Traffic Modeling Language User Guide

Release 0.1.0

Last updated November 19, 2015

Alberto Fernández Isabel

Contents

1. Introduction

Author(s) of this document: Alberto Fernández-Isabel

1.1 What is this TML?

This Traffic Modeling Language (TML) is based on individuals involved in road traffic. The roles considered for these individuals are drivers (with their vehicles), pedestrians and passengers.

The concepts of TML are grouped in three main clusters: a *Mental* cluster where the mental state and features of people are represented, an *Environment* cluster to consider the scenario where individuals interact, and *Interactive* cluster for developing the interactions and intentions of individuals.

1.2 Features

The following list shows the features and tools that support this TML:

- **Ecore.** TML is developed by this metamodeling language. Ecore provides a wide set of tools for supporting the generation of model specifications or their source code transformations.
- **Eclipse Modeling Framework (EMF)**. It is a framework provided by the Eclipse Modeling Project (see https://eclipse.org/modeling/emf/). It allows generating the metamodel and design its structure.
- **Graphical Editing Framework (GEF)**. It is a framework provided by the Eclipse Modeling Project (see http://www.eclipse.org/gef/). It supports the generation of graphical editors and the design and development of model specifications.
- **Graphical editor.** It is the tool in charge of developing model specifications. It provides a canvas and a palette in order to generate elements based on the references and metaclases of the metamodel.
- **Code generator**. It is the tool used to transform model specifications to Java source code. Also, it is able to ease the code specialisation in order to adapt it to target traffic simulation platform (see Code Generator user guide).

1.3 About this guide

This user guide should help you to get started with TML. It begin introducing an overview of it highlighting its main clusters. Chapter 2 delves into the concepts in which it is based and the different structures the metamodel presents.

2. TML concepts and structure

Author(s) of this document: Alberto Fernández-Isabel

2.1 Main concepts

The proposed metamodel has been designed keeping in mind to facilitate understanding of traffic problems and adapting its Modeling Language (ML) to the changing needs. To achieve these objectives, the metamodel adopts an Agent-Based Modeling (ABM) approach, organised through conceptual clusters, and composition and inheritance hierarchies.

Metamodel concepts are closely related to the agent paradigm, and in particular with models that consider the mental state. Thus, the *Knowledge* metaclass encompasses the beliefs and intentions of individuals, while the *KComponent* metaclass is responsible for organising the possible types of knowledge (for example, the driving experience or familiarity with the planned route). The characteristics of individuals are represented by the *Profile* metaclass, and the kinds of features for the *KComponent* metaclass (e.g. age or degree of stress). The objectives are represented by the *Goal* metaclass. Meanwhile, information from the environment is considered by *Environment* metaclass whose components are specified by *EComponent* metaclass (e.g. traffic or environmental conditions).

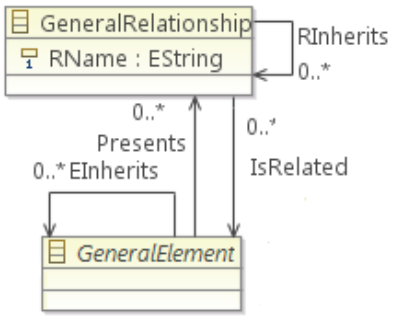
Regarding the structure of concepts, these are classified into three clusters. *Mental* and *Environment* clusters are responsible for describing the various concepts derived from the road traffic literature. Both clusters share similar structures. *Mental* cluster considers internal status of the participants in traffic, while *Environment* cluster includes the Driver-Vehicle-Environment (DVE) approach, focusing on the different forms of interaction among the individuals involved in road traffic and the surrounding environment. Meanwhile, the *Interactive* cluster focuses on the representation of the objectives and actions of those involved in traffic and usage. In order to do that, it follows a design inspired by the Agent Oriented Software Engineering (AOSE) methodologies that integrates a perception, reasoning and action cycle.

The core element of the metamodel is the *Person* metaclass. It represents the type of people involved in traffic. According to their means of transport, these people can take the role of drivers, passengers and pedestrians. *Person* instances can interact with an instance of *Environment*. This interaction is direct (in the case of pedestrians by reference *Perceives*) or indirectly (for drivers and passengers by reference *Interacts*) as a vehicle is used for it. Its objectives are described by *Goal* instances, and possible ways to achieve them are represented by *Task* instances. *Evaluator* instances interpret available information (perhaps generating new information) and determine how people must act according to the circumstances observed (i.e. potential tasks that can run in given circumstances). Finally, *Actuator* instances run scheduled tasks.

The above elements are organised into inheritance hierarchies. These hierarchies provide the specialisation of concepts and the necessary structure to the metamodel. All concepts extend from the *GeneralElement* metaclass. This metaclass provides the *EInherits* reference to represent inheritance among elements of the same type in model specifications. Meanwhile, the *GeneralRelationship* metaclass supports the introduction of relations (e.g. affects or influences) among others. The reference *RInherits* allows its specialisation. Both types of references are limited by OCL constraints. For example, these restrictions only allow inheritance between instances of the same type of metaclasses (in the case of an instance of *Knowledge*, it can only be extended using an instance of *EInherits* reference to another instance of *Knowledge*).

Metaclasses also include predefined attributes and methods. The first can be specific of a metaclass, (as *AvailableArea* in the *Environment* metaclass), or common, with similar meaning and name (such as *XName* attributes, for example *KName* or *VName*). Nevertheless, note that the particular case of the *Id* attribute in *Person* metaclass, which has a similar meaning to these *XName* attributes, although the names are different. Furthermore, methods are containers for specifications describing behaviour or how to derive certain attributes from others.

The internal structure of the *Mental* and *Environment* clusters allows hierarchies composition using *XComponent* metaclasses (e.g. *VComponent* or *KComponent*). These metaclasses can be decomposed into others of the same type. All this compositions are limited by OCL restrictions. For example, a *Profile* instance can be decomposed only into instances *PComponent*, while these *PComponent* instances can only be decomposed into others of the same type.



2.2 Mental cluster

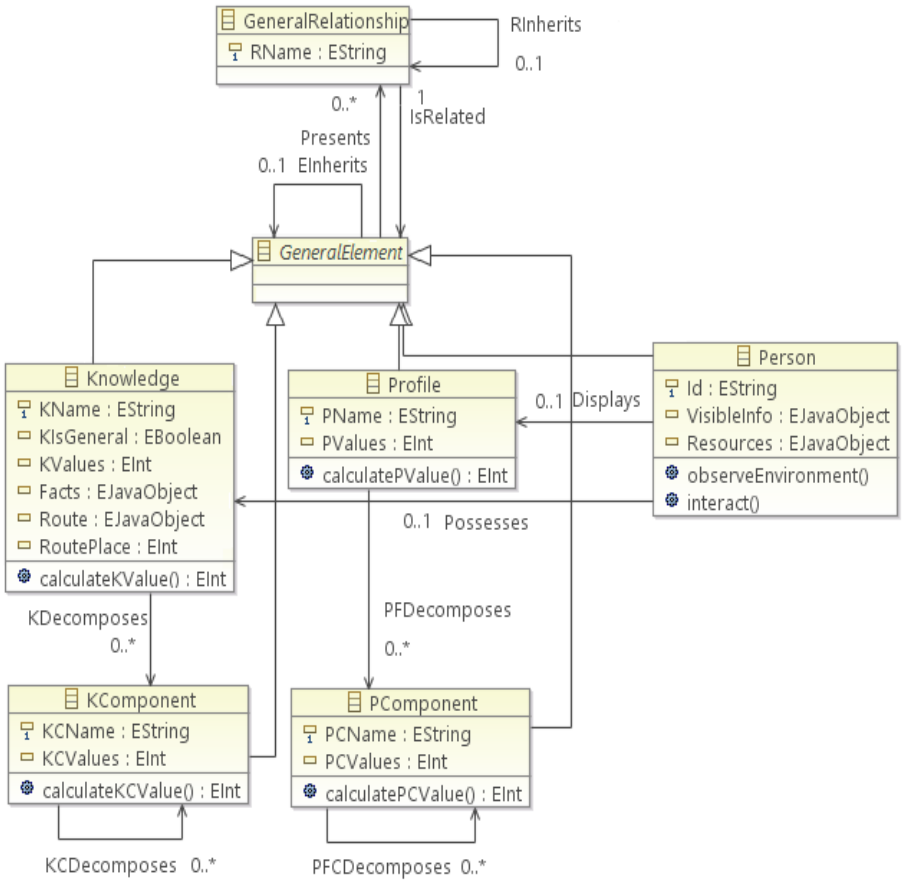
Mental cluster represents the various items that can appear in the domain of road traffic and that can influence the behaviour of individuals. These concepts are classified as characteristics or current states of individuals.

This cluster consists of three main metaclasses: *Person*, *Profile* and *Knowledge*. *Profile* represents the different characteristics of the people involved in traffic (for example, age or fatigue). *Knowledge* considers mental status and the current knowledge (except the objectives

of the person), and plans. This information is stored by the attribute *Facts*. *Knowledge* can be factual (e.g. street signs), procedural (e.g. how to overtake a vehicle), and policy (e.g. drivers must respect the safety distance from other vehicles). The plans of individuals regarding the traffic are often given by the route to follow. This route is taken into account by the *Route* attribute, while the progress on the route by individuals is given by the *RoutePlace* attribute.

Both knowledge of people and their characteristics can specify information that does not change during the simulation (e.g. the gender or the meaning of the signs), or that it does (e.g. the level of stress or mood). To view these aspects, the metamodel provides *XValues* attributes associated to each metaclass and their corresponding methods *calculateXValue*.

Instances of *Knowledge* metaclass and their composition *KComponent* metaclasses may represent information that belongs to individuals or globally, being available to all participants in the simulation. For example, speed limits are common to a model, but some participants may ignore them. *KIsGeneral* attribute differentiates both uses

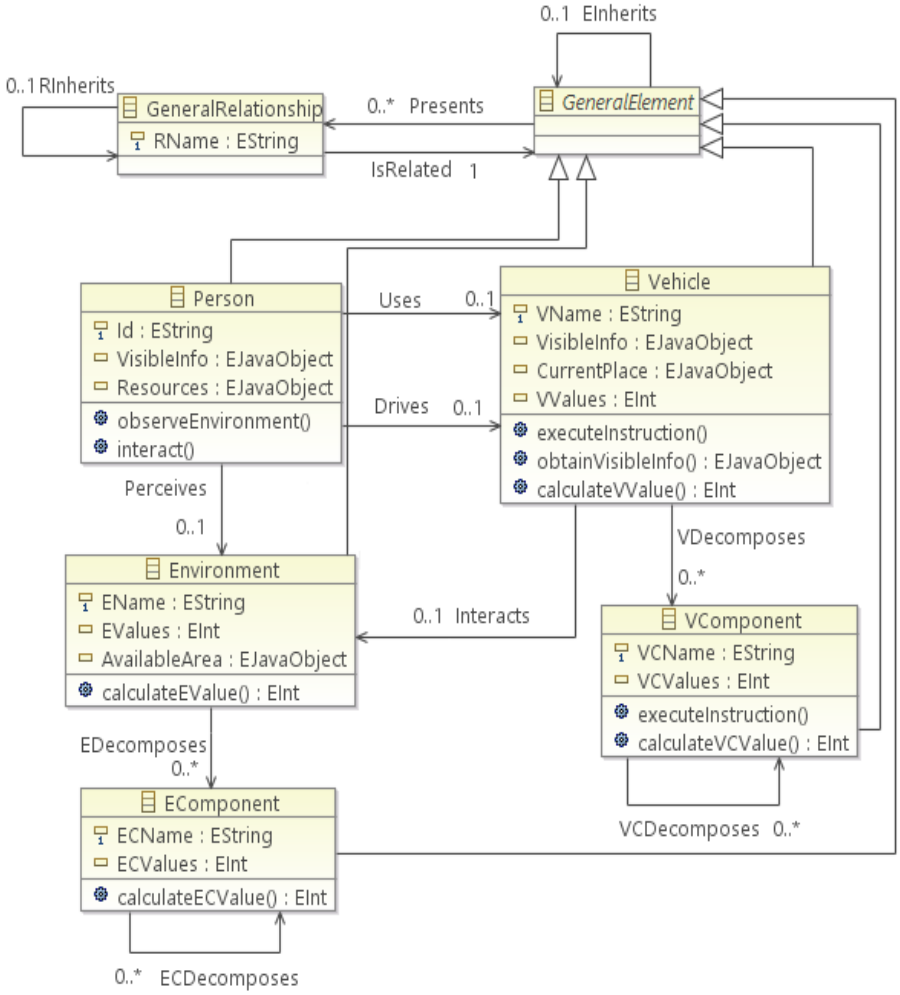


2.3 Environment cluster

Road traffic occurs in an environment that provides certain physical conditions (e.g. time, lane width, or obstacles). The *Environment* cluster considers these issues by adapting concepts of DVE model focusing on the role played by individuals involved in road traffic. It supposes that these individuals can obtain information from the environment (in the case of any person participating in traffic) and the vehicle in which they are (only drivers and passengers). These considerations and related elements can be extended to accommodate other similar theories.

The cluster is in charge of describing how people relate to their means of transport and the environment that surrounds them. To carry out these descriptions, it uses three main metaclasses: *Person*, *Environment*, and *Vehicle*.

Drivers use the reference *Drives* to relate instances of *Person* metaclass with instances of the *Vehicle* metaclass, while passengers adopt *Uses* reference. Pedestrians do not use any of these relationships as they do not relate to any mean of transport.



Environment metaclass represents where people (i.e. instances of the *Person* metaclass) interact, including physical conditions that may occur (e.g. weather and the characteristics of the road). These conditions are considered by instances of the *EComponent* metaclass.

Vehicle metaclass symbolises the means of transport used by each individual. Drivers and passengers relate to *Environment* instances through its vehicles, but only drivers can perform actions on them to act on the vehicle and the environment. For pedestrians, they present a direct reference with the environment. *VComponent* metaclasses eases the insertion of characteristics of the means of transport (e.g. engine or vehicle size).

The mutual influences between instances of *Person*, *Environment* and *Vehicle* through their relationships are partly represented in the metamodel by some attributes and methods. The *Environment* metaclass has an *AvailableArea* attribute that indicates the global scenario where the simulation occurs. *Person* and *Vehicle* metaclasses include *VisibleInfo* attribute to specify what information coming from attribute *AvailableArea* of the *Environment* instance can be perceived. The *Person* metaclass has the *observeEnvironment* method to update the perception of the environment to modify the *VisibleInfo* attributes of their own instances. Also, it presents an *interact* method to carry out interactions with the environment and with other individuals. The *Vehicle* metaclass presents a method called *obtainVisibleInfo* in charge of conducting a similar task to *observeEnvironment* method of the *Person* metaclass. In addition, this metaclass has the *executeInstruction* method intended to perform specific tasks of the vehicle (for example, moving a mirror or turn lights). *VComponent* metaclass presents this same method with a similar functionality.

2.4 Interactive cluster

Interactive cluster considers how the instances of *Person* acting on traffic situations described with elements of *Mental* (see Section 2.2) and *Environment* (see Section 2.3) clusters. The components are organised into two groups. The first one describes the goals of individuals and their abilities to try to reach them. The second represents the elements that perform a perception, reasoning and acting cycle.

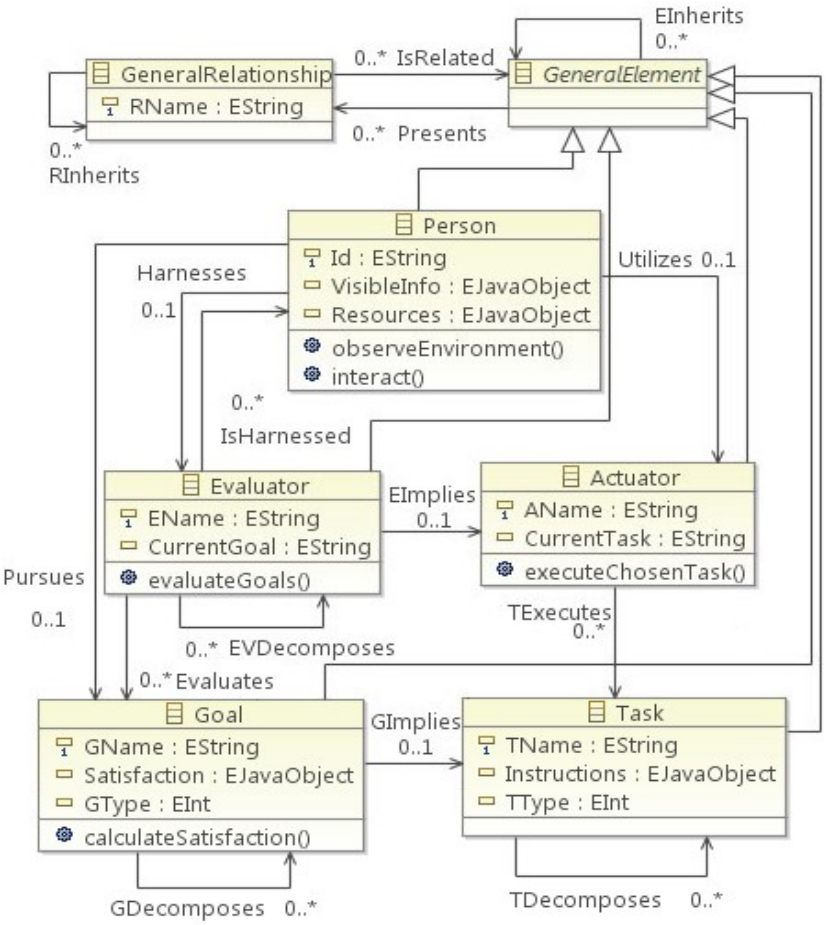
The first group includes *Goal* and *Task* metaclasses. These two concepts are taken from AOSE where agent-based methodologies used them to specify the Multi-Agent Systems (MAS). These methodologies include architecture-specific performance where actors play multiple roles according to their different skills, knowledge and goals. Agents try to enforce the conditions of satisfaction of their different goals. To do this, agents have tasks related to the goals, so that the execution of the tasks is potentially able to satisfy the goals. These tasks are enabled and enforceable in accordance with conditions that depend on the mental state of the agents and the environment. Both tasks and goals can be decomposed into others respectively.

The second group has the elements of an instance of *Person* that are in charge of assessing the known state (environmental and internal agent) and executing actions. In the perception, reasoning and acting cycle the information perceived from the environment is stored in the

elements of the *Environment* cluster (including the *Person* metaclass), the reasoning is achieved by the instances of the *Evaluator* metaclass, and the acting is carried out by instances of the *Actuator* metaclass.

Evaluator instances can be decomposed into others through the *EVDecomposes* reference, allowing distribution of responsibilities among them. *Actuator* instances cannot be decomposed. Each *Person* instance (i.e. a type of person) can only present one *Actuator* instance with it joined through *Utilizes* reference. Instances of both metaclasses can be refined using inheritance through *EInherits* references.

Regarding the performance of the cluster, *Evaluator* instances evaluate the information obtained from the *Environment*, *Vehicle*, *Profile* and *Knowledge* instances linked to their *Person* instance, its components, and other possible elements related through *GeneralRelationship* instances. To carry out this process these *Evaluator* instances have the predefined method *evaluateGoals*. With it they update the internal state of their *Person* instance. All this information determines the current state of *Goal* instances, that is, if they are satisfied or not. For it they use the *calculateSatisfaction* method. Once a candidate *Goal* instance is selected to be executed, an *Actuator* instance collects its associated tasks. These *Task* instances are performed using the *executeChosenTask* method following the atomic instructions in their *Instructions* attributes or executing their children *Task* instances.


EInherits
0..*
GeneralRelationship
RName : EString
0..* IsRelated
0..* Presents
GeneralElement
0..*
RInherits
Person
Id : EString
VisibleInfo : EJavaObject
Resources : EJavaObject
observeEnvironment()
interact()
Harnesses
0..1
Utilizes 0..1
0..*
IsHarnessed
Evaluator
EName : EString
CurrentGoal : EString
evaluateGoals()
EImplies
0..1
Actuator
AName : EString
CurrentTask : EString
executeChosenTask()
Pursues
0..1
TExecutes
0..*
0..* EVDecomposes
0..* Evaluates
Goal
GName : EString
Satisfaction : EJavaObject
GType : EInt
calculateSatisfaction()
GImplies
0..1
Task
TName : EString
Instructions : EJavaObject
TType : EInt
GDecomposes 0..*
TDecomposes 0..*