Simulation of Road Traffic Applying Model-Driven Engineering

Alberto Fernández-Isabel[a] and Rubén Fuentes-Fernández[a]

[a]Research Group on Agent-based, Social & Interdisciplinary Applications, Complutense University of Madrid, c. Profesor José García Santesmases 9, Madrid, 28040

KEYWORD

Road traffic; Simulation; Modelling language; Intelligent agent; Model-Driven Development; Code generation

ABSTRACT

Road traffic is an important phenomenon in modern societies. The study of its different aspects in the multiple scenarios where it occurs is relevant for a huge number of problems. At the same time, its scale and complexity make it hard to study. Traffic simulations can alleviate these difficulties, simplifying the scenarios to consider and controlling the amount of variables. However, their development also presents difficulties. The main ones come from the need to integrate the way of working of researchers and developers from multiple fields. Model-Driven Engineering (MDE) addresses these problems using Modelling Languages (MLs) and semi-automatic transformations to organise and describe the development, from requirements to code. This paper presents a domain-specific MDE framework for simulations of road traffic. It comprises an extensible ML, support tools, and development guidelines. The ML adopts an agent-based approach, which is focused on the roles of individuals in road traffic and their decision-making. A case study shows the process to model a traffic theory with the ML, and how to specialise that specification in an existing target platform and its simulations. The results are the basis for comparison with related work.

1. Introduction

Road traffic is a phenomenon that involves multiple perspectives and scenarios. For this reason, its study in real settings demands considering large sets of variables that represent a variety of aspects, such as the subjective security perception of individuals, multiple interactions among them, leisure organisation, and health issues. Also, the individuals involved play multiple roles (i.e. driver, pedestrian, or passenger), being able to establish complex relationships among individuals of the same or different role. This complexity makes compulsory looking for means to limit the considered variables in order to focus only on the relevant aspects. Traffic simulations appear as a key tool in a possible solution for these problems. Nevertheless, simulations present their own limitations (Crooks et al., 2008). Some of the most relevant emerge from the misunderstandings in multidisciplinary teams, and the difficulties to adapt simulations to changing requirements and to guarantee that abstract models are faithfully translated to code by manual development processes where unintended mistakes and assumptions are frequent.

Model-Driven Engineering (MDE) (France and Rumpe, 2007) has been proposed to address these problems. It uses development artefacts with a higher level of abstraction than source code, that include all the relevant information for the transition from requirements to the final simulation, and that can be oriented to specific user profiles. Approaches based on it are organised around *models* and semi-automatic *transformations* among artefacts (e.g. to generate source code or documentation from models). Its processes usually adopt an iterative and incremental approach that allows introducing modifications in artefacts at any moment. The main weakness of these proposals is the effort required to develop their specific infrastructure, as these are not mainstream.

However, the benefits of MDE (for instance in artefact reutilisation among projects) usually surpass this initial investment.

The framework introduced in this paper provides a complete and integrative MDE infrastructure intended to develop road traffic simulations from theories of the domain. It comprehends a Traffic Modelling Language (TML), two main supporting tools (a graphical editor for model specifications and a code generator), and a development process.

The TML is intended to be a Domain-Specific ML (DSML) for road traffic and has been designed to be able to integrate different theories according to the simulation needs. It is organised to consider the different roles played by individuals in traffic (i.e. drivers, pedestrians, and passengers).

Following MDE practices, a metamodel specifies the TML. It is structured in conceptual clusters using inheritance and composition hierarchies. Inheritance allows the specialisation of concepts, while composition relates elements that share the same purposes or are part of the same functional group. There are three clusters: a *Mental cluster* to represent the knowledge and features of individuals, an *Environment cluster* to consider the information related to the traffic scenario, and an *Interactive cluster* to describe the decision-making and the interactions among individuals and the surrounding environment.

The support tools are compliant with the TML. The graphical editor allows specifying and validating model specifications. These specifications are the inputs of the code generator tool, which supports the source code transformation. It provides a set of functionalities to modify and specialise this code according to the requirements of the target simulation platform. To achieve these operations it uses source code templates based on the primitives of the TML.

The proposed process covers the complete development cycle, from the analysis of the problem to the execution of the simulation. It considers the roles of traffic expert and programmer. It is organised as an iterative and incremental workflow with five main phases and several internal ones.

A case study illustrates the suitability of the framework to adapt road traffic theories to simulations requirements. In this case, the chosen target traffic simulation platform is MATSim (Transport Systems Planning and Transport Telematics Group, Transport Planning Group and Senozon Company, 2015). The development process guides users during the different phases, indicating how to carry out tasks and the appropriate tools for them.

The rest of the paper is organised as follows. Section 2 introduces the foundations of MDE and its related tools. Next sections describe our framework. Section 3 addresses the TML, Section 4 the tools, and Section 5 presents the main phases of the development process. The case study in Section 6 illustrates the use of the framework. Finally, Section 7 compares our and related work. Finally, Section 8 discusses some conclusions and future work.

2. Model-Driven Engineering

MDE (France and Rumpe, 2007; Kent, 2002) is a general software development approach organised around *models*. A *model* is a representation of an element (in this case, an object, system or idea), which is described in a different form than the entity itself. It captures essential aspects of the element shown for a certain purpose (e.g. develop code or understand an existing system). In projects, developers iteratively and incrementally add and refine elements to models. This cycle moves the development from abstract models (which are close to requirements and in our case to traffic theories) to concrete ones (which are close to the target platform and source code). *Transformations* are automated processes to perform repetitive changes in models (e.g. the modification of models to introduce specific patterns or specialisations to a target platform). Other elements of development (e.g. source code or documentation) are generated in the same way, as they can be obtained from transformations using models and manual adjustments.

In order to allow their automated processing, models must be defined in formal ways. MDE achieves this

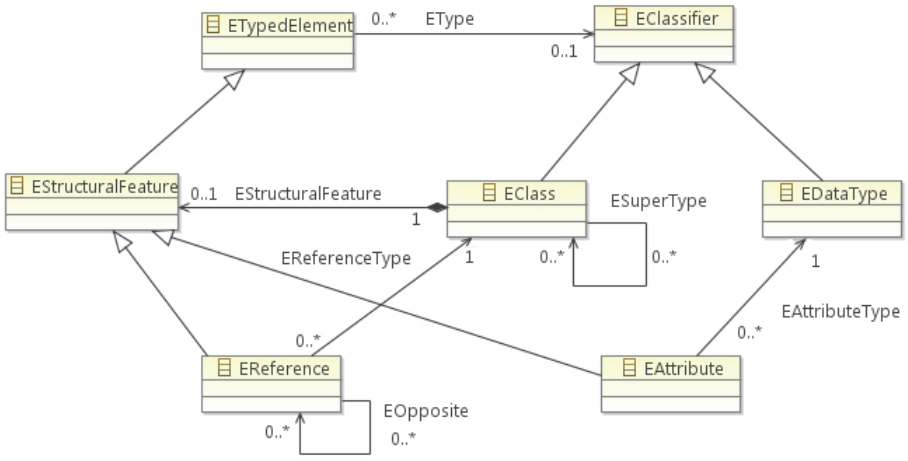

Figure 1: Main Ecore primitives extracted from (Steinberg et al., 2008).

purpose with the specification of MLs to which models must conform. There are different types of ML, but the most used in MDE are graphical graph-oriented languages (Bézivin, 2006), and the usual way to define them is with *metamodels*. Metamodels in turn are specified using meta-modelling languages. There are also several of these languages, like the Meta-Object Facility (MOF) (Object Management Group, 2015a) and Ecore (Gronback and Merks, 2008). MOF is used in the definition of the Object Management Group (OMG) standards, such as the Unified Modeling Language (UML) (Object Management Group, 2015b) and SPEM (Software & Systems Process Engineering Metamodel) (Object Management Group, 2008). Its lack of extensive tool support makes that most of development approaches choose Ecore, the alternative of Eclipse projects for MDE. Nevertheless, these languages have many features in common, as Ecore is almost aligned with the subset of MOF known as Essential MOF (EMOF). Both support the definition of constraints using the Object Constraint Language (OCL) (Object Management Group, 2014).

Some of Ecore main primitives are illustrated in Figure 1. *EClass* instances group elements that share characteristics. They are similar to entities at the model level (i.e. classes). *EClass* instances can contain *EAttribute* and *EReference* instances. *EAttribute* instances are features which domain are the primitive *EDataTypes* ()e.g integer, boolean or characters). *EReference* instances represent binary and oriented references that relate two *EClass* instances. There are two types of them: containment and non-containment. The *ESuperType* reference supports the inheritance mechanism among *EClass* instances. An *EOperation* instance defines a certain behaviour in an *EClass* instance. *EPackage* instances allow generating groups of elements in a metamodel. They can be included within other identifying them in the namespace through a URI (Uniform Resource Identifier). *Eenum* instances represent the enumerated types defined within an *EEnumLiteral* list. An *EEnumLiteral* instance includes the values of an enumerated type calling them by specific names. Finally, *EAnnotation* instances allow introducing notes or comments referencing one of the above instances.

Eclipse provides a wide range of supporting tools for the design of MLs and the specification of their associated models. It also eases the automatic generation of source code using generic templates based on the meta-classes of the metamodel that defines a ML. These functions are performed by several frameworks (Gronback and Merks, 2008), being the core the Eclipse Modelling Framework (EMF) (Steinberg et al., 2008)

and the Graphical Editing Framework (GEF) (Rubel et al., 2011). The first one mainly deals with the design of the ML and the transformations to source code. The second one allows implementing graphical editors using Eclipse plug-ins based on Ecore metamodels. These editors can be configured adding properties to be considered through OCL restrictions.

Transformations are used to generate artefacts from others. They can be classified according to their input and the output they produce as (Czarnecki and Helsen, 2006): Model to Model (M2M) (Wimmer and Burgueño, 2013) (e.g. refining a model with design information), Model to Text (M2T) (e.g. generation of documentation or code from models), and Text to Model (T2M) (e.g. reverse engineering to generate models that describe a given code).

Transformations are implemented in different ways, including the use of general purpose programming languages and specific transformation languages. In the first case, the transformation becomes a module that uses programming interfaces to handle 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 has the advantage of being able to reuse proven artefacts and tools coming from mainstream development approaches (e.g. object-oriented programming or XML processing (Object Management Group, 2003)), and to facilitate adjusting the execution of transformations. The second approach eases the understanding and analysis of the correspondences between inputs and outputs.

Our MDE development framework is based on the meta-modelling language Ecore and the Eclipse tools. The TML (see Section 3) is specified using the EMF facilities, while the supporting tools (see Section 4) use functionalities and code from EMF and GEF.

3. Traffic Modelling Language

The TML is focused on modelling the behaviour of individuals involved in road traffic. Given the variety of needs and theories that can be considered in traffic studies, it is important to provide mechanisms that facilitate the adaptation of the language to specific settings through the modelling of additional theories within its conceptual framework. It must be noted that the language is mainly intended for the early stages of development, so it is biased to traffic concepts and less to design details.

Following prevalent practices in MDE, a metamodel (Steinberg et al., 2008) describes the TML, in this case with Ecore. It introduces a set of meta-elements that represent the concepts, relationships, and properties that specify the TML conceptual framework.

Metamodel concepts are closely related to those of the agent paradigm (Shoham, 1993), and in particular with models that consider the mental state (Bresciani et al., 2004; Pavón et al., 2005). They are classified into three clusters. The *Mental cluster* considers the features and internal state of participants in traffic following mainly (Shinar, 1978). The *Environment cluster* incorporates the elements from the DVE model (Amditis et al., 2010), focusing on the different types of interactions among individuals involved in traffic and the surrounding environment. The *Interactive cluster* represents the goals and actions of those involved in traffic. It describes a perception, reasoning, and acting cycle inspired by the agent literature.

The core element of the metamodel is the *Person* meta-class. It represents a type of individual involved in traffic. According to their means of transport, these people can take the role of drivers, passengers, or pedestrians. *Person* instances can interact with an *Environment* instance. This interaction is direct (in the case of pedestrians through instances of the *Perceives* reference) or indirect (for drivers and passengers through instances of the *Interacts* reference) as a vehicle is used in it. The information individuals own (e.g. norms or experience) is described with instances of the *Knowledge* meta-class using the *Possesses* reference to relate them to *Person* instances. Their features are represented by the *Profile* meta-class. This is related to *Person* instances through the *Displays* reference. Regarding to the purposes of individuals, they are described by *Goal* instances, and

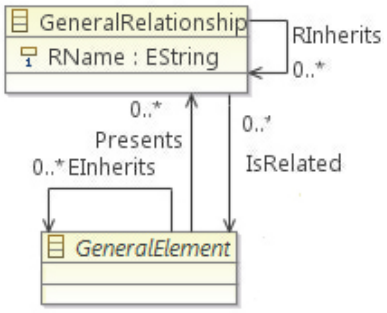

Figure 2: Excerpt of the basic structure of the metamodel.

the possible ways to achieve them by *Task* instances. *Evaluator* instances examine the available information (perhaps generating new one) and determine how people must act according to the circumstances observed (i.e. potential tasks that can run in given circumstances). Finally, *Actuator* instances run the scheduled tasks.

The meta-classes include predefined attributes and methods. The first ones may be specific for a meta-class, (e.g. *Facts* or *Route* in the *Knowledge* meta-class), or shared by a set of meta-classes having a similar meaning and name (e.g. *XName* attributes such as *EName* or *PName*). The second ones are containers for specifications that describe behaviours or how to derive certain attributes from others. For instance, in a model, fragments that specify code or formulas can be inserted in the body of these methods. This information can be used later to guide transformations.

The metamodel uses inheritance hierarchies in order to provide the necessary specialisation of concepts and structure to its elements. Concepts extend from the *GeneralElement* meta-class (see Figure 2), that uses the *EInherits* reference to represent inheritance among elements of the same type in model specifications. The *GeneralRelationship* meta-class (see Figure 2) supports introducing relationships (e.g. impact or affect) among these elements. Its *RInherits* reference allows creating extended instances.

Other type of hierarchies appear in the both *Mental* and *Environment* clusters. Composition hierarchies are considered among main elements (e.g. *Knowledge* or *Environment*) and their respective *XComponent* element (e.g. *KComponent* or *EComponent*). The latter can be decomposed into others of the same type, building complex structures.

Both types of hierarchies are constrained applying OCL restrictions. In the case of inheritance, a constraint only allows references among instances of meta-classes of the same type (e.g. an *Evaluator* instance can be only extended to another of the same type). In composition hierarchies, a constraint controls that both elements might be properly related (e.g. a *Vehicle* instance can be decomposed only into *VComponent* instances, and these only into other *VComponent* instances).

Mental and *Environment* clusters contain *calculateXValue* methods (e.g. *calculateKValue* or *calculatePValue*) and their associated attributes *XValues* (e.g. *KValue* or *PValue*) for establishing and calculating a specific impact factor among the instances of their meta-classes.

Next sub-sections give more details about clusters. Section 3.1 introduces the *Mental cluster*, Section 3.2 focuses on the *Environment cluster* and its concepts related to the DVE model, and Section 3.3 describes the *Interactive cluster*.

3.1 Mental cluster

The *Mental cluster* (see Figure 3) describes the current mental state (e.g. experience or any other type of learnt knowledge) and the particular features of individuals involved in road traffic. These elements affect the exhibited behaviour and the actions these individuals perform (Shinar, 1978).

This cluster comprises three main meta-classes: *Person*, *Profile*, and *Knowledge*. *Profile* represents the particular features of the participants in traffic (e.g. gender or impatience). *Knowledge* describes the people' mental state and current information, except their goals. The attribute *Facts* store this information, being able to be classified. Knowledge can be factual (e.g. traffic signs on the road), procedural (e.g. how to carry out a specific manoeuvre with the vehicle), and normative (e.g. the driver must drive below the maximum speed on roads). Regarding the plans of individuals in traffic, they are often given by their route. This route is considered by the *Route* attribute, and the progress in it by the *RoutePlace* attribute.

Both knowledge of people and their particular characteristics can describe information that does not change during the simulation (e.g. meaning of signs or stature of individuals), or that it does (e.g. respect to signs or levels of aggression).

The instances of the *Knowledge* meta-class and their meta-classes related through composition hierarchy (i.e. *KComponent*) can embody information pertaining to individuals or restricted groups of them (e.g. its own experience or a specific route), or global information accessible to every participant (e.g. the length of a stretch of road under construction). The *KIsGeneral* attribute discriminates between both uses.

3.2 Environment cluster

Traffic occurs in a scenario that provides certain physical conditions. These can change over time (e.g. width of the road or tyre grip) or not (e.g. situation of a specific building or size of a vehicle). The *Environment cluster* (see Figure 4) considers these issues adapting the concepts of the DVE model (Amditis et al., 2010), focusing on the role played by drivers, pedestrians, and passengers in traffic. This cluster considers that individuals can obtain information from the environment (the case of any person participating in traffic) and from the vehicle where they are (only drivers and passengers). The DVE concepts and their related elements in the TML can be extended to accommodate similar theories about the interactions among people and the environment in road traffic. For instance, models based on reactive automatons (Ehlert and Rothkrantz, 2001) or considerations about external factors that increase the risk of accidents (Doherty et al., 1998).

This cluster is responsible for describing how people relate to their means of transport and the environment that surrounds them. To carry out these descriptions, it uses three main meta-classes: *Person*, *Environment*, and *Vehicle*.

Person allows considering the different roles that people can play in road traffic. In the case of drivers, the *Drives* reference is used to relate instances of this meta-class to instances of the *Vehicle* meta-class, while for passengers the *Uses* reference must be applied. Pedestrians do not use any of these relationships as they are not related to means of transport.

The *Environment* meta-class represents the scenario where people (i.e. instances of the *Person* meta-class) interact, including physical conditions that may occur (e.g. weather and traffic conditions). These features and conditions of the environment are described by instances of the *EComponent* meta-class. A model specification compliant with the TML must present at most one *Environment* instance.

The *Vehicle* meta-class represents the mean of transport used by each individual. Drivers and passengers are related to *Environment* instances through their vehicles, but only drivers can execute actions on them in order to influence the vehicle or the environment. The *VComponent* meta-class allows introducing features or parts of vehicles (e.g. engine horsepower or vehicle size).

The mutual influences among instances of the *Person*, *Environment*, and *Vehicle* meta-classes are largely represented in the cluster through some predefined attributes and methods. The *Environment* meta-class includes an attribute called *AvailableArea* in order to indicate the scenario where traffic occurs at a given moment. The *Person* and *Vehicle* meta-classes present an attribute named *VisibleInfo* to specify what information provided by the *AvailableArea* attribute of an *Environment* instance can be observed. The *Person* meta-class also has an *observeEnvironment* method to update the perception of the environment modifying its *VisibleInfo* attribute. It also presents an *Interact* method to execute its interactions with the environment and with other individuals. The *Vehicle* meta-class provides a method called *obtainVisibleInfo*, which plays a similar role to the *observeEnvironment* method in the *Person* meta-class. In addition, the *Vehicle* meta-class has an *executeInstruction* method intended to execute specific actions of the vehicle (e.g. moving a rear-view mirror or turn on lights). The *VComponent* meta-class presents this same method with a similar functionality.

3.3 Interactive cluster

The *Interactive cluster* (see Figure 5) illustrates how *Person* instances act. This acting is organised in a cycle of perception, reasoning, and acting, which makes use of descriptions of the goals and capabilities of individuals. The cycle includes the decision-making to choose the best possible action to achieve the person's goals given certain traffic situations, which are represented with the elements provided by the *Mental cluster* (see Figure 3) and the *Environment cluster* (see Figure 4). The cycle also includes the execution of the selected actions. The cluster describes this information with two groups of elements: the first one represents the goals of individuals and their abilities to try to reach them; the second one the elements to implement the cycle.

The first group comprises the *Goal* and *Task* meta-classes. These two concepts are taken from AOSE, where agent-based methodologies as Tropos (Bresciani et al., 2004) and INGENIAS (Pavón et al., 2005) use them to specify Multi-Agent Systems (MAS) (Van Der Hoek and Wooldridge, 2008). These methodologies include a specific architecture of acting where actors (i.e. agents) play various roles that give them different skills, knowledge, and goals. The agents try to enforce the conditions of satisfaction of their different goals. To do this, agents have tasks linked to goals, so that the execution of these tasks is potentially able to satisfy their goals.

In the TML, the *Mental cluster* represents the mental state of agents, and the *Environment cluster* collects the information from the scenario where traffic occurs and the vehicle (only when an individual plays the driver or passenger role). The *Goal* meta-class describes a state of some elements related to road traffic that a person aspires to maintain or achieve, while the *Task* meta-class models the skills of this person. Both meta-classes provide specific attributes to define these aspects. *Goal* meta-class uses the *Satisfaction* attribute to represent its fulfilment conditions and a method called *calculateSatisfaction* in order to manage these conditions. The *Task* meta-class includes the *Instructions* attribute to specify the atomic actions that are carried out to perform it. These actions are executed to achieve the satisfaction of those goals to which tasks are associated through an instance of the *GImplies* reference.

These meta-classes present a composition hierarchy where they can be decomposed into others of the same type (restricted by OCL expressions). The structure of this hierarchy is similar to the other two clusters (i.e. a main element and a *XComponent*). In this case the semantics are different, being similar to the compositions of tasks and goals adopted by the agent-based methodologies previously introduced. Thus, they are related to fulfilment instead of determining specific features of an element. The *Goal* and *Task* meta-classes own the *GType* and *TType* attributes in order to specify the type of those compositions. This promotes both meta-classes can support various semantic structures and classifications. Currently, the *GType* attribute describes the different types of satisfaction compositions allowed for the goal, while the *Ttype* attribute indicates whether the current task is carried out completing one or a specific set of its sub-tasks.

The second group has the elements of a *Person* instance that are responsible for assessing the known state (both environmental and internal to the agent) and execute actions. This group incorporates a classic cycle of

perception, reasoning and acting for agents (Lind, 2001). In it, the perceived and extracted information from the scenario is gathered in elements of the *Mental cluster*, while the reasoning is achieved by instances of the *Evaluator* meta-class. Finally, the acting is implemented by instances of the *Actuator* meta-class.

Evaluator instances can be arranged in a hierarchical composition (i.e. they can be decomposed into other *Evaluator* instances through the *EVDecomposes* reference). This allows the distribution of responsibilities among them, but only one is related to the *Person* instance and vice versa through *IsHarnessed* and *Harnesses* references respectively. On the contrary, *Actuator* instances do not have a hierarchical composition (i.e. *Actuator* instances cannot be decomposed into others). Each *Person* instance (i.e. a type of individual) may be related only to one *Actuator* instance through the *Utilizes* reference (see Figure 5).

The previous elements implement the cycle as follows. *Evaluator* instances evaluate the information obtained from the *Environment*, *Vehicle*, *Profile*, and *Knowledge* instances linked to their *Person* instance, their *XComponent* elements, and other possible elements related through *GeneralRelationship* instances. To carry out this process, these *Evaluator* instances present the predefined method *evaluateGoals*. Using it, they can update the internal state of their *Person* instance. All this information determines the current state of *Goal* instances, that is, if they are satisfied or not. For this evaluation, these instances have the *calculateSatisfaction* method. Once a candidate *Goal* instance is selected for execution, an *Actuator* instance must collect its associated tasks. These *Task* instances are executed through the *executeChosenTask* method, following the atomic instructions provided by their *Instructions* attributes or executing their children *Task* instances.

4. Supporting tools

The MDE framework introduced in this work presents two supporting tools: a graphical editor and a code generator. The first one is used to develop model specifications compliant with the TML, and the second for the semi-automatic transformation to source code of these specifications. This code can be specialised in order to adapt it to the requirements of target simulation platforms. Both tools are built using functionalities of the EMF (Steinberg et al., 2008) and the GEF (Rubel et al., 2011).

The graphical editor is an Eclipse plug-in that provides assistance and guidance to users during the model specifications development. In order to achieve this, it uses two XMI files (Object Management Group, 2015c), one to store the design of the model specification and another for the graphical location of the elements contained in the model. The former also allows the validation of the current model specification, ensuring its compliance with the TML and the OCL constraints included. The interface of this tool provides a canvas and a palette to display the specifications and the concepts contained in the metamodel respectively.

The code generator is a tool particularly developed to achieve the multiple functions related to the transformations of the model specifications produced by the graphical editor. It is used to modify the code templates EMF generates and specialise this source code according to the requirements of the target simulation platform. Most of these operations are partially automated through wizards in order to provide support and guide to users during the process. This allows playing a preponderant role to traffic experts, while programmers only work in the different development phases where the insertion or modification of source code is indispensable. A text editor and a compiler are integrated into the tool in order to support these features.

Once a model specification is loaded, the graphical interface of the tool allows visualising the data contained in the file, providing an intuitive tab-based navigation among its elements. When one of these elements is selected, the tool displays information about its associated methods, its composition and inheritance hierarchies, and the related *GeneralRelationship* instances (see Figure 6).

As said before, the code generator is able to achieve multiple purposes through graphical wizards. They can be grouped as functionalities related to code generation and modification, configuration and documentation, structural changes modifications and platform specialisation.

The insertion of source code is a feature related to code generation that provides the possibility of modifying the body of the methods of different classes introducing code-snippets, or creating new attributes and methods updating the original class completely. This functionality is based on the code templates EMF generates by default. These operations demand programming skills and are supported by the graphical interface and the integrated modules (i.e. the compiler and the text editor). The tool also provides a user guide and on-line assistance.

The configuration and documentation functionality produces different XML files. In the first case, the configuration file sets the initial values of the *XValues* attribute (see Section 3). In the second case, the tool generates the project documentation.

The cluster integration is the operation included within the structural changes functionalities. This is achieved in two steps: the integration of two incomplete and complementary model specifications (i.e. the elements of the first cannot belong to a cluster provided by the second and vice versa), and the addition of new *GeneralRelationship* instances in order to model the impact an element exerts on another in the resulting new model specification. Two graphical wizards guide users during the process indicating the possible references to add according to the TML, or the *GeneralRelationship* instances that are incomplete (i.e. they do not present an origin or a destination yet). This feature promotes the reutilisation of model specifications.

The code specialisation eases the adaptation of the current source code to the demands of target simulation platform. This feature presents two main operations: the dynamic insertion of external files in the path of the compiler and the generation of new classes.

The dynamic insertion operation handles and stores external libraries (and their dependencies). These files usually come from the target platform. Once the library is selected, the process is managed internally by the tool, so it is transparent to users. It allows the inclusion of items not related to the TML in the source code of a class of the current model specification.

The generation of new classes is an operation that is achieved with the support of an assistant. These classes can be empty, extended from others provided by the external libraries, or predefined. The latter can be built using additional templates and are intended to automate tasks that are similar in every project (e.g. read the model specification XMI file). These classes are considered by the compiler dynamically. This feature allows the instantiation and use of them in other classes of the current project.

The code generator produces two types of outcomes: a plug-in directly usable as a library in the target traffic platform, or a new platform that integrates the aspects provided by the model specification and the target platform. In both cases, the generation process is supported by graphical wizards that make it more intuitive to users. Also, associated documentation and configuration files can be produced.

These development tools support our MDE framework intended to adapt traffic theories and generate source code to simulate them. They ease the process through graphical wizards, which provide the appropriate functionality to examine elements and integrate multiple artefacts (e.g. model specifications or predefined classes). They also promote reusability and incremental development, reducing as a consequence manual coding and the need of programmers.

5. Development process

The development process used to generate traffic simulations covers from the moment when experts study the problem and choose to model certain traffic theories, until the instant the simulation is executed on the target traffic platform. Here, it is described using SPEM (Object Management Group, 2008). It identifies five main phases (see Figure 7): *Preliminary theory evaluation*, *Model design*, *Preliminary code generation*, *Platform specialisation*, and *Simulation*. Although the representation of the process in the diagram shows a linear workflow in order to simplify it, the process must be considered as incremental and iterative (i.e. spiral

development). This means that users can return to a previous phase from the current one and continue the process from there.

There are two main roles involved in the process: traffic expert and programmer. The first one achieves most of tasks, evaluating the selected traffic theory, designing the current model and validating the simulation. The second one mainly works in the *Preliminary code generation* and *Platform specialisation* main phases (also in the *Simulation* main phase in the case of a plug-in outcome). It is responsible for encoding operations (e.g. changes in the body of the predefined methods or programming new ones), and introducing new elements (e.g. specific classes or methods to implement the specialisation of the model specification to a target platform).

The *Preliminary theory evaluation* main phase is the first in the process. In it, experts assess the selected traffic theory and try to generate a *modelling plan* of the future model specification. This plan identifies the elements extracted from the traffic theory and their correspondences to concepts provided by the TML. If there is an acceptable mapping, then the process continues through the next phase. In other case, if experts fail to reach an appropriate *modelling plan* of the theory, it is discarded. Alternatively they could consider a review of the TML, and consequently of the supporting tools, in order to incorporate the requirements presented by that theory.

The *Model design* main phase develops the model specification according to the *modelling plan* obtained in the previous phase. In this phase, traffic experts introduce modifications incrementally, inserting new elements and checking their compliance with the constraints of the TML. Once the specification is completed and validated, users can go forward with the next phase.

The *Preliminary code generation* main phase is in charge of generating the source code using the model specification. This phase introduces specific code snippets in the generated classes through the code generator. These snippets take as basis the original templates EMF generates from the classes in the TML metamodel. The code generator supports through assistants the creation of new methods and attributes in a class, and the insertion of specific code in the body of methods. It is also responsible for compiling these modifications and saving the current state of the project. For instance, this inserted code is used to encode some mathematical formulas or quantify the influence of interactions among individuals in a specific attribute.

In the *Platform specialisation* main phase, the source code is modified to adapt it to the requirements of the target platform. To do that, new classes are created with the purpose of packaging the model specifications and the decision-making of individuals. This step considers the information and specifications available about the target platform.

This phase has an optional internal phase that allows integrating complementary incomplete model specifications. It considers two possible types of these specifications: models that contain elements from the *Mental cluster* and the *Environment cluster*, and models only with elements from the *Interactive cluster*. This particular issue is directly related to the domain literature, where there are traffic theories only focused on the decision-making of individuals and others that only consider mental aspects and features of participants. Also, it promotes reusability being able to produce complete model specifications (i.e. a model specification with elements of the three conceptual clusters of the TML) based on two different traffic theories.

This main phase produces as output two possible types of archives, which are the outcomes of the code generator tool (see Section 4). The first one is a plug-in directly usable in the target platform as a library. Thus, the platform can be modified introducing new source code to consider the aspects supplied by the model specification contained in the library. This encourages the development of traffic platforms specialised in the proposed TML. The second one implements an integration of the current model specification and the target platform, creating an enhanced traffic platform that encompasses aspects of the model specification. In both cases, the associated documentation (i.e. Javadoc) and a configuration file are provided. This last file can be modified to change the behaviour of the individuals considered in the traffic simulation.

Finally, the *Simulation main phase* addresses the simulation process. In it, the traffic platform collects the

values set in the configuration file and then executes the simulation. If the file produced in the previous main phase is a plug-in, in this phase it is added as a library to the target platform. Then, the platform must be adapted introducing modifications in the existing classes or even creating new ones. This process allows achieving simulations that consider the new features provided by this library.

6. Case Study

The case study describes the use of the MDE framework to produce an enhanced traffic platform. This modified platform is based on an existing one and includes a model specification that adapts several domain theories.

The considered theoretical works are: the application of a formula based on fuzzy logic (Pappis and Mamdani, 1977) to calculate the *XValues* attributes of the instances of the *Mental* and *Environment* clusters (see Section 3) from the related elements; the theory on driving risk factors in (Schieber and Thompson, 1996); and the decision-making theory from (Fernandez-Isabel and Fuentes-Fernandez, 2015), completed here with an equivalent structure for pedestrians. The target platform to extend is MATSim (Transport Systems Planning and Transport Telematics Group, Transport Planning Group and Senozon Company, 2015).

The theory in (Schieber and Thompson, 1996) describes groups of risk factors related to traffic accidents among young pedestrians. These groups hierarchically classify certain features, like the involvement drivers or features of the surrounded environment. This organisation conforms a taxonomy where certain factors depend upon or are influenced by others.

The traffic theory does not consider the DVE approach (Amditis et al., 2010) integrated in the TML. Modelling it with the TML requires a more detailed study to determine how to fit the factors provided in the theory as components of the main meta-classes (e.g. *Knowledge* or *Environment*). It can be observed this theory does not consider elements related to the decision-making of individuals, as these risk factors can be described only using the instances of meta-classes provided by the *Mental cluster* and the *Environment cluster*. Therefore, it is necessary to select a complementary traffic theory focused on individual decisions to generate a complete specification model.

Following the development process (see Section 5), in the *Preliminary theory evaluation* main phase, a *modelling plan* is developed. This pursues finding equivalences among the factors considered in the theory based on young pedestrians and the concepts from the TML. Note that the names of the factors used in the traffic theory has been modified and shortened to facilitate their display in diagrams.

Four main groups of concepts are specified: *Pedestrian Profile* (i.e. *PProfile*), *Environment*, *Vehicle* and *Driver Profile* (i.e. *DProfile*). The first group includes all the elements related to the features of young pedestrians (e.g. *Family factors* or *Anatomic development*). The second group comprises the factors present in the environment (e.g. *Layout of road* or *Type of road*). These factors are common to drivers and pedestrians. The third group describes the factors of vehicles (e.g. *Design* or *Speed*). These factors are not considered as related to the environment, as it happens in the original work. As the TML is based on the DVE approach and this considers vehicles apart from the environment, these factors are part of a *Vehicle* instance. Finally, the fourth group considers the features of drivers (e.g. *Driver fatigue* or *Use of alcohol*). In both cases (i.e. pedestrians and drivers), concepts related to the *Knowledge* meta-class do not appear.

The original decision-making theory for people involved in traffic (Fernandez-Isabel and Fuentes-Fernandez, 2015) only considers drivers. Here, it is extended to pedestrians with the activities described in (Gipps and Marksjö, 1985). This work introduces a set of activities that pedestrians carry out when they interact with other elements of road traffic. Comparing both theories, it follows that drivers and pedestrians have many common goals (e.g. both search obstacles or accelerate) in those works. Therefore, both types of people can be modelled pursuing a similar tree structure of *Goal* instances with *AND* and *OR* compositions. However, these goals are related to different tasks and conditions. For instance, a pedestrian cannot overtake another in the same way as

a driver overtakes another vehicle.

In the *Model design* main phase, the model specifications are designed in the graphical editor following the *modelling plan*. Thus, a *Person* instance for drivers is created. It is linked with the suitable references to *Profile*, *Vehicle*, and *Knowledge* instances. The first presents three *PComponent* instances as children: *UseofAlcohol*, *ChildrenintheRoad*, and *DriverFatigue*. The second is related to two children *VComponent* instances called *Design* and *Speed* (see Figure 8). The latter is extracted from the division into two new factors of the attribute from the original theory called *Traffic density and speed*. This division is necessary because the TML follows the DVE approach. Thus, the traffic density is considered part of the scenario, while the speed is related to vehicles. The *Knowledge* instance does not present a composition hierarchy.

The *Environment* instance is the root of all the environmental factors that influence both drivers and pedestrians. It is decomposed into five *EComponent* instances: *TypeofRoad*, *LayoutofRoad*, *TrafficSignalsandIslands*, *AdverseTrafficConditions*, and *SocialEnvironment*. The last one is originally a factor related to young pedestrians, but in the *modelling plan* it has been reclassified as a component of the environment. It is decomposed into an *EffectofOthers* instance, which at the same time is composed by three instances: *CrossingGuards*, *Accompanying-Adult*, and *Peers*. The *AdverseTrafficConditions* instance is decomposed into five instances: *Darkness*, *Inclement-Weather*, *RoadsideParking*, *TrafficDensity*, and *NeightbordhoodCharacteristics*. The *TypeofRoad* instance considers types of roads through its *EValues* attribute (e.g. *Local Driveway* or *Street*).

A *Person* instance for pedestrians is only linked to *Knowledge* and *Profile* instances. The first one does not present a composition hierarchy, while the second has two levels according to the original classification. The first level includes the *AnatomicDevelopment*, *StateofDevelopment*, and *FamilyFactors PComponent* instances. The first one is decomposed into three instances: *Stature*, *ReflectiveClothes*, and *SuddenAppearance*. The second instance is also decomposed into three instances called *AttentionSpan*, *WalkingSpeed*, and *MidblockDart-outs*. The third instance is composed by other three instances: *Poverty*, *Crowding*, and *CareofLocalAuthorities* (see Figure 9).

Once the pedestrian factors model specification is completed (i.e. the specification with elements belonging to the *Mental cluster* and the *Environmental cluster*), the next step is designing the model specification related to the decision-making (i.e. the specification with only elements included in the *Interactive cluster*). Note that this model specification is extended from another one created for drivers (Fernandez-Isabel and Fuentes-Fernandez, 2015), so for avoiding overlapped names the pedestrian instances begin with a *P*.

The root goal introduced for pedestrian is described by the *PArrivedFastDestination Goal* instance. This instance contains an *AND* composition through the *Goal* instances *PEndedRoute* and *PActuated*. The last one has an OR composition with seven instances corresponding to actions: *PMoved*, *PDodged*, *PAccelerated*, *PSearchedObstacle*, *PUpdatedEnvironment*, *PSlowed*, and *PTurned*. *PMoved* has an OR composition that consists of three instances (*PContinuedPath*, *PCrossedWay*, and *PPassedCrossing*), while *PDodged* has an OR composition with two instances (*GDodgedLeft* and *GDodgedRight*). *PSearchedObstacle* has a composition with four instances: *PSearchedForward*, *PSearchedBackward*, *PSearchedLeft*, and *PSearchedRight*. Finally, *PTurned* is decomposed into the *PChangedDirection* instance.

Each *Goal* instance has its *Task* responsible for providing instructions when it is executed. These *Task* instances for pedestrians have also changed their name to distinguish them from *Task* instances of drivers.

Then, in the *Preliminary code generation* main phase, both model specifications are transformed to source code by the code generator tool. Moreover, the fuzzy logic formula is codified and inserted in the body of the appropriate *calculateXValue* methods of the pedestrian factors specification. The *evaluateGoals* and *executeChosenTask* methods of the *Evaluator* and *Actuator* instances are redefined in order to consider the pertinent *Goal* and *Task* instances respectively.

After completing these changes in both model specifications and save their projects, they are integrated in order to generate a complete specification that includes both traffic theories. This operation is achieved in the

Platform specialisation main phase of the development process.

The model specification related to the factors presents two different *Person* instances (*Driver* and *Pedestrian*). The first instance is linked to the root *Goal* instance (i.e. *ArrivedFastDestination*) through the *Pursues* reference. This allows drivers to access the tree structure of goals with the associated tasks. Then, the second instance is linked to the *PArrivedFastDestination Goal* using another *Pursues* reference. After that, each *Person* instance is linked to its corresponding *Evaluator* root instance through the *IsHarnessed* and *Harness* references. Finally, the two *Actuator* instances (one for each type of person) are respectively linked to *Driver* and *Pedestrian* instances using the *Utilizes* reference.

The next step is the insertion of *GeneralRelationship* instances among the elements provided by both model specifications. These instances are created to indicate which elements of the *Mental* and *Environment* clusters impact on which *Task* instances and vice versa. For instance, the *Darkness* and *InclementWeather EComponent* instances respectively affect *Task* instances *SearchObstacle* and *Accelerate*.

In the case of elements that belong to the *Mental* or *Environment* clusters, the body of their *calculateXValue* methods must be updated to consider these influences. Also, the body of the *executeChosenTask* method of the *Actuator* instance has to be modified in order to alter the *XValues* attribute of the appropriate element related to the executed task. All these aspects must be considered by the *Evaluator* instances. Fro this purpose, appropriate code snippets must be added to the body of the *evaluateGoals* methods.

When this step is completed, users carry out the platform adaptation to MATSim. MATSim is based on agents, but it only considers route configuration through a path to follow (i.e. the interactions and decisions of participants in traffic are not contemplated). The platform environment offers different functionalities to work with these elements.

The adaptation starts with users loading the platform and its dependencies in the code generator tool. This adds them to its compiler.

The generation of the enhanced platform requires developing and adding a new class using the appropriate graphical wizard provided by the code generator. This class must combine the elements of the model specification and the platform through programming procedures (i.e. the model specification is encapsulated in a class which is integrated in MATSim).

Some classes provided by MATSim must be extended in order to introduce the model specification structure in the decision-making of its agents. Its original classes only plan the route, and now they have to achieve multiple actions related to drivers and pedestrians (e.g. overtake or dodge) and also preserve their original functionalities. They are integrated into the project using the code generator tool, which allows the compiler considers them.

The *Instructions* attribute and the *executeIntructions* method of *Task* instances must be redefined. They need to be specialised according to the requirements of MATSim and its source code. This allows producing the suitable platform-specific actions to exhibit the suitable behaviour.

Once the specialisation of the source code has been done, a configuration file is created through the appropriate wizard of the code generator tool. It contains the initial parameters of the *XValues* attributes of the elements that belong to the *Mental* and *Environment* clusters. These parameters can be modified to obtain different levels of impact among the elements. Another class is also generated in the code generator and integrated into the project in order to load this configuration file into the simulation. Predefined classes are provided by the tool in order to simplify this step. When these operations are completed, a new directory is generated through the code generator. It contains the enhanced platform in a compressed file, the configuration file, sub-directories with its associated documentation, and the libraries and dependencies it requires to run.

Finally, in the *Simulation* main phase the new platform is set up and executed in a similar way to the original MATSim. Traffic experts can then study the result according to their original needs.

7. Related work

Road traffic simulation encompasses multiple areas of research. In this case, this approach is focused on two main issues: how to model the aspects that affect people' behaviour and the development process of simulations. The first one considers the state of participants in traffic and their particular features, alongside the aspects related to the environment that influence their behaviour. The second one describes how to organise development projects for these simulations.

These simulations can be classified according to the way they consider the individuals involved in traffic. Thus, microscopic simulations (Paruchuri et al., 2002) are focused on a particular individual or a small group of them, while macroscopic simulations (Van Den Berg et al., 2003) contemplate the traffic flows. Mesoscopic simulations (Tolujew and Alcalá, 2004) are hybrids based on the traffic flows but they are able to centre the attention in an individual in a specific moment.

Regarding the existing road traffic simulation platforms, they mainly represent individuals through basic entities that follow a predefined route (Transport Systems Planning and Transport Telematics Group, Transport Planning Group and Senozon Company, 2015; Behrisch et al., 2011). Although some of them exhibit some types of random behaviours, the organisation of their particular characteristics, the decision-making, and their interaction with other participants or with the surrounded environment are contemplated in a simplified way. Furthermore, there are roles of individuals involved in traffic that are commonly dismissed in these platforms although pedestrians and the impact of passengers over drivers are also important aspects to consider. For instance, (Visual Solutions, Incorporated, 2015) allows using pedestrians that carry out interactions with the elements of the scenario and drivers, but the impact of the close individuals is not considered.

The proposed metamodel allows describing the multiple roles of participants in traffic (i.e. drivers, pedestrians and passengers). It is focused on microscopic models, as it is in this type of design where it can represent the different artefacts of individuals (e.g. instances of *Person* and *Vehicle*). Mesoscopic models could also be integrated in the ML with certain restrictions. The ABM approach considered by the TML eases this point, as this is frequently adopted for this type of models (Vasirani and Ossowski, 2009).

Another point of discussion is related to the modelling of individuals involved in traffic. As there is not an accepted standard, multiple approaches have been developed for different purposes. These models range from simple ones, where only reactions are considered, to quite complex, where different levels of deliberation are included. For instance, a simple approach is introduced in (Doniec et al., 2008), where agents adopt a set of basic rules to react to the surrounding environment. Another more complex is described in (Burmeister et al., 1997), where driver's purposes and actions are organised in workflows. These allow considering the multiple situations that can befall until individuals reach their destinations.

The modelling of the decision and interactions achieved by the individuals can be organised through hierarchical architectures. These structures usually present multiple abstraction layers. For instance, (Michon, 1985) proposes the Michon's hierarchical control model for drivers, where their decisions are classified in different levels. In the case of the metamodel, it provides a hierarchical composition of most of its concepts, but the definition of these type of layer as required by hierarchical architectures is not considered.

The discussion in literature about which of the characteristics of participants and the environment have influence on road traffic is an open issue. Approaches such as (Greenberg, 1959; Paruchuri et al., 2002) revise some of these features. The metamodel is intentionally open to these considerations. Main meta-classes (e.g. *Vehicle* or *Knowledge*) belong to the *Environment* or *Mental* clusters present a hierarchical composition in order to classify a wide set of different elements or features. The metamodel also allows describing additional aspects (i.e. via the *GeneralElement* meta-class) and influence relationships among elements (i.e. through the *GeneralRelationship* meta-class), and specialising them using inheritance hierarchies. These capabilities ease the customisation of the TML in order to cover the larger amount of demands of the road traffic domain.

In relation to the development process, most of revised approaches do not indicate the type of approach they

adopt. When it is specified, it is usually a traditional development process based on source code and manual coding, where model specifications are used only to generate documentation. The benefits of MDE in this domain have been already pointed out in the related literature (Fuentes-Fernández et al., 2012): explicit and visual description of the information, development oriented to specific domain profiles (i.e. traffic experts), and artefacts with higher abstraction and reusability.

8. Conclusions

This paper has presented a MDE framework for traffic simulations. It is focused on the behaviour of individuals and their interactions among them and the environment. The framework has been designed with special attention to ease the modification of existing elements and the integration of new ones. The other main goal is giving a greater autonomy in the development of simulations to traffic experts, so they can participate more actively in it. The framework includes the TML, two specific tools, and the development process.

The TML is focused on the participants in traffic. It adopts an approach based on agents (Janssen, 2005) and incorporates the DVE model (Amditis et al., 2010) to organise its concepts. Extending this model, it assumes that people can play three roles in traffic: drivers, pedestrians, and passengers. These roles determine their potential relationships with the environment.

Elements in the TML are structured around the specification of different aspects of traffic problems. These aspects are addressed through three conceptual clusters: *Mental*, *Environment*, and *Interactive*. The first one is based on the theories of (Shinar, 1978) and describes the features and current knowledge of an individual or group of them. The second one is based on the DVE model. This model considers that drivers interact with their vehicles and the rest of the environment to obtain information to guide their actions. This cluster extends this model to consider the other two types of people involved in traffic, pedestrians and passengers. Finally, the *Interactive cluster* adapts elements and concepts from the agent paradigm (Van Der Hoek and Wooldridge, 2008), particularly notions related to decisions inspired by the BDI model (Rao and Georgeff, 1992) and methodologies as Tropos (Bresciani et al., 2004) or INGENIAS (Pavón et al., 2005). This cluster models a perception, reasoning, and action cycle based on goals that can be achieved through tasks.

The TML is oriented to provide a flexible and extensible conceptual framework in order to facilitate the integration of different theories and aspects related to road traffic. To do this, it includes mechanisms to support inheritance and composition hierarchies. Inheritance allows obtaining specialisations in the language concepts (both entities and relationships). Composition supports establishing the parts of a specific at multiple nested levels.

Supporting tools are built on top of the Eclipse MDE frameworks EMF (Steinberg et al., 2008) and GEF (Rubel et al., 2011). The graphical editor allows the visual specification of models conforming to the TML. Its implementation is a direct use of GEF. These specifications can be validated to guarantee its compliance. The code generator carries out the transformations of model specifications to source code. It also provides through graphical wizards a set of functionalities that guide users in the generation of specific source code for a target traffic simulation platform.

Regarding the development process, it is an iterative and incremental procedure organised into five main phases: *Preliminary theory evaluation*, *Model design*, *Preliminary code generation*, *Platform specialisation*, and *Simulation*. The first phase evaluates the selected traffic theory and makes a preliminary specification model only with the key concepts to consider. The second phase develops the specifications based on the approach of the previous phase. The third one transforms the specifications to source code. This source code is modified to complete the bodies of the predefined methods and add non-existent attributes and methods. Also, it develops and codifies the mechanisms of influence between the instances of the elements that belong to the *Mental* and *Environment* clusters. The fourth phase specialises the source code to a target platform. The results of this phase

can be two: a plug-in library directly embedded in the target platform, or an enhanced platform that contains the original one with modifications in order to consider the new model specification. Finally, the fifth phase sets up the target platform if necessary, and runs the simulation.

The case study illustrates the use of the MDE framework. The supporting tools were applied to create a model specification according to a specific road traffic theory, its specialisation to the requirements of the MATSim platform, and the generation of the source code associated. The MATSim platform presents a route optimisation functionality that does not consider interactions among individuals and only generates a path to follow. It is improved generating decision-making actions based on (Fernandez-Isabel and Fuentes-Fernandez, 2015; Gipps and Marksjö, 1985) through a goal-task hierarchical structure with *OR* and *AND* compositions. This structure is managed by a perception, reasoning, and acting cycle specified with instances of the *Evaluator* and *Actuator* meta-classes. The model also integrates a taxonomy that considers the risk factors of young pedestrians (Schieber and Thompson, 1996) and their relationships with drivers and the rest of the environment. Thus, people' actions in the resulting simulation are affected by these multiple factors, producing different behaviours in the participants according to their values.

The presented approach has several open issues. The TML structure must be reviewed in order to simplify it. For instance, a new abstract meta-class that extends from *GeneralElement* could be added to encapsulate the *XValues* and *XName* attributes of the *Mental* and *Environment* clusters meta-classes. Moreover, a notion of time could be added to the TML to model traffic events. Regarding tools, the current graphical editor could be integrated in the code generator tool, centralising the development process in only one tool. In order to improve the maintenance of tools when there are changes in the metamodel, the use of meta-editors is being currently considered.

9. 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.

10. References

Amditis, A., Pagle, K., Joshi, S., and Bekiaris, E., 2010. Driver–Vehicle–Environment monitoring for on-board driver support systems: Lessons learned from design and implementation. *Applied Ergonomics*, 41(2):225–235.

Behrisch, M., Bieker, L., Erdmann, J., and Krajzewicz, D., 2011. Sumo-simulation of urban mobility-an overview. In *SIMUL 2011, The Third International Conference on Advances in System Simulation*, pages 55–60.

Bézivin, J., 2006. Model driven engineering: An emerging technical space. In *Generative and Transformational Techniques in Software Engineering*, pages 36–64. Springer.

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., and Mylopoulos, J., 2004. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236.

Burmeister, B., Haddadi, A., and Matylis, G., 1997. Application of multi-agent systems in traffic and transportation. In *IEEE Transactions on Software Engineering*, volume 144, pages 51–60. IET.

Crooks, A., Castle, C., and Batty, M., 2008. Key challenges in agent-based modelling for geo-spatial simulation. *Computers, Environment and Urban Systems*, 32(6):417–430.

Czarnecki, K. and Helsen, S., 2006. Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3):621–645.

Doherty, S. T., Andrey, J. C., and MacGregor, C., 1998. The situational risks of young drivers: The influence of passengers, time of day and day of week on accident rates. *Accident Analysis & Prevention*, 30(1):45–52.

Doniec, A., Mandiau, R., Piechowiak, S., and Espié, S., 2008. A behavioral multi-agent model for road traffic simulation. *Engineering Applications of Artificial Intelligence*, 21(8):1443–1454.

Ehlert, P. A. and Rothkrantz, L. J., 2001. A reactive driving agent for microscopic traffic simulation. In *Proceedings of the 15th European Simulation Multiconference, Prague, Czech Republic*, pages 943–949.

Fernandez-Isabel, A. and Fuentes-Fernandez, R., 2015. Developing an integrative Modelling Language for enhancing road traffic simulations. In *Computer Science and Information Systems (FedCSIS), 2015 Federated Conference on*, pages 1745–1756. IEEE.

France, R. and Rumpe, B., 2007. Model-driven development of complex software: A research roadmap. In *2007 Future of Software Engineering*, pages 37–54. IEEE Computer Society.

Fuentes-Fernández, R., Hassan, S., Pavón, J., Galán, J. M., and López-Paredes, A., 2012. Metamodels for role-driven agent-based modelling. *Computational and Mathematical Organization Theory*, 18(1):91–112.

Gipps, P. G. and Marksjö, B., 1985. A micro-simulation model for pedestrian flows. *Mathematics and computers in simulation*, 27(2):95–105.

Greenberg, H., 1959. An analysis of traffic flow. *Operations Research*, 7(1):79–85.

Gronback, R. C. and Merks, E., 2008. Model-driven architecture at eclipse. *The European Journal for the Informatics Professional*.

Janssen, M. A., 2005. Agent-based modelling. *Modelling in Ecological Economics*, pages 155–172. ISBN: 978-1-78195-866-7.

Kent, S., 2002. Model driven engineering. In *Integrated Formal Methods*, pages 286–298. Springer.

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

Michon, J. A., 1985. A critical view of driver behavior models: what do we know, what should we do? In *Human Behavior and Traffic Safety*, pages 485–524. Springer.

Object Management Group, 2003. eXtensible Modelling Language (XML), Version 1.1. `http://www.omg.org/spec/XML/`. [Online: accessed 14-Dec-2015].

Object Management Group, 2008. Software & Systems Process Engineering Meta-Model (SPEM), Version 2.0. `http://www.omg.org/spec/SPEM/`. [Online: accessed 14-Dec-2015].

Object Management Group, 2014. Object Constraint Language (OCL), Version 2.4. `http://www.omg.org/spec/OCL/`. [Online: accessed 14-Dec-2015].

Object Management Group, 2015a. Meta-Object Facility (MOF) Core Specification, Version 2.5. `http://www.omg.org/spec/MOF/`. [Online: accessed 14-Dec-2015].

Object Management Group, 2015b. Unified Modelling Language (UML), Version 2.5. `http://www.omg.org/spec/UML/`. [Online: accessed 14-Dec-2015].

Object Management Group, 2015c. XML Metadata Interchange (XMI), Version 2.5.1. `http://www.omg.org/spec/XMI/`. [Online: accessed 14-Dec-2015].

Pappis, C. P. and Mamdani, E. H., 1977. A fuzzy logic controller for a traffic junction. *Systems, Man and Cybernetics, IEEE Transactions on*, 7(10):707–717.

Paruchuri, P., Pullalarevu, A. R., and Karlapalem, K., 2002. Multi agent simulation of unorganized traffic. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 1*, pages 176–183. ACM.

Pavón, J., Gómez-Sanz, J. J., and Fuentes, R., 2005. The INGENIAS methodology and tools. *Agent-Oriented Methodologies*, 9:236–276.

Rao, A. S. and Georgeff, M. P., 1992. An abstract architecture for rational agents. In *Proceedings of Knowledge Representation and Reasoning (KR&R-92)*, volume 92, pages 439–449.

Rubel, D., Wren, J., and Clayberg, E., 2011. *The Eclipse Graphical Editing Framework (GEF)*. Addison-Wesley Professional.

Schieber, R. A. and Thompson, N., 1996. Developmental risk factors for childhood pedestrian injuries. *Injury Prevention*, 2(3):228.

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

Shoham, Y., 1993. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92.

Steinberg, D., Budinsky, F., Merks, E., and Paternostro, M., 2008. *EMF: Eclipse Modeling Framework*. Pearson Education.

Tolujew, J. and Alcalá, F., 2004. A mesoscopic approach to modeling and simulation of pedestrian traffic flows. In *Proceedings of the 18th European Simulation Multi-Conference, SCS International, Ghent*, pages 13–16.

Transport Systems Planning and Transport Telematics Group, Transport Planning Group and Senozon Company, 2015. MATSim, Multi-agent transport simulation. `http://www.matsim.org/`. [Online: accessed 14-Dec-2015].

Van Den Berg, M., Hegyi, A., De Schutter, B., and Hellendoorn, J., 2003. A macroscopic traffic flow model for integrated control of freeway and urban traffic networks. In *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, volume 3, pages 2774–2779. IEEE.

Van Der Hoek, W. and Wooldridge, M., 2008. Multi-agent systems. *Foundations of Artificial Intelligence*, 3:887–928.

Vasirani, M. and Ossowski, S., 2009. 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*, pages 617–624. International Foundation for Autonomous Agents and Multiagent Systems.

Visual Solutions, Incorporated, 2015. VisSim, A graphical language for simulation and model-based embedded development. `http://www.vissim.com`. [Online: accessed 14-Dec-2015].

Wimmer, M. and Burgueño, L., 2013. Testing M2T/T2M Transformations. In *Model-Driven Engineering Languages and Systems*, pages 203–219. Springer.

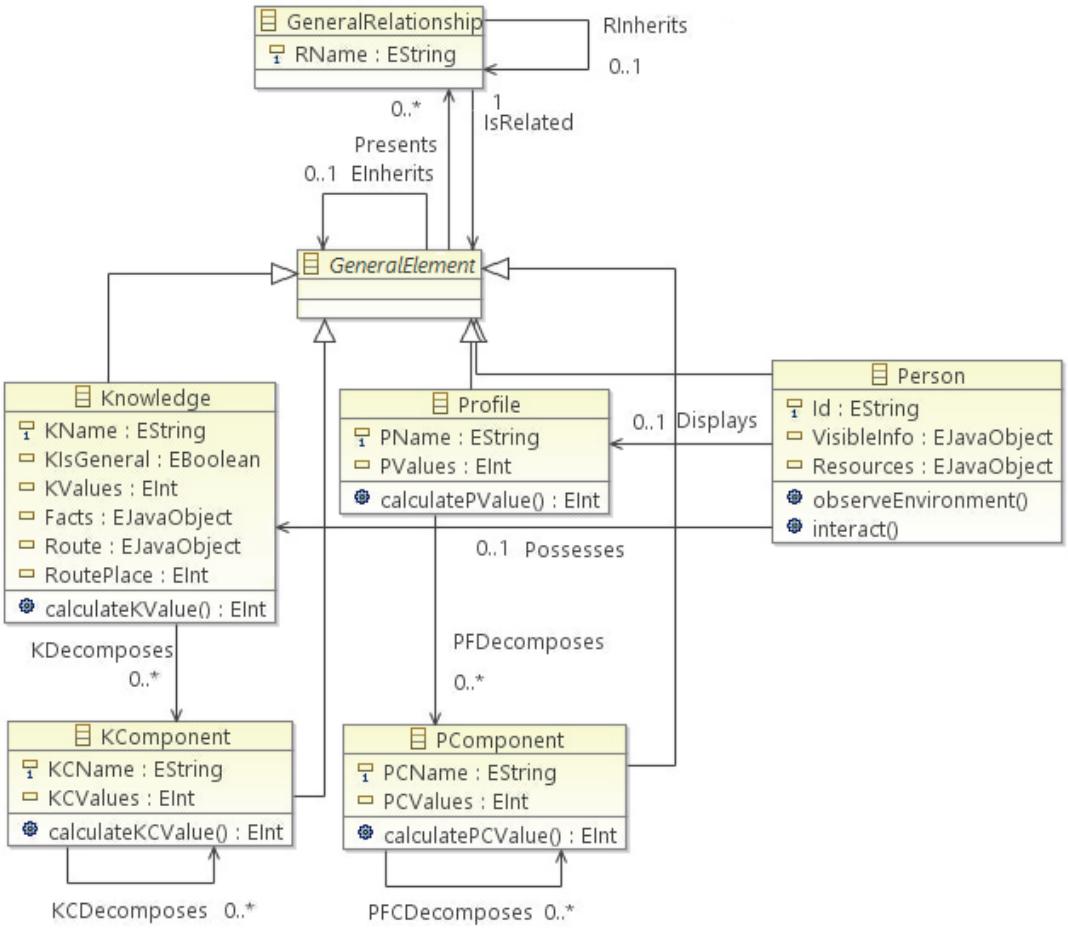

Figure 3: Excerpt of the Mental cluster

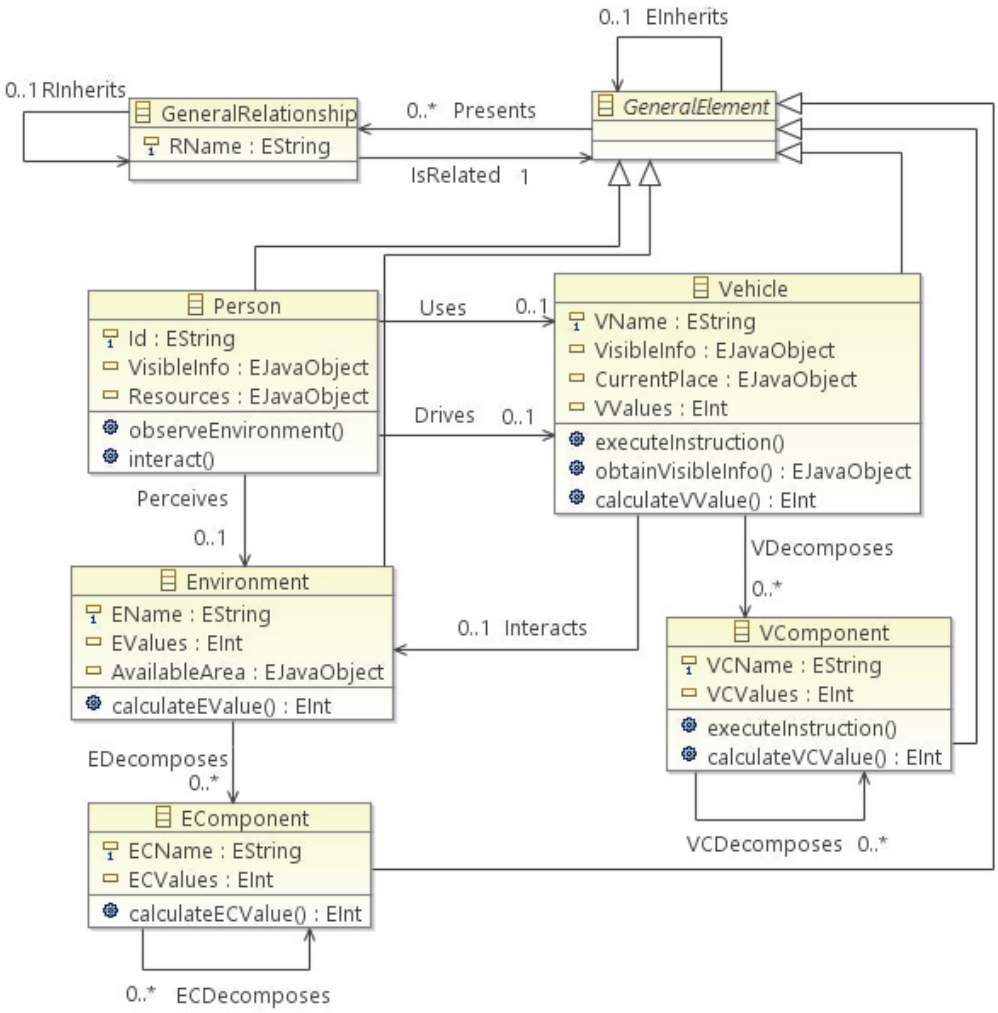

Figure 4: Excerpt of the Environment cluster.

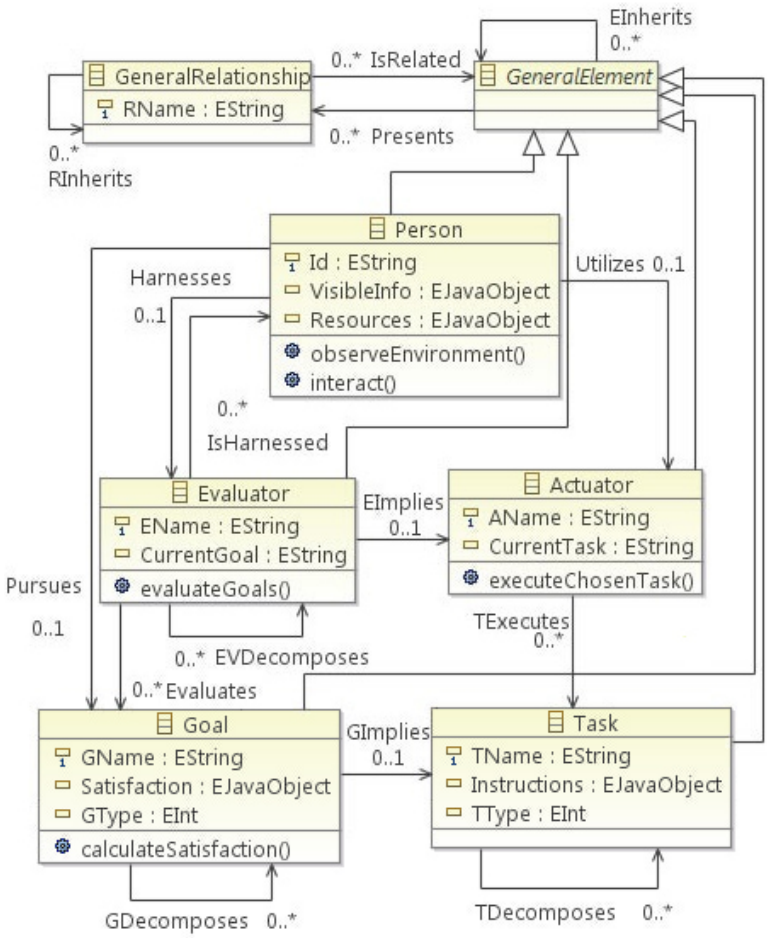

Figure 5: Excerpt of the Interactive cluster

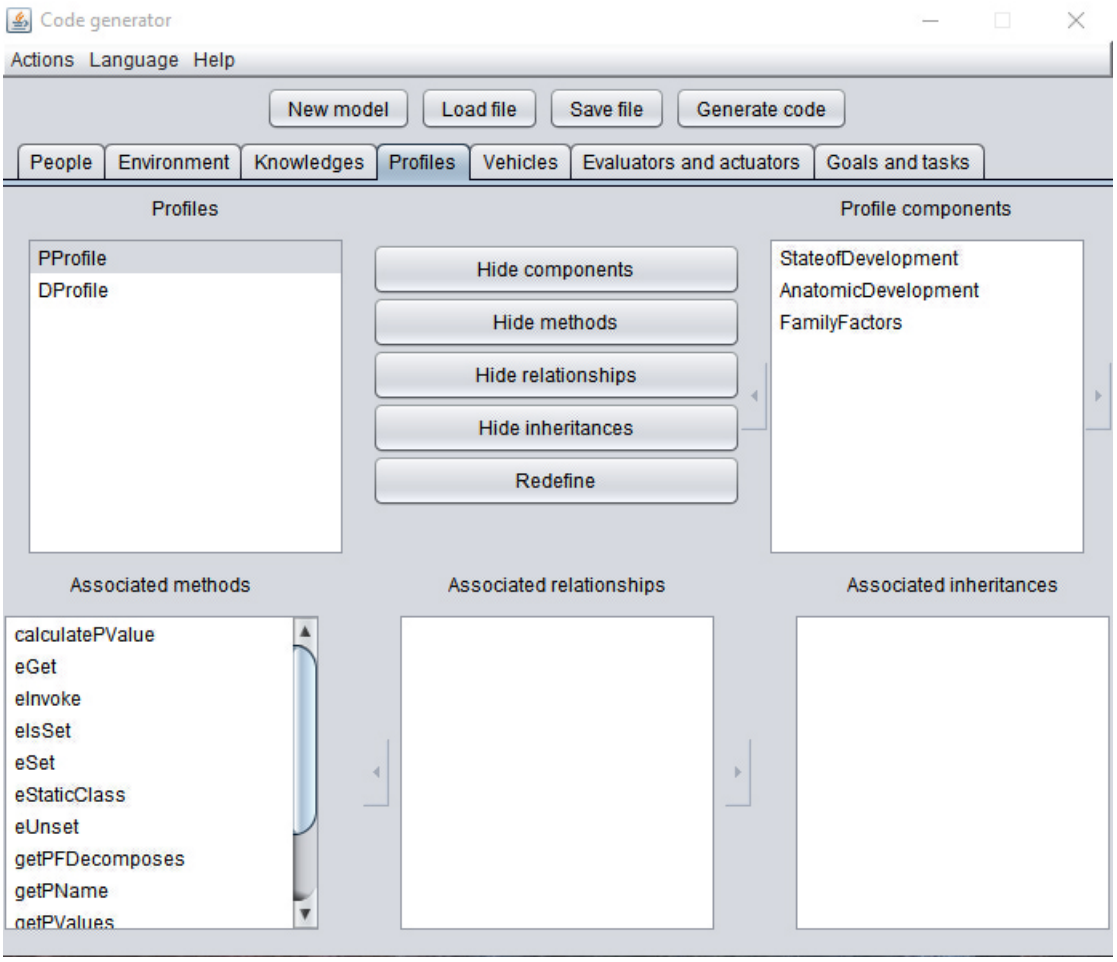

Figure 6: Snapshot of the code generator interface.

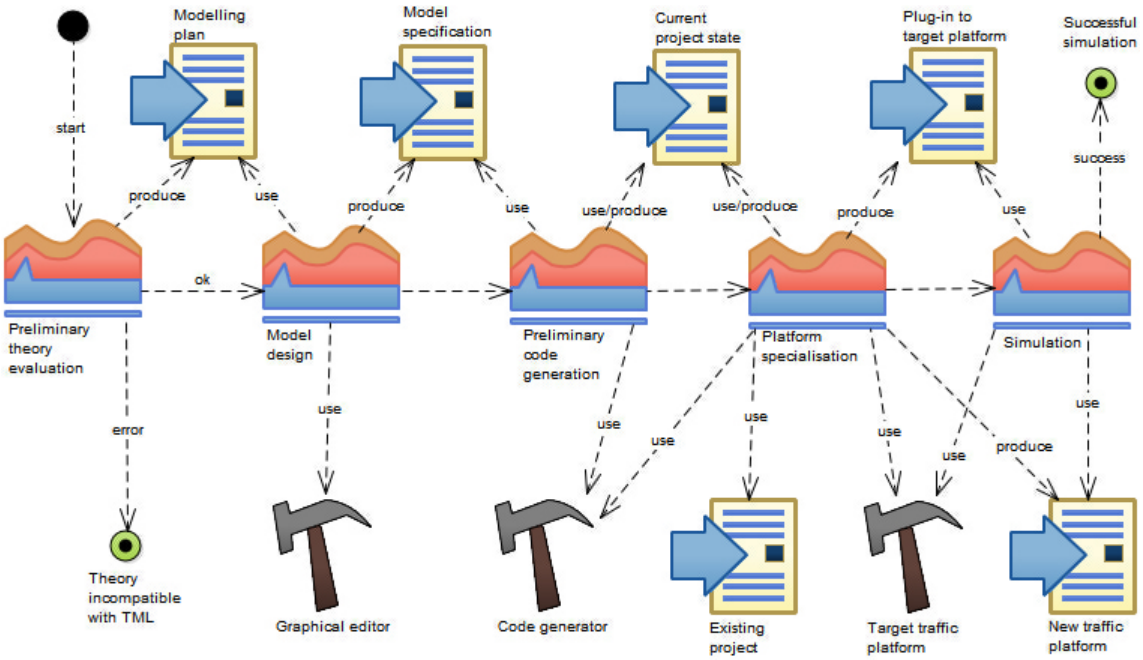

Figure 7: Main phases of the development process.

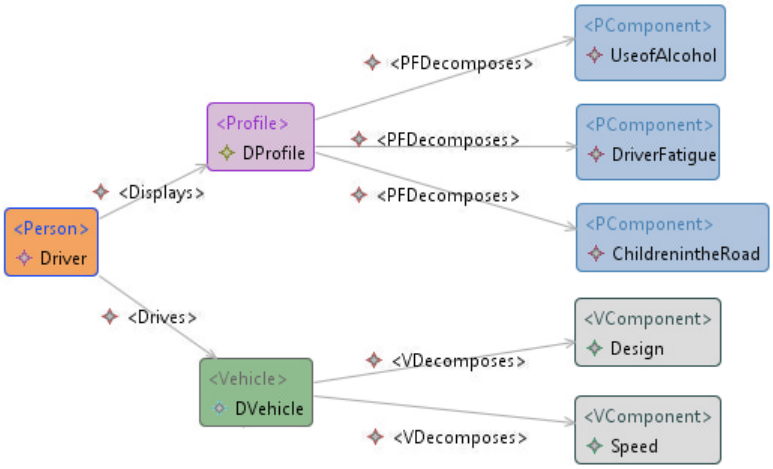

Figure 8: Excerpt of the element structure related to drivers.

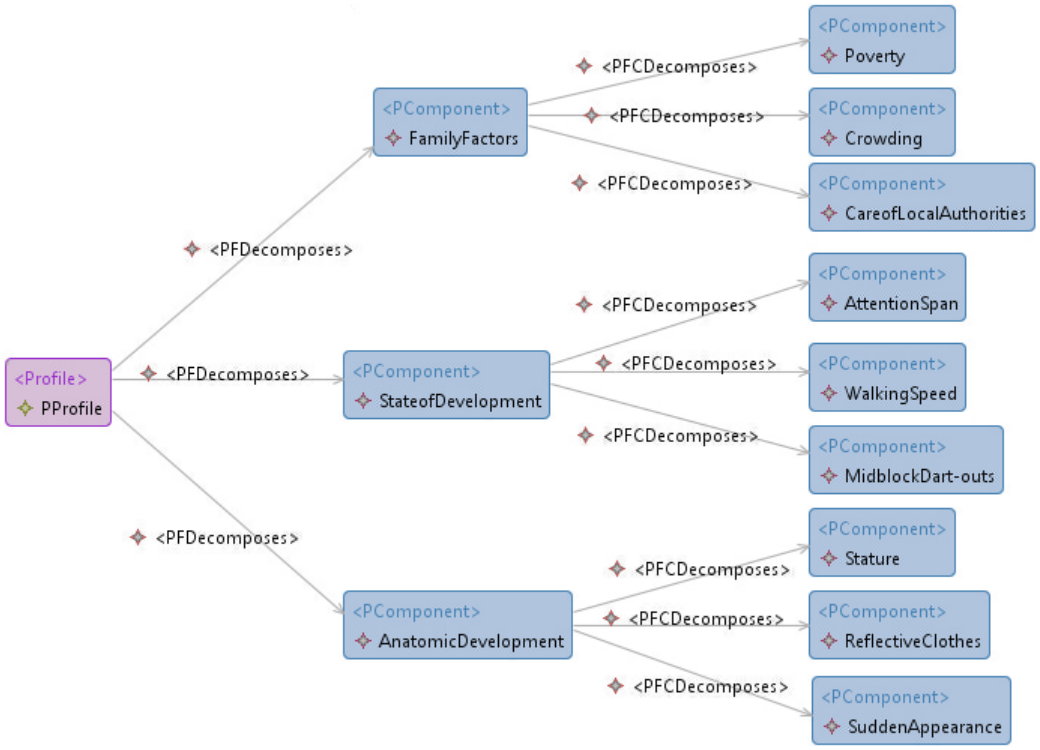

Figure 9: Excerpt of the concept structure related to pedestrians.