Proceedings of the 2004 Winter Simulation Conference
R. Ingalls, M. Rossetti, J. Smith, B. Peters, eds.

COMPUTER AUTOMATED MULTI-PARADIGM MODELLING FOR ANALYSIS AND DESIGN OF TRAFFIC NETWORKS

Hans Vangheluwe

School of Computer Science
McGill University
Montréal, Québec H3A 2A7, CANADA

Juan de Lara

ETS Informática
Universidad Autonóma de Madrid
Madrid, SPAIN

ABSTRACT

Computer Automated Multi-Paradigm Modelling (CAMPaM) is an enabler for *domain-specific* analysis and design. Traffic, a new untimed visual formalism for vehicle traffic networks, is introduced. The *syntax* of Traffic models is meta-modelled in the Entity-Relationship Diagrams formalism. From this, augmented with concrete syntax information, a visual modelling environment is synthesized using our CAMPaM tool AToM^3, *A Tool for Multi-formalism and Meta-Modelling*. The *semantics* of the Traffic formalism is subsequently modelled by mapping Traffic models onto Petri Net models. As models' abstract syntax is graph-like, *graph rewriting* can be used to transform models. The advantages of a domain-specific formalism such as Traffic as opposed to a generic formalism such as Petri Nets are presented. We demonstrate how mapping onto Petri Nets allows one to employ the vast array of Petri Net analysis techniques. A Coverability Graph is generated and *conservation analysis* is automated by transforming this graph into an Integer Linear Programming specification.

1 INTRODUCTION

Computer Automated Multi-Paradigm Modelling (CAMPaM) (Mosterman and Vangheluwe 2002) aims to simplify the modelling of complex systems by combining three orthogonal directions of research:

- *Meta-Modelling*, which models (the syntax of) modelling formalisms;
- *Model Abstraction*, concerned with the relationship between models at different levels of abstraction;
- *Multi-Formalism modelling*, concerned with the coupling of and transformation between models described in different formalisms.

In the sequel, we focus on meta-modelling and on multi-formalism modelling to build a *domain-specific* modelling environment for the Traffic formalism.

Meta-modelling can help in defining high abstraction level notations. With meta-modelling, we can describe, using a high-level, graphical notation, the (possibly graphical) syntax of languages for particular needs: domain-specific visual languages. Such languages have the potential to greatly increase system quality and reduce development costs, as they are notations tailored to specific needs.

Some languages such as the UML are rigourously defined through meta-modelling. But meta-modelling the syntax of a language is only one side of the coin. One needs to formally specify the semantics of a language. We may be interested in defining a language's *operational* semantics (*i.e.,* how models described in the language are simulated or executed), or its *denotational* or *transformational* semantics (*i.e.,* defining a mapping onto another well-defined language; this may include code generation when mapping onto a virtual machine). We may also wish to *optimize* the models (*i.e.,* reduce the complexity without removing salient features). As models, meta-models and meta-metamodels can all be described as attributed, typed graphs, we present Graph Grammars (Ehrig, Engels, Kreowski, and Rozenberg 1999), a formal, graphical and high-level notation to specify the model transformations.

We have implemented these meta-modelling and graph transformation concepts in a tool called AToM^3, *A Tool for Multi-formalism and Meta-Modelling*. AToM^3's design has been described in (de Lara and Vangheluwe 2004, de Lara and Vangheluwe 2002, de Lara Jaramillo, Vangheluwe, and Alfonseca Moreno 2003). In AToM^3, we follow the maxim *everything is a model*. That is, not only formalisms and transformations are modelled explicitly, but also composite types and the user interfaces of the generated tools. In fact, the entire AToM^3 tool was bootstrapped from a small kernel with code-generating capabilities.

Section 2 introduces the Traffic formalism for modelling vehicle traffic networks. Section 3 demonstrates, by means of the Traffic meta-model, the meta-modelling concepts and how they are implemented in AToM^3. Section 4 discusses graph rewriting. Section 4.1 gives the semantics of the Traffic formalism by mapping it onto the Petri Net formalism.

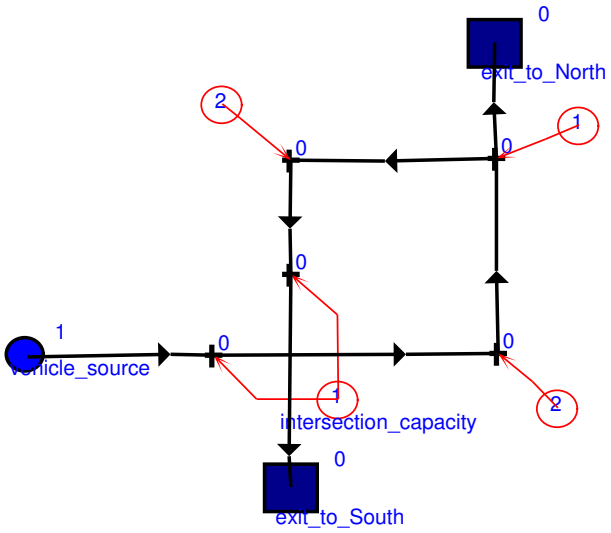

Figure 1: A Traffic Model

Subsequently, the Petri Net resulting from such a transformation is transformed into a Coverability Graph. Further transformation into an Integer Linear Programming problem allows for conservation analysis. Finally, section 4.2 shows an alternate semantics for the Traffic formalism. Section 5 draws some conclusions on CAMPaM in general and domain-specific modelling in particular.

2 THE TRAFFIC FORMALISM

Domain- and formalism-specific modelling have the potential to greatly improve productivity (Kelly and Tolvanen 2000). They are able to exploit features inherent to a specific domain or formalism. This will for example enable specific analysis techniques or the synthesis of efficient code.

To illustrate domain-specific modelling, we introduce the Traffic formalism, a new visual notation specific to the vehicle traffic domain (Papacostas and Prevedouros 1992). It is of course possible to model traffic systems using a variety of modelling and simulation languages such as GPSS, DEVS, and Petri Nets. We choose not to do this, but rather build a Traffic-specific modelling environment. This maximally constrains users, allowing them, by construction, to *only* build syntactically and (for as far as this can be statically checked) semantically correct models. Furthermore, the Traffic-specific, visual syntax used matches the users' mental model of the problem domain. Note how all advantages of the aforementioned formalisms are not lost as we will map Traffic models onto them. In this article, the Traffic semantics is expressed by mapping onto Petri Nets.

Figure 1 shows a small traffic system in which vehicles arrive into the system via a `vehicle_source`, go straight across an intersection (when no other vehicles are present), turn left on a short road section which can only hold two vehicles, and either leave the system through `exit_to_North`, or turn left. Turning left brings them across another short road section which can only hold two vehicles, back to the first intersection. After succesfully crossing this intersection, they leave via `exit_to_South`.

Vehicle arrival is denoted by a filled circle. Vehicle departure is denoted by a filled rectangle. A cross denotes a road section which can have a time-varying number of vehicles in it. Road sections are connected by arrows. Multiple arrows departing from a single road section indicates a choice. A capacity constraint circle may be connected to a number of road sections. The total number of vehicles in all those sections may not exceed the capacity. It is clear that this notation is specific to the vehicle traffic domain and that it allows for the description of a plethora of traffic configurations. Note how we have chosen to make Traffic an un-timed formalism to allow for high abstraction level, conservative analysis.

3 META-MODELLING

When modelling complex physical or logical systems it is desirable to use the *most appropriate formalism* to optimally describe their different aspects or components. In this case, one has to solve the problem of building and interconnecting a plethora of different tools, especially built for each formalism. *Meta-Modelling* alleviates these problems.

Meta-modelling (Engstrom and Krueger 2000, Karsai, Nordstrom, Ledeczi, and Sztipanovits 2000) is the explicit modelling of a class of models, *i.e.,* of a modelling language. A meta-model $M_{\mathcal{L}}$ of a modelling language $\mathcal{L}$ is a model (with textual or visual syntax) in its own right which *specifies* precisely which models m are elements of $\mathcal{L}$.

Modelling environments based on meta-modelling will either *check*, by means of a meta-model $M_{\mathcal{L}}$ whether a given model m is in $\mathcal{L}$, or they will *constrain* the modeller during the incremental model construction process such that only elements of $\mathcal{L}$ can be constructed. Note how the latter approach, though possibly more efficient, due to its incremental nature –of construction and consequently of checking– may render certain valid models in $\mathcal{L}$ unreachable through incremental construction.

The advantages of meta-modelling are numerous. Firstly, an *explicit* model of a modelling language can serve as *documentation* and as *specification*. Such a specification can be the basis for the *analysis* of properties of models in the language. From the meta-model, a modelling environment may be *automatically* generated. The flexibility of the approach is tremendous: new languages can be designed by simply *modifying* parts of a meta-model. As this modification is explicitly applied to models, the relationship between different variants of a modelling language is apparent. Above all, with an appropriate meta-modelling tool, modifying a meta-model and subsequently generating a possibly visual

modelling tool is orders of magnitude *faster* than developing such a tool by hand. The tool synthesis is *repeatable* and *less error-prone* than hand-crafting.

As meta-models are models in their own right, they must be elements of a modelling language (or put differently, expressed in a particular formalism). This modelling language can be specified in a so-called *meta-meta-model*. Note how the "meta" qualifier is obviously *relative* to the original model.

Though an arbitrary number of meta-levels are possible in principle; in practice, some modelling languages/formalisms such as Entity-Relationship Diagrams (ERD) and UML Class Diagrams are expressive enough to be expressed in themselves. That is, the meta-model of such a language $\mathcal{L}$ is a model in language $\mathcal{L}$. From the implementation point of view, this allows one to *bootstrap* a meta-modelling environment. This is often referred to as *meta-circular* interpretation.

3.1 A Traffic Meta-Model

As an example, we briefly describe how to build a meta-model for the Traffic formalism with AToM^3. In AToM^3, the default meta-formalism is Entity-Relationship Diagrams. To define the meta-model, one has to provide an *abstract* syntax (denoting entities of the formalism, their attributes, relationships and constraints) as well as a *concrete* graphical syntax (how the entities and relationships should be rendered in a visual interactive tool, as well as the possible graphical constraints). The Traffic meta-model shown in Figure 2 prescribes which entities are allowed in the formalism with their attributes and how they may be connected. Not shown is the definition of the graphical appearance (seen in Figure 1) of these entities, global attributes (such as the model name, and author) nor are constraints.

Once the formalism is modelled, AToM^3 generates Python (`www.python.org`) code which can be loaded by the AToM^3 kernel. Once this compiled Traffic meta-model is loaded, the tool only accepts valid Traffic models. Using AToM^3, the effort to produce a customized visual modelling tool can be reduced to just a few hours for typical formalisms.

4 MODEL TRANSFORMATION

The *transformation* of models is a crucial element in all model-based endeavours. As models, meta-models, and meta-meta-models are all in essence attributed, typed graphs, we can transform them by means of graph rewriting. The rewriting is specified in the form of Graph Grammar (Ehrig, Engels, Kreowski, and Rozenberg 1999) models. These are a generalization, for graphs, of Chomsky grammars. They are composed of rules. Each rule consists of Left Hand Side (LHS) and Right Hand Side (RHS) graphs. Rules are evaluated against an input graph, called the host graph. If a matching is found between the LHS of a rule and a sub-graph of the host graph, then the rule can be applied. When a rule is applied, the matching sub-graph of the host graph is replaced by the RHS of the rule. Rules can have applicability conditions, as well as actions to be performed when the rule is applied. Some graph rewriting systems have control mechanisms to determine the order in which rules are checked. After a rule matching and subsequent application, the graph rewriting system starts the search again. The graph grammar execution ends when no more matching rules are found.

On the one hand, graph grammars have some advantages over specifying the transformation to be done on the graph using a traditional programming language. Graph grammars are a natural, formal, visual, declarative and high-level representation of the transformation. The theoretical foundations of graph rewriting systems may assist in proving correctness and convergence properties of the transformation tool. On the other hand, the use of graph grammars is constrained by efficiency. In the most general case, subgraph isomorphism testing is NP-complete. However, the use of small subgraphs on the LHS of graph grammar rules, as well as using node and edge types and attributes can greatly reduce the search space. This is the case with the majority of formalisms we are interested in. It is noted that a possible performance penalty is a small price to pay for explicit, reusable, easy to maintain models of transformation. In cases where performance is a real bottleneck, graph grammars can still be used as an *executable specification* to be used as the starting point for an efficient *manual implementation*.

Graph grammars for formalism transformation are particularly useful for the modelling and analysis of complex systems. Models of such systems consist of many components or views, possibly at different levels of abstraction. Due to the diversity of these models, we use different formalisms to describe each one of them. To analyse the entire system, one cannot look at properties of components or views in isolation, but the system should be understood as a whole. Therefore, in Computer Automated Multi-Paradigm Modelling we have proposed to *transform* each component or view into a single common formalism for subsequent analysis and simulation (Vangheluwe 2000).

4.1 Traffic Semantics

In addition to the syntax of the Traffic formalism modelled in section 3.1, we still need to model its *semantics*. One option would be to describe the operational semantics of the formalism (*i.e.,* how vehicles move through the model) by constructing a simulator by hand or by building a Graph Grammar model of the dynamics. We have chosen to map Traffic models onto Petri Net (Murata 1989) models instead. Not only does this *define* the meaning of the the Traffic formal-

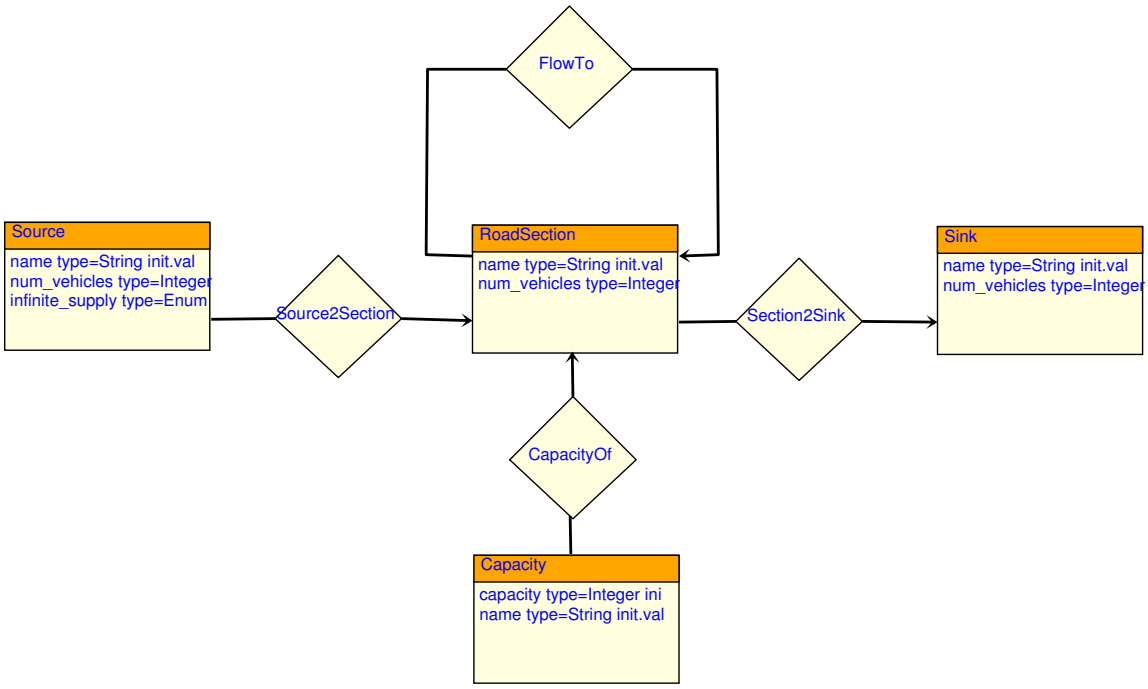

Figure 2: Entity Relationship Diagram Meta-Model of Traffic

ism, but it allows for the use of existing Petri Net analysis, optimization and simulation techniques and tools.

Figures 3 and 4 depict our Graph Grammar model of the mapping. The model starts with an initial action followed by nine rules. Each rule has a LHS and a RHS as well as an optional pre-condition and post-action. Nodes and connections in LHSs and RHSs are identified by means of labels (numbers). If a number appears on both the LHS and the RHS of a rule, the node or connection is retained when the rule is applied. If the number appears only on the LHS, the node or connection is deleted when the rule is applied. Finally, if the number appears only on the RHS, the node or connection is created when the rule is applied. Node and connection attributes in LHSs must be provided with attribute values which will be compared with the node and connection attributes of the host graph during the matching process. These attributes can be set to $\langle ANY \rangle$, or may have specific values. In the RHS, we can specify changed attribute values for those nodes which also appear in the LHS. In AToM^3, we can either copy the value of the attributes of the LHS (this appears as $\langle COPIED \rangle$ in the figure), specify a new value, or associate arbitrary Python code to compute the attribute value, possibly based on other nodes' attributes. Obviously, we must specify the attribute values of the newly created nodes or connections.

In the inital action of our model, all `RoadSection` nodes are marked as unvisited (to avoid infinite application of rule 1). Rule 1 transforms Traffic `RoadSection` nodes into Petri Net `Places`, with a link to the original `RoadSection` node. Rule 2 transforms Traffic `FlowTo` connections between `RoadSection` nodes into Petri Net `Transitions` with appropriate Petri Net arcs. Rule 3 creates a Petri Net `Place` for each Traffic `Capacity` node, copying the `capacity` and `name` attributes and keeping a link between both nodes. Rule 4 creates a direct link between a Petri Net `Capacity` node and a Traffic `RoadSection` node it pertains to. The no longer needed link between the Traffic `Capacity` node and the Traffic `RoadSection` node is removed. Rule 5 removes the no longer needed Traffic `Capacity` nodes. Rules 6, 7 and 8 implement Petri Net capacity constraints as described by Murata (1989). Rules 6 and 7 add appropriate input and output arcs. Rule 8 adjusts the number of tokens in Petri Net `Capacity` nodes to reflect the initial number of vehicles in capacity constrained `RoadSection` nodes. Finally, rule 9 removes the no longer needed Traffic `RoadSection` nodes as well as dangling edges.

Note how for simplicity, the rules pertaining to vehicle `Sources` and `Sinks` have not been included.

Note also how a GenericGraph formalism are used as a "helper" during graph transformations, in particular from one formalism to another. GenericGraph edges are used to keep links between Traffic and Petri Net nodes. This is cleaner than adding "helper" relationships to either of those two formalisms or than using some of the relationships of those two formalisms out-of-context (this "hack" is possible as some checking is disabled when specifying Graph Grammar rules).

Figure 5 illustrates the application of the rules. It starts

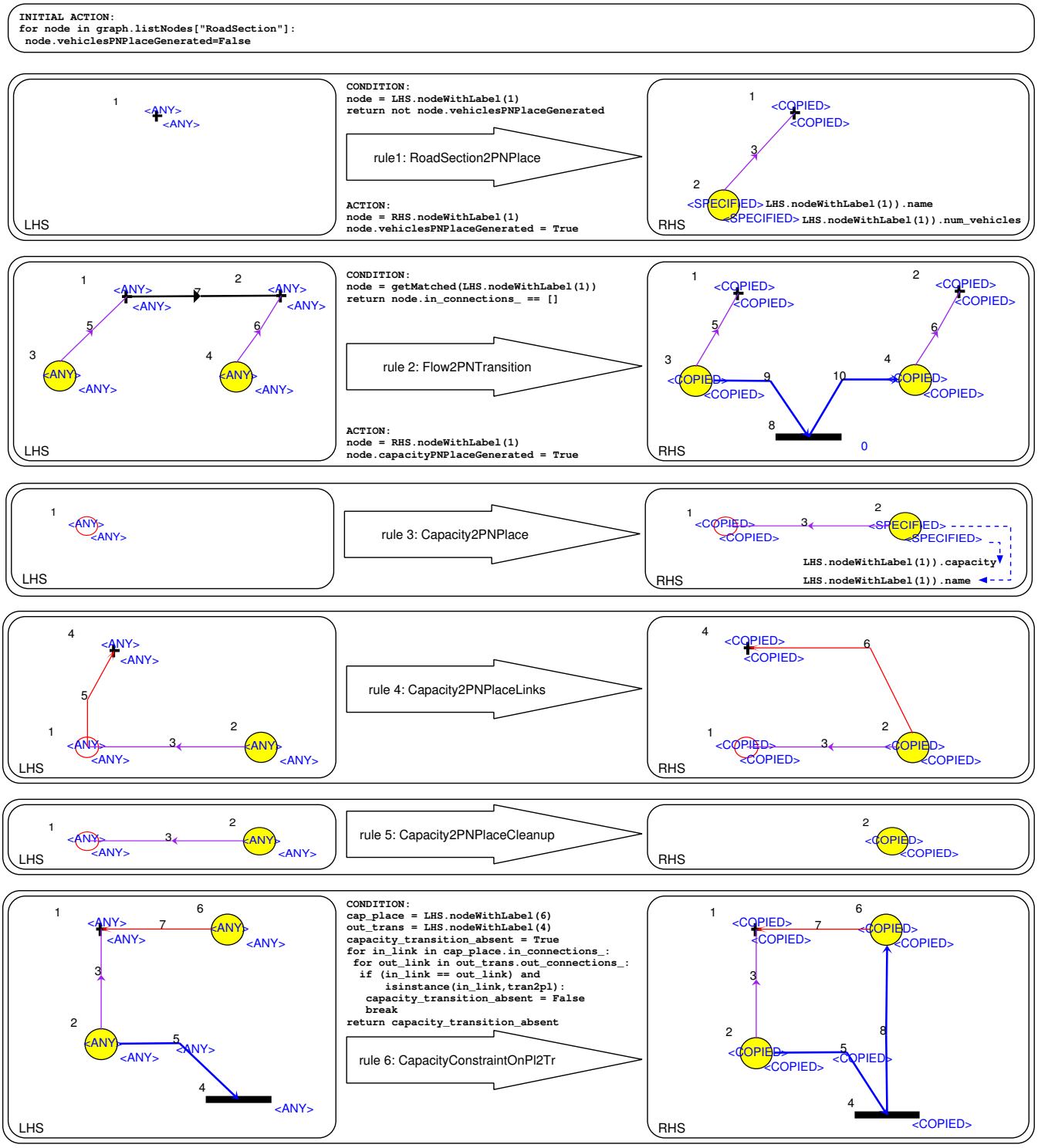

Figure 3: Traffic to Petri Net Transformation Rules (part 1)

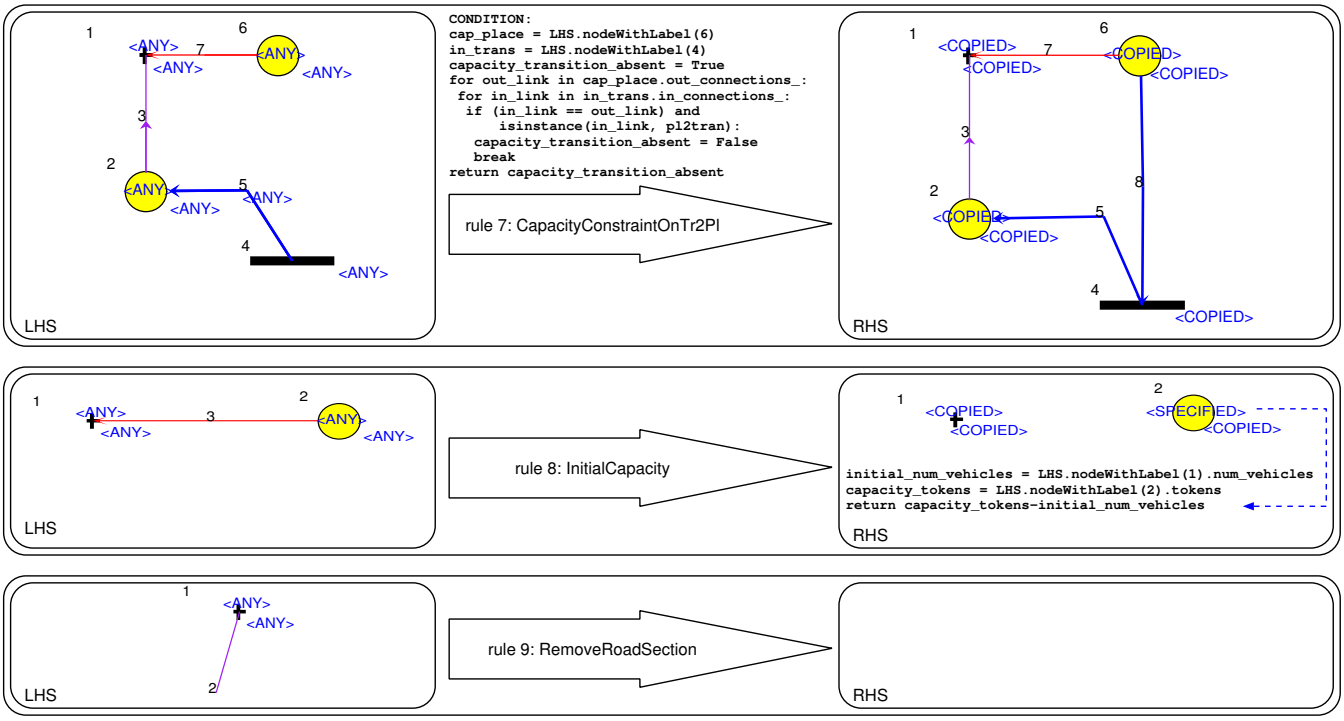

Figure 4: Traffic to Petri Net Transformation Rules (Part 2)

from an extremely simple Traffic model with two connected road segments. The first segment is initially populated by two vehicles, the second by one. In total, no more than four vehicles may be present in both segments. The transformation ends with a Petri Net representing the behaviour of the Traffic model.

A Closed Traffic System

For a less trivial example, Figure 6 adds more feedback to the Traffic model in Figure 1, making the system autonomous. Applying our transformation yields the Petri Net model depicted in Figure 7.

This model may now be used to analyze and simulate the system. For analysis, we generate the Coverabilty Graph (a Reachability Graph dealing with possibly infinite markings) shown in Figure 8. The Coverability Graph allows for *liveness* analysis of the Traffic system. In particular, as there are no nodes with outgoing edges in this graph, we conclude that *deadlock* cannot occur.

Murata (1989) defines a Petri net with initial state x_0 *conservative* with respect to a vector of integer weights $\gamma = [\gamma_1, \gamma_2, \ldots, \gamma_n]$ if

$$\sum_{i=1}^{n} \gamma_i x(p_i) = constant$$

for all states in all possible sample paths from x_0, with $x(p)$ the marking (number of tokens) of place p.

We traverse the Coverability Tree and generate a matrix representation of the above conservation equations. After Gauss elimination, we produce an Integer Linear Programming specification which we solve with the `lp_solve` code (`http://www.geocities.com/lpsolve`). This leads to the following set of conservation equations:

```
1.0 x(turn1_CAP) + 1.0 x(turn1)                    = 1.0
1.0 x(turn2_CAP) + 1.0 x(turn2)                    = 1.0
1.0 x(top_CAP) + 1.0 x(to_N_or_W)                  = 1.0
1.0 x(bot_CAP) + 1.0 x(bot_W2E) + 1.0 x(bot_N2S)   = 1.0
1.0 x(cars) + 1.0 x(bot_W2E) + 1.0 x(turn1) +
1.0 x(to_N_or_W) + 1.0 x(turn2) + 1.0 x(bot_N2S)   = 2.0
```

These equations can easily be verified on the original Traffic model. The first three equations correspond to capacity constraints on `turn1`, `turn2`, and `to_N_or_W` respectively. The fourth equation corresponds to the capacity constraint on the bottom intersection. The last equation expresses that the total number of vehicles in the system is conserved and is 2. The above is a "basic" set of conservation equations: any linear combination of the above is also valid. As we solved an Integer Linear Programming problem, there is no guarantee that this solution set is complete nor minimal (though it is in this case).

4.2 Alternate Semantics

When one of the exit routes of the `to_N_or_W` road section is full, the other one will be used in the current semantics.

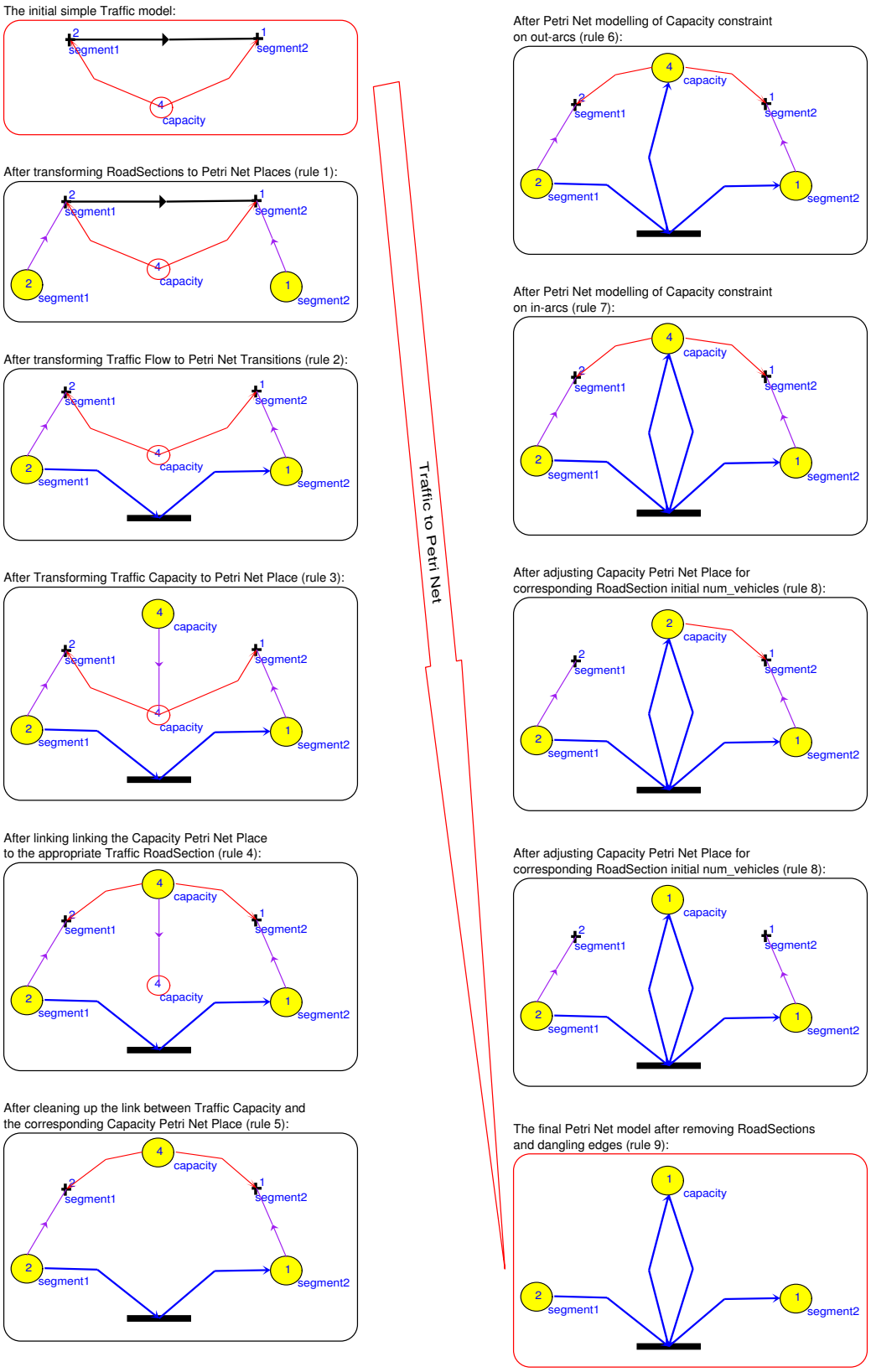

Figure 5: Steps in the Transformation from Traffic to Petri Net

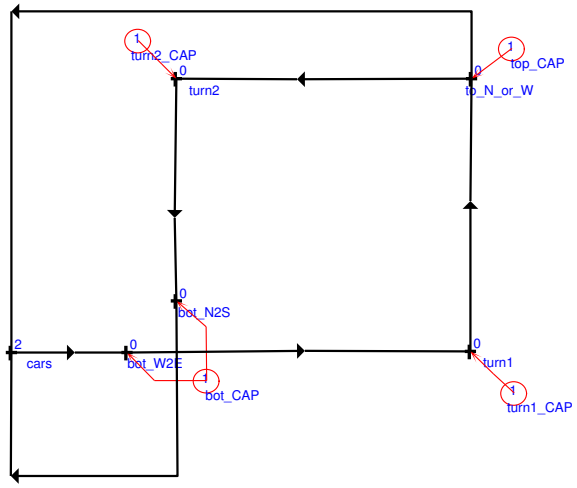

Figure 6: A Closed Traffic System

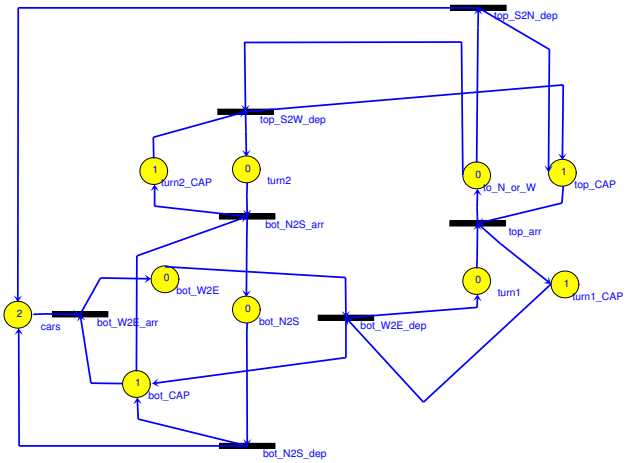

Figure 7: The Generated Petri Net Model

Modifying the Graph Grammar model of Traffic semantics allows us to generate Figure 9 instead in which a vehicle decides on which of the exits of a road section to take *before* entering it.

5 CONCLUSIONS

In this paper, we have presented the Computer Automated Multi-Paradigm (CAMPaM) framework for model-based development founded on a combination of meta-modelling and graph transformation. By means of meta-modelling we graphically specify the syntax of models we want to deal with. By means of graph transformation we graphically define the kinds of manipulations that we can apply to these models. These manipulations typically include defining operational or denotational semantics, transformations into other formalisms, code generation and optimization. Using graph rewriting has the advantage that transformations are explicitely modelled. We have implemented these concepts in our AToM^3 CAMPaM tool following the *everything is a model* philosophy.

To illustrate our approach, we have modelled the Traffic formalism dedicated to vehicle traffic network modelling. The syntax of Traffic was meta-modelled and the semantics was given by mapping Traffic models onto Petri Net models. The mapping was modelled using a Graph Grammar. The resulting Petri Net model was transformed to a Coverability Graph which was the basis for further analysis. In particular, an Integer Linear Programming problem was generated from the Coverability Graph to determine conserved quantities.

We plan to extend our work by modelling a TimedTraffic formalism whose semantics will be given by mapping it onto Timed Transition Petri Nets, DEVS, and GPSS. The different target formalisms each have their own analysis and simulation tools. We will also move across abstraction levels by mapping TimedTraffic onto Traffic for the conservative, untimed analysis described here.

ACKNOWLEDGMENTS

Juan de Lara's work has been partially sponsored by the Spanish Interdepartmental Commission of Science and Technology (CICYT), project number 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. The authors wish to thank Ms. Sokhom Pheng for her work on the Petri Net conservation analysis during her "Modelling and Simulation Based Design" project at McGill University.

REFERENCES

de Lara, J., and H. Vangheluwe. 2002, April. AToM^3: A tool for multi-formalism and meta-modelling. In *European Joint Conference on Theory And Practice of Software (ETAPS), Fundamental Approaches to Software Engineering (FASE)*, LNCS 2306, 174 – 188: Springer-Verlag. Grenoble, France.

de Lara, J., and H. Vangheluwe. 2004, June - August. Defining visual notations and their manipulation through meta-modelling and graph transformation. *Journal of Visual Languages and Computing* 15 (3 - 4): 309–330. Special Issue on Domain-Specific Modeling with Visual Languages.

de Lara Jaramillo, J., H. Vangheluwe, and M. Alfonseca Moreno. 2003, February. Using meta-modelling and graph grammars to create modelling environments. In *Electronic Notes in Theoretical Computer Science*, ed. P. Bottoni and M. Minas, Volume 72: Elsevier. 15 pages. `www.elsevier.nl/locate/entcs/volume72.html`.

Figure 8: The Generated Coverability Graph

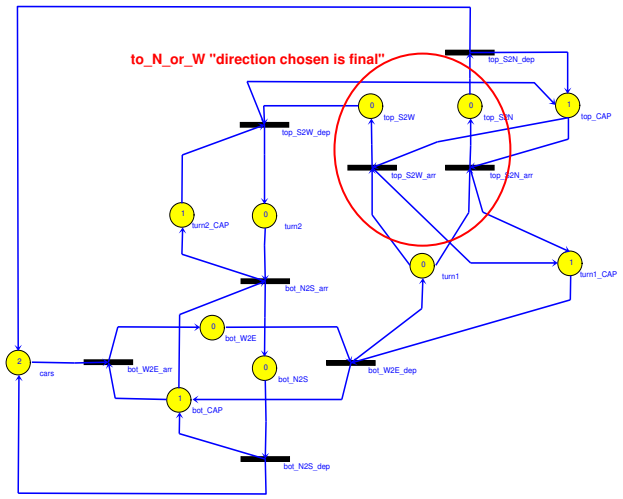

Figure 9: "Direction Chosen Is Final" Semantics

Ehrig, H., G. Engels, H.-J. Kreowski, and G. Rozenberg. 1999. *Handbook of graph grammars and computing by graph transformation. vol. 2: Applications, languages, and tools*. World Scientific.

Engstrom, E., and J. Krueger. 2000, September. A Meta-Modeler's Job is Never Done: Building and Evolving Domain-Specific Tools With DOME. In *Proceedings of the IEEE International Symposium on Computer Aided Control System Design*, 83–88. Anchorage, Alaska.

Karsai, G., G. Nordstrom, A. Ledeczi, and J. Sztipanovits. 2000, September. Specifying Graphical Modeling Systems Using Constraint-based Metamodels. In *Proceedings of the IEEE International Symposium on Computer Aided Control System Design*, 89–94. Anchorage, Alaska.

Kelly, S., and J.-P. Tolvanen. 2000. Visual domain-specific modeling: Benefits and experiences of using metacase tools. In *Proceedings of the International workshop on Model Engineering, ECOOP 2000*, ed. J. Bezivin and J. Ernst, 9 pp. `http://www.metamodel.com/IWME00/`.

Mosterman, P. J., and H. Vangheluwe. 2002. Computer Automated Multi-Paradigm Modeling. *ACM Transactions on Modeling and Computer Simulation* 12 (4): 1–7. Special Issue Guest Editorial.

Murata, T. 1989, April. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE* 77 (4): 541–580.

Papacostas, C., and P. Prevedouros. 1992, October. *Transportation engineering and planning*. Second ed. Prentice Hall.

Vangheluwe, H. 2000, September. DEVS as a common denominator for multi-formalism hybrid systems modelling. In *IEEE International Symposium on Computer-Aided Control System Design*, ed. A. Varga, 129–134: IEEE Computer Society Press. Anchorage, Alaska.

AUTHOR BIOGRAPHIES

HANS VANGHELUWE is an Assistant Professor in the School of Computer Science at McGill University, Montréal, Canada where he teaches Modelling and Simulation, as well as Software Design. He heads the Modelling, Simulation and Design Lab (MSDL). He has been the Principal Investigator of a number of research projects on the development of a multi-formalism theory for Modelling and Simulation. Some of this work has led to the WEST++ tool, which was commercialised for use in the design and optimization of Waste Water Treatment Plants. He was the co-founder and coordinator of the European Union's ESPRIT Basic Research Working Group 8467 "Simulation in Europe", and a founding member of the Modelica Design Team. His current research is focused on the development of the AToM^3 tool for Computer Aided Multi-Paradigm Modelling (CAMPaM), on domain-specific visual modelling and on modular formalism for discrete-event simulation and code synthesis. His e-mail address is `<hv@cs.mcgill.ca>`, and his web page is `<www.cs.mcgill.ca/~hv>`.

JUAN DE LARA is an Associate Professor at the Universidad Autonóma (UAM) de Madrid in Spain, where he teaches software engineering, automata theory as well as modelling and simulation. His research interests include Web Based Simulation, Meta-Modelling, Grah Transformation, Distance Learning, and Social Agents. He recieved his PhD in June 2000 at UAM in Computer Science. During 2001, as a post-doctoral researcher in the MSDL, he created the AToM^3 prototype. Later, he also spent several periods at TU Berlin working on graph transformation. His e-mail address is `<Juan.Lara@ii.uam.es>`, and his web page is `<www.ii.uam.es/~jlara>`.