Model-Driven Engineering of Simulations for Smart Roads

Alberto Fernández-Isabel, Rubén Fuentes-Fernández
Department of Software Engineering and Artificial Intelligence
Univesidad Complutense de Madrid
Madrid, Spain
afernandezisabel@ucm.es, ruben@fdi.ucm.es

Abstract

Smart Roads (SRs) are systems that provide traffic-related services, based on a combination of sensor and actuator networks deployed in roads, vehicles, and surrounding elements. They are complex distributed systems that involve multiple heterogeneous components and technologies. This makes their development a challenging and costly process. Simulations are a key tool to deal with these issues, as they allow developing and testing in fully controlled environments with simplified software components. Nevertheless, they still need to consider multiple perspectives (e.g. experts and designers), which frequently cause problems to understand and validate them. Model-Driven Engineering of simulations appears as a solution. It uses models to represent explicitly these perspectives, and transformations to link them and generate new artifacts (including code). This paper presents a framework to develop simulations of SRs following this approach. Its base is an existing modeling language related to road traffic which is adapted to specify the aspects of these systems (i.e. sensors, networks, and services), and their context (i.e. users, vehicles, and their environment). A process guides its use in the transition from abstract models to code supported by tailored tools. A case study on a system to track vehicles using sensors in roads illustrates its use.

1 Introduction

The availability of affordable sensors and actuators suitable for traffic settings is leading experts to redesign the related facilities. The goal is that they become *smart* environments, able to gather and analyze information, and react to it [Figueiredo *et al.*, 2001]. Intelligent Transportation Systems (ITSs) integrate these environments to provide services like [Figueiredo *et al.*, 2001] [Varaiya, 1993] vehicle tracking, congestion detection, or identification of road conditions. In this context, Smart Roads (SRs) [Wang *et al.*, 2006] are systems where the key devices are those deployed in roads and their elements.

The development of systems for SRs (and in general of ITSs) presents multiple challenges [Figueiredo *et al.*, 2001] [Varaiya, 1993] [Wang *et al.*, 2006]. First, these are complex and distributed systems. They comprehend multiple and heterogeneous software and hardware components. Their operating conditions are changing and demanding, as they are frequently deployed over wide geographical areas and partly outdoors. Among other issues, this implies that they have to deal with the failure and redeployment of components, energy saving, and limited and intermittent connectivity. Second, they affect activities that involve living beings and uncontrolled environments, so carrying out realistic and exhaustive testing is difficult and expensive.

Simulations help to mitigate the previous problems [Pursula, 1999]. With them, experts can control and observe the relevant variables of a problem, and designers perform an incremental development of systems. However, simulations also present some important drawbacks [Axtell and Epstein, 1994]. People involved in their development have different backgrounds, e.g. authorities, traffic experts, systems designers, or programmers of simulations and control. It is difficult for them having a complete understanding of the simulation at the different levels of abstraction and its multiple facets. For this reason, there are frequent problems to validate that simulation results correspond to the initial requirements.

Model-Driven Engineering (MDE) [Schmidt, 2006] can be used to address these issues [Fuentes-Fernández *et al.*, 2012]. In these developments, participants specify systems mainly using *models*. *Transformations* perform recurrent modifications of models and other artifacts, and describe mappings among them. In the case of simulations for SRs, traffic experts would model the abstract system (i.e. independently of specific platforms), and designers would ground it to specific devices and target simulation platforms. Part of the transition between both groups of models could be automated with transformations. For instance, abstract sensors and actuators usually correspond to certain classes in the target platform. In this way, the development of simulations becomes an iterative and incremental process of refining models and transformations where all the information is explicit. Since models have a higher level of abstraction than code, and transformations describe the relevant correspondences, this approach facilitates the exchange and discussion of information on simulations and artifact reutilization.

The adoption of MDE to develop SR simulations needs to have available an infrastructure that includes several elements. Domain Specific Modeling Languages (DSMLs) define the vocabulary to specify models in a determined context. There must be also languages to describe transformations, though here both specific Transformation Languages (TLs) and general-purpose programming languages are used. Participants need tools to work with these elements, such as model editors, transformation engines, or code generators. Finally, processes guide participants in developments with these elements. This work introduces a DSML for the high-level specification of simulations of SRs. Work with it is based on a process with tailored tools.

The DSML is adapted from a previous one related to road traffic [Fernandez-Isabel and Fuentes-Fernandez, 2015]. It is formed by three clusters according to the context they consider: a *behavioral cluster*, an *environment cluster* and an *interactive cluster*. The first describes the profiles and behavior of individuals, while the second takes into account the place where the simulation occurs and its elements. The last one uses elements commonly used by Agent-Oriented Software Engineering (AOSE) [Argente *et al.*, 2009] (i.e. goals and tasks) and a perception, evaluation, action cycle to represent the decision-making of individuals.

New primitives are introduced to model the main *elements* of SRs. These are the sensors and actuators that provide the interface of the system with the external world. That world includes people, their vehicles, and the environment. The environment in turn includes, at least, roads, signals, and general conditions (e.g. weather, daytime, or type of road). Part of these concepts are extracted from research in related domains, including Agent-Based Modeling (ABM) [Axtell and Epstein, 1994], traffic simulations [Fernandez-Isabel and Fuentes-Fernandez, 2015], and sensor networks [Fuentes-Fernández *et al.*, 2009].

Most of elements in these problems are represented in the DSML as *model elements*. These are related to the original DSML or to SRs components, having the latter an internal state, and an interface with methods to consult and manipulate them. Examples of components are sensors, actuators, and vehicles. A particular type of element are the *spots*. They represent components in the environment that can be observed and manipulated by devices (i.e. sensors and actuators). For instance, a tracking magnetic sensor can detect the passing of a bodywork *spot* of a vehicle.

SRs elements with complex behaviors are modeled as *agents*. They are defined in a similar way of individuals involved in road traffic (i.e. in terms of the goals they pursue and the information they have), and their capabilities to manipulate both their internal and the environment states.

The language also includes general mechanisms of inheritance between concepts and definition of instances of types in the adaptation to SRs. They facilitate the modification to different modeling needs through extensions of the language and the specification of simulations using models.

The related tools are a model editor and a code generator. Experts use the first one to specify graphically models compliant with the DSML. It is based on the INGENME [Pavón *et al.*, 2011] meta-editor. Designers and programmers use the generator (adapted from [Fernandez-Isabel and Fuentes-Fernandez, 2015] and based on Eclipse [Steinberg *et al.*, 2008] frameworks) to map model elements to code templates. These templates are fragments of code with marks corresponding to primitives of the DSML. Then, the code generator reads the models and mappings, instantiates the templates, and generates the code of simulations.

The case study that illustrates this approach is the simulation of the system in [Karpiriski *et al.*, 2006]. That work presents an architecture with road sensors to track vehicles. This case study models the system and generates its code for the JADE agent platform [Bellifemine *et al.*, 2007]. This illustrates how working in this way facilitates understanding the different aspects of the simulation and reduces the effort to code it.

The rest of the paper is organized as follows. Section 2 makes an introduction to MDE. Section 3 presents the DSML, while Section 4 the development guidelines of the proposal. Section 5 describes the support tools, and Section 6 applies the framework to the case study of tracking vehicles. Then, Section 7 compares the approach and its results with related work. Finally, Section 8 discusses the conclusions and future work.

2 Background

MDE [Schmidt, 2006; Kent, 2002] is an approach to software development based on *models* and *transformations*. Models are specifications of information regarding the system to build. Transformations are automated modifications of models and other artifacts to generate new products. In this context, developers work specifying their models incrementally, and running transformations to integrate models or perform certain modifications (e.g. adding design information or generating code). All the information relevant for the development is thus presented as models or transformations, so developers have it explicitly described. This improves traceability between artifacts across development. Working effectively in this way requires having support tools for certain tasks.

Models are described following Modeling Languages (MLs) that establish their primitives and constraints, so all developers can interpret them in similar ways. Model editors support developers when specifying models, and guarantee the compliance of models with their MLs. In order to allow this functionality, MLs are defined formally. There are alternatives for this definition depending on the ML features and the context and needs of its use. Domain Specific Modeling Languages (DSML) [Luoma *et al.*, 2004] are MLs oriented only to one context.

Graphical graph-oriented MLs are the most popular ones in contexts such as Software Engineering and graphical simulations. Their models specify graphs where entities are connected by links, and all of them can have related properties. Metamodels are the most widely used means to specify these languages [Steinberg *et al.*, 2008]. The description of metamodels relies on meta-modeling languages such as the Meta-Object Facility (MOF), Ecore [Steinberg *et al.*, 2008], or Graph-Object-Property-Relationship-Role (GOPRR) [Smolander, 1993]. MOF is used by the Object-

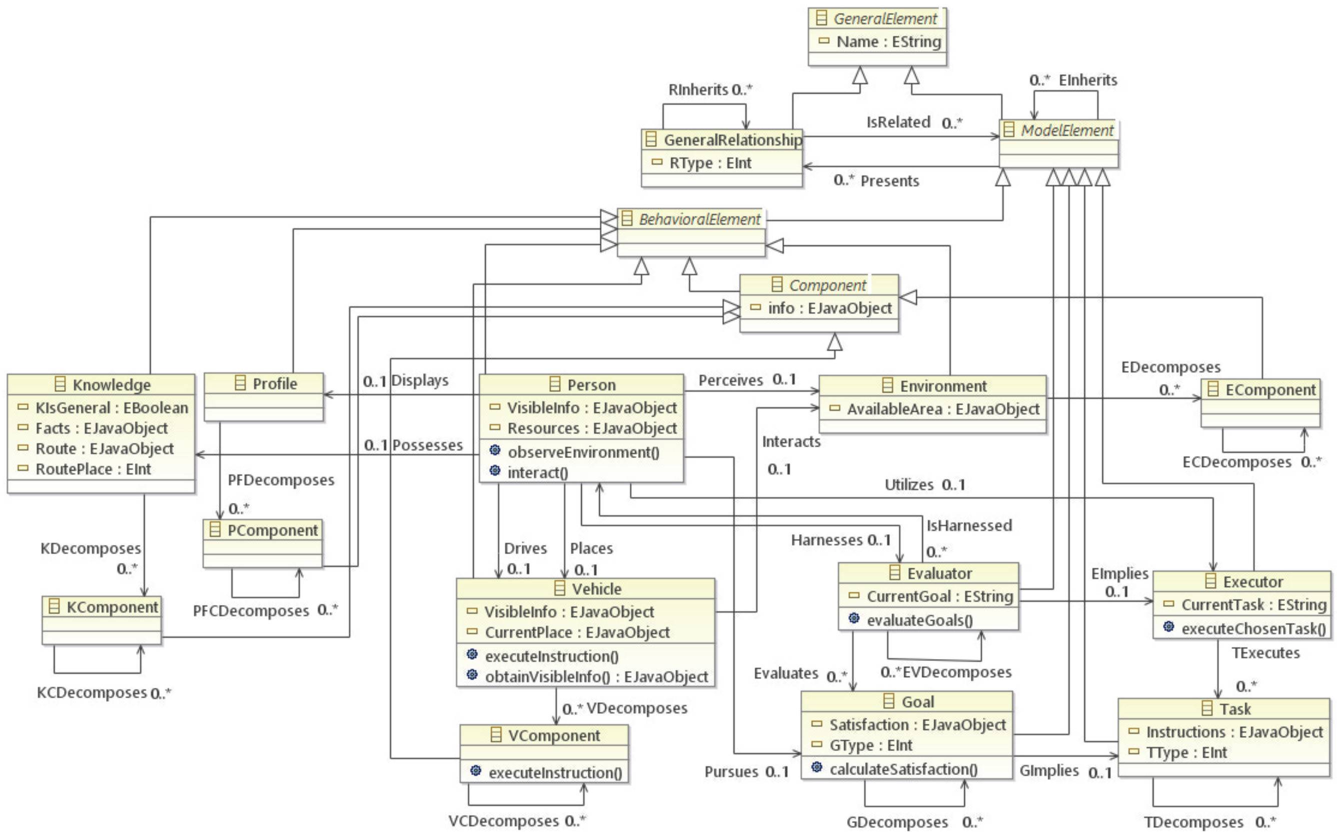

Figure 1: Excerpt of the road traffic metamodel.

Management Group (OMG) to define standards such as the Unified Modeling Language (UML). Ecore is almost aligned with Essential MOF, a subset of MOF. It is supported by the Eclipse communities related to MDE with a complete and widely supported set of tools. GOPRR has a richer set of primitives than the previous two languages, as it allows for instance the direct definition of n-ary relationships. INGENME [Pavón *et al.*, 2011] is a framework for it.

The implementation of transformations has two main approaches. They can be implemented as modules in mainstream programming languages; or they can be described with specific transformation languages and executed by an engine. The first approach reuses existing expertise and resources, and it is usually more efficient. Examples of it are INGENME [Pavón *et al.*, 2011] and the traffic simulation framework in [Fernandez-Isabel and Fuentes-Fernandez, 2015], both based on Java and XML. The second approach makes easier to examine the mappings between the source and target artifacts of the transformation. Examples of it are Eclipse projects such as JET and ATL [Steinberg *et al.*, 2008].

This work adopts GOPRR as its meta-modeling language. Its tools are based on the INGENME MDE framework.

3 Domain specific modeling language

The foundations of the DSML for SRs is adopted from another DSML focused on general purpose road traffic [Fernandez-Isabel and Fuentes-Fernandez, 2015]. It is enhanced modifying its structure and introducing various abstract entities. The *model element* is the main one. It allows describing elements from road traffic (e.g. profile and knowledge of individuals involved in traffic) and specific components related to SRs (e.g. sensors and spots).

Regarding the naming, nodes are the meta-classes and links are meta-relationships among them. Meta-relationships with triangles represent inheritance and with filled diamonds aggregation. The attributes and adornments of the previous elements are meta-properties. It is specified with a GOPRR metamodel, being this notation similar to that also used in MOF and Ecore [Steinberg *et al.*, 2008].

The original metamodel is introduced in Section 3.1 where its structure and meta-classes are explained. The DSML extension developed to represent the SRs components is described in Section 3.2.

3.1 Traffic DSML

The DSML is originally focused on modeling the behavior of individuals involved in road traffic (i.e. drivers, pedestrians and passengers). It is a flexible language that presents specific mechanisms to ease its fitness to the large amount of theories evaluated by traffic studies.

It is described by a metamodel [Steinberg *et al.*, 2008] which provides a set of meta-entities in order to represent the

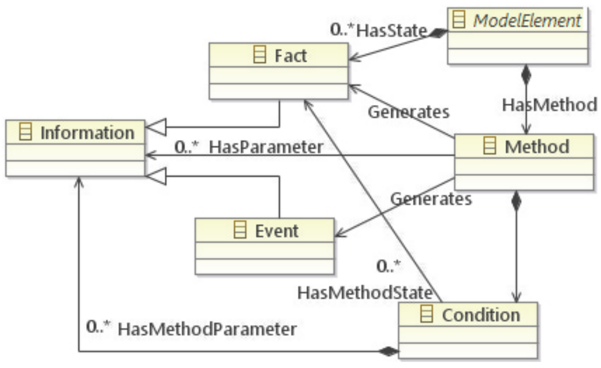

Figure 2: *Information* and *ModelElement* related elements.

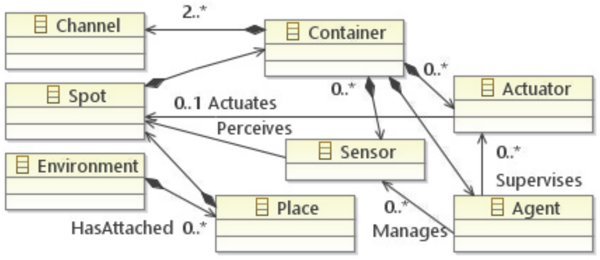

Figure 3: *Environment* and *Container* related elements.

notions, relationships, properties and explicit constraints.

Metamodel concepts are inspired in AOSE [Argente *et al.*, 2009] and are classified into three clusters. The *mental cluster* (i.e. *profile*, *knowledge* and their *components*) is based on [Shinar, 1978] and considers the features and internal state of individuals. The *environment cluster* (i.e. *environment*, *vehicle* and their respective *components* describes the elements extracted from DVE model [Amditis *et al.*, 2010]. The *interactive cluster* (i.e. *evaluator*, *executor*, *goal* and *task*) represents the decision-making of individuals. It includes a perception, evaluation, and acting cycle.

Regarding the metamodel bedrock, it revolves around the *person* notion (see Fig. 1). It symbolizes a kind of human being involved in road traffic. According to their means of transport and how they interact with them, these people can take different roles (i.e. drivers, passengers, or pedestrians). Thus, *person* are able to interact with an *environment*. This interaction is immediate (for pedestrians) or indirect (in the case of drivers and passengers). The information people have is illustrated with the *knowledge*, while their features are represented by the *profile*. The purposes of people involved in traffic are described by *goals*, and the actions to carry out them by *tasks*. *Evaluators* study the information obtained from the *environment* and decide how the individuals must act according to it. *Tasks* for achieving their implicit instructions are picked up by an *executor*.

The metamodel uses inheritance hierarchies with the purpose of providing notion specializations and a flexible structure. The main one is the *general element* from which the *model element* and the *general relationship* extend (see Fig. 1). The first acts as a basis for the traffic DSML and the SRs DSML parts and is extended to the *behavioral element* and the meta-classes involved in the *interactive cluster*. The *behavioral element* is the parent meta-class of the main elements (i.e. not *components*) that compound the *mental* and *environment* clusters. *Component* extended from it and is also the parent of the *components* of both clusters. The second supports introducing relations (e.g. affect or impact) between the rest of entities that are extended from *model element*.

Another kind of hierarchies are considered in this part of the metamodel. In the both *mental* and *environment* clusters composition hierarchies are introduced between main elements (e.g. *profile* or *vehicle*) and their respective *components* (e.g. *pcomponent* or *vcomponent*). These *components* can be decomposed into others of the same type, promoting the creation of complex structures.

3.2 DSML extension to SRs

This module of the DSML is mainly focused on people moving in their vehicles in roads unless there are other elements in the environment such as traffic signals, obstacles, and weather. These elements can be observed with sensors, and systems that can actuate on them using actuators. The DSML makes of these concepts its core categories.

Elements are modeled in terms of the information they manage. There are two basic types of *information*: *facts* are internal to elements, and *events* can be perceived from outside.

The root concept that embraces both modules of DSML is the *ModelElement* entity (see Fig. 2). In this case, it is characterized in terms of an identifier, an internal *state*, an interface compose of *methods*, and the *events* it can generate. The internal state is a set of *facts*. A method is defined by its parameters and results, which are *information*. It can also have execution *conditions* defined in terms of their parameters and the internal state of its model element. Methods can be internal (only accessible from the component) or external (accessible from other components).

The *environment* meta-class of the traffic DSML is related to a set of elements over a *map*. These are the *places*, and can be located in *sections* or *junctions*. Examples of places are things in the environment, like the road surface or protective fence. *Environment* has attributes (e.g. *AvailableArea*) to store the relevant information of the *map*. This latter is represented through a graph that describes the road *sections* that link two *junctions* (one when the section in an entering or exit point).

Places contain *spots* (see Fig. 3). These are the components that sensors can actually observe and where actuators can act upon. For instance, a vehicle have several *spots*, e.g. the bodywork, the electronic system, or the engine. *Sensors* and *actuators* constitute the interface of systems with the external environment. They run on containers attached to *spots*. Besides this, a *sensor* is linked by the *perceives* relationship to the *spot* it observes, and an *actuator* by the *actuates* relationship to the *spot* it affects. In both cases, *sensors* and *actuators* access to the external interface of the *spot*, i.e. they use its external *methods*. For instance, a *rain sensor* runs in a *container* of the bodywork, where it perceives the raindrops from an *abstract weather spot*. The *containers* of a system are linked through *communication channels*.

Beyond these elements, complex entities related to SRs are modeled as *agents*. Typical agents are the controllers of sensors and actuators. The control of components can be represented directly with their methods, but controller agents are recommended when there are complex algorithms and communication with other controllers.

Similarity to a *person* (see Fig. 1), an *agent* has an identifier, and *goals* that it can achieve through *tasks* that manipulate information. As the environment of SR systems is unpredictable, the execution of a task can fail or not to produce the expected results. Thus, a goal defines satisfaction conditions in terms of information to indicate when it has been fulfilled.

Tasks can be organized (or decomposed) into others according to the traffic DSML (see Fig. 1). Also, they can be linked to *methods* related to the information they produce and consume. In this way, tasks can use methods of components, including sensors and actuators.

Agents communicate among them using *notifications*. These are a type of event addressed to a certain *agent* identifier.

Agents behave internally following a perceive-reflect-act cycle. First, they execute those tasks that imply access to external components (e.g. sensors and other elements related to SRs) to gather data. Then, they update their internal state, both *facts* and *goals*. Finally, they pre-select for execution those *tasks* that can satisfy some of their still non-fulfilled goals. Among them, they choose one to actually execute.

The external elements that *agents* can access are those linked to them using *manages* relationships. For controllers, these relationship can be only with other elements in their *containers*. In contrary, *persons* can relate only to *sensors* and *actuators* from *containers*, or other elements outside *containers* like *vehicles* or *components* in the *environment*.

This part of the language also includes some general mechanisms applicable to most of the previous concepts. It supports inheritance of concepts and relationships to allow their specialization. For instance, the concept of component can have additional and different features according to the target simulation platform.

4 Development guidelines

The framework to develop SR simulations provides guidelines to model using the previous DSML and support tools. They include 11 activities. The process is decomposed in two different stages. The first one (nodes 1-9) is focused on the expert work and specifications with the proposed DSML. The second one (nodes 10-11) deals with the design of simulation. After concluding the first one with the specific infrastructure of this work, the second part can be addressed with a MDE methodology for general software development. Given that our DSML oriented to SRs follows ABM [Axtell and Epstein, 1994], methodologies from AOSE are a suitable choice [Argente *et al.*, 2009]. Both ABM and AOSE make of agents their core concept. Though there are differences among specific works, most of them conceptualize agents in terms of mental entities and communication capabilities, and consider the existence in their environment of artifacts they can use. This common core facilitates the transition from abstract to design models with transformations.

The first stage is organized around the services that the SR should provide. It starts identifying potential services pending to specify (activity 1). If there are any, work follows with its definition in terms of the information it needs and it provides, and the actions it should take (activity 2). This information and actions appear in elements of the system and its environment that next activities specify.

The service interacts with *spots*, either observing or changing them. Experts identify them and their potential *containers*, and specify them as *model elements* (activity 3).

The service also communicates with *spots* using the system *sensors* (described in activity 4) and *actuators* (in activity 5). These devices are initially specified as *model elements*. In case that their functioning needs complex control or communications with other containers, they also need controller *agents*. *Channels* must be added for those *containers* that need to be linked.

The *spots* identified in previous activities are located in elements of the SR environment. These elements are *places*, *vehicles* (considered in activity 6), *persons* (activity 7) and (in activity 8) other *components* from environment (i.e. *ecomponents*). All of them are specified as *model elements*. They also have a location in the environment. Thus, these activities also define the *map* and locate the places in it.

The last element to specify is the behavior of individuals (in activity 9). It is defined in terms of the *goals* and *tasks* adapting existing road traffic theories [Fernandez-Isabel and Fuentes-Fernandez, 2015], and the steps of the perception, evaluation and acting cycle. As people act on the environment, tasks to check the actual result of their actions and update its information must be included (e.g. route path or position in the environment).

When all the services have been identified, the process can move to the design of the simulation. Transformations map abstract to design models (activity 10). These transformations can be reused when they are available from other projects with the same target AOSE methodology. Then, the design models act as the initial specification for the simulation in that methodology (activity 11). For instance, our work can be easily linked to the INGENIAS AOSE methodology [Pavón *et al.*, 2005]. The INGENME [Pavón *et al.*, 2011] infrastructure our work uses is the same of INGENIAS, and both share similar definitions of concepts such as *agent*, *goal*, *task*, and *fact*.

5 Support tools

This MDE approach uses two development tools to achieve the different steps of the process (see Section 4). A model editor based on INGENME supports the specification of models compliant with the DSML. It is the main tool of the first stage. For the second stage, most of AOSE methodologies have their own tailored model editors and transformation tools. Here, the last steps of code generation are achieved with a

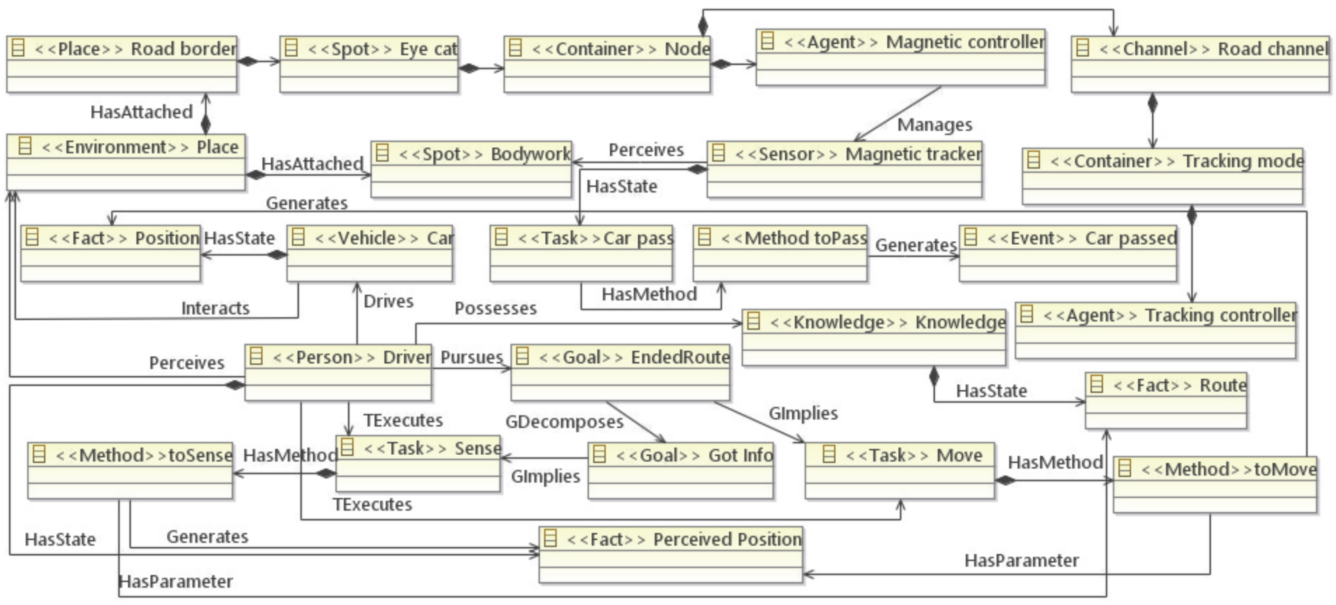

Figure 4: Excerpt of the simulation model for the system to track vehicles. Stereotypes indicate DSML types.

code generator adapted from [Fernandez-Isabel and Fuentes-Fernandez, 2015]. Then, designers specify graphically the mappings from elements in models to classes in the target platform. The generator outputs the simulation code from this information.

The INGENME model editor is based on INGENIAS methodology. It present a graphical canvas where models can be captured. The resulting outcome can be exported as a XML file. It promotes the modularity of the proposal and the ability to maintain the independence among the different artifacts created.

The code generator is a graphical tool implemented in Java which provides an engine to generate source code from a model specification. In this case, the specification comes in form of XML file from the model editor. This tool is mainly focused on easing the work to developers. In order to that, it provides an intuitive navigation through the elements of the models, and uses multiple wizards to guide the users in the achievement of the most complex tasks. It also takes as input a metamodel, code templates compliant with it and libraries from the target simulation platform. The templates support the preliminary automated code generation while the libraries can be used to adapt the simulation platform to the models requirements. To achieve it, the tool offers the possibility of creating new classes (empty or extended) through which producing a new simulation platform specialized in specific models [Fernandez-Isabel and Fuentes-Fernandez, 2015].

6 Case study

This section applies the previous framework (see Sections 3 and 4) to develop the simulation of one of the services for SRs described in [Karpiriski *et al.*, 2006]. It is a vehicle tracking service based on magnetic sensors located in nodes of cat eyes every few meters in road borders. These sensors are able to perceive car passing. Nodes know their relative order and distance to others. They make up an ad-hoc network to exchange information, so they can determine the position and speed of vehicles.

The first stage of the process generates the abstract model of the simulation with the DSML. Fig. 4 shows part of it.

Activities 1 and 2 identify the services the SR offers. The definition of the problem points out only to the *tracking vehicle* service. Activity 3 identifies the *spots* related to it. There are two. The sensors are placed in *eye cats* in the road borders. These sensors track the passing of vehicles sensing their metal, for instance in their *bodywork*. Activities 4 and 5 consider the related *sensors* and *actuators*. There are *magnetic tracker sensors*, but no *actuators*. The specification of *car vehicles* in activity 6 does not consider any particular feature of them. In the proposed DSML, decisions regarding maneuvers are placed in *persons*. Activity 7 models them. The original work does not introduce any specific model of drivers. The problem also identifies elements in the environment (see activity 8) and models them in particular as *ecomponents* (see Section 3.1) or *places* (e.g. *road borders*).

The previous first round allows identifying the main concepts of the problem, and their types and relationships. A second round is focused on their state and functionality. Given that individuals with driver role and their actions trigger most of activities in the system, the analysis starts with them.

A simple path-following behavior is proposed for *persons* with driver role. Following a perception, reflect and acting cycle (no evaluation is considered), there is a first step of calculating position and a second of moving. The first one corresponds to *goal got info* and *task sense*, and the second to *goal ended route* and *task move* (see Fig. 4). *Sense* generates the *fact perceived position* consulting the *method toSense*. This fact is compared with the *fact route* to determine if the car has arrived to its destination. That is the satisfaction condition of *goal ended route*. If the goal is unfulfilled, the *task move* can be triggered. It calls a *method toMove* that updates the *car* and its *fact position*.

In the system, several elements do not have specific state or methods, as only their location is relevant. This is the case of *road border* and *eye cat*. In order to have a precise location of sensors, road borders are modeled as multiple elements of this type, each one with its own location.

The *eye cat spot* has *node containers* for the sensors and controllers of the system. The *magnetic tracker sensor* triggers events when it perceives a car. Its *magnetic controller* has a *task pass car* to sense that event (see Fig. 4), and generates through the *toPass method* a *car passed event* addressed to a *central tracking controller* (not shown in the diagram). This would implement the services for end users described in [Karpiriski *et al.*, 2006]. Given the constraints imposed by the DSML, the model needs to introduce a *tracking node container* for this last controller. In order to enable communication between controllers in different containers, these are connected with a *road channel*.

The previous discussions has not considered the *map*. The studied work focuses on two-way single carriageways. With the DSML, these correspond to *junctions* that connect at least two *sections*, one for each direction. Crossroads are *junctions* that connect more *sections*. The previous *model elements* (vehicles and road borders) are placed in them.

These steps complete the first stage and the abstract model of the simulation. Activity 10 maps the abstractions of that model to those of INGENIAS [Pavón *et al.*, 2005]. Then, activity 11 follows the steps of this methodology adding the design information required for the JADE platform [Bellifemine *et al.*, 2007]. There are several entities of the DSML with direct mappings to JADE: agent to agent and task to behavior. Others need to be mapped to specific ad-hoc classes. This is the case of the model elements, facts, goals, spots, sensors and actuators. Their implementation only requires attributes and methods to get and set their values, as agents manage the updates of the simulation state.

7 Related work

The presented framework is mainly related to the modeling and development of SRs. This section discusses existing alternatives for them.

Under the label of SRs literature presents a variety of complex heterogeneous systems [Figueiredo *et al.*, 2001] [Varaiya, 1993] [Wang *et al.*, 2006]: they provide different services, for a wide range of users, and integrating multiple devices. Nevertheless, several common elements can be abstracted in most of them [Varaiya, 1993] [Wang *et al.*, 2006] [Sun *et al.*, 2006]. There are sensors and actuators embedded in an environment that includes roads and their elements, weather, vehicles, and sometimes people. Vehicles and people are usually modeled focusing on their movement. In the case of simulation [Kotusevski and Hawick, 2009], works offer more complex models of vehicles and people moving, with paths to follow and principles of movement (e.g. collision avoidance or observe traffic norms). These elements are considered in the proposed DSML, though the use of *components*, *agents*, and *information* makes possible setting up richer models than in other works. The DSML also offers extension mechanisms frequently disregarded in other works.

In most cases, there is no information about the adopted development process (see for instance the already mentioned works). However, this is a key aspect to evaluate approaches regarding, for instance, ease of adoption and modeling or costs. When there is some information on that [Pursula, 1999] [Kotusevski and Hawick, 2009], it usually shows approaches with a manual transition from abstract models to code, and focused on the later. The advantages of MDE in this context were pointed out in the introduction: explicit definition of all the information related to that transition, which facilitates its discussion and reutilization, and support tools for it. There are already some works in this line in the context of traffic studies [Fernandez-Isabel and Fuentes-Fernandez, 2015] [Vangheluwe and De Lara, 2004]. Their main differences with our work are their focus and purposes (general traffic versus SRs) and the use of infrastructures less adopted than ours (in [Vangheluwe and De Lara, 2004] graph rewriting grammars for transformations), which hinders their adoption.

8 Conclusions and future work

This paper has introduced a framework for the model-driven development of simulations of SRs. It is based on a DSML and tailored standard infrastructures.

The DSML adopts a previous one focused on road traffic that integrates concepts from studies and simulations of traffic [Fernandez-Isabel and Fuentes-Fernandez, 2015]. It is adapted through ABM concepts (e.g. agents) [Axtell and Epstein, 1994] and elements related to sensor networks [Fuentes-Fernández *et al.*, 2009]. Based on a general concept of *model element* (could be a modeling entity or a component with state and an interface that manipulates information), it introduces new concepts related to sensors, actuators, spots in the environment and vehicles. A higher level of abstraction comes from agents, which are used to describe complex controllers.

The framework also proposes a development process based on this DSML. It comprehends a specific first stage to specify the abstract models of simulations, and connects with an AOSE MDE methodology [Argente *et al.*, 2009] for the low level design and code generation. A model editor based on INGENME [Pavón *et al.*, 2011] and a code generator supports it. This latter is a graphical engine based on wizards that allow guiding users in some of the complex steps of the development process.

The case study has shown how a simulation can be specified to a large extent with the DSML, including the behavior of its components. Only algorithms to manipulate information are demoted to code templates. This approach focus development efforts on models and transformation, which can be reused more easily than code.

The previous work has still several open issues. The SR module of the DSML needs extensions to consider aspects such as time or specific constraints. There are studies on these issues but further work is required to integrate them. The process needs to incorporate additional advice on how to use the DSML and deal with the design. In particular, that design should be consistent with the expected semantics of the DSML. Finally, additional experiments are needed with other works and target platforms to validate the approach. Specifically, the combination of road traffic theories and services for SRs taking advantage of the potential of the DSML described here (it is a general purpose traffic DSML adapted to SRs) is another important next step to consider.

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. Also, we acknowledge support from the Programa de Creacin y Consolidacin de Grupos de Investigacin (UCM-BSCH GR35/10-A).

References

[Amditis *et al.*, 2010] Angelos Amditis, Katia Pagle, Somya Joshi, and Evangelos Bekiaris. Driver–vehicle–environment monitoring for on-board driver support systems: Lessons learned from design and implementation. *Applied Ergonomics*, 41(2):225–235, 2010.

[Argente *et al.*, 2009] Estefanía Argente, Ghassan Beydoun, Rubén Fuentes-Fernández, Brian Henderson-Sellers, and Graham Low. Modelling with agents. In *Agent-Oriented Software Engineering X*, pages 157–168. Springer, 2009.

[Axtell and Epstein, 1994] Robert L Axtell and Joshua M Epstein. Agent-based modeling: understanding our creations. *The Bulletin of the Santa Fe Institute*, 9(2):28–32, 1994.

[Bellifemine *et al.*, 2007] Fabio Luigi Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing multi-agent systems with JADE*, volume 7. John Wiley & Sons, 2007.

[Fernandez-Isabel and Fuentes-Fernandez, 2015] Alberto Fernandez-Isabel and Ruben Fuentes-Fernandez. 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, 2015.

[Figueiredo *et al.*, 2001] Lino Figueiredo, Isabel Jesus, JA Tenreiro Machado, J Ferreira, and JL Martins De Carvalho. Towards the development of intelligent transportation systems. In *Intelligent Transportation Systems*, volume 88, pages 1206–1211, 2001.

[Fuentes-Fernández *et al.*, 2009] Rubén Fuentes-Fernández, María Guijarro, and Gonzalo Pajares. A multi-agent system architecture for sensor networks. *Sensors*, 9(12):10244–10269, 2009.

[Fuentes-Fernández *et al.*, 2012] Rubén Fuentes-Fernández, Samer Hassan, Juan Pavón, José M Galán, and Adolfo López-Paredes. Metamodels for role-driven agent-based modelling. *Computational and Mathematical Organization Theory*, 18(1):91–112, 2012.

[Karpiriski *et al.*, 2006] M Karpiriski, Aline Senart, and Vinny Cahill. Sensor networks for smart roads. In *Pervasive Computing and Communications Workshops, 2006. PerCom Workshops 2006. Fourth Annual IEEE International Conference on*, pages 5–pp. IEEE, 2006.

[Kent, 2002] Stuart Kent. Model driven engineering. In *Integrated formal methods*, pages 286–298. Springer, 2002.

[Kotusevski and Hawick, 2009] G Kotusevski and KA Hawick. A review of traffic simulation software. *Computer Science. Institute of Information and Mathematical Sciences, Massey University*, 2009.

[Luoma *et al.*, 2004] Janne Luoma, Steven Kelly, and Juha-Pekka Tolvanen. Defining domain-specific modeling languages: Collected experiences. In *4 th Workshop on Domain-Specific Modeling*, 2004.

[Pavón *et al.*, 2005] Juan Pavón, Jorge J Gómez-Sanz, and Rubén Fuentes. The ingenias methodology and tools. *Agent-oriented methodologies*, 9:236–276, 2005.

[Pavón *et al.*, 2011] Juan Pavón, Jorge Gómez-Sanz, and Adolfo López Paredes. The sicossys approach to sos engineering. In *System of systems engineering (SoSE), 2011 6th international conference on*, pages 179–184. IEEE, 2011.

[Pursula, 1999] Matti Pursula. Simulation of traffic systems-an overview. *Journal of Geographic Information and Decision Analysis*, 3(1):1–8, 1999.

[Schmidt, 2006] Douglas C Schmidt. Model-driven engineering. *COMPUTER-IEEE COMPUTER SOCIETY-*, 39(2):25, 2006.

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

[Smolander, 1993] Kari Smolander. Goprr: a proposal for a meta level model. *University of Jyväskylä, Finland*, 1993.

[Steinberg *et al.*, 2008] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. *EMF: eclipse modeling framework*. Pearson Education, 2008.

[Sun *et al.*, 2006] Zehang Sun, George Bebis, and Ronald Miller. On-road vehicle detection: A review. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(5):694–711, 2006.

[Vangheluwe and De Lara, 2004] Hans Vangheluwe and Juan De Lara. Computer automated multi-paradigm modelling for analysis and design of traffic networks. In *Proceedings of the 36th conference on Winter simulation*, pages 249–258. Winter Simulation Conference, 2004.

[Varaiya, 1993] Pravin Varaiya. Smart cars on smart roads: problems of control. *Automatic Control, IEEE Transactions on*, 38(2):195–207, 1993.

[Wang *et al.*, 2006] Fei-Yue Wang, Daniel Zeng, and Liuqing Yang. Smart cars on smart roads: an ieee intelligent transportation systems society update. *IEEE Pervasive Computing*, (4):68–69, 2006.