IEICE TRANS. INF. & SYST., VOL.E99–D, NO.2 FEBRUARY 2016

PAPER

An Integrative Modelling Language for Agent-Based Simulation of Traffic

Alberto FERNÁNDEZ-ISABEL[†a)], *Nonmember and* **Rubén FUENTES-FERNÁNDEZ**[†b)], *Member*

SUMMARY Traffic is a key aspect of everyday life. Its study, as it happens with other complex phenomena, has found in simulation a basic tool. However, the use of simulations faces important limitations. Building them requires considering different aspects of traffic (e.g. urbanism, car features, and individual drivers) with their specific theories, that must be integrated to provide a coherent model. There is also a variety of simulation platforms with different requirements. Many of these problems demand multidisciplinary teams, where the different backgrounds can hinder the communication and validation of simulations. The Model-Driven Engineering (MDE) of simulations has been proposed in other fields to address these issues. Such approaches develop graphical Modelling Languages (MLs) that researchers use to model their problems, and then semi-automatically generate simulations from those models. Working in this way promotes communication, platform independence, incremental development, and reutilisation. This paper presents the first steps for a MDE framework for traffic simulations. It introduces a tailored extensible ML for domain experts. The ML is focused on human actions, so it adopts an Agent-Based Modelling perspective. Regarding traffic aspects, it includes concepts commonly found in related literature following the Driver-Vehicle-Environment model. The language is also suitable to accommodate additional theories using its extension mechanisms. The approach is supported by an infrastructure developed using Eclipse MDE projects: the ML is specified with Ecore, and a model editor and a code generator tools are provided. A case study illustrates how to develop a simulation based on a driver's behaviour theory for a specific target platform using these elements.

key words: *traffic simulation, road behaviour, agent-based modelling, model-driven engineering, metamodel*

1. Introduction

Road traffic has a great influence in modern societies. Its study in real settings is difficult, given its scale, complexity, and potential impact on the wellbeing of people. For this reason, researchers resort frequently to simulations [1]. However, these also present limitations [2]. The literature points out the difficulties with discussing and aligning simulation models at different levels of abstraction (e.g. social theory and code design), for people with heterogeneous backgrounds, or implemented in different platforms. These issues make it hard to guarantee that the resulting simulation faithfully reflects the initial abstract model [3].

The use of model-driven approaches has been proposed to overcome those limitations [2]. Model-Driven Engineering (MDE) [4] organises development projects around *models*, which are compliant with well-defined Modelling Languages (MLs). For traffic simulation, this approach implies developing MLs to specify simulations from different perspectives (e.g. driver's behaviour, car functioning, and environment) and at different levels of abstraction (e.g. those of traffic experts and developers). From the definition of MLs, developers provide graphical editors for their specifications and tools to transform them. *Transformations* are used to generate semi-automatically from models most of the other required artefacts, e.g. code or documentation. In this way, the process of developing simulations becomes iterative, by refining abstract models to others more specific, until generating source code from them is possible.

The initial effort to develop this infrastructure is higher than just manually developing a simulation. However, it pays off with reutilisation across simulations and the explicit description of all the information used to develop them [2]. MLs and transformations can be incrementally defined, tested, and reused across different projects. For instance, a ML for urban environments is applicable to different cities. As these elements include all the information necessary to specify the simulation and its refinement to code, they facilitate the traceability of artefacts for analysis and verification.

There have been some partial efforts regarding MDE for traffic simulations. Several integrative models have tried to provide the basis to build incrementally models of traffic (e.g. [5], [6]). However, they usually detail through text some aspects or rely on their simulation platforms to describe the semantics. These descriptions are not suitable for defining a ML for MDE. Their translation to (semi-)formal languages is not evident, and/or they mix domain and computational issues. There are also works regarding development processes, but are general discussions. They do not consider the activities or infrastructure required (e.g. [7]), or the specific features of traffic simulations (e.g. their scale or how to deal with highly dynamic components) (e.g. [2]).

Our research is focused on the development of a complete MDE approach for traffic simulations. It pursues being suitable for studies at both the micro (i.e. individual) and macro (i.e. group) levels, considering different theories and analysis focuses, and platform-independence. This paper introduces the *Traffic ML* (TML), that is its core, and more briefly the development guidelines and tools (a model editor and a code generator).

The TML adopts Agent-Based Modelling (ABM) [8]

Manuscript received April 22, 2015.
Manuscript revised September 8, 2015.
Manuscript publicized October 27, 2015.

[†]The authors are with GRASIA (Research Group on Agent-based, Social & Interdisciplinary Applications), University Complutense of Madrid, Madrid, Spain.
a) E-mail: afernandezisabel@ucm.es
b) E-mail: ruben@fdi.ucm.es

DOI: 10.1587/transinf.2015EDP7156

and follows the Driver-Vehicle-Environment (DVE) model [9]. Its agents are intentional entities that interact among them and with the environment where they are situated. ABM has shown to be useful to deal simultaneously with individual and group aspects for non-linear systems, and facilitates the transition from abstract models to code. The DVE model assumes that drivers use their vehicles to establish relationships with the environment. The dynamic interactions among elements influence their individual behaviours.

The conceptual framework, along with inheritance and composition mechanisms, facilitates the specification of other theories with the TML. These new models are used as the basis to specify simulations with those theories.

A case study illustrates the use of the framework. It specifies and integrates the theoretical *Drivability* Model (DM) [10]. DM considers the influence of driver and environment features on driving capabilities. The Simulation of Urban MObility (SUMO) platform [11] is used as the target. It has a model for vehicle control based on paths to follow. The case study shows how the TML supports describing the theoretical model, and this specification facilitates further modifications (adding the platform model) and reduces the coding effort to develop the simulation.

The rest of the paper is organised as follows. Section 2 introduces the basic concepts of MDE and related tools. Then, Sect. 3 discusses the TML, and Sect. 4 briefly presents the development guidelines and tools. The case study in Sect. 5 applies these elements to specify a simulation and generate its code. It guides the comparison with related work in Sect. 6. Finally, Sect. 7 discusses some conclusions and future work.

2. Model-Driven Engineering

MDE [4] is an approach for system development structured around *models*, as opposed to traditional approaches that can be considered as *code*-centric. Its processes are mainly organised as an iterative and incremental specification of models, where developers refine and add new information to them at each step. Along this process, certain repetitive changes are automated with *transformations*. For instance, adding patterns or platform specific information to design models. Work with other artefacts (e.g. code, tests, and documentation), follows a similar path, as developers usually generate them from models using transformations and manually adjusting.

This kind of development needs means to define formally MLs in order to facilitate that tools process them and their models. *Metamodels* are currently the main mechanism for this purpose when dealing with graph-oriented languages, which are the most popular ones [12]. They commonly specify the abstract syntax of MLs, though they can also be used for their concrete syntax and semantics. Metamodels are in turn defined using meta-modelling languages [12], such as the Meta-Object Facility (MOF) and Ecore. Ecore [13] is used in multiple Eclipse MDE projects organised around the Eclipse Modelling Framework (EMF) [13]. Moreover, it is almost aligned with Essential MOF (EMOF), a subset of MOF [13]. Our work uses these Eclipse projects, so the TML is specified using Ecore.

In an Ecore metamodel, an instance of *EClass* represents a set of similar entities (its instances at the model level). It can group *EAttribute* and *EReference* elements. *EAttribute* instances are used to associate properties of *EDataTypes* (i.e. primitive types) to *EClass* instances. These primitive types include, for instance, integer, float, Boolean, char, and string. An *EReference* represents a binary and directed relationship between two *EClass* instances. These references can represent containment and non-containment relationships. The *EReferenceType* of a given *EReference* instance is determined by its target *EClass*. Two *EClass* instances can also be linked by an extension relationship (i.e. *ESuperType*), which represents inheritance. Ecore also incorporates the concept of *EPackage* to group elements of the metamodel. For a diagram, there must be an instance of *EClass* that contains all its elements.

Transformations are the other key component of MDE. According to their input and output, they are classified as: Model-to-Model (M2M), Model-to-Text (M2T), and Text-to-Model (T2M). These transformations are implemented in different ways, including the use of general-purpose programming languages and transformation languages. In the first case, the transformation becomes a module that uses programming interfaces to manipulate its inputs and outputs. In the second case, the transformation is written in a specific language for transformations and an engine executes it. The first approach allows reusing already available experience and tools from mainstream development approaches (e.g. object-oriented programming and XML processing), and fine-tuning the execution of the transformation. The second approach facilitates understanding and examining the mapping between inputs and outputs. Our work adopts the first one, as it makes use of techniques from object-oriented reflection-based programming.

3. Traffic Modelling Language

Road traffic is a complex phenomenon that involves large numbers of interacting components with heterogeneous features and structures. There are not general and agreed theories for its different aspects, so researchers apply different ones according to their own background and the goals of each study. There are also multiple simulation platforms with different modelling approaches. The analysis objectives, theories, and infrastructure constitute here the *modelling setting* of a problem.

A ML suitable to represent the previous information should be designed having in mind to ease the problem comprehension and adaptation to new modelling settings. To achieve these goals, the metamodel that specifies the TML is organised to a large extent through inheritance and composition hierarchies and clustering of concepts. Its conceptual framework is based on ABM [8] and the DVE model [9].

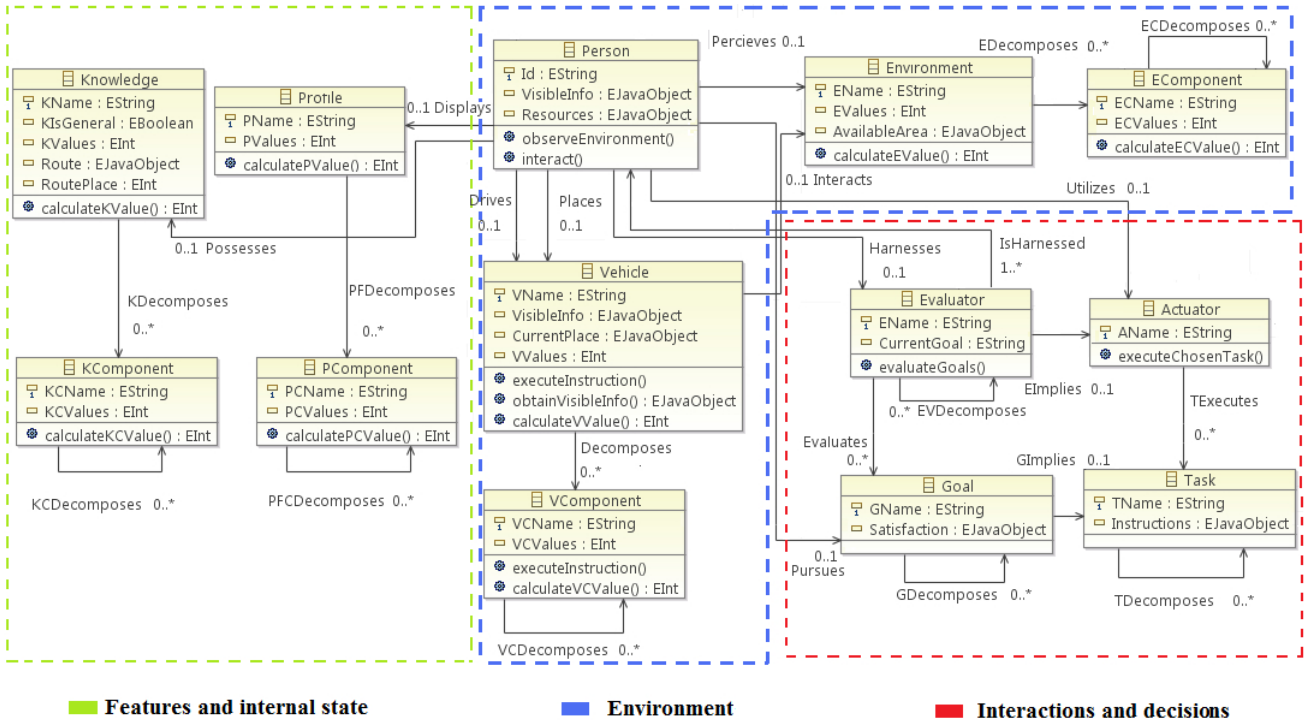

Fig. 1 Main primitives of the traffic metamodel.

Figure 1 shows part of its specification with Ecore [13].

Inheritance hierarchies arrange concepts using generalisation relationships. They support the incremental definition of concepts, and facilitate the modification of the metamodel. All concepts inherit from the *GeneralElement* meta-class (not shown in the figure). It introduces the *EInherits* reference to represent inheritance between concept instances in models. In the same way, the *GeneralRelationship* meta-class supports the introduction of arbitrary directed relationships with attributes between concepts, and its specialisation using the *RInherits* reference.

Some concepts (e.g. vehicle or environment) may have complex structures. In order to represent them, the metamodel decouples *container* concepts from their *components* using a *composition pattern*. The container meta-classes are the roots of composition hierarchies of component meta-classes. These components can be structured in turn in components of the same hierarchy. An example of this pattern is the decomposition of the meta-class *Vehicle* in the meta-class *VComponent* (see Fig. 1). This organisation supports at the model level arbitrary whole-part relationships. In them, the child components of a given one represent features/parts depending on its root component.

Clusters are specific parts of the metamodel. They group components that share features, e.g. similar semantics, purpose, or container. There are three (see Fig. 1): *features and internal state*, *environment*, and *interactions and decisions*. They highlight the commonalities of their elements, and clarify their relationships with other clusters.

Besides clusters, an instance of the *TrafficModel* meta-class must always gather all the components in a specification (not shown in the figure). Ecore requires the existence of a root element that groups all the others in a diagram.

The previous mechanisms account for the static description of simulations. However, some features and behaviours in traffic are inherently dynamic, e.g. the driver's workload or the choice of goals according to the situation. Moreover, in the detailed specifications, experts need means to describe how the mutual influences among components through relationships affect them. The metamodel considers *methods* as placeholders for these indications about dynamic calculations. For example, they can have attached at the model level code snippets to specify algorithms.

In order to provide guidance in modelling, the TML also offers a conceptual framework. It helps to identify the relevant elements in a problem. Following the DVE model, the core concepts here are *Persons* (which are agents) interacting with an *Environment* that includes *Vehicles*. Next sub-sections discuss them in detail following the clusters.

3.1 Environment

Traffic occurs in an environment that sets certain physical conditions (e.g. weather, lane width, or signals in a street). The metamodel represents it with the *Environment* meta-class and its components. A given model includes a unique instance of *Environment* perceived by all participants.

People perceive and act on the environment in different ways depending on the artefacts they use in the interaction. The metamodel accounts for these differences including the *Vehicle* meta-class and its components to represent means of transport. Drivers and passengers relate to the external *Envi-*

ronment through their instances of *Vehicle*, but only drivers can use them to act on that environment. The driver and passengers of a vehicle relate among them directly using the instance of *Environment*. Pedestrians have a direct relationship with the environment.

With this representation, all the elements in traffic external to people are represented as instances of components of the environment (i.e. *EComponent*) or vehicles (i.e. *VComponent*). Different subtypes of *GeneralRelationship* with specific meanings can be introduced to indicate how these components are related. For example, a navigation system is modelled as an instance of *VComponent*, and a relationship *Perceives* introduced to link the navigator and the person who uses it. An inter-vehicle communication network could be an instance of *EComponent* that links instances of *Vehicle*, and where these vehicles produce / use messages (also instances of *EComponent*) using communication devices (new instances of *VComponent*), all of them linked by new types of relationships.

3.2 People Features and Internal State

This cluster characterises the state and features of a *Person* using the *Knowledge* and *Profile* meta-classes. The first one represents any kind of mental entity (but the objectives) that people use when dealing with traffic. It includes factual (e.g. speed limits, routes, or actual positions of vehicles), procedural (e.g. how to perform a lane change or cross a street with traffic lights), and moral and normative (e.g. drivers should not exceed speed limits) knowledge. The second one represents people features (e.g. gender, age, or fatigue). Both of them can be decomposed into components.

The instances of the *Knowledge* meta-class can represent information pertaining only to an individual or global to all participants. For example, laws about traffic are global, but some participants in the simulation can ignore them. The *KIsGeneral* attribute differentiates both uses. When a *Knowledge* instance belongs to a *Person* instance, a *Possesses* reference instance must also link them.

3.3 Interactions and Decisions

The metamodel describes people (i.e. instances of *Person*) in the environment (i.e. instances of *Environment* and *Vehicle*). These *Persons* are agents that behave according to a perceive-reason-act cycle [14]. Perception studies the environment (concepts in Sect. 3.1) to change the agent's features and mental state (concepts in Sect. 3.2). Reasoning processes these features and state to generate additional information, and decide what should be done and with which actions. Acting carries out actions that change the agent and environment states. Agents repeat this cycle over time. The components that implement the cycle are organised in two groups: one to define what a person wants and is potentially able to do; another to represent what s/he actually does.

The first group includes the *Goal* and *Task* meta-classes. A *Goal* represents a state of the world the agent wants to keep or achieve. A *Task* represents the actions the agent is able to perform to try to change its own or the environment state. Both meta-classes include specific attributes to characterise them, goals with their satisfaction conditions (i.e. *Satisfaction*) and tasks with the atomic actions that implement them (i.e. *Instructions*). Goals are linked to those tasks which execution is potentially able to satisfy them.

Both *Goal* and *Task* instances can be decomposed into instances of their same meta-classes, as it occurs with other *components* of the metamodel. However, the semantics are different in this case. While previous decompose relationships indicate a whole-part link, here they are related to satisfaction. When a goal is decomposed into child goals, the satisfaction of any of them implies the satisfaction of the parent goal. In the case of tasks, the completed execution of all child tasks implies finishing the parent task.

The second group has the components that model the execution of the agent's behaviour cycle. *Evaluator* instances carry out the perceive and reason parts, and *Actuator* instances execute the selected tasks. These instances only work with those elements linked to their instance of *Person*. For example, an instance of *Person* can be linked to an instance of *Vehicle* with an instance of the *EComponent* navigator. An instance of *Evaluator* for that agent is able to navigate a *Perceives* relationship to get the route information from the navigator, and assert it as an instance of *KComponent* in the mental state of the agent. In order to represent the different processes of the cycle, *Evaluator* instances can be composed of others using the *EVDecomposes* reference. This facilitates making up evaluators through composition.

4. Development Guidelines and Tools

The MDE framework for traffic simulations includes guidelines to apply the TML using several support tools. Among them, there are a graphical editor for models and a code generator. Both are implemented using EMF [13], the Graphical Editing Framework (GEF) [15], and the Graphical Modelling Project (GMP)[†]. They use tailored XML files to store information.

The editor allows specifying models compliant with the metamodel. It is a plug-in generated using GMP from the traffic metamodel (see Sect. 3). The tool provides a visual interface with a canvas to describe models using the elements available from palettes. These palettes are populated with the concepts and relationships from the metamodel. Additional constraints are introduced using the Object Constraint Language (OCL) [16]. For instance, they avoid that a component can have itself as a part, and force some compatibility rules for inheritance, e.g. a class cannot be a subclass of instances of *Person* and *Vehicle* simultaneously using *EInherits* instances.

The code generator is the tool to work with M2T transformations. It is a tailored tool implemented in Java. Its input is the traffic metamodel, a model compliant with it,

[†]http://eclipse.org/modeling/gmp/

and the source code EMF generates for the metamodel metaclasses. It also allows importing external files (e.g. libraries to access the target simulation platform) and elements of a finer grain (e.g. the code to specify algorithms). It outputs the code of the simulation. Its tasks are organised around graphical wizards. Among the supported functionalities are: the selection of entities in models for code generation; the injection of code templates for them; a limited form of model integration, by adding instances of already defined relationship types to link models; and the generation of configuration files for simulations. The tool also supports the integration of tailored wizards. Code templates use a specific markup language to refer to the model elements (e.g. entities, attributes, and relationships).

In order to guide developers in the implementation of wizards and the specification of code templates, the generator includes samples. For instance, *calculateValue* methods (see Sect. 3) have a default implementation using fuzzy logic [17]. This is a quite intuitive way to account for the influences among concepts across relationships in models. Anyway, users are expected to be able to reuse from other projects most of the code needed for their models.

With these tools, the proposed development process includes the following activities:

1. *Specify the theory model*. This activity uses the model editor to create a TML model of the theories to consider. It may require adapting these theories to its conceptual framework. The definition of concepts in the TML guides this specification. The *theory model* provides the types to use in the simulation. This activity is usually decomposed into three:
    a. *Specify the context*. This activity provides the context of agents' behaviour. It uses the *features and internal state* and *environment* clusters.
    b. *Specify acting*. It defines the instances of elements from the *interactions and decisions* cluster required to define the agents' cycle.
    c. *Specify dynamics*. The previous activities just consider the static perspective of the simulation. Hints on dynamics are provided in methods. If the markup language for templates is used, the generator can directly interpret it when producing code.
2. *Specify the simulation model*. This activity refines the *theory model* with specialisations for the simulation. Its sub-activities are similar to those of activity 1.
3. *Map the simulation model to code templates* . The code generator provides implementations for the elements of the TML using the default EMF code, and these can be used for the *simulation model*. When other implementations are required, experts need to link code templates to the related model elements. These templates can be reused from other projects or developed from scratch.
4. *Generate instances*. A configuration wizard of the generator reads the code for the simulation types. Then, it allows the user graphically specifying instances of those types, with their identification and initial attribute values. This wizard is platform-dependent.
5. *Code generation*. The generator uses al the previous information to generate the source code and configuration files for the simulation as the final step.

A detailed example of the application of this process with some variations can be found in [18]. It considers the simulation of Intelligent Transportation Systems (ITSs).

The process and the functionalities of the tools effectively support the MDE approach for traffic simulations. They facilitate the examination and modification of the different artefacts (i.e. models, code templates, and mappings) involved. They also improve reuse, even across projects, as correspondences among artefacts are explicit and users can modify them. Finally, they reduce the need of manual coding, as relevant parts of code are shared by multiple elements or imported from other projects.

5. Case Study

The case study illustrates the use of the TML to specify a simulation that integrates two works, and generate its code. The first work is the DM [10], which is focused on the influence of the person and environment features on driving capabilities. Then, the simulation selects as its target platform SUMO [11], which has a model for vehicle control based on path following. Diagrams in this section are snapshots from the developed model editor.

The DM [10] describes the dynamic changes happening in the capacity level of drivers, i.e. their *drivability*. These changes depend on their own features and environmental factors. Drivers' features are classified into three main groups: *knowledge and skills*, *risk awareness*, and *individual resources*. Under the category *environmental factors* appear aspects such as *vehicle type/status*, *traffic hazards*, or *weather conditions*. The person's *workload* during the driving activity is also considered. All these features change over time, frequently because of the influence of other features. Their actual value also depends on their initial value, except for the *workload*, which is completely calculated from other features.

Figure 2 shows an excerpt of the specification of the DM using the TML. Only the concepts relevant for the case study appear. This will be the result of activity 1.

The first task to build the *theory model* of a work is classifying its concepts into the main categories of the TML. In the case of the DM, it is mainly related to activity 1a. If this classification was not possible, the metamodel should be reviewed to add the required concepts and relationships.

The DM classifies its concepts into categories similar to those of the TML, though there are some exceptions. The category *knowledge and skills* includes elements that are instances of the *Knowledge* and *Profile* TML meta-classes. For instance, *generic*, *driving experience*, and *driver training* are forms of the TML *Knowledge*; on the other side, *self-awareness* is part of the TML *Profile*. In the category *risk*

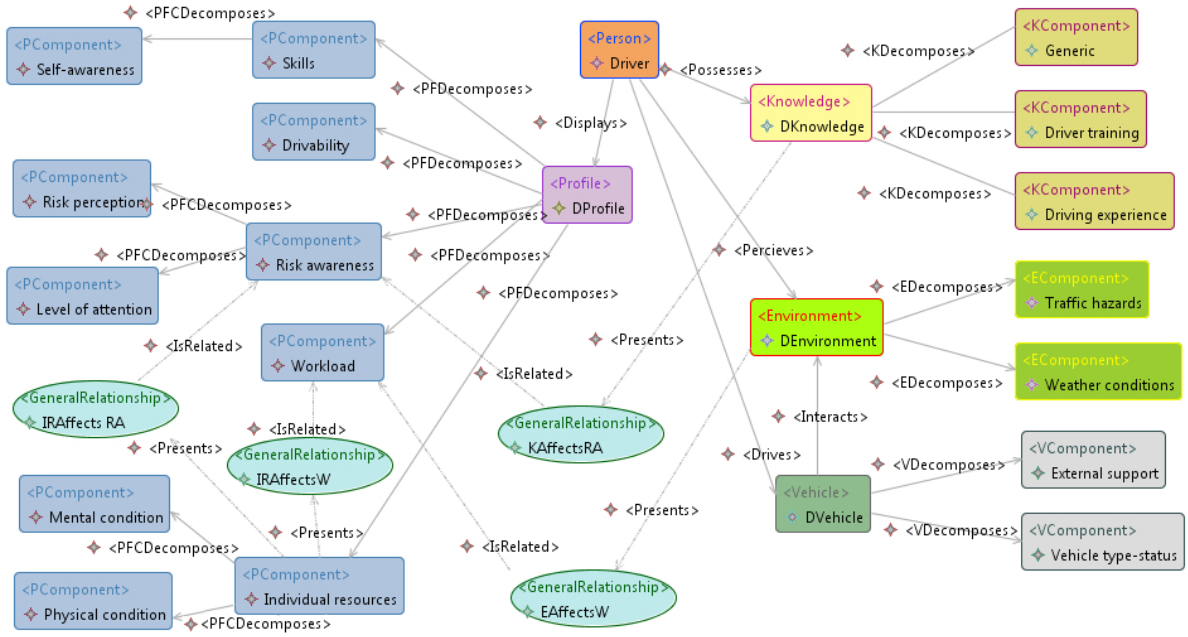

Fig. 2 Excerpt of the TML specification of the *Drivability* model.

awareness, all the concepts are part of the profile, except the *external support*. It represents assistance systems (e.g. a navigator or inter-vehicle communication device), that are part of the vehicle or the environment. The concepts of the category *individual resources* and the *workload* are also part of the profile. For the category *environmental factors*, most concepts are part of the TML *Environment*. Only the *vehicle type-status* is a component of the TML *Vehicle*.

In order to keep as far as possible the original classification of DM concepts, the specification introduces four root elements: the *DKnowledge*, *DProfile*, *DVehicle*, and *DEnvironment* instances of the corresponding root types. These group the remaining concepts.

Concepts from the DM category *knowledge and skills* that are knowledge become instances of the TML *KComponent*. For the other person-related concepts that are part of the TML *Profile*, the specification introduces new instances of *PComponent* that represent their original categories, e.g. *Workload* and *Risk awareness*. The actual DM concepts are represented as instances of *PComponent* grouped under the previous ones using composition relationships.

The category *environmental factors* is modelled with the *DEnvironment* instance of the TML *Environment*. Its concepts become *EComponent* instances linked to it with the exception of the *vehicle type-status*.

The *DVehicle* groups two concepts from previous categories related to vehicles. The *external support* from *risk-awareness* and the *vehicle type-status* from *environmental factors*, which become *VComponent* instances.

The previous concepts must be linked using relations. Inheritance and composition are part of the TML, but others have to be introduced as instances of *GeneralRelationship*.

The DM considers several relationships of mutual influence between concepts. For instance, *individual resources* and *environmental factors* affect the *workload*, and *knowledge* to *risk awareness*. These influences are represented with the new *Affects* instances of *GeneralRelationship*. They indicate that a source concept (i.e. *Presents* role) has influence on the value or behaviour of a target concept (i.e. *IsRelated* role). There is a different instance for each pair of source and target concepts. As they represent types of relationship, if there was only one instance of *Affects*, it would indicate that every source concept can be related to every target concept. This is not the intended semantics.

Some of the previous concepts could alternatively be modelled as attributes of TML classes, for example, the *vehicle type-status* or the *workload*. However, such alternative does not allow representing complex structures of parts (which could be necessary for the first example) or relationships with other concepts (which is required for the second).

The DM only accounts for factors that influence driving capabilities. It does not indicate how the actual decision-making and acting is. This case study considers for that purpose the control of vehicles in the SUMO platform. This information corresponds to activity 2, with those sub-activities corresponding to activities 1b and 1c.

The behaviour of a SUMO vehicle is characterised in terms of a path to follow and parameters that indicate how to execute it. SUMO uses a map of *edges* with *lanes* that connect *junctions*. The path (i.e. *route*) the vehicle follows is a sequence of *edges*. Among the parameters affecting driving appear the *impatience* (i.e. the willingness to progress even if that hinders other vehicle movements) and the *speedFactor* (i.e. the multiplier for speed limits that gives the maximum acceptable speed according to the driver). To model these aspects, the *simulation model* (see Fig. 3) adds several new elements to the previous *theory model* (see Fig. 2).

Both *speedFactor* and *impatience* are part of the

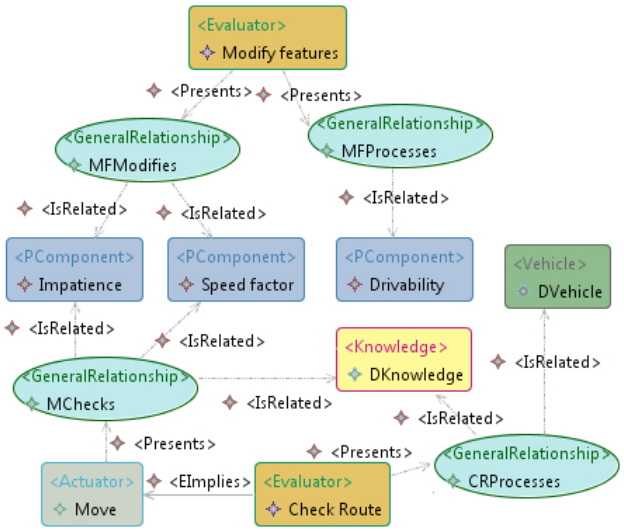

Fig. 3 Excerpt of the TML specification for the SUMO platform.

driver's profile, as they represent how somebody behaves driving. Thus, they are modelled as instances of *PComponent*. The model has also to indicate that the *drivability* feature affects them. The related *Affects* instances of *GeneralRelationhip* are also introduced (not shown in the figure).

In order to monitor the vehicle movement, several elements from the metamodel (see Fig. 1) are needed. The *DVehicle* instance puts in use the *CurrentPlace* attribute that characterise its location in the environment. The *Route* attribute of the driver's *Knowledge* is used to specify the SUMO path her/his vehicle follows.

The actual processing of the previous information is modelled with instances of *Evaluator* and *Actuator*. The *Affects* relationships are considered with an instance *Modify features* of *Evaluator* that reads the *Drivability* and sets the values of *Speed factor* and *Impatience*. Another evaluator *Check route* checks the position of the *DVehicle* in the *Route* to establish if an instance *Arrive destination* of *Goal* (not shown in the figure) is fulfilled. If not, an instance *Move* of *Actuator* reads the information from the *Route*, *Impatience*, and *Speed factor* to generate the next movement.

When the specification is ready, the code generator allows working with the source code. In this case, the code generation needs to output the XML files that define the simulation in SUMO.

The first step is activity 3, which maps suitable code templates to model elements. Most of concepts in the *specification model* uses a default template that corresponds to a class that includes all their attributes and methods to get and set them. Concepts that are target of *Affects* relationships use a template similar to the previous one, but it adds a *calculate* method for each attribute in the concept. Its implementation uses a default fuzzy logic implementation [17] provided in the metamodel. After code generation in activity 5, programmers need to adjust these fuzzy implementations according to the expected thresholds. The *DVehicle* does not need to update its position, as SUMO does it automatically. The *Modify features* evaluator just needs a template with the algorithm for the methods that calculate from the *Drivability* input the *Impatience* and *Speed factor* outputs. The *Check route* evaluator is empty, as the platform does its test. Finally, the *Move* actuator needs a specific template for the SUMO platform. The template provides the implementation of a method that generates an XML fragment for the platform with the specification of the vehicle and its type and route. In this case, the only possible task is moving until arriving at the destination.

Here, activity 5 does not produce directly the simulation code. The generator outputs code that is used to generate the XML file that is the simulation specification for SUMO. After running the generator, experts launch the *Move* actuator for every driver type, and generate XML fragments. The SUMO specification is the concatenation of these fragments.

With this approach, researchers can follow a MDE process to develop traffic simulations. They worked with models most of time. The TML could be extended to support the *Drivability* and SUMO models. Only when using the code generator researchers needed to consider code as templates. Many templates can be reused for several concepts. In the case study, only minor adjustments were needed for reused templates. There were a few platform specific templates that have to be implemented from scratch, in particular that for the actuator. They could be reused for other simulations with SUMO. The model editor provided basic functionality to build the models. The code generator offered a richer support, particularly to explore models and templates, and make mappings. Template examples were also very helpful.

6. Related Work

The simulation of road traffic integrates multiple areas of research. For this work, the most relevant ones are modelling approaches, the study of people and environment features and their mutual influences, and development processes.

Simulations can describe phenomena at different granularity levels. Microscopic models focus on the behaviour of individual components, macroscopic ones on the dynamics of the overall system (or groups in it), and mesoscopic models combine features of the others. Every approach has its own trade-offs [19]. Microscopic models support the detailed analysis of individuals and their influence in the behaviour of the complete system, but strongly limit the size of populations because of the required computational resources. Macroscopic simulations can consider larger populations, but mostly disregard individuals. Mesoscopic models try to balance the features of the previous approaches, but it is difficult to determine what is better to model at each level. Traffic simulations present examples of all of them: [20], which uses fluid dynamics to model traffic streams, is a macroscopic model; [21] introduces a behavioural model for vehicles, so it is microscopic; and [6] is a mesosocopic example with microscopic components to control individual vehicles and macroscopic elements for general traffic flows.

In this classification, the TML is a microscopic model. Its entities represent individual participants in traffic (e.g. instances of *Person* and *Vehicle*). Nevertheless, ABM is also suitable for mesoscopic models [19]. The design of the metamodel can take advantage of existing approaches to integrate social aspects. In fact, *Knowledge* instances can already be attached to the overall model, providing means to introduce information that affects groups of entities.

Regarding the internal modelling of individual participants in traffic, there are not widely accepted approaches. Models range from reactive (e.g. cellular automata in [22]) to deliberative (e.g. rules for manoeuvres in [21]). These approaches can be combined in hierarchical architectures, where there are several decision and/or acting abstraction layers. An example of them is the Michon's hierarchical control model for drivers [23].

The proposed metamodel is suitable for both reactive and deliberative agents. More deliberative individuals correspond to agents that consider instances of the *Knowledge* and *Goal* meta-classes. As these elements are optional, they can be disregarded for reactive individuals. In both cases, the way of making decisions and acting is encapsulated in instances of the *Evaluator* and *Actuator* meta-classes. Their adaptation to a given internal model will require specific instances to accommodate the relevant information.

Currently, the metamodel supports the hierarchical composition of most of its elements, but not the definition of abstraction layers. Its concepts are organised in clusters, and the composition pattern only represents whole-part relationships among instances of the same meta-class.

Another issue is which features of traffic elements need to be considered. Works like [10], [17], [20], [24] review literature on the topic. Most of the proposed features can be accommodated into the categories of the DVE [9] and ABM [8] approaches that are the basis of the TML. Moreover, the metamodel is intentionally open in this aspect. Meta-classes such as *Knowledge*, *Goal*, and *Environment* are containers to classify other elements. They can be detailed with additional constituents (i.e. through *compose* relationships) or extended (i.e. inheritance). This supports including new primitive features in models, as done in the case study.

The development process adopted by most of the previous works, when it is considered, is based on manual coding. Sometimes, libraries and wizards support the process. The advantages of adopting MDE approaches in this context were already pointed out in the introduction [2]: explicit representation of all the information involved, traceability of artefacts, improved reusability, and higher involvement of experts. Some works have already contributed in this line.

Existing works can provide artefacts for MDE developments. Models (e.g. [5], [6], [24]) contribute with conceptual frameworks and parts of metamodels to define new MLs. For instance, the case study incorporates concepts from [10] at the model level, but they could be promoted to the metamodel in a similar way if needed. Simulations (e.g. [5], [6], [11]) help to understand the semantics of their theoretical models. They can also be the source of code fragments to reuse in code generation.

There are few examples of complete MDE approaches in this field. Two of them are [7], [25]. The first one [7] only offers a general discussion on the models considered at different levels of abstraction. The second [25] applies the Computer Automated Multi-Paradigm Modelling. It defines a metamodel for traffic environments and vehicles (without considering their components), specifies its semantics with Petri Nets, and proposes using graph-rewriting rules for transformations. The main drawback of these approaches is the difficulties to extend them (e.g. adapting the metamodel or integrating new tools), as they do not use formalisms and standards widely adopted in the MDE community.

7. Conclusions

This paper has presented the TML that is the basis of a MDE framework for traffic simulations. It also explained more briefly the development guidelines and tools to use it. The MDE approach facilitates the specification and exchange of knowledge among experts, combining different theories and simulation platforms, and reusing artefacts.

The TML is designed to provide a basis for integrating different perspectives on traffic. It includes the key concepts of traffic following the DVE model [9], and a general acting model from ABM [8], [14]. Most of existing works can accommodate their concepts into its categories. The organisation of its metamodel is based on clustering and inheritance and composition hierarchies. It also allows specifying arbitrary interconnected structures among concepts. This design supports introducing the concepts, relationships, and properties needed to deal with new modelling needs.

The use of the TML is based on a graphical model editor and a code generator. They are developed using Eclipse projects [13], [15].

The case study illustrated the use of this infrastructure. According to the initial goals, it has shown how the metamodel can be used to integrate works on traffic [10], [11] and produce compliant models. These models were the basis to produce the simulation code. When introducing new types of component, users only needed to map them to code templates, either reusing or developing these. After that, they could automatically generate a new simulation. Most of this effort can be reused for other simulations.

This work is ongoing research with several open issues. Firstly, the TML needs to be tested with additional theories about traffic to validate its primitives and structure. For instance, the introduction of social norms or explicit information about time has not been checked. Secondly, the current TML makes a limited used of OCL [16] constraints. Some facilities could be added to manage them graphically at the model level. This would allow, for instance, that experts limit the kinds of inheritance allowed in their models. It would be also interesting being able to inject real data on traffic dynamics in simulations. This could add information on drivers' usual manoeuvres in certain places,

traffic jams, or traffic lights programming. Such functionality requires incorporating the modelling of simulation instances (i.e. instances of instances of the meta-classes in the traffic metamodel), instead of managing them from the code generator as now. Finally, more testing is required to check whether this infrastructure is suitable to satisfy different analysis needs, e.g. urban planning or norm changes.

Acknowledgments

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

References

[1] M. Pursula, "Simulation of Traffic Systems – An overview," Journal of Geographic Information and Decision Analysis, vol.3, no.1, pp.1–8, 1999.
[2] 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.
[3] R.L. Axtell and J.M. Epstein, "Agent-based modeling: understanding our creations," The Bulletin of the Santa Fe Institute, vol.9, no.2, pp.28–32, 1994.
[4] R. France and B. Rumpe, "Model-driven Development of Complex Software: A Research Roadmap," 2007 Future of Software Engineering (FoSE 2007), pp.37–54, IEEE, 2007.
[5] M.V. Aerde, B. Hellinga, M. Baker, and H. Rakha, "INTEGRATION: An Overview of Traffic Simulation Features," 1996 Transportation Research Board Annual Meeting, pp.1–14, 1996.
[6] Q. Yang, H.N. Koutsopoulos, and M.E. Ben-Akiva, "Simulation laboratory for evaluating dynamic traffic management systems," Transportation Research Record, vol.1710, no.1, pp.122–130, 2000.
[7] D. Ni, "A framework for new generation transportation simulation," 38th Winter Simulation Conference (WSC 2006), ed. L.F. Perrone, B.G. Lawson, J. Liu, and F.P. Wieland, pp.1508–1514, Winter Simulation Conference, 2006.
[8] N. Gilbert and K. Troitzsch, Simulation for the social scientist, McGraw-Hill International, 2005.
[9] 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.
[10] E. Bekiaris, A. Amditis, and M. Panou, "DRIVABILITY: a new concept for modelling driving performance," Cognition, Technology & Work, vol.5, no.2, pp.152–161, 2003.
[11] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "SUMO – Simulation of Urban MObility - An overview," 3rd International Conference on Advances in System Simulation (SIMUL 2011), pp.55–60, IARIA, 2011.
[12] J. Bézivin, "Model Driven Engineering: An Emerging Technical Space," Lecture Notes in Computer Science, vol.4143, pp.36–64, 2006.
[13] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, EMF: Elipse Modeling Framework, Addison-Wesley Professional, 2008.
[14] J.P. Müller, M. Pischel, and M. Thiel, "Modeling Reactive Behaviour in Vertically Layered Agent Architectures," Lecture Notes in Computer Science, vol.890, pp.261–276, 1995.
[15] D. Rubel, J. Wren, and E. Clayberg, The Eclipse Graphical Editing Framework (GEF), Addison-Wesley Professional, 2011.
[16] Object Management Group, "Object Constraint Language, Version 2.4." http://www.omg.org/spec/OCL/2.4, 2014.
[17] T.I. Ören and N. Ghasem-Aghaee, "Personality representation processable in fuzzy logic for human behavior simulation," 2003 Summer Computer Simulation Conference (SCSC 2003), pp.11–18, The Society for Modeling and Simulation International, 2003.
[18] A. Fernández-Isabel and R. Fuentes-Fernández, "Analysis of Intelligent Transportation Systems Using Model-Driven Simulations," SENSORS, vol.15, no.6, pp.14116–14141, 2015.
[19] D. Servat, E. Perrier, J.P. Treuil, and A. Drogoul, "When Agents Emerge from Agents: Introducing Multi-Scale Viewpoints in Multi-Agent Simulations," Lecture Notes in Computer Science, vol.1534, pp.183–198, 1998.
[20] H. Greenberg, "An analysis of traffic flow," Operations Research, vol.7, no.1, pp.79–85, 1959.
[21] P.A.M. Ehlert and L.J.M. Rothkrantz, "Microscopic traffic simulation with reactive driving agents," 2001 IEEE Intelligent Transportation Systems (ITSC 2001), pp.860–865, IEEE, 2001.
[22] S. Maerivoet and B.D. Moor, "Cellular automata models of road traffic," Physics Reports, vol.419, no.1, pp.1–64, 2005.
[23] J.A. Michon, "A Critical View of Driver Behavior Models: What Do We Know, What Should We Do?," in Human Behavior and Traffic Safety, ed. L. Evans and R.C. Schwing, pp.485–524, Springer, 1985.
[24] P. Paruchuri, A.R. Pullalarevu, and K. Karlapalem, "Multi-Agent Simulation of Unorganized Traffic," 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002), pp.176–183, ACM, 2002.
[25] H. Vangheluwe and J. de Lara, "Computer Automated Multi-Paradigm Modelling for Analysis and Design of Traffic Networks," 36th Winter Simulation Conference, (WSC 2004), ed. R.G. Ingalls, M.D. Rossetti, J.S. Smith, and B.A. Peters, pp.249–258, Winter Simulation Conference, 2004.

Alberto Fernández-Isabel is a Ph.D. student of Computer Science at the Complutense University of Madrid, Spain, where he collaborates with the GRASIA research group. He currently works at the Spanish National Research Council (CSIC) as consultant and computer engineer. His main research interests are on social simulation, in particular applied to social behaviour and traffic problems.

Rubén Fuentes-Fernández holds a Ph.D. in Computer Science from the Complutense University of Madrid, Spain. He is Associate Professor at this university since 2010, and member of the GRASIA research group. Previously, he worked as consultant in database systems. Rubén is (co-)author of more than 80 papers published in international journals, book chapters, and conference proceedings. His main research interests are related to the application of Social Sciences to the development of information systems, model-driven engineering, agent-oriented methodologies, social simulation, and ambient intelligence.