducing manual coding.

V. Case study

The case study shows the use of the MDE infrastructure (i.e. the TML and the development tools) in order to produce a road traffic simulation. It integrates a model specification (compliant with the TML) that adapts a theory of the domain with the specialisation to a target traffic simulation platform.

In this case, the selected traffic theory [12] describes a classification of potential risk factors for drivers, and how these factors can influence their behaviour. It is modelled with the proposed TML, and uses the resulting model for generating source code through a semi-automatic process. Specific code snippets and classes are inserted for generating an adaptation that can run a simulation using the MATSim platform [13].

The original classification of risk factors presents multiple aspects structured around two main concepts: *Individual differences* and *Situational factors*. This classification does not follow the DVE [7] approach required by the TML, but its adaptation seems feasible, as both cover common aspects. For instance, the *Vehicle size* factor in [12] can be represented through the *VComponent* meta-class, the *Age* factor with the *PComponent* meta-class, and the *Trip purpose* factor using the *KComponent* meta-class.

The first step to model the traffic theory is focused on elaborating a *modelling plan*. This describes an initial evaluation of the elements coming from the traffic theory to model that can match with the types of the metamodel. Once this task is completed, these elements are mapped to the selected types of the *Mental* and *Environmental* clusters (see Sections III-A and III-B) that fit properly with them. This planning produces a *starting schema* as a result. These guidelines are represented as a model using the graphical editor tool as follows.

Users start producing a simple structure that contains only the needed instances from the root meta-classes of the metamodel to represent the concepts of the theory (i.e. the starting schema). In this case, the *FPerson* class (an instance of the *Person* meta-class) is the root of the model design. The other classes are related to it using their appropriate references, e.g. connecting the *FEnvironment* class (an instance of the *Environment* meta-class) and the *FKnowledge* class (an instance of the *Knowledge* meta-class) to the *FPerson* class.

The previous structure is the basis to integrate the rest of the theory. After creating it, users add the elements of the TML that represent the other factors considered in [12], and link them with the relevant main elements following the *modelling plan*. The *FProfile* class acts as a root of its own tree substructure, being decomposed into two *PComponent* children. They represent *Individual Differences* and *Individual* factors from the theory. Each one of them is in turn decomposed into several children (e.g. *Individual Differences* into *Age* and *Gender*; and *Individual* into *Impairment* and *Hurry/Distraction*). The *FKnowledge* class is decomposed into two *KComponent* children (i.e. *Trip purpose* and *Length of drive*). The *FVehicle* class is decomposed into two *VComponent* children (i.e. *Size* and *Performance Characteristics*). Finally, the *FEnvironment* class is decomposed into four *EComponent* children (e.g. *Weather* and *Road condition*). This completes the adaptation of the original model to the TML (see Fig. 5).

Once the model specification based on the *Mental* and *Environmental* clusters is completed, the next step is designing the elements of the *Interactive cluster*. This can be done with the graphical editor tool adding the elements previously defined, or with the graphical editor integrated in the code generator tool. In order to show how models can be reused, the second option is chosen, developing a self-contained model specification.

A self-contained specification is an independent model that comprises only elements of the *Interactive cluster* and *GeneralRelationship* instances. These components can be developed according to the requirements of a particular simulation platform, promoting their specialisation. The resulting model can be merged with other models based on the *Mental* and the *Environmental* clusters, which facilitates the integration of multiple traffic theories with the target platform.

In this case, the self-contained model takes as basis the approach presented in [23]. It provides a *Goal* and *Task* tree structure with *AND* and *OR* compositions. That comes from agent-based methodologies and the BDI model [6]. This tree structure presents tasks associated with most of the goals. These tasks achieve the actions of the individuals following a set of instructions.

The root goal called *ArrivedFastDestination* represents the basic goal of individuals involved in traffic. It is decomposed into two sub-goals that must be fulfilled (i.e. *AND* composition): *Actuated* and *EndedRoute*. In turn, the *Actuated* goal is decomposed into a set of alternative goals (i.e. *OR* composition) that represents the different actions individuals can choose while they are interacting in road traffic (see Fig. 6). These goals are decomposed into the alternatives to achieve them (e.g. the *SearchedObstacle* goal is decomposed into the *SearchedOnLeft* or *SearchedForward* sub-goals). When any of these goals that represent actions is satisfied, the *Actuated* goal is satisfied too. Meanwhile, the end of the route is

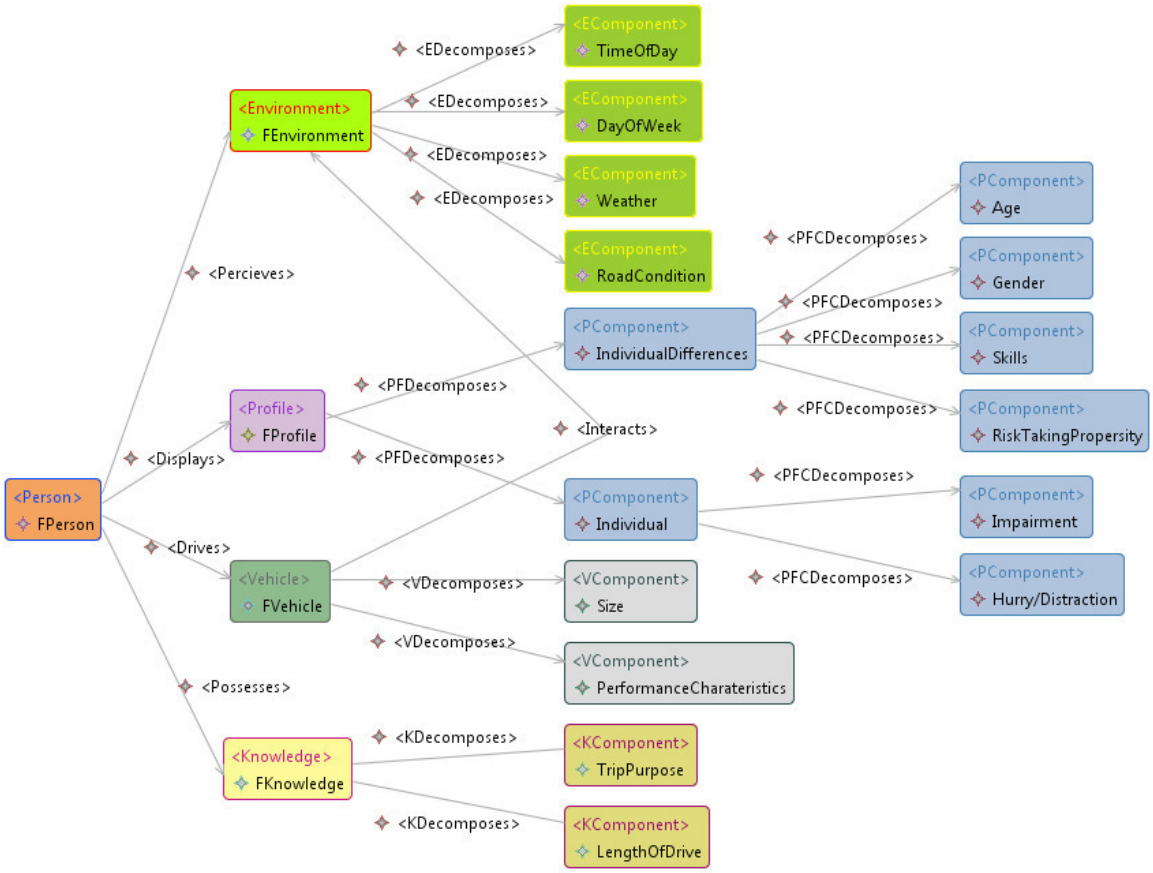

Fig. 5. Excerpt of the TML specification of the *Factors* model.

checked (*EndedRoute* goal satisfaction). If the *EndedRoute* goal is satisfied, then individuals have achieved their purpose, satisfying the *ArrivedFastDestination* root goal; if not, the process starts again. This sequence of actions models the processes in real-life, and assumes that at least one action must be done to reach the root goal.

In this self-contained model specification, *Goal* and *Task* instances present their own attributes and methods to manage these type of compositions. In *Goal* instances, the *GType* attribute indicates the type of goal composition (i.e. *AND* or *OR*). Code snippets are inserted into the body of the *calculateSatisfaction* method. These code snippets check if the sub-goal elements are satisfied. In *Task* instances, the *TType* attribute indicates the type of task composition (i.e. *AND* or *OR*), while code snippets complete the body of the *setInstructions* method. These code snippets validate if the associated sub-tasks and the atomic instructions are achieved successfully.

The *Interactive cluster* of the metamodel provides the means to model a *perceive-reason-act* cycle [11] of people. It uses the *Evaluator* and *Actuator* meta-classes based on [23].

In this case, *Evaluators* have a hierarchical decomposition that follows the goal tree. The root *Goal ArrivedFastDestination* is considered only by the *EvaluateDestination Evaluator* instance, the *EndedRoute Goal* is checked only by the *EvaluateRoute Evaluator* instance, and the *Actuated Goal* and the rest of *Goals* related to actions are controlled by the *EvaluateActions Evaluator* instance. This last *Evaluator* is in charge of selecting the best *Goal*. It considers the input parameters provided by both the *Mental* and the *Environmental* cluster, or the other two *Evaluator* instances. It also evaluates if the satisfaction of the selected *Goal* is produced in order to check if its parent *Goal* instances can be satisfied.

Once the structure in charge of evaluating the goals is completed, an *Actuator* instance is added to the self-contained model specification. It considers every task, executing its instructions when the appropriate evaluator selects its associated goal.

After that, this part of the model specification must be stored using the corresponding wizard of the code generator tool to generate a compressed file. In turn, the model based on the *Mental* and *Environmental* clusters generated in the graphical editor is loaded as input by the code generator tool. Then, navigating through the elements of the model specification, code snippets can be inserted into the *setXValues* body method of each one of the elements with the purpose of redefining the

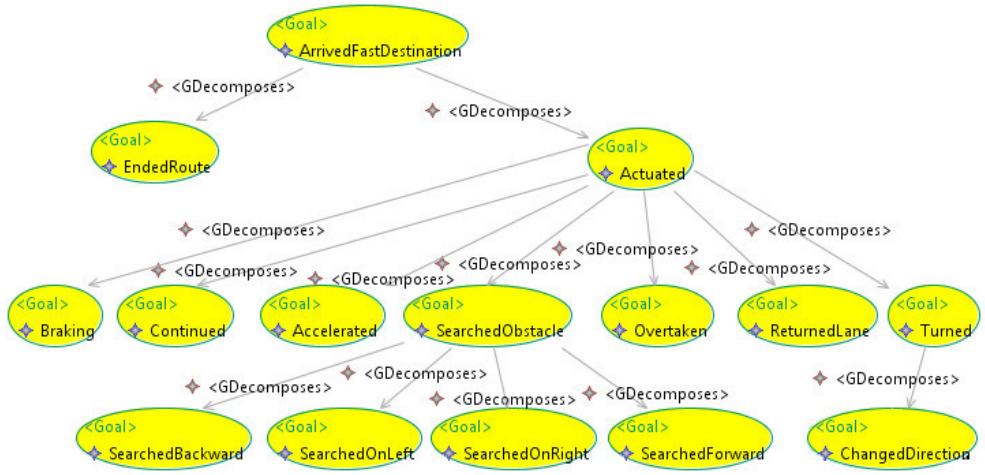

Fig. 6. Excerpt of the goal tree structure of the self-contained model specification.

calculation procedure of the *XValue* attributes (see Section III). In this case, the code snippet applies a formula based on Fuzzy logic [24]. The value of an element is obtained adding every value of the children and its own value and dividing this result by its number of children plus one, establishing a relationship among the children components and the parent. This step can be changed to consider other formulas and theories.

This model based on the *Mental* and *Environmental* clusters can be stored using the appropriate code generator functionality. This allows reusing the fuzzy formula and the structure generated in other projects.

As soon as the fuzzy logic is inserted into the model, the wizard in charge of the cluster integration functionality can be selected to apply it. The wizard links the *FPerson* instance with the root *Goal* instance of the self-contained model specification (i.e. *ArrivedFastDestination*) through a *Pursues* reference. Then, the *FPerson* instance is linked to the *Evaluator* root instance (i.e *EvaluateDestination*) and vice versa using the references *Harnesses* and *IsHarnessed* respectively. Finally, the *Actuator* instance is connected to the *FPerson* instance by means of *Utilizes* reference. When the process is completed and every reference is established, both clusters become a single model specification.

The same wizard supports the integration of *GeneralRelationship* instances among both clusters of the single model specification. These *GeneralRelationship* instances indicate the influence of target elements over the *Task* instances associated with each *Goal* instance. These *GeneralRelationships* are considered by the appropriate *Evaluator* instance in order to select the best candidate *Goal*. It evaluates the *Xvalues* attributes (previously configured using Fuzzy logic) of each related element of the *Mental* and *Environmental* clusters in order to generate real-time decisions.

Here, the addition of *GeneralRelationship* instances follows the factors structure. *Environment* factors (i.e. *RoadCondition* or *TimeOfDay*) directly affect *Overtake* or *Brake* instances,

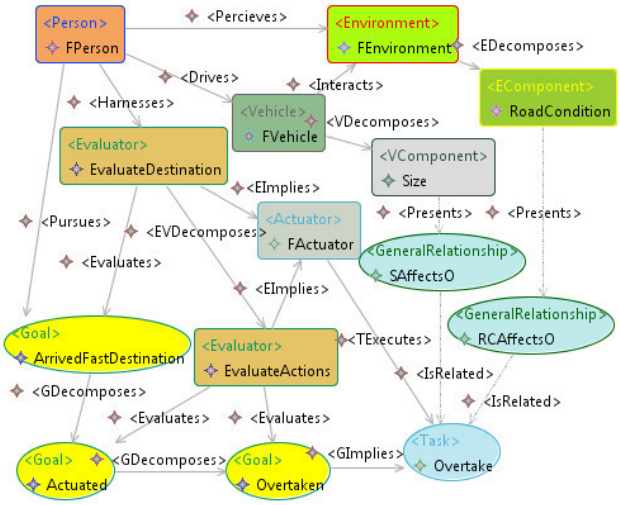

Fig. 7. Excerpt of the elements implicated in the overtaken interaction.

establishing multiple *GeneralRelationship* instances among them (see example in Fig.7). *IndividualDifferences* factors (i.e. *Age* or *RiskTakingPropensity*) affect *Accelerate* or *ReturnLane Task* instances. *Individual* factors (i.e. *Impairment* or *Hurry/Distraction*) affect *SearchObstacle* or *Accelerate Task* instances. *Vehicle* factors (i.e. *Size* or *PerformanceCharacteristics*) affect *Accelerate* or *Turn Task* instances. *Knowledge* factors (i.e. *TripPurpose* or *LengthOfDrive*) affect *Overtake* or *SearchObstacle Task* instances.

When the integration of these *GeneralRelationship* instances is completed, all these concepts and other possible evaluation criteria must be considered by the *Evaluator* instances. For this, the appropriate code snippets must be inserted into the body of the *evaluateGoals* method.

In order to illustrate how the code generator tool generates specific source code for a target platform, this case study considers MATSim. This agent platform presents different functionalities, but it only supports route configuration and optimisation using a path to follow (i.e. the interactions of individuals are not considered). It is made available to the code generator tool as an external compressed file, adding all its libraries and dependencies to the compiler.

The integration requires developing and adding a new class using the related wizard. This class is in charge of merging the model specification and the platform by establishing communications between them through programming procedures (i.e. the model specification is encapsulated being able to be integrated as a class in the MATSim source code).

Some classes of the MATSim platform need to be extended in order to consider the model specification structure. The original classes only plan the route, and now they must also consider, for instance, overtaking or lane changes. The new classes are integrated into the project, being considered by the compiler of the code generator.

The *Instructions* attribute of *Task* instances must be adapted and redefined to the way of functioning and the source code provided by MATSim. This allows generating the proper platform-specific actions to get the intended behaviour.

Once the specialisation is achieved, a configuration file is generated through the corresponding wizard of the code generator tool. The initial parameters of the *XValues* attributes are defined. These parameters can be modified with the purpose of obtaining different influences of elements. Also, another class is developed in the code generator and integrated into the path of its compiler in order to load this configuration file when the road traffic simulation starts.

Finally, the code generator tool produces a compressed file. It directly runs the MATSim platform with the embedded model specification generated and integrating the configuration file.

VI. Related work

Road traffic simulation is related to multiple areas of research. The presented approach considers the modelling of people's behaviour and environmental features affecting traffic, and the development process of simulations.

Existing road traffic simulation platforms are mainly based on multiple drivers that follow paths, though some of them allow random behaviours. The differentiation of their features, the decision-making and the interaction among them or with the environment are considered only in limited ways [13], [25].

Pedestrians and the influence of people around (e.g. passengers) over drivers are also important elements to evaluate, but frequently disregarded. For instance, [26] can model pedestrian interacting with the environment and drivers, but passengers and their possible impact are not contemplated.

The proposed metamodel considers the different roles of the individuals involved in traffic (i.e. drivers, pedestrians and passengers). It corresponds to microscopic models, as it models the multiple individual artefacts involved in road traffic (e.g. instances of *Person* and *Vehicle*). Although not fully considered now, mesoscopic models (i.e. those combining the individual and group levels), could also be integrated in the ML. The ABM approach adopted in the ML facilitates this extension, as it is frequently adopted for such kind of models [27]. The metamodel structure improves existing approaches in order to embody multiple social features. Most of approaches consider a fixed set of these features and their relationships, e.g. [9]. *Knowledge* instances are designed to be specialised and combined, providing instruments to add facts that affect groups of entities or the overall simulation.

Regarding the internal modelling of individual participants involved in traffic, there is not a widely accepted approach. Models range from simple, mainly reactive ones, to quite complex, usually deliberative. For instance, in [28] agents use simple logical rules to interact with the environment. This environment is mainly composed of crossroads where agents react to the behaviour of others. A more complex approach appears in [29], where driver's actions are decomposed into workflows considering the multiple situations that can occur during their execution.

The decisions achieved by agents in the previous approaches can be combined in hierarchical architectures, where there are several abstraction layers that organize acting. An example of this is the Michon's hierarchical control model for drivers [30]. The metamodel supports the hierarchical composition of most of its elements, but not the definition of abstraction layers as required by hierarchical architectures.

Another point of discussion in literature is related to which of the features of participants and the environment have influence on road traffic. Approaches such as [31], [32] review some of these features. The metamodel is intentionally open in this aspect. Meta-classes such as *Vehicle* and *Environment* in the *Environmental cluster*, and *Knowledge* and *Profile* in the *Mental cluster*, present sub-components to classify other related elements or characteristics. The metamodel also allows introducing additional concepts (i.e. through the *GeneralElement* meta-class) and relationships (i.e. through the *GeneralRelationship* meta-class), and extending them using the different inheritance hierarchies. These aspects entail that the TML is highly customisable for the multiple requirements of the road traffic domain.

Regarding the development process, most of reviewed works do not cite the approach they follow. Those that do it, in general assume common development processes focused on source code, where models play only a documentation and communication role. The advantages of MDE in this scenario have been already discussed in the related literature [33]: explicit representation of the information, higher involvement of experts, enhanced model validation, and reusability.

VII. Conclusions

This paper has presented a metamodel that defines a TML to support a MDE approach for the development of road traffic simulations. It defines this extensible ML focused on the behaviour of individuals. Development tools compliant

with the metamodel are provided to support the process. The adoption of MDE facilitates the exchange of information among groups of experts with different backgrounds. It also promotes the reutilisation of artefacts between projects, as there is a clear separation of concerns and all the information is explicit. For instance, this facilitates deployments in multiple platforms. It also encourages the incremental development, as models and transformations can be more easily modified than code.

The metamodel is designed with the purpose of being able to integrate multiple theoretical works from the domain. It follows an ABM [8] approach in order to consider the social interactions of the individuals involved in road traffic. It has three main clusters: a *Mental cluster*, an *Environmental cluster*, and an *Interactive cluster*. The first one includes the different psychological attributes that can influence the behaviour of individuals [4]. These concepts are classified into two groups: the features of people and their current state. It considers aspects of the mental state in the agent paradigm [5], particularly the BDI model [6]. The second cluster is based on the DVE approach [7] for modelling the different interactions of the individuals involved in road traffic, considering the relationships among vehicles, environment, and people. Many of the existing studies can fit their concepts into this structure, as it includes widely accepted notions to describe traffic settings. The last cluster uses concepts like goal and task. A *perceive-reason-act* cycle [11] is integrated through the evaluator and actuator concepts.

The meta-classes of the metamodel are designed to support internal hierarchical substructures, e.g. the container *Profile* meta-class and its sub-components with the *PComponent* meta-class. Inheritance between elements of the same type is introduced to make possible specialisations. Other types of relationships among elements are also considered.

Development tools are based on Eclipse facilities [17], [18]. The graphical editor provides a visual interface and a palette for describing the model specifications. These specifications can be validated to guarantee its compliance with the ML. The code generator provides a set of functionalities that guide users in the production of source code for a given target traffic simulation platform.

The case study exemplifies the use of the complete MDE infrastructure. The development tools support the design of a model specification according to a theory, its specialisation to the MATSim platform, and the generation of the source code associated. The MATSim platform presents a route optimisation feature to simulate individuals involved in traffic. This feature does not consider interactions among individuals and only generates a path to follow. Here, it is improved adding decision-making actions based on [23] through a goal-task hierarchical structure with *OR* and *AND* compositions. A tailored *perceive-reason-act* cycle [11] is integrated using the *Evaluator* and *Actuator* meta-classes. Also, the resulting model integrates a taxonomy related to the traffic domain based on the risk factors for drivers [12]. Individual actions are influenced by the related factors, producing different behaviours in individuals when those factors change.

The presented approach has several open issues. The TML must be tested with other types of road traffic theories (e.g. interactions among drivers and pedestrians) in order to check its primitives and structure. The development tools must also be used to generate source code specialisations for other traffic agent platforms (e.g. SUMO [25] or VISSIM [26]). The introduction of social norms, the influence of traffic signals (e.g. crossings or traffic lights) and the types of vehicles (e.g. ambulances or motorbikes) could be considered.

ACKNOWLEDGMENT

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, and 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.

REFERENCES

[1] A. Crooks, C. Castle, and M. Batty, "Key challenges in agent-based modelling for geo-spatial simulation," *Computers, Environment and Urban Systems*, vol. 32, no. 6, pp. 417–430, 2008.

[2] R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," in *2007 Future of Software Engineering*. IEEE Computer Society, 2007, pp. 37–54.

[3] A. Van Deursen, P. Klint, and J. Visser, "Domain-specific languages: An annotated bibliography." *Sigplan Notices*, vol. 35, no. 6, pp. 26–36, 2000.

[4] D. Shinar, *Psychology on the Road. The Human Factor in Traffic Safety*. John Wiley & Sons, 1978.

[5] Y. Shoham, "Agent-oriented programming," *Artificial Intelligence*, vol. 60, no. 1, pp. 51–92, 1993.

[6] A. S. Rao and M. P. Georgeff, "An abstract architecture for rational agents," in *Proceedings of Knowledge Representation and Reasoning (KR&R-92)*, vol. 92, 1992, pp. 439–449.

[7] A. Amditis, K. Pagle, S. Joshi, and E. Bekiaris, "Driver–vehicle–environment monitoring for on-board driver support systems: Lessons learned from design and implementation," *Applied Ergonomics*, vol. 41, no. 2, pp. 225–235, 2010.

[8] M. A. Janssen, "Agent-based modelling," *Modelling in Ecological Economics*, pp. 155–172, 2005.

[9] J. Pavón, J. J. Gómez-Sanz, and R. Fuentes, "The INGENIAS methodology and tools," *Agent-Oriented Methodologies*, vol. 9, pp. 236–276, 2005.

[10] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, "Tropos: An agent-oriented software development methodology," *Autonomous Agents and Multi-Agent Systems*, vol. 8, no. 3, pp. 203–236, 2004.

[11] J. Lind, "Issues in agent-oriented software engineering," in *Proceedings of the First International Workshop on Agent-Oriented Software Engineering (AOSE)*. Springer, 2001, pp. 45–58.

[12] T. A. Ranney, "Psychological factors that influence car-following and car-following model development," *Transportation Research Part F: Traffic Psychology and Behaviour*, vol. 2, no. 4, pp. 213–219, 1999.

[13] Transport Systems Planning and Transport Telematics Group, Transport Planning Group and Senozon Company, "MATSim, Multi-agent transport simulation," http://www.matsim.org/, 2015, [Online: accessed 08-May-2015].

[14] J. Bézivin, "Model driven engineering: An emerging technical space," in *Generative and Transformational Techniques in Software Engineering*. Springer, 2006, pp. 36–64.

[15] S. Kent, "Model driven engineering," in *Integrated Formal Methods*. Springer, 2002, pp. 286–298.

[16] Object Management Group, "Meta-Object Facility (MOF) Core Specification, Version 2.4.2," 2014.

[17] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro, *EMF: Eclipse Modeling Framework*. Pearson Education, 2008.
[18] D. Rubel, J. Wren, and E. Clayberg, *The Eclipse Graphical Editing Framework (GEF)*. Addison-Wesley Professional, 2011.
[19] Object Management Group, "Object Constraint Language (OCL), Version 2.4," http://www.omg.org/, 2014, [Online: accessed 07-May-2015].
[20] K. Czarnecki and S. Helsen, "Feature-based survey of model transformation approaches," *IBM Systems Journal*, vol. 45, no. 3, pp. 621–645, 2006.
[21] M. Wimmer and L. Burgueño, "Testing m2t/t2m transformations," in *Model-Driven Engineering Languages and Systems*. Springer, 2013, pp. 203–219.
[22] W. Van Der Hoek and M. Wooldridge, "Multi-agent systems," *Foundations of Artificial Intelligence*, vol. 3, pp. 887–928, 2008.
[23] A. Fernández-Isabel and R. Fuentes-Fernández, "An agent-based platform for traffic simulation," in *Soft Computing Models in Industrial and Environmental Applications, 6th International Conference SOCO 2011*. Springer, 2011, pp. 505–514.
[24] C. P. Pappis and E. H. Mamdani, "A fuzzy logic controller for a traffic junction," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 7, no. 10, pp. 707–717, 1977.
[25] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "Sumo-simulation of urban mobility-an overview," in *SIMUL 2011, The Third International Conference on Advances in System Simulation*, 2011, pp. 55–60.
[26] Visual Solutions, Incorporated, "VisSim, A graphical language for simulation and model-based embedded development," http://www.vissim.com, 2015, [Online: accessed 08-May-2015].
[27] M. Vasirani and S. Ossowski, "A market-inspired approach to reservation-based urban road traffic management," in *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*. International Foundation for Autonomous Agents and Multiagent Systems, 2009, pp. 617–624.
[28] A. Doniec, R. Mandiau, S. Piechowiak, and S. Espié, "A behavioral multi-agent model for road traffic simulation," *Engineering Applications of Artificial Intelligence*, vol. 21, no. 8, pp. 1443–1454, 2008.
[29] B. Burmeister, A. Haddadi, and G. Matylis, "Application of multi-agent systems in traffic and transportation," in *IEEE Transactions on Software Engineering*, vol. 144, no. 1. IET, 1997, pp. 51–60.
[30] J. A. Michon, "A critical view of driver behavior models: what do we know, what should we do?" in *Human Behavior and Traffic Safety*. Springer, 1985, pp. 485–524.
[31] H. Greenberg, "An analysis of traffic flow," *Operations Research*, vol. 7, no. 1, pp. 79–85, 1959.
[32] P. Paruchuri, A. R. Pullalarevu, and K. Karlapalem, "Multi agent simulation of unorganized traffic," in *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 1*. ACM, 2002, pp. 176–183.
[33] R. Fuentes-Fernández, S. Hassan, J. Pavón, J. M. Galán, and A. López-Paredes, "Metamodels for role-driven agent-based modelling," *Computational and Mathematical Organization Theory*, vol. 18, no. 1, pp. 91–112, 2012.