MODELLING AND ANALYSIS OF TRAFFIC NETWORKS BASED ON GRAPH TRANSFORMATION

Juan de Lara [1], Hans Vangheluwe [2], Pieter J. Mosterman [3]
[1] *Escuela Politécnica Superior, Universidad Autónoma de Madrid*
Address: Campus Cantoblanco, Ctra. Colmenar Km. 15., 28045 Madrid (Spain)
Phone: (+34) 91 497 22 77, Fax: (+34) 91 497 22 35, e-mail:Juan.Lara@ii.uam.es
[2] *School of Computer Science, McGill University*
Address: 3480 University Street, Montreal, Canada H3A 2A7
Phone: +1 (514) 398 44 46, Fax: +1 (514) 398 38 83, e-mail: hv@cs.mcgill.ca
[3] *The Mathworks Inc.*
Address: 3 Apple Hill Dr., Natick, Massachusetts 01760, USA
Phone: +1 (508) 657 7765, Fax: +1 (508) 647 7712, e-mail: pieter_j_mosterman@mathworks.com

abstract: We present the formal definition of a domain specific visual language (*Traffic*) for the area of traffic networks. The syntax has been specified by means of meta-modelling. For the semantics, two approaches have been followed. In the first one, graph transformation is used to specify an operational semantics. In the second one we include timing information and a denotational semantics is defined in terms of Timed Transition Petri Nets (TTPN). The transformation from the *Traffic* formalism into TTPN was also defined by graph transformation. Both approaches have been used for the analysis of *Traffic* models. The ideas have been implemented in the AToM^3 tool and are illustrated with examples.
Keywords: Meta-Modelling, Graph Transformation, Domain Specific Visual Languages, Traffic Networks, Petri nets.

1 INTRODUCTION

Domain Specific Visual Languages (DSVL) are specialized notations for specific business areas, such as manufacturing, logistics or traffic networks (Vangheluwe and de Lara, 2004) . They offer high-level building blocks that encapsulate patterns commonly used in the specific domain. Thus, in well understood domains, DSVL have the potential to greatly increase productivity, due to the power of such high-level primitives. Nonetheless, one important issue is how to formally define the syntax and semantics of such DSVL.

The approach we take in the present work is to formally define the DSVL syntax by means of meta-modelling. In meta-modelling, a model (called meta-model) describes the valid models in the DSVL. For the specification of such meta-models, one can use languages such as UML class or entity relationship diagrams. In addition, constraint languages such as OCL can be used to reduce the kind of admissible models. A meta-model defines the abstract syntax of a visual language. In addition, information has to be given about how the elements are visualized. This is called the concrete syntax. With information about the concrete and abstract syntax, a meta-modelling tool is able to generate a customized modelling environment for the defined language (de Lara and Vangheluwe 2002).

For a formal specification of the DSVL semantics, we use graph transformation (Rozenberg, 1997) (Ehrig et al. 1999). In this framework, and in a similar way as Chomsky grammars, rules specify modifications to be performed on the models. In this way, one can use graph transformation to specify the operational semantics of the DSVL (by defining a simulator). Another approach is to define a transformation of the original DSVL into a formally defined semantic domain. This can be seen as the definition of a denotational semantics. For both approaches, graph transformation has the advantage of being a graphical and formal framework, which makes transformations subject to analysis. In the case of the definition of the operational semantics by graph transformation, one can use the theortical results of graph grammars (Rozenberg 1997) for behaviour

analysis. In the case of specification of model transformations into semantic domains, theoretical results of graph grammars can be used to verify the correctness of the transformation. Once the transformation is performed, it is possible to use the analysis techniques of the target formalism.

In this paper, we present an improvement of our *Traffic* DSVL (Vangheluwe and de Lara, 2004). This language allows the user to build traffic networks with intersections, vehicles sources and sinks and maximum capacities for road segments. The semantics of the formalism were defined in (Vangheluwe and de Lara, 2004) by a model transformation into Petri nets. In this work, we describe the operational semantics by means of graph transformation. This allows us to use graph transformation techniques to specify global safety properties and ensure that they are met by the semantics. In addition, this definition permits the animation of the models in our AToM^3 tool. We have also improved the *Traffic* formalism with traffic lights and timing information and defined a transformation for the timed model into TTPN (Ramchandani, 1973), which allows for simulation and performance evaluation.

The rest of the paper is organized as follows: section 2 presents a brief introduction to graph transformation. Section 3 shows the traffic formalism and how it has been implemented in AToM^3. Section 4 gives the (untimed) operational semantics in terms of graph transformation systems. Section 5 shows the timed semantics by a transformation into TTPN. In section 6 we discuss some related research. Finally, section 7 terminates with the conclusions and proposes directions for future work.

2 GRAPH TRANSFORMATION

Graph grammars are a generalization of Chomsky grammars for graphs (Rozenberg, 1997) (Ehrig et al. 1999). Graph grammars are composed of production rules, each having graphs in its left and right hand sides (LHS and RHS). In the *Double Pushout Approach* (DPO), productions have the form:

$$p : L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R \qquad (1)$$

where L (left hand side), K (interface graph) and R (right hand side) are graphs and l and r are (usually injective) morphisms. That is, K is the set of nodes and edges that are preserved by the production, L-K is the set of nodes and edges that are deleted and R-K is the set of nodes and edges that are created by the production. The diagram in Figure 1 sketches the application of a rule on a graph G, resulting in graph H.

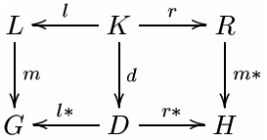

Fig. 1 Application of a Rule to Graph G

Thus, in order to apply a production to a graph G, a match *m* should be found between the production's LHS L and the graph G. This can be either an injective or non-injective morphism. The next step is to delete all the elements in G matched with elements of L-K. Finally, the elements of R-K are added. Note how this process can be expressed in terms of category theory as two pushouts in category **Graph**. Additionally, the double pushout approach needs two additional conditions. The *dangling condition* specifies that if an edge is not deleted its source and target nodes should be preserved. The identification condition specifies that if two nodes or edges are matched into a single node or edge in the host graph (via a non-injective morphism), then both should be preserved.

Productions can be extended with sets of application conditions (AC) (Heckel and Wagner, 1995) of the form:

$$\{P \xrightarrow{c_i} Q\} \qquad (2)$$

and a morphism x from L to P. This means that in order to apply the rule to a host graph, if an occurrence of P is found then an occurrence of Q must be found in order for the rule to be applicable. Note that, if c_i is empty, we have a negative application condition (NAC). In this case, if an occurence of graph P is found, then the rule is not applicable. If $x=id_L$, then we have a positive application condition.

3 THE TRAFFIC FORMALISM

In this section we use meta-modelling to formally define the syntax of the Traffic formalism. We first define an untimed version of the formalism, and later extend it with timing information. Traffic models are made of "RoadSections", which can be connected by "FlowTo" relationships. "RoadSections" contain the number of vehicles at a certain moment. "Capacity" entities limit the number of vehicles that can be present at the same time in a number of "RoadSections". "Source" entities generate vehicles, while "Sink" entities consume vehicles, eliminating them from the model. Finally, "TrafficLights" can be placed in "RoadSections" by means of the "ControlledSection" relationship. This models the fact that the semaphore is physically placed in a certain road segment. The second relationship "Direction" indicates which outgoing road the traffic light is controlling. Two traffic lights can be synchronized, in such a way that one changes to red when the other changes to green.

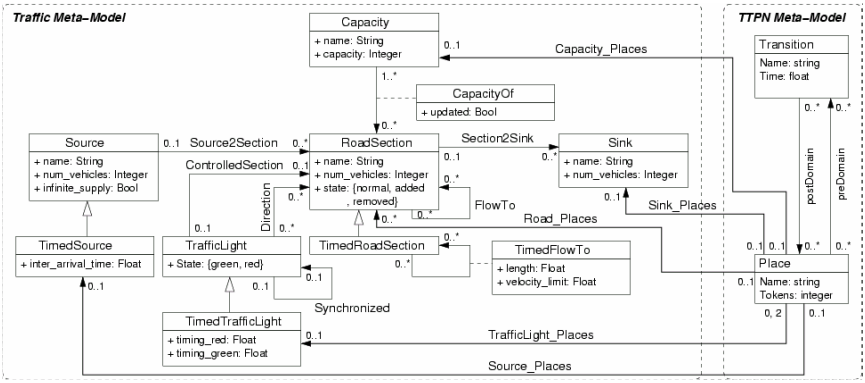

Fig. 2: The Traffic Meta-model (left)

We have extended the untimed model with timing information. This is done by adding new classes that add the timing attributes. For example, the "TimedSource" entity adds the "inter_arrival_time" attribute to indicate the time interval at which new vehicles should be generated. In this way two kinds of models can be built. In untimed models, it is possible to use only the untimed version of the classes. In timed models, only the timed version can be used, together with instances of classes "Sink" and "Capacity" which do not have a timed counterpart. Note how both formalisms can be used to specify a traffic network at two different levels of abstraction. The untimed models have less detail (do not considers timing information) and permit different kinds of analyses (for example mapping into untimed Petri nets and subsequently reachability analysis). Note how, this higher level of abstraction can always be deduced from the timed model simply by ignoring the timing information.

With this meta-model and information about how the elements should be visualized, the AToM^3 tool (de Lara and Vangheluwe, 2002) is able to generate a modelling environment for the Traffic formalism. Figure 3 shows an example with a simple (timed) Traffic model.

The circle-like icon on the left is a "source" entity, the square-like icons on the top-right and bottom are "sinks". Two synchronized traffic lights control the perpendicular road sections of the crossing. Note also that a customized user interface is generated for the defined visual language. It allows inserting the domain specific elements and to execute further functionality specified as Python programs or as graph transformation.

The meta-model presented before only defines the abstract and concrete syntax of Traffic models. To define the semantics, AToM^3 allows the definition of graph transformation rules. The next

section shows two approaches for the definition of such semantics.

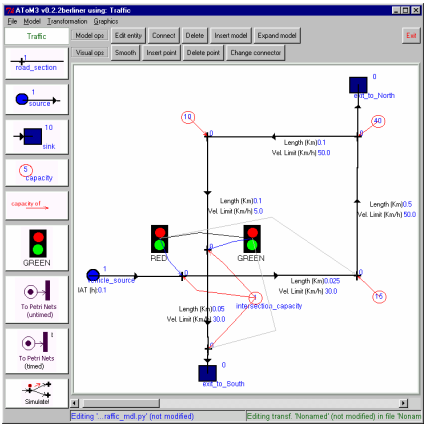

Fig. 3: Building a Traffic model with AToM^3.

4 OPERATIONAL SEMANTICS (UNTIMED)

In this section we show how to use graph transformation rules to define the operational semantics of untimed Traffic models. In addition, graph transformation techniques permit the definition of global conditions, undesired states or safety properties that must hold in all possible executions of the models (Heckel and Wagner, 1995). The global conditions can be translated into local application conditions for each rule in the transformation (see section 2). In this way, the transformation system satisfies the safety properties by construction.

Figure 4 shows some of the rules of the graph grammar that defines the operational semantics. The interface graph K mentioned in section 2 is ommited, but the morphisms between LHS and RHS are indicated by numbers (only in nodes). In this way, the elements of the interface graph K are those with the same number in LHS and RHS.

Rule “Generate Vehicle” moves a vehicle from a “Source” node into a connected “RoadSegment” node. The number of vehicles in the “Source” node is decreased if it does not have an infinite capacity. The NAC prohibits the application of the rule if the “RoadSegment” node is connected to a “Capacity” node which has a capacity of zero. Note how, if the rule is applied, the state of the “RoadSegment” node is changed to “added”. This means that all the connected “Capacity” nodes should be decreased. This is performed by rule “Update Capacities”.

Rule “Change Traffic Light State” simply changes the traffic light state. There are additional rules not shown in this paper, for example to move vehicles between two “RoadSegment” nodes, to increase the capacity, to consume vehicles by “Sink” nodes, and to synchronize traffic lights.

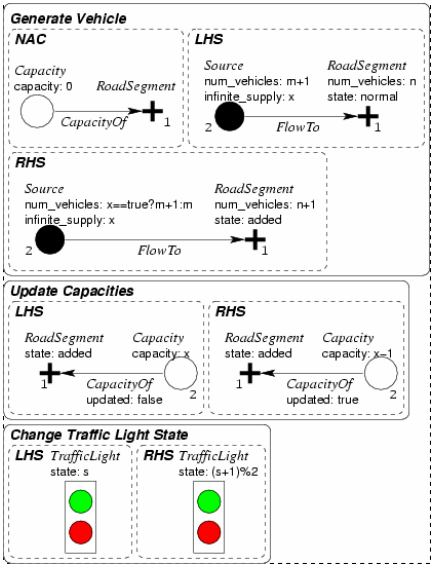

Fig. 4: Some Rules for the Definition of Traffic Operational Semantics (Untimed).

The graph grammar defines the semantics of a given start model as all the reachable models that result from the application of the rules. Note how there is no control flow for the execution of the rules, but they are tried at random. Execution finishes when there is no applicable rule.

Figure 5 shows a simple global safety condition, which specifies a non desired situation in Traffic models. The condition specifies that two semaphores which control an intersection cannot be in green at the same time. Following the procedure shown in (Heckel and Wagner, 1995) it is possible to translate this global negative application

condition into local (pre-)conditions for each rule in the graph grammar. In this way, we assure that the non desired situation cannot happen in any reachable model. For example, for rule "Change Traffic Light State", the safety condition induces a NAC that does not let the rule to be applied if it sets to green a traffic light that is in an intersection and there is already another one in green.

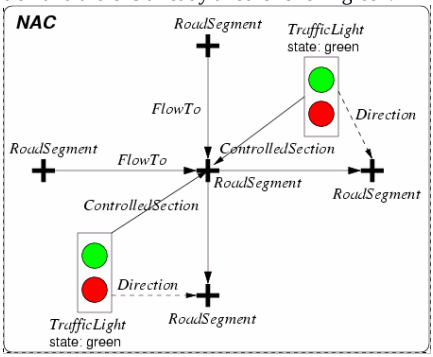

Fig. 5: A Safety Condition.

Additional graph grammar techniques allow further analysis of models. For example, we can use critical pair analysis (Heckel et al., 2002) to check if rules are independent. If they are, then they can be applied in parallel. In our example, the possibility to apply the rules for moving vehicles in parallel shows the independent parts of the traffic network.

5 DENOTATIONAL SEMANTICS (TIMED)

In this section we briefly present a graph transformation system for translation of timed traffic models into TTPN. Note how, in a timed model (which is made of instances of the timed version of the classes in the meta-model) it is still possible to apply the transformation of previous section. The reason is that in AToM^3 a node in a rule can match with nodes of the same class, or any of their subclasses in the meta-model (Bardohl et al., 2004). However the (denotational) semantics we present in this section is only applicable to timed models.

The graph grammar rules for the translation are similar to the ones we showed in (Vangheluwe and de Lara 2004), but the target formalism in the present work is TTPN, where transitions have a delay. In this way, transitions have to be enabled for a certain period of time before they can fire. We use this delay to simulate the time it takes a vehicle to move from one "RoadSegment" to another.

For the specification of the transformation, we use the source language meta-model, as initial model are instances of this meta-model. After the transformation, the resulting models are instances of the TTPN meta-model. During the transformation, we use auxiliary elements to relate source language elements to target language elements. For example, during the transformation, each "RoadSegment" is assigned a "Place". But before the transformation ends, "RoadSegments" and their connection to the newly created "Places" are deleted. The relationships between both meta-models are shown in Figure 2.

Some of the transformation rules are shown in Figure 6. The first rule associates a unique "Place" to each "RoadSegment". Note how the NAC prohibits assigning more than one "Place" to each "RoadSegment". The second rule creates a "Transition" for each "FlowTo" relationship between two "RoadSegment" nodes. The delay for the transition is calculated in seconds (velocity and length for "FlowTo" were in km/h and km respectivelly).

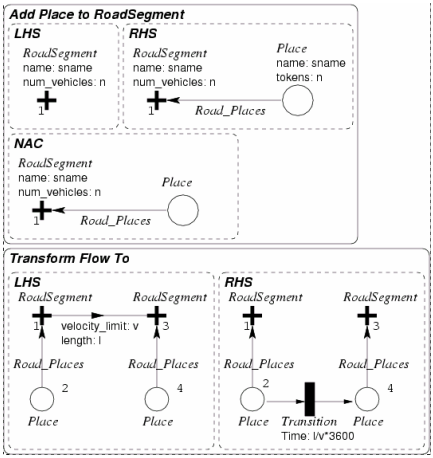

Fig. 6: Some Rules for the Transformation into TTPN

The semantics we use in case of conflicts of enabled transitions is preselection with a random probability. Otherwise the transition representing the shorter segment would always be fired when in conflict with other transitions. We also use "atomic firing" (tokens stay in places until transitions fire), "infinite server" semantics (transitions are provided with infinite timers) and "enabling memory" (i.e. timers of transitions being disabled due to a transition firing are reset).

We can use graph transformation techniques [some of them presented in (de Lara and Taentzer, 2004)] to analyze the transformation itself. For example, we are interested in termination, confluence (that a unique target model can be obtained with the transformation), syntactic consistency (that the target model we obtain is a correct instance of the target meta-model) and semantic consistency (that the transformation preserves some semantic properties, like behaviour).

Once the model has been translated, we can use Petri net techniques for analysis. In (Vangheluwe and de Lara, 2004) we used untimed Petri net techniques for analysis, in particular the ones based on the coverability graph. With timing information, we can use simulation to evaluate the performance of the designed system.

Figure 8 shows the resulting TTPN model from the transformation of the model in Figure 3.

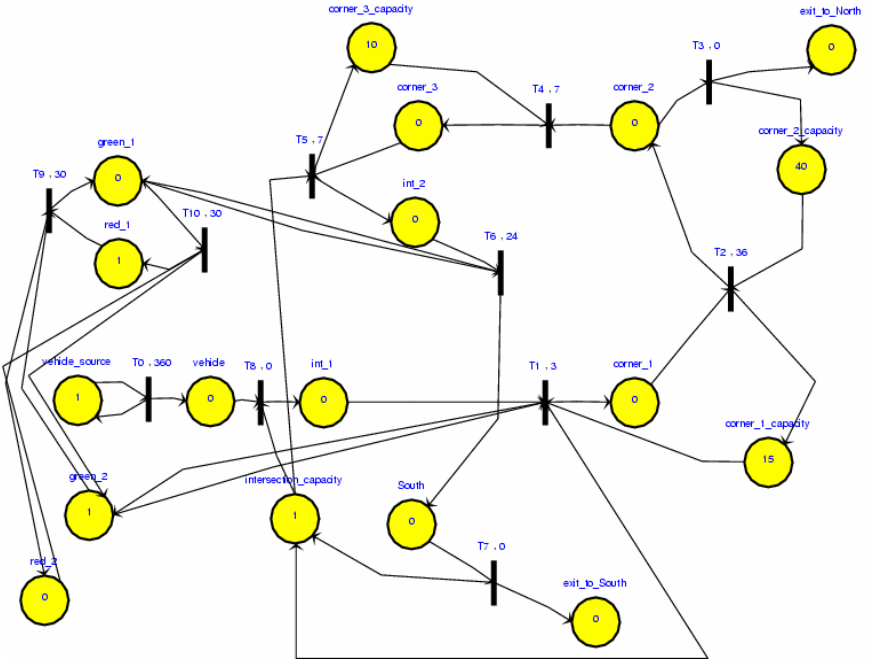

Fig. 8: TTPN Model Resulting from the Transformation of Model in Fig. 3

6 RELATED WORK

There are several tools that allow the definition of Visual Languages and their manipulation

by means of graph grammars. Among them GenGED (which does not follow an approach based on meta-models). With respect to the analysis of graph transformations, it is possible to use the attributed graph grammar (AGG) too (Taentzer et al., 1999), which allows finding critical pairs. In our case, we have used AToM^3 for the definition of the visual language and the specification of the transformations. We built a translator to export from the AToM^3 format into AGG (de Lara and Taentzer, 2004), which allows the analysis of the graph transformation rules.

Extensive research has been carried out in the area of traffic network modelling and simulation [for a survey, see for example (Pursula 1999)]. Nonetheless, the presented work is original in the sense that uses a combination of meta-modelling and graph transformation for the definition and analysis of the system semantics.

7 CONCLUSIONS

In this paper we have presented a DSVL for the definition of traffic networks. We have defined its semantics by means of graph transformation and by its mapping into untimed and timed transition Petri nets. In the first case, graph transformation allows the definition of the operational semantics and the specification of safety properties that the system behaviour must preserve. The mapping into Petri nets is also specified by means of graph transformation, which allows the analysis of the transformation itself. Once the model is transformed into a Petri net, we can exploit Petri net analysis and synthesis techniques to apply to the model in the DSVL.

Note how mappings of Traffic models into other simulation formalisms, such as GPSS (Schriber, 1974) and DEVS (Zeigler et al. 2000) are also possible. This allows the use of the different formalisms analysis and simulation tools.

ACKNOWLEDGEMENTS: Juan de Lara's work has been partially sponsored by a grant from the E.U. SEGRAVIS research network (HPRN-CT-2002-00) and the Spanish Ministry of Science and Technology (TIC2002-01948). Hans Vangheluwe gratefully acknowledges partial support for this work by a National Sciences and Engineering Research Council of Canada (NSERC) Individual Research Grant.

LITERATURE

Bardohl, R., Ehrig, H., de Lara J., and Taentzer, G. 2004. "Integrating Meta Modelling with Graph Transformation for Efficient Visual Language Definition and Model Manipulation". In proceedings of ETAPS/FASE'04, LNCS 2984, pp.: 214-228. Springer.

Bardohl, R. 2002. "A Visual Environment for Visual Languages". Science of Computer Programming 44,pp.: 181-203. See also the GenGED home page: http://tfs.cs.tu-berlin.de/~genged/

Ehrig, H., Engels, G., Kreowski, H.-J. and Rozenbergs, G. editors. 1999. The Handbook of Graph Grammars and Computing by Graph Transformation. Vol 2. World Scientific.

Heckel, R., Wagner, A. 1995. "Ensuring consistency of conditional graph rewriting - a constructive approach". Proc. of SEGRAGRA 1995, Joint COMPUGRAPH/SEMAGRAPH Workshop on Graph Rewriting and Computation, In ENTCS Vol 2, 1995.

Heckel, R., Küster, J. M., Taentzer, G. 2002. "Confluence of Typed Attributed Graph Transformation Systems". In ICGT'2002. LNCS 2505, pp.: 161-176. Springer.

de Lara, J., Taentzer, G. 2004. "Automated Model Transformation and its Validation with AToM^3 and AGG". In DIAGRAMS'2004 (Cambridge, UK). Lecture Notes in Artificial Intelligence 2980, pp.: 182-198. Springer.

de Lara, J., Vangheluwe, H. 2002. "AToM^3: A Tool for Multi-Formalism Modelling and Meta-Modelling". In ETAPS/FASE'02, Lecture Notes in Computer Science 2306, pp.: 174 - 188. Springer-Verlag. See also the AToM3 home page at: http://atom3.cs.mcgill.ca

Pursula, M., 1999. "Simulation of Traffic Systems - An Overview". Journal of Geographic Information and Decision Analysis, vol.3(1), pp. 1-8.

Ramchandani, C. 1973. "Performance Evaluation of Asynchronous Concurrent Systems by

Timed Petri Nets". Ph.D. Thesis, Massachusetts Inst. of Tech., Cambridge.

Rozenberg, G. (ed) 1997. "Handbook of Graph Grammars and Computing by Graph Transformation". World Scientific. Volume 1.

Schriber, T. J., "Simulation Using GPSS". Wiley, 1974.

Taentzer G., Ermel C., Rudolf M. 1999. "The AGG Approach: Language and Tool Environment", in (Ehrig et al. 1999), See also the AGG Home Page: http://tfs.cs.tu-berlin.de/agg

Vangheluwe, H., de Lara, J. 2004. "Computer Automated Multi-paradigm Modelling for Analysis and Design of Traffic Networks", to appear in proc. 2004 Winter Simulation Conference. Washington, USA.

Zeigler, B. P., Praehofer, H., Kim, T. G. 2000. "Theory of Modeling and Simulation 2nd Edition. Integrating Discrete Event and Continuous Complex Dynamic Systems". Academic Press.