Computer Science and Information Systems 14(1):219–237 DOI: 10.2298/CSIS161010001F

Extending a Generic Traffic Model to Specific Agent Platform Requirements

Alberto Fernández-Isabel and Rubén Fuentes-Fernández

Complutense University of Madrid
Department of Software Engineering and Artificial Intelligence
c/ Profesor José García Santesmases, 9, 28040, Madrid, Spain
afernandezisabel@ucm.es, ruben@fdi.ucm.es

Abstract. Road traffic and its influence over individuals is an important aspect of our life nowadays. Its study in order to understand its dynamics and the factors that affect it is a relevant field of research. Traffic simulations have become a fundamental tool for these studies. They provide a controlled environment to analyse traffic settings. However, they present some shortcomings. One of the main ones is the need of multidisciplinary groups of experts to work with complex models. Communication problems and misunderstandings frequently appear in them, which produce mistakes and bring increased costs. Some works have addressed these issues adopting abstract concepts that can act as bridges among different groups to model and implement simulations. Works that use intelligent agents to represent individuals, and their related simulation platforms, belong to this category. Nevertheless, these platforms are still programmer-oriented, so other experts find difficult to ground their abstract models in them. As a further step, Model-Driven Engineering (MDE) has been proposed to work with models and simulations. It offers the possibility of working with models at multiple levels of abstraction and focused on different aspects. These models can be oriented to specific experts' backgrounds. The work presented follows this approach and introduces a generic Modelling Language (ML) through a model, that can be specialized to meet different needs in road traffic simulations. The case study illustrates how that model can be successively modified to model people' behaviour in traffic both at the traffic expert and platform-oriented levels. This allows reducing the learning curve of experts with backgrounds non-related to software simulations.

Keywords: road traffic, model transformation, model-driven engineering, agent platform, modelling language, agent-based modelling

1. Introduction

Road traffic is a widespread phenomenon in our society. Every person that lives in a town has to face it and deal with the different situations it produces or influences. This leads researchers from multiple areas to develop studies where the road traffic and its factors are analysed in order to improve its understanding and forecast its evolution and problems.

These studies face to various obstacles concerning the difficulty of modelling the complex behaviour of individuals. This behaviour includes the relationships among these people, where they play different types of roles (i.e. driver, passenger or pedestrian), and their mental state.

The interactions among the participants can be homogeneous (i.e. individuals of the same role) or heterogeneous (i.e. individuals of different roles), and have multiple nuances. The latter are related to the influence produced by others through various ways.

Regarding the mental state, each individual has her/his particular information: facts (e.g. the route to follow) and goals (e.g. to save fuel). Although not part of a state, we also consider here as part of it the traits that influence its behaviour (e.g. the age as it influences the response time). This information is modelled in terms of specific entities and relationships. They allow to include in models information about decisions and reactions that also affect the relationships among elements in traffic.

Simulation has become a key element in order to mitigate these kinds of problems. It allows working with only sets of selected variables, controlling the mutual influences among individuals and the environment (e.g. modifying weather conditions or the configuration of traffic lights) [15]. Also, it avoids the problems that empirical studies might produce in the real world (e.g. traffic jams or unnecessary risky situations to people).

However, simulations have their own drawbacks [9]. They require multidisciplinary groups where the understanding between the researchers can be tedious. Communication problems and the different backgrounds of individuals might produce undesirable situations. For instance, bugs or mistakes in code due to a misconception in the design are usually detected in late phases of the development. These situations imply the increase of the costs and delays.

Trying to reduce this communication gaps, the use of Agent-Based Modelling (ABM) [8] for simulation is quite common. Agents represent the individuals and their goals and actions. They establish relationships among them and with the environment, and are able to modify their behaviour in consequence. Experts from different domains can easily grasp these abstractions that, at the same time, can be further specialised form a variety of domains. In order to support these features, there are agent platforms [31] that provide among other services life cycle and information management, and communication channels.

These platforms also present their own problems. They usually have a steep learning curve for experts without previous experience in agent-based simulations. Moreover, there is a quite heterogeneous landscape of platforms (e.g. Jade [3] or NetLogo[37]) and agent methodologies (e.g. INGENIAS [30] or Gaia [6]). They consider similar but not the same concepts and their support is also different. This situation increases the difficulties to compare models and results, and to select the proper resources for a given study.

Model-Driven Engineering (MDE) [21, 33] can be helpful to address these problems [14]. It is a development approach which focuses on Modelling Languages (MLs) [28] to produce models that define systems. These MLs are specialised for different abstraction levels and experts.

Models can be used to produce or modify artefacts (e.g. source code) using automatic transformations. In this way, the development process becomes iterative and incremental, promoting the reutilisation of elements. The process also makes explicit all the information required to develop the system, and facilitates experts their understanding and manipulation.

A well-established framework for MDE is Model-Driven Architecture (MDA) [22] of the Object Management Group (OMG) [35]. It is focused on object-oriented development using OMG standards, like the Meta-object Facility (MOF) [2] and Unified Mod-

elling Language (UML) [13]. This framework recommends the elaboration of two different types of models to develop the system: Platform Independent Model (PIM) and Platform Specific Model (PSM). The first presents analysis and design models remaining an abstract level that discards concrete implementation guidelines. The second addresses the models extending them in order to include the necessary information to generate a specific runtime system.

This work introduces a generic ML at the level of PIM [22] for developing road traffic simulations. It provides a high-level specification where relationships are based on inheritance and decomposition of entities. These primitives provide the means to extend the ML so it can integrate existing road traffic theories using its guidelines to produce extended PIMs. These new languages are used to define specific vocabularies and methodologies. Then, they are specialised for target agent platforms generating PSMs [22].

The case study that illustrates the approach introduces two types of transformations from the generic ML. The first one produces an extended PIM based on road traffic models and theories extracted from the literature [1, 34]. It also takes common concepts used in agent methodologies (e.g. goals and tasks [5, 30]). The resulting model is extended using a second transformation to generate the ML for the PSM. This transformation is oriented to the A-globe agent platform [36] and considers specific platform details. Thus, it allows producing model specifications according to the requirements of traffic theories using the agent platform.

The rest of paper is structured as follows. First sections are devoted to introduce the background of the work. Section 2 presents the basics of MDE and its tools, with a particular focus on the transformation processes, while Section 3 delves into agent platforms and methodologies, and their concepts. The presentation of our work starts in Section 4. It introduces the generic ML, its use guidelines and the transformation process. Section 5 applies this framework to the case study. Section 6 compares and discusses our and related work, placing the proposal in context. Finally, Section 7 presents some conclusions about the proposal and future work.

2. Model-Driven Engineering

Model-Driven Engineering (MDE) [21, 33] is a development approach that uses models to specify systems. These models can be described at different abstraction levels according to the needs of participants (e.g. models for theoretical experts are more abstract than those to generate source code), or from different perspectives (e.g. security or architecture). In order to ease the generation of these artefacts that are commonly related, MDE includes semi-automatic transformations.

The approach is based on the use of well-defined MLs to make possible the development process and the corresponding transformations. This requires that MLs present specific definitions in order to validate the models created through their primitives.

There are multiple possibilities to define MLs. The most common uses metamodels. This is strongly related to the fact that most of typical MDE MLs are graphical and graph-based. They depict conceptual graphs with properties [4] (e.g. UML [13]).

A metamodel is a model that indicates certain primitives and constraints to define a ML. They are defined using meta-MLs in a similar way to models. Examples of these types of language are MOF [2], which is used to specify UML, or Ecore in Eclipse

Projects [38]. The primitives of these languages are oriented to define different types of graphs. For instance, Ecore has an *EClass* node linked to *EAttribute* and *EReference* nodes.

Regarding transformations, they can be classified according to their inputs and the outputs they produce. There are transformations that generate new models from others (i.e. Model to Model), source code from a specific model or group of them (i.e. Model to Text), and models from source code using reverse engineering (Text to Model) [10]. Also, they can generate other types of artefact related to the development process (e.g. documentation or validation tests). This leads to an incremental development process: models can be transformed into others more specific until the requirements of the target platform are satisfied.

Transformations are implemented in different ways: using general purpose programming languages (e.g. Java) and transformation languages. The first approach develops a module that uses programming interfaces for manipulating its inputs and outputs (e.g. a code generator tool). This leads to reuse the already tested artefacts and tools of mainstream development approaches (e.g. object-based programming). In the second case, the transformation is developed using a specific language for transformations and an engine executes the instructions. It eases the comprehension and analysis of the input artefacts and the resulting ones in the outputs.

MDA [22] is a specific development framework defined by the OMG [35]. It provides guidelines in order to generate MDE projects with a desirable set of properties.

The guidelines start defining a Computation Independent Model (CIM). It considers the requirements and does not have any information about the system to develop. Then, an initial Platform Independent Model (PIM) is produced. This is a high-level abstraction model and is independent of any implementation technology. Once it is generated, it can be transformed into one or more Platform Specific Models (PSMs). These are focused on representing the entities related to the implementation technology. A PSM can be transformed in order to link more than one technology. It is the last modelling step, from which the process generates the source code which should be directly usable in the target platform or technology.

MDE, and its implementation with MDA, are expected to produce improvements on system development regarding productivity, portability, interoperability and maintenance and documentation. Manipulating information at the level of models and transformations should facilitate the experts work by promoting and explicit manipulation of all the required information.

In the presented approach, MDA guidelines are followed in order to generate more specific models from a generic one related to road traffic. The CIM is not considered in this case, while the PIM and PSM are defined by UML [13]. Transformations are implemented through ATL [19]. This is a language based on rules specially designed to these issues.

3. Agent Methodologies and Platforms

Agents [39] are autonomous and intentional software entities that interact with the environment around and among others. They can establish communication through pieces of informations (e.g. messages) and can be organised in groups. Agents usually have a goal

that can be decomposed into others, and a set of tasks to fulfil them. Evaluators and actuators are other components related to them. They consider the best goal to accomplish and execute the tasks associated to a specific selected goal respectively.

Groups formed by agents can be organised into Multi-Agent Systems (MAS) [39]. They consist of an environment, different objects and agents, the relationships among these entities and a set of operations that might be achieved by them. The latter must be able to modify the environment and have influence over the rest of individuals. These systems are used in multiple domains, and in particular for simulations. Here, they provide the possibility of creating artificial universes for testing theories related to certain behaviours (e.g. road traffic simulations [29]).

There are multiple methodologies to develop agent-oriented systems. These include a body of methods and guidelines to developed in a systematic and coordinated way such systems [16]. Many of them are based on MDE, and include support tools.

These methodologies can be classified into three main types according to their focus: requirement-driven methodologies, agent-oriented methodologies and multi-agent system methodologies. The first type is focused on eliciting requisites through the same concepts of the structured development techniques used in the programming paradigm. An example of this type of methodology is Tropos [5]. The second type models the problems of the systems to solve. In order to do that, it extends the concepts of the object-oriented methodology and apply them to MAS. An instance of this methodology is Prometheus [27]. The third has as foundations MAS and their organisation. The individuals (i.e. agents) are considered through a collective perspective discussing their roles and importance within the group. INGENIAS [30] and Gaia [6] are traditional examples of this type of methodologies.

Regarding agent platforms, there is not a standard (e.g. OMG MASIF [26] or FIPA [31]). These platforms provide the infrastructure to implement MAS in different scenarios. They are usually composed by a set of components that conforms an organisation, security protocols, and communication channels to allow the interaction among agents. They can be classified into two broad groups: simulation-oriented platforms and agent-oriented platforms. The first ones have as a main issue the development of simulations and present components related to them (e.g. libraries for controlling and analysing experiments). Examples of these type of platforms are Swarm [24] and NetLogo [37]. The second ones are used to develop general-purpose systems that can also produce simulations. Instances of these platforms are Jade [3] and A-globe [36].

This work is mainly focused on developing PSMs for agent-oriented platforms. Nevertheless, introducing some modifications in the PIM could be affordable, as the purpose of the ML produced is the same for both types. This purpose is based on the premise that a PSM must represent the different elements and requisites the target platform proposes to implement a specific MAS related to road traffic simulations.

4. Road Traffic Modelling

This work proposes a MDA infrastructure [22] for road traffic agent-based simulations. It proposes a PIM able to generate other models more specific applying different automatic transformations.

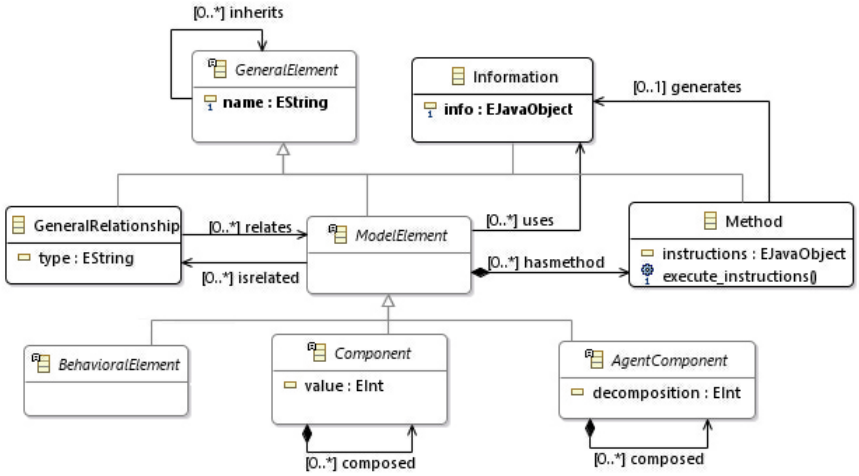

Fig. 1. Excerpt of the generic Platform Independent Model (PIM) illustrating its primitives.

The generic PIM presents different inheritance and composition relationships to ease the generation of more specific models. Transformations use these relationships in order to achieve the appropriate modifications during the development of these extended models.

Then, a more specific PIM is produced from the original. It must encompass existing road traffic theories for individuals involved in it. They are extracted from the literature of the domain focused on the behaviour, traits and decision-making process of participants. Moreover, it has to adopt concepts and related components according to agent methodologies (e.g. goals and tasks).

Regarding the PSMs, they are created in order to adapt the ML produced by the more specific PIM. Its purpose is to model the structure of target agent platform and adopt its specific requirements. Thus, a complete agent platform must be able to be developed using these primitives. Also, it allows integrating the components that are provided by these general purpose platforms in a specific language for road traffic.

Finally, source code is generated through the appropriate transformation process. Then, the body of the methods produced are fulfilled with the required information. Once this step is completed (i.e. developed and tested), the road traffic simulation might be run properly using the agent platform and agents would apply the selected traffic theories during their decision-making processes.

Section 4.1 introduces the generic PIM and its primitives and entities. It also illustrates the meaning of its different classes and how they establish relations with others.

4.1. Generic Platform Specific Model

The generic PIM is focused on providing a set of flexible primitives. It includes references and entities that ease the transformation operations. These are defined with the purpose of facilitating the specification of multiple road traffic theories and agent-oriented platforms.

The references provided by the model store the information and link the different entities in order to generate coherency. They can be classified into four main types: *inheritance*, *composition*, *relation* and *method-related references*.

The *inheritance* references can be decomposed into two: *original* and *generated* inheritance. The first one is used in the model to link types, and represented by a triangular arrow. The second is an explicit inheritance that can be used in models (see Fig. 1). It eases the extension of the entity that uses it (e.g. *inherits* reference).

The *composition* links generates the possibility of decomposing an entity into others in extended model versions. This promotes the organisation of individual traits extracted from road traffic literature or goal-based trees, which are common in agent methodologies [30]. Instances of this type of relations are the *composed* references.

About the *relation* references, they link two or more entities to provide a communication channel that allows sharing information among them. Examples of this type of references in the PIM are *relates* and *isrelated*. In this case, both introduce guidelines to insert specific relationships with bi-directional communication.

The *method-related* relationships link methods with the entities that use them. This use can be classified into two orthogonal perspectives: an entity generates certain information through a method or an entity needs some information produced by others. Example of the first usage is *hasmethod* relationship while *uses* reference concerns to the second.

Regarding the PIM entities, the key element is *GeneralElement*. It is an abstract class that acts as the root of the rest of classes of the model. It has an attribute *name* which identifies unequivocally each element (a constraint is inserted in order to consider this limitation). There are four types of elements that inherit from it (i.e. child classes): *GeneralRelationship*, *ModelElement*, *Information* and *Method* (see Fig. 1). The *inherits* reference allows generating extended versions of the *GeneralElement* and for extension of these child classes (e.g. *Method* or *ModelElement* entities can be extended to more specific classes).

GeneralRelationship entity stores information about the relationships produced among entities. It is related to *ModelElement* through the *relation* reference *relates*. An attribute *type* represents the possibility of generating different types of relationships.

ModelElement is an abstract class that is linked to *Method* and *GeneralRelationship* entities. The first indicates which methods has each *ModelElement* class through *hasmethod* reference. The second is linked to *GeneralRelationships* using *isrelated* reference to complete the communication channel among other *ModelElement* entities. This class is also related to *Information* entity through *uses* reference. This latter allows obtaining the desirable data previously stored.

Regarding to the extended classes from *ModelElement* entity, it has three which are abstract: *BehavioralElement*, *Component* and *AgentComponent*. The first is in charge of representing the different elements extracted from road traffic theories (e.g. the mental state of individuals or the environment where a simulation is performed). The second describes the components associated to these theories (e.g. the traits of a group of individuals or road traffic conditions like weather or the state of traffic lights). It has an attribute *value*. This allows introducing the appropriate values related to the influence of the component over the rest of them. This may also represent how the component modifies the behaviour of an individual. The last class considers the components related to agent

methodologies and platforms. It presents an attribute *decomposition*. This identifies the composition rule this kind of components has (e.g. goals can be decomposed into others in some agent methodologies [5, 30]). Both types of component entities (i.e. *Component* and *AgentComponent*) have their *composed* references to achieve these type of operations.

The *Information* class illustrates the data managed by road traffic simulations and individuals (i.e. agents). The *info* attribute stores this information. It must exist in each piece of information (i.e *Information* instances) though it could be empty. This is controlled through a specific constraint.

The *Method* entity represents the methods used by each one of the *ModelElement* instances to achieve some operation. It has an *instructions* attribute which is in charge of storing the steps to follow and the possible parameters needed. The *execute_instructions* method performs the operation according to the *instructions*.

Finally, delving into the PIM cardinality, most of references goes from zero to n. This pursues to provide flexibility to support more specific models (e.g. a PIM oriented to road traffic theories or a PSM with agent platform requirements). The exception is the *generates* reference from the *Method* class to the *Information* class. It indicates that a single method is able to modify only a given piece of information and no other else.

5. Case Study

The case study shows the MDA infrastructure (i.e. the generic PIM, its primitives and the transformations) in order to produce an extended PIM and a PSM. The first encompasses road traffic guidelines oriented to individuals introduced in [34]. They are related to the mental state and traits of individuals. The interactions between individuals (i.e. drivers, pedestrians and passengers) and the environment are considered based on [1]. The decision-making process based on concepts presented in agent methodologies (e.g. Tropos [5] or INGENIAS [30]) is also modelled. The second is adapted to generate a ML that allows producing the elements used in the architecture of the A-globe [36] agent platform and its requirements.

Section 5.1 presents the steps achieved to develop the extended PIM and the primitives introduced to model the road traffic theories and the elements related to agent methodologies. Section 5.2 illustrates the last part of the MDA infrastructure (i.e. the generation of the PSM), introducing the features and requirements of A-globe agent platform and how they are adopted by the PSM.

5.1. PIM extension to include road traffic theories

Addressing the development of the specific PIM, the first step consists of identifying the main entity. In this case, the model is focused on individuals that are represented by agents. Therefore, an *Agent* class is created (see Fig. 2). It extends the *ModelElement* class (see Section 4.1). New attributes and methods are introduced. The attribute *visibleinfo* is in charge of storing the information obtained for the agent from the part of environment it can observe. The attribute *type* provides a tool to organise the individuals involved in traffic by roles (e.g. driver, pedestrian or passenger). Thus, each agent represents a determined type of participant during the simulation. The method *observeEnvironment* obtains the information and modifies the value of the attribute *visibleinfo*, while the method *interacts*

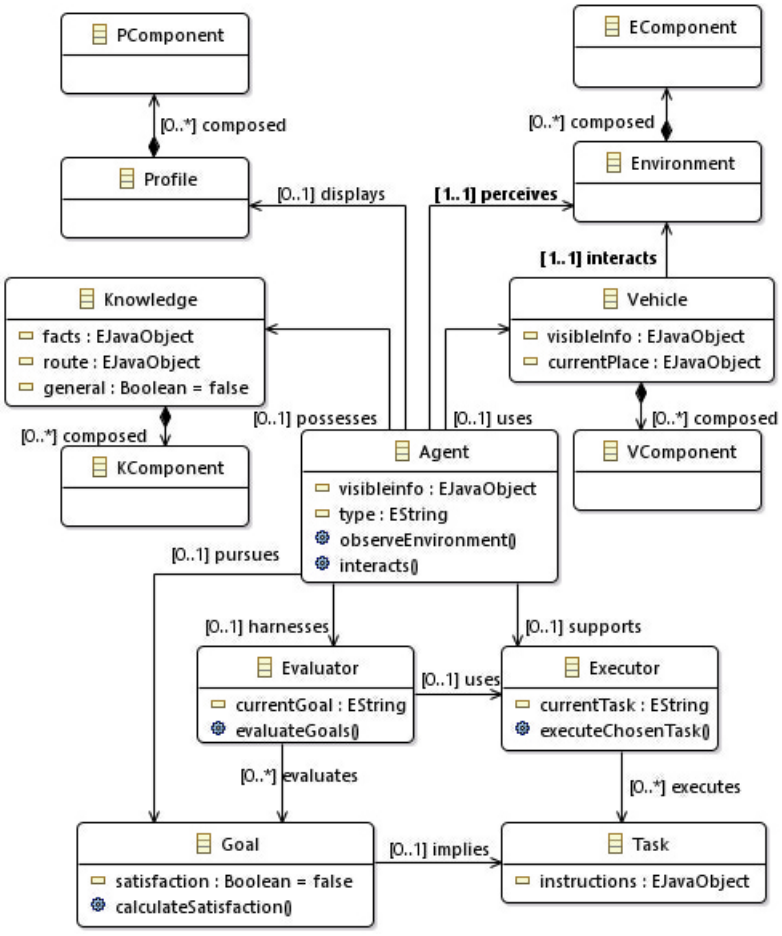

Fig. 2. Excerpt of the specific PIM for addressing traffic theories and agent methodologies.

is responsible for achieving the operations the agent needs in order to progress in the environment around. The *Agent* class presents new references that relates a set of new entities to it. These entities extend the *BehavioralElement* class.

In the case of the *displays* reference, it targets the *Profile* class. The cardinality of this relationship indicates that an agent can only have a determined profile (but the profile can be shared by more than one agent). The *Profile* class describes the set of traits of participants in traffic. It has a *composed* link to the *PComponent* class. The latter extends the *Component* entity and represents a specific trait (e.g. age or gender).

The *possesses* relationship links the *Agent* entity and the *Knowledge* class. This illustrates the mental state of the individual (i.e. agent). It has three attributes: *facts*, *route* and *general*. The first is in charge of storing the information a person has (e.g. the how to drive or how to dodge other individuals). The second indicates the steps to follow to arrive to the desired destination. The last one provides an special functionality for gen-

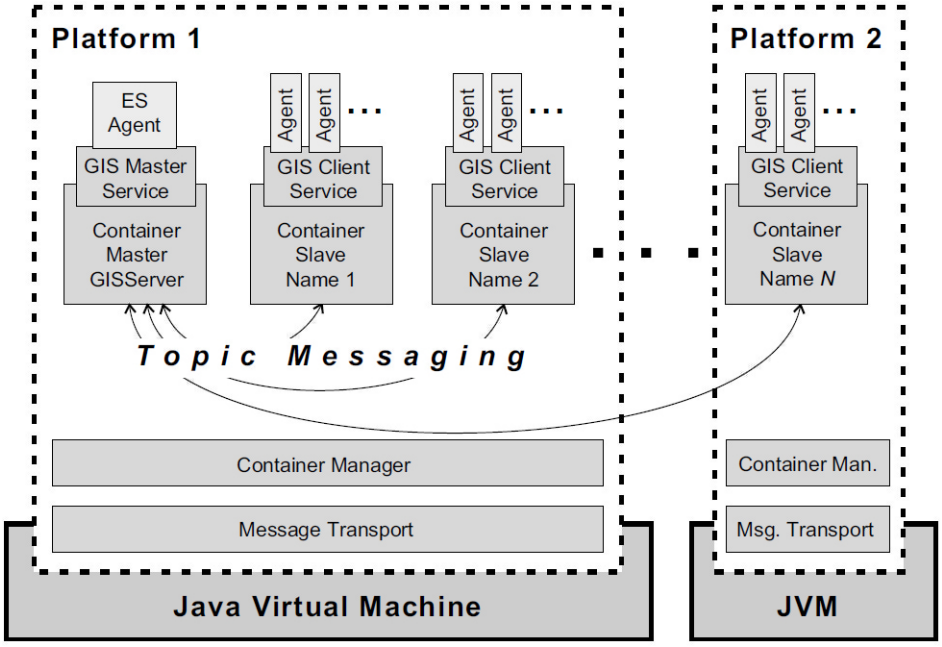

Fig. 3. Excerpt of the A-globe platform architecture extracted from [36].

erating general knowledge to simulations. This latter is not a property of any *Agent* but everyone can consult it in order to know information. In this case, the *facts* attribute stores this generic data (e.g. a lane closure or and accident in a particular point). The *Knowledge* entity has a *composed* reference to the *KComponent* class. This describes the multiple elements related to the mental state of the participant (e.g. trip purpose). Both *Profile* and *Knowledge* with their *Component* classes provide the primitives to adapt to the PIM the traffic theories extracted from [34].

The reference *uses* allows the interaction among the individual and the *Vehicle*. Therefore, only agents with driver or passenger role need it. The *Vehicle* entity presents two attributes: *visibleinfo* and *currentPlace*. The first is similar to the attribute with the same name included into the *Agent* class. Thus, it stores the information that can be obtained being inside the vehicle. The second illustrates the place where the vehicle is situated being able to modify it according to the travel. The *Vehicle* entity has a *composed* relationship to the *VComponent* entity. This reference eases the interaction among the vehicle and its components (e.g. steering wheel or lights). Moreover, the *interacts* relationship links the *Vehicle* to the *Environment* class. This allows establishing relations among the vehicle and the environment (e.g. modifications in the vehicle behaviour due to the road or weather conditions).

To consider that vehicle and environment are different entities is strongly related to Driver-Vehicle-Environment (DVE) model extracted from the literature of the domain [1]. It specifies for the vehicle its own interactions with people (i.e. driver and possible

passengers), while the rest of the environment (e.g. roads or traffic lights) interacts with agents using others more generic.

The reference *perceives* relates the *Agent* class to the *Environment* entity. It provides to individuals the possibility of observing the environment around and obtaining information. Therefore, every role played by agents (i.e. driver, pedestrian or passenger) can achieve this interaction. Nevertheless, drivers and passengers are able to obtain extra information using the *interacts* reference (e.g. a driver looks in the rear-view mirror). The *Environment* class presents a *composed* reference to the *EComponent* entity in order to consider the elements of the environment (e.g. the map or the situation of traffic signals). The *EComponent* class extends the *component* entity (see Section 4.1), which allows being decomposed into other ones of the same type.

Regarding the classes that support agents in a simulation, they extends *AgentComponent* entity. It has been identified four class types: *Evaluator*, *Executor*, *Goal* and *Task*. But in this case, only the first three have the respective references related to *Agent* entity. These are *harnesses*, *support*, and *pursues* respectively.

The relationship *harnesses* links the *Agent* to the *Evaluator* class. This is in charge of evaluating the available goals of the agents and choose the most appropriate to them in each moment. It presents an attribute *currentGoal* that describes the selected goal to satisfy. The method *evaluateGoal* is responsible for achieving these operations and modifying the *currentGoal* attribute according to its results. *Evaluator* presents two references: *evaluates* and *uses*. The first one relates this class to the *Goals* of agents, while the second connects to *Executor* entity.

The reference *supports* joins *Agent* entity to *Executor* class. This class plays the role of an actuator and is responsible for executing the operations presented in each *Task*. These *Task* entities are associated to *Goal* entities following the typical guidelines from Agent-Oriented Software Engineering (AOSE). The *Executor* class has the *currentTask* attribute and the *executeChosenTask* method. The first illustrates the name of the current *Task* that is going to be executed, while the second modifies this attribute selecting the *Task* entities associated to the current goal selected by the *Evaluator* (see Fig. 2). The *executes* reference allows establishing these relationships to *Task* classes. The *Evaluator* and *Executor* classes alongside the *Environment*, *Vehicle*, *Knowledge* and *Profile* entities carry out a *perceive-reason-act* cycle [23].

The link *pursues* relates the *Agent* entity to the *Goal* class. This class presents an attribute *satisfaction* that indicates if a goal is satisfied or not according to agent methodologies guidelines (e.g. INGENIAS [30]). The method *calculateSatisfaction* modifies this attribute and evaluates if the conditions provided by the current goals are properly considered. The relationship *implies* communicates each *Goal* entity with the *Task* entities it has associated.

As it was previously indicated, the *Task* class is not directly connected to *Agent* entity. It is highly dependent on the *Goal* class. It has an attribute *instructions* that encompasses the different operations to be achieved by the *Executor* entity. The proper fulfilment of the instructions of the current task and the others that could be related to the current goal, produces the satisfaction of this.

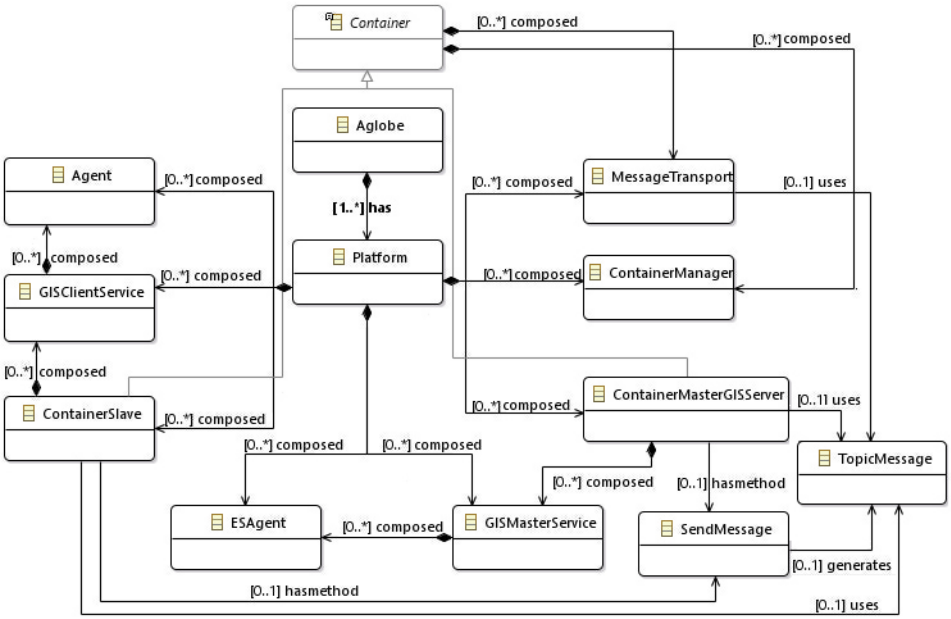

Fig. 4. Excerpt of the PSM which models the A-globe general architecture.

5.2. PSM adapted to A-globe agent platform

Once the specific PIM is developed, the next step consists of generating the PSM adapted to the A-globe agent platform [36]. In this case, the fist task is evaluating its infrastructure.

A-globe presents multiple components that provide functionality included inside different platforms (see Fig. 3). These are distributed instances of A-globe running in a specific host. In order to a simulation runs, A-globe must present at least one main of these latter. If more than one of these instances are created, then they are slaves of the main one.

Delving into the components of a platform, there are a message transport and a container manager. The first provide the communication channels among agents, while the second is responsible for the containers that store agents and services. Here, in a similar way to platforms architecture, there is a main container and a set of slaves which are dependent on it.

As A-globe is an agent platform specially designed to real-world simulations, each container presents a Geographical Information System (GIS) service. The main container has also an Environment Simulator (ES) agent, while the rest of them only present agents (in this case individuals related to road traffic). The GIS service is in charge of indicating the situation of the individuals while the ES agent generates the simulation of related parameters (e.g. physical location or movement in time).

When the evaluation of the architecture of the agent platform is concluded, the PSM that adopts its structure is developed. Its entities extend *AgentComponent* (see Fig. 1), except those that represent specific agents related to road traffic (i.e. *Agent* class) or message communication (i.e. *TopicMessage* and *SendMessage*).

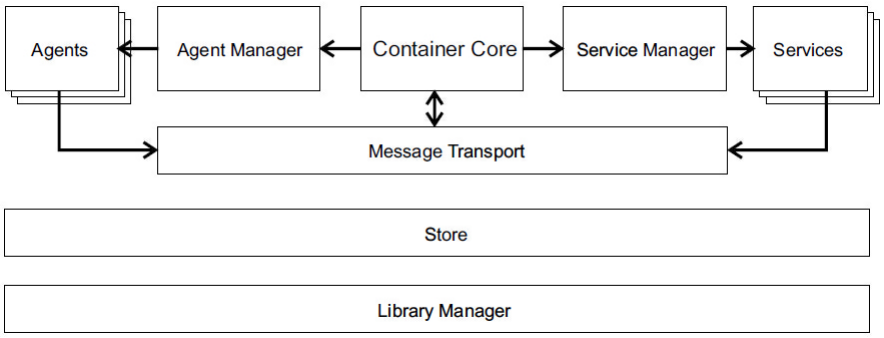

Fig. 5. Excerpt of the A-globe architecture illustrating the agent containers extracted from [36].

In this model *A-globe* is the main entity and is related to the *Platform* class through the reference *has*. The cardinality of this relationships goes from 1 to n in order to represent the possible platforms the A-globe architecture has (see Fig. 4).

The *Platform* class presents different relationships to the rest of the entities. It has eight *composed* references where the cardinality goes from zero to n. These allow establishing the relationships among the *MessageTransport* and *ContainerManager*, which are presented in each one of the types of platforms (i.e. main platform or slave platform).

In the case of a main platform, it must have instances of the *ContainerMasterGISServer*, *GISMasteService* and *ESAgent* classes, which describe the elements of the same name in the A-globe architecture (see Fig. 3). They are connected through *composed* relationships that ease the communication among them. If the *Platform* class is a slave, then it is related to *Agent* (i.e. drivers, pedestrians or passengers), *GISClientService* and *ContainerSlave* classes. These are linked through *composed* references in a similar way to the others previously introduced.

The communication among agents and services is achieved in the PSM with the *TopicMessage* and *SendMessage* entities. The first extends the *Information* class of the generic PIM (see Fig. 1) and contains the data which is sent among the A-globe agents and services. The second inherits from *Method* class and presents the *generates* relationship that links it to *TopicMessage*. Both types of containers (i.e. *ContainerSlave* and *ContainerMasterGISServer*) has *hasmethod* reference in order to provide the communication functionality. This is because in A-globe the containers are in charge of sending messages between agents and services. *TopicMessage* has *uses* relationships to the *MessageTransport*, *ContainerSlave* and *ContainerMasterGISServer* entities in order to complete the operations of sending and receiving messages.

The PSM has two inheritance references among the both types of containers and a general *Container* class. It is related to the architecture of the agent containers, as they share similar elements that have to be also modelled. These elements are: the library manager, the store, the message transport and its agents and services, the container core that presents a bi-directional communication with the message transport and the managers of both agents and services (see Fig. 5).

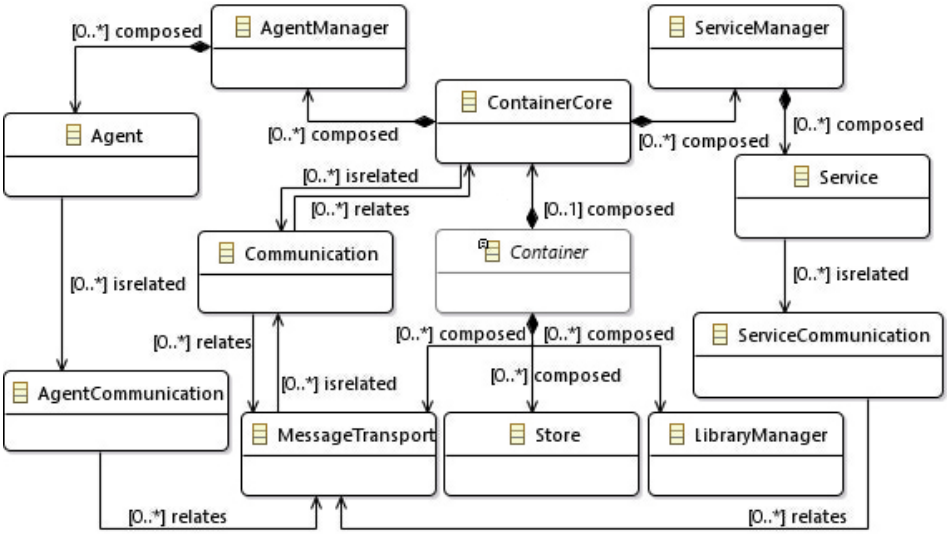

Fig. 6. Excerpt of the PSM for the A-globe agent container architecture.

Thus, the PSM is completed with the container components. These elements also extend *AgentComponent* class with the exception of *Communication*, *AgentCommunication* and *ServiceCommunication* entities which extend *GeneralRelationship* class (see Fig. 1).

The *Container* entity presents *composed* references to *MessageTransport*, *Store*, LibraryManager and *ContainerCore* (in this case with cardinality from 0 to 1) clases. This latter has its own *composed* relationships to *AgentManager* and *ServiceManager* entities. In turn, these managers are connected to *Agent* and *Service* entities through *composed* links (see Fig. 6).

The *ContainerCore* class represents the bi-directional communication to *MessageTransport*, In order to do it, it uses the *Communication* class that establish these relationships through *relates* and *isrelated* references. Similar references are used among the *Agent* class and *MessageTransport* entity using the *AgentCommunication* class, and analogously between the *Service* entity and the *MessageTransport* class through the *ServiceCommunication* entity. After that, it can be considered that the A-globe agent platform is integrated in the model and the PSM satisfies the requirements related to its architecture.

Once the PSM is completely developed, a model specification can be generated. This must consider a specific road traffic theory and a decision-making based on a determined goal and task hierarchy [11]. Then, the associated source can be generated through a semi-automatic transformation. The body methods of the element must be fulfilled while the parameters of the simulation have to be initialised. Then, needed classes (e.g. the class with the main method) are also generated in order to run the road traffic simulation.

Finally, it must be indicated that the transformations among the different models could be implemented using ATL [19] rules. It allows automatising the development process and promotes the reutilisation of the artefacts generated.

6. Related Work

MDE embraces multiple areas of research related to our work. This approach is focused on three specific issues. The first consists of the definition of a generic PIM that can be extended and specialised until producing a ML related to road traffic simulations. The second introduces the MDA framework that guides users to incorporate traffic theories, decision-making processes and agent platform requirements to the PIM. The third concerns to the adoption of the traffic theories during the specialisation of the PIM. This is a key step in this MDA proposal. It is related to the fact that the PIM has to encompass a wide amount of types of road traffic theories. Thus, the adaptation of the primitives to the theoretical elements and their relationships becomes a very demanding task.

Road traffic simulations that adopt MDE use models to develop the interactions of individuals with the environment. It can generate difficulties in the references among entities due to the inherent complexity of human relationships. One proposal to mitigate this effect is based on the definition of generic metamodels. These present a high level perspective where the elements are connected through simple relations. These provide development guidelines that ease the generation of more complex primitives in MLs. [7] and [11] are examples of this perspective. The first one illustrates a metamodel with generic primitives. It is able to generate a ML adapted to web-based traffic simulations, but presents drawbacks in the decision-making process. The elements considered in road traffic theories do not affect this process. It occurs due to the individuals in the simulation have a simple behaviour based on path-following. The second is focused only on certain studies related to road traffic. Therefore, the primitives provided are not enough flexible in order to consider the multiple perspectives related to the domain.

The MDE applied to agent-based simulations is another point of view related to the development of generic metamodels. In this case, these models provide guidelines to achieve cooperation and interaction instructions among agents. This eases the evaluation and testing of these operations. A MDA process to reduce the gap between the designs and the implementation is an important issue to consider. Examples of this perspective are [20] and [32]. The first produces a high-level conceptual model that uses the community metaphor to support the creation of goal-oriented organisations. The second presents an agent-based metamodel oriented to social interactions using INGENIAS methodology [30]. Both have as shortcoming the lack of flexibility in the primitives of languages.

MDA can be used in other issues non-related to simulations. For instance, it is widely used in fields associated to testing. This is related to the fact that it presents a process based on transformations (usually automatised), which can be applied to models in order to generate others more specific. Here, it provides the necessary certitude and reutilisation capability to generate unitary test where bugs are easy identified and controlled. An example of this usage is illustrated in [18].

Regarding the road traffic theories, there are multiple of them that are organised through hierarchical structures. These usually have levels and abstraction layers to classify the elements considered. [25] is an example of this organisations. It presents a hierarchical control model only for drivers where their decisions and considerations are assorted in different levels. The generic PIM presented in this paper has primitives to generate composition relationships among entities in order to achieve these hierarchical structures. The use of layers to classify the elements has not been considered relevant in this case.

Nevertheless, there is not a standard theory related to participants in road traffic that embraces the behavioural elements and decision-making processes. The selections of which are the most important traits of individuals or which elements present in the environment around affect people is an open issue. Examples of approaches that review these components and try to generate theories of road traffic are [1, 34]. The generic PIM is intentionally open in order to address this issue. Thus, the cardinality of the references is not very restricted, and the inheritance links promote a specialisation of the model in next steps of the development allowing embracing multiple theories. Also, a specific entity (i.e. *GeneralRelationship*) has been included to ease the bi-directional relationships between components extracted. This entity presents its own inheritance primitives to provide specialisation if it is necessary in these interactions. All these characteristics facilitate the generation of specific PIMs that consider multiple traffic theories.

7. Conclusions

This paper has introduced a MDA infrastructure [22] for the development of road traffic simulations. Its core element is a general PIM that can be specialised to provide tailored PIMs, and from these the related PSMs.

These PIMs include the existing road traffic theories, the traits of participants in traffic identified and the decision-making process based on the concepts of AOSE. Therefore, PIMs formalise the information that traffic experts manage.

The PSM adapts the PIMs considering the requirements of agent platforms (e.g. Jade [3]). These requirements are focused on the architecture of the platform and how it works, establishing the appropriate relationships among the elements it presents. It also must integrate the agent platform, being able to produce a model specification of a traffic simulation.

The last step of the process is in charge of transforming the PSM to source code. The body of the methods must be fulfilled inserting the different instructions in order to generate the simulation. Part of them are completely generated by transformations (e.g. standard accessor methods), but for specific algorithms code snippets must be provided. Additional classes need also to be created (e.g. a class with the main method). In this step, developers also run standard code tests with the simulation before proceeding to validation tests with traffic experts. Some of these tests are generated with transformations from models.

The transformations among models (i.e. from the generic PIM to specific PIM, and from this latter to PSM) can be achieved using model-to-model transformation languages like ATL [19]. It eases the process introducing automatic steps and promotes the reutilisation of the artefacts developed.

Regarding the case study, the generic PIM is extended to a specific PIM using three different traffic theories. In order to do that, new references among the new elements have been created adapting the original model to another one that contains them.

The A-globe [36] agent platform has been selected to be modelled with the PSM. A *Platform* concept is included with a cardinality from 1 to n. It is related to the fact that A-globe must have at least one main platform in order to run properly. This element presents multiple *composed* references to the other elements identified in the analysis of the A-globe architecture.

Finally, a model specification was developed following the primitives introduced by the PSM. It considers specific road traffic theories and integrates a hierarchical structure to generate a given decision-making process [11].

This approach has several open issues. The simulations generated do not present a graphical interface where the individuals (i.e. agents) can show the different processes and interactions they perform. The PSM should also be tested to include the requisites of other road traffic simulation platforms based on agents (e.g. MATSim [17]). Smart Roads (SR) is a related field where this kind of developments could be useful. Applying them in that domain requires new primitives [12]. Also, time events (e.g. an accident during the simulation in a specific moment) have to be considered by creating proper primitives.

Acknowledgments. This work has been done in the context of the project "Collaborative Ambient Assisted Living Design (ColoSAAL)" (grant TIN2014-57028-R) supported by the Spanish Ministry for Economy and Competitiveness, the research programme MOSI-AGIL-CM (grant S2013/ICE-3019) supported by the Autonomous Region of Madrid and co-funded by EU Structural Funds FSE and FEDER, and the "Programa de Creación y Consolidación de Grupos de Investigación" (UCM-BSCH GR35/10-A).

References

1. Amditis, A., Pagle, K., Joshi, S., Bekiaris, E.: Driver–Vehicle–Environment monitoring for on-board driver support systems: Lessons learned from design and implementation. Applied Ergonomics 41(2), 225–235 (2010)
2. Atkinson, C., Kuhne, T.: Model-driven development: a metamodeling foundation. IEEE Software 20(5), 36–41 (2003)
3. Bellifemine, F., Poggi, A., Rimassa, G.: JADE: a FIPA2000 compliant agent development environment. In: Proceedings of the 5th International Conference on Autonomous Agents. pp. 216–217. ACM (2001)
4. Bézivin, J.: Model driven engineering: An emerging technical space. In: Generative and Transformational Techniques in Software Engineering, pp. 36–64. Springer (2006)
5. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems 8(3), 203–236 (2004)
6. Cernuzzi, L., Juan, T., Sterling, L., Zambonelli, F.: The Gaia methodology. In: Methodologies and Software Engineering for Agent Systems, pp. 69–88. Springer (2004)
7. Çetinkaya, D.: A model driven approach to web-based traffic simulation. In: Proceedings of the Symposium on Theory of Modeling & Simulation. p. 14. Society for Computer Simulation International (2016)
8. Chen, X., Zhan, F.B.: Agent-based modelling and simulation of urban evacuation: relative effectiveness of simultaneous and staged evacuation strategies. Journal of the Operational Research Society 59(1), 25–33 (2008)
9. Crooks, A., Castle, C., Batty, M.: Key challenges in agent-based modelling for geo-spatial simulation. Computers, Environment and Urban Systems 32(6), 417–430 (2008)
10. Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM Systems Journal 45(3), 621–645 (2006)
11. Fernández-Isabel, A., Fuentes-Fernández, R.: Simulation of road traffic applying model-driven engineering. Advances in Distributed Computing and Artificial Intelligence Journal 4(2), 1–24 (2015)

12. Fernández-Isabel, A., Fuentes-Fernández, R.: Model-driven engineering of simulations for smart roads. In: Proceedings of the 9th International Workshop on Agents in Traffic and Transportation (ATT 2016). vol. 1678. CEUR-WS.org (2016)
13. Fowler, M.: UML distilled: a brief guide to the standard object modeling language. Addison-Wesley Professional (2004)
14. Fuentes-Fernández, R., Hassan, S., Pavón, J., Galán, J.M., López-Paredes, A.: Metamodels for role-driven agent-based modelling. Computational and Mathematical Organization Theory 18(1), 91–112 (2012)
15. Goodwin, L.C.: Weather impacts on arterial traffic flow. Federal Highway Administration, Washington, prepared for Road Weather Management Program (2002)
16. Henderson-Sellers, B.: Agent-oriented methodologies. IGI Global (2005)
17. Horni, A., Nagel, K., Axhausen, K.W.: The multi-agent transport simulation MATSim. Ubiquity, London 9 (2016)
18. Javed, A.Z., Strooper, P.A., Watson, G.: Automated generation of test cases using model-driven architecture. In: Proceedings of the 2nd International Workshop on Automation of Software Test (AST 2007). pp. 3–3. IEEE (2007)
19. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: A model transformation tool. Science of Computer Programming 72(1), 31–39 (2008)
20. Jung, Y., Lee, J., Kim, M.: Multi-agent based community computing system development with the model driven architecture. In: Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006). pp. 1329–1331. ACM (2006)
21. Kent, S.: Model driven engineering. In: Integrated Formal Methods. pp. 286–298. Springer (2002)
22. Kleppe, A.G., Warmer, J.B., Bast, W.: MDA explained: the model driven architecture: practice and promise. Addison-Wesley Professional (2003)
23. Lind, J.: Issues in agent-oriented software engineering. In: Agent-Oriented Software Engineering. pp. 45–58. Springer (2001)
24. Luna, F., Stefansson, B.: Economic simulations in Swarm: Agent-based modelling and object oriented programming, vol. 14. Springer Science & Business Media (2012)
25. Michon, J.A.: A critical view of driver behavior models: what do we know, what should we do? In: Human Behavior and Traffic Safety, pp. 485–524. Springer (1985)
26. Milojicic, D., Breugst, M., Busse, I., Campbell, J., Covaci, S., Friedman, B., Kosaka, K., Lange, D., Ono, K., Oshima, M., et al.: MASIF: The OMG mobile agent system interoperability facility. Personal Technologies 2(2), 117–129 (1998)
27. Padgham, L., Winikoff, M.: Prometheus: A practical agent-oriented methodology. Agent-Oriented Methodologies pp. 107–135 (2005)
28. Paige, R.F., Ostroff, J.S., Brooke, P.J.: Principles for modeling language design. Information and Software Technology 42(10), 665–675 (2000)
29. Paruchuri, P., Pullalarevu, A.R., Karlapalem, K.: Multi agent simulation of unorganized traffic. In: Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002): part 1. pp. 176–183. ACM (2002)
30. Pavón, J., Gómez-Sanz, J.J., Fuentes, R.: The INGENIAS methodology and tools. Agent-Oriented Methodologies 9, 236–276 (2005)
31. Poslad, S., Buckle, P., Hadingham, R.: The FIPA-OS agent platform: Open source for open standards. In: Proceedings of the 5th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents (PAAMS 2000). vol. 355, p. 368 (2000)
32. Sansores, C., Pavón, J.: Agent-based simulation replication: A model driven architecture approach. In: Mexican International Conference on Artificial Intelligence. pp. 244–253. Springer (2005)
33. Schmidt, D.C.: Model-driven engineering. IEEE Computer Society 39(2), 25–31 (2006)
34. Shinar, D.: Psychology on the road. The human factor in traffic safety. New York: Wiley (1978)

35. Siegel, J.: OMG Overview: CORBA and the OMA in Enterprise Computing. Communications of the ACM 41(10), 37–43 (1998)
36. Šišlák, D., Rehák, M., Pěchouček, M., Rollo, M., Pavlíček, D.: A-globe: Agent development platform with inaccessibility and mobility support. In: Software Agent-Based Applications, Platforms and Development Kits, pp. 21–46. Springer (2005)
37. Sklar, E.: Netlogo, a multi-agent simulation environment. Artificial Life 13(3), 303–311 (2007)
38. Steinberg, D., Budinsky, F., Merks, E., Paternostro, M.: EMF: eclipse modeling framework. Pearson Education (2008)
39. Wooldridge, M.: An introduction to multiagent systems. John Wiley & Sons (2009)

Alberto Fernández-Isabel holds a PhD in Computer Science from the Complutense University of Madrid, Spain, where he collaborates with the GRASIA research group. He currently works at Rey Juan Carlos I University of Madrid, Spain, as postdoctoral fellow and professor. His main research interests are on social simulation, in particular applied to social behaviour and road traffic problems, model-driven engineering, agent-oriented methodologies, and natural language processing, specially focused on sentiment analysis.

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

Received: October 10, 2016; Accepted: January 11, 2017.