Domain-Specific Visual Modelling in AToM^3

Hans Vangheluwe
School of Computer Science
McGill University
Montréal, Québec, CANADA

Juan de Lara
ETS Informática
Universidad Autonóma de Madrid
Madrid, SPAIN

Abstract

Domain- and formalism-specific modelling have the potential to greatly improve productivity. 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. The Traffic formalism, a new visual notation specific to the vehicle traffic domain is introduced. Using AToM^3, A Tool for Multi-formalism and Meta-Modelling, a Traffic-specific modelling environment is built. This demonstrates how meta-modelling and graph rewriting allow one to rapidly develop complete visual domain-specific modelling environments. The underlying philosophy of the work is to "model everything". Finally, a number of directions for future work on domain-specific visual modelling environments are suggested.

1 Introduction

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 [11, 1]. Such languages have the potential to greatly increase system quality and reduce development costs, as they are notations tailored to specific needs.

Some (families of) 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 use graph grammars [13], a formal, graphical and high-level notation to specify the model transformations. As the Graph Grammar formalism can be meta-modelled in its own right, a visual environment for manipulating transformation models can also be automatically generated.

We have implemented these meta-modelling and graph transformation concepts in AToM^3, *A Tool for Multi-formalism and Meta-Modelling*. AToM^3's design has been described in [5, 3, 6]. The power of AToM^3 has been demonstrated by modelling the DEVS formalism, Petri Nets and Statecharts, GPSS, Causal Block Diagrams, and flow diagrams [4]. 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.

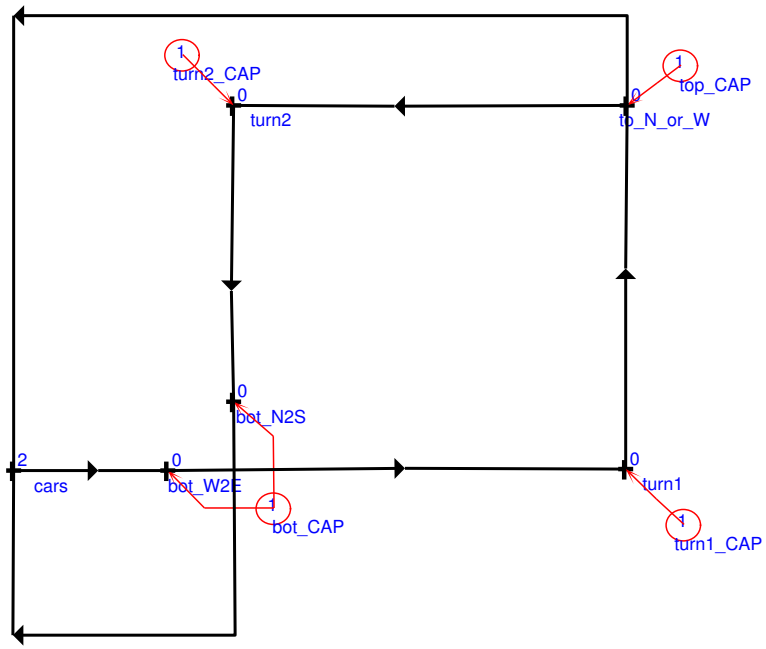

Figure 1: A Traffic Model

2 The Traffic Formalism

To illustrate domain-specific modelling, we introduce the Traffic formalism, a new visual notation specific to the vehicle traffic domain. 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 closed traffic system in which two vehicles are initially present (in the location `cars`), go straight across an intersection (when no other vehicles are present), turn left on a short road section which can only hold one vehicle, and either go back to their initial location or turn left. Turning left brings them across another short road section which can only hold one vehicle, back to the first intersection. After succesfully crossing this intersection, they again go back to their initial location.

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 host of different tools, especially built for each formalism. *Meta-modelling* [9, 10] alleviates these problems.

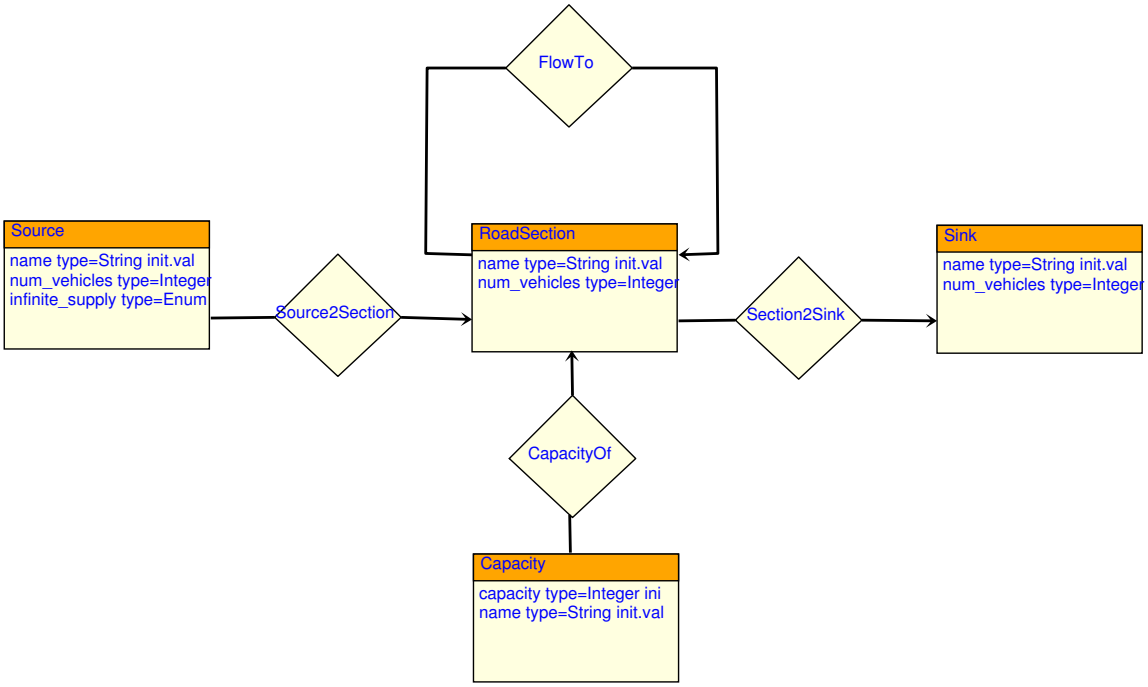

Figure 2: Entity Relationship Diagram Meta-Model of Traffic

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 in many cases, models, meta-models, and meta-meta-models are all attributed, typed graphs. These graphs can be transformed by means of graph rewriting. The rewriting is specified in the form of models in the Graph Grammar formalism. Graph grammars are a generalization of Chomsky grammars, for graphs [13] [7]. Graph grammars are composed of production rules, each having graphs in their left and right hand sides (LHS and RHS).

In addition to the syntax of the Traffic formalism modelled above, 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 [12] models instead. Not only does this *define* the meaning of the the Traffic formalism, but it allows for the use of existing Petri Net analysis, optimization and simulation techniques and tools.

Figure 3 depicts our Graph Grammar model of the mapping. The model starts with an initial

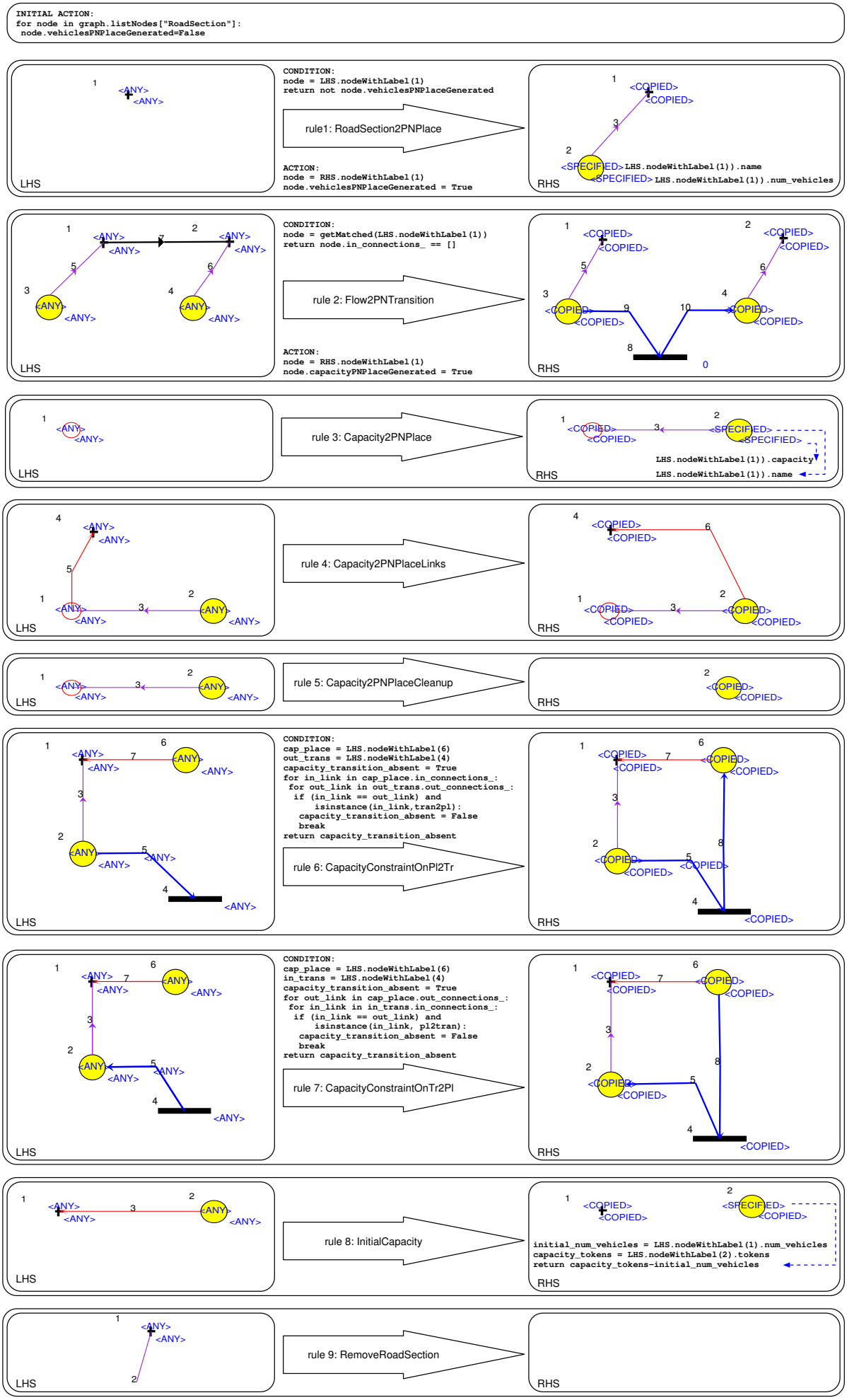

Figure 3: Traffic to Petri Net Transformation Rules

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 in [12]. 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 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.

Applying our transformation to the Traffic system depiced in Figure 1 yields the Petri Net model depicted in Figure 4.

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 5. 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.

5 Suggestions for Future Research

It is important to model not only the visual syntax of formalisms, but also the way in which a user will interact with models in those formalisms. This modelling of *reactive behaviour* of domain-specific graphical user interfaces needs to (i) be intuitive, (ii) support analysis, simulation, and above all code synthesis and (iii) be seemlessly integrated and consistent with the meta-model and with possible modelled model transformations. Some early experience with Statechart models of the reactive behaviour of AToM^3 has been promising. As shown in Figure 6 AToM^3's user interface is modelled, simulated and synthesized entirely using DCharts [8]. In our experience, the Statechart/DChart formalisms are not modular enough for our purposes and the link with declarative (layout as well as semantic) constraints is still unclear.

Transformations need to be possible to/from and between textual model descriptions. For this,

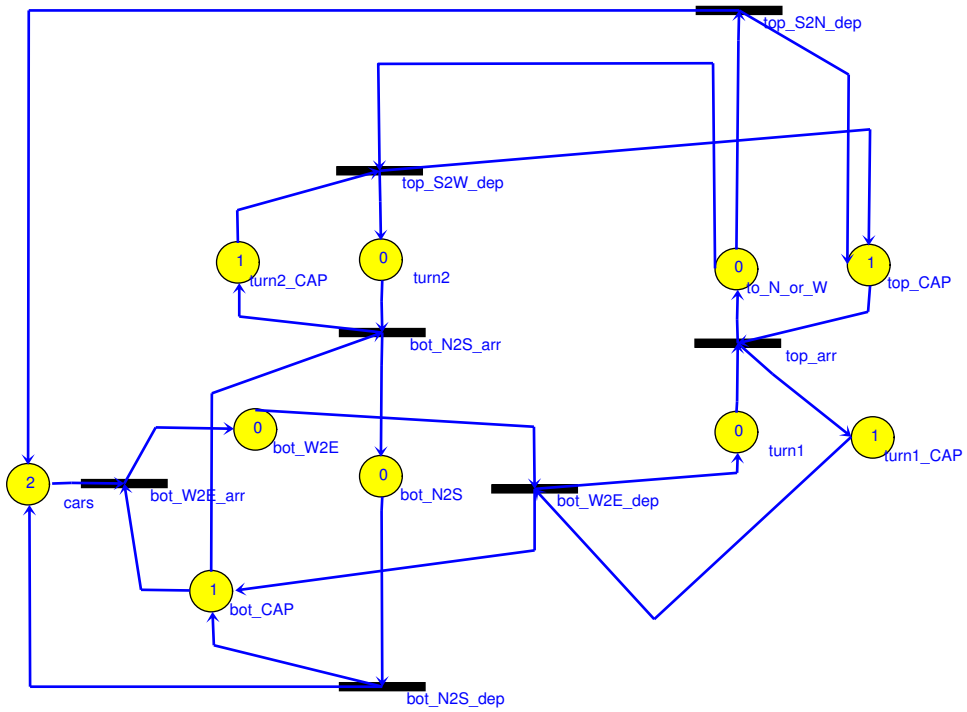

Figure 4: The Generated Petri Net Model

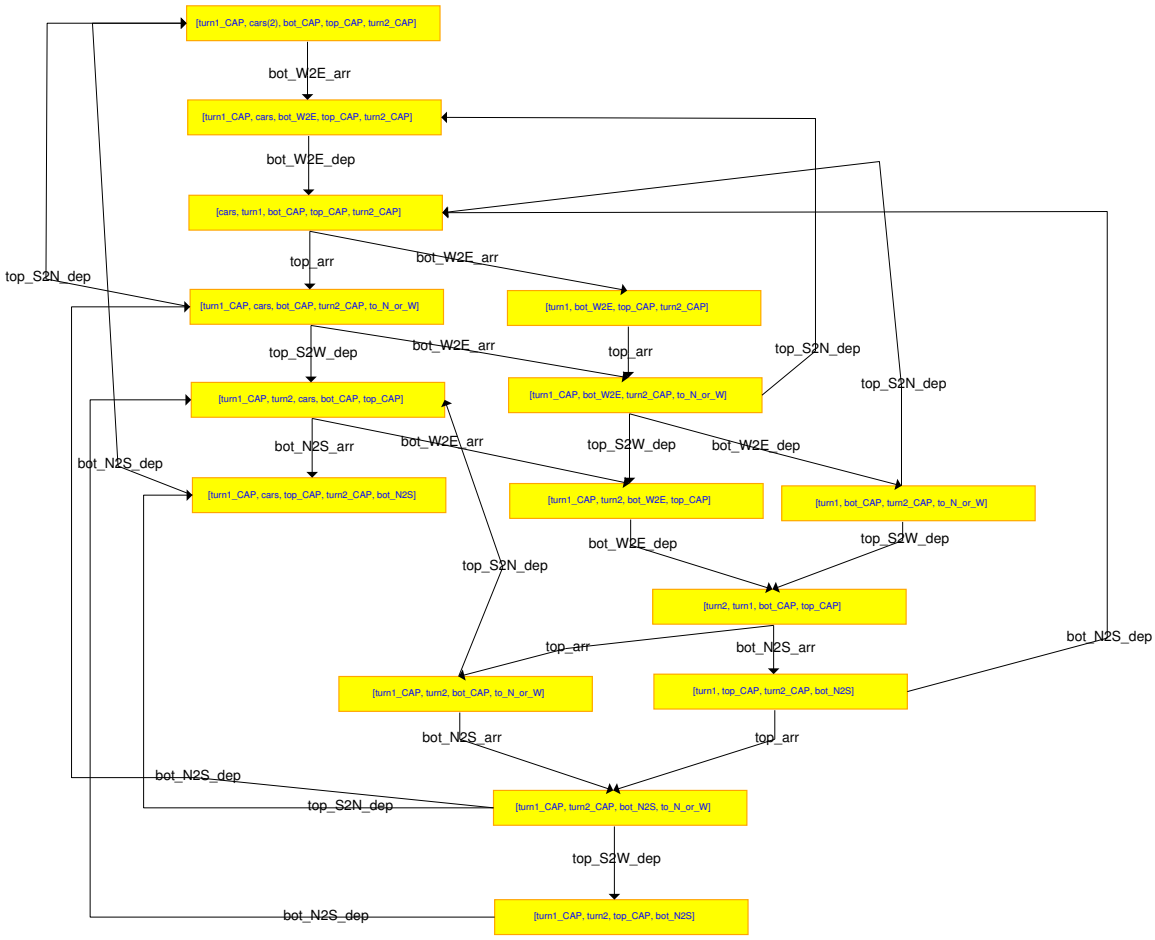

Figure 5: The Generated Coverability Graph

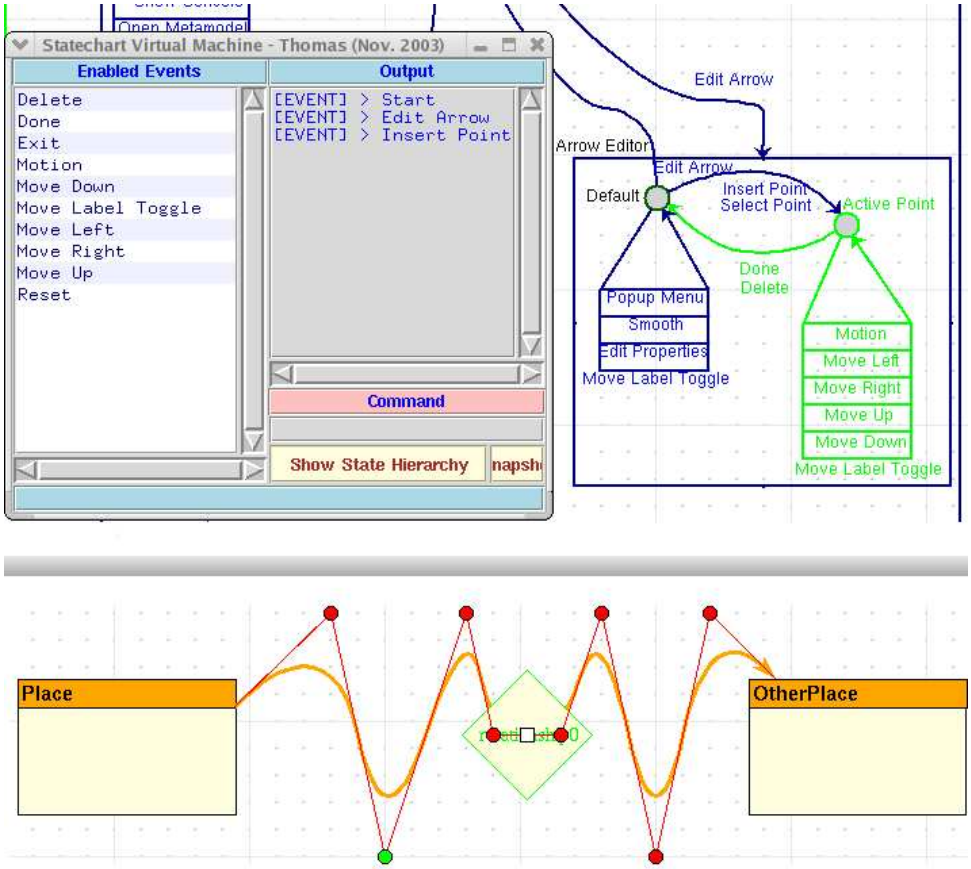

Figure 6: The GUI's reactive behaviour modelled and in action

graph grammars are not very suitable. Furthermore, if graph grammars are used, it may sometimes be more appropriate to describe patterns in a textual way, as in GReAT [2]. Apart from this fact, a textual syntax is most certainly needed for a target-code-independent *action language*. Due to our involvement in the design of the Modelica language (`www.modelica.org`), we are currently experimenting with Modelica to make it suitable for multi-formalism and meta-modelling.

Models should be used for (i) analysis and verification, (ii) simulation, and (iii) synthesis (by appropriate transformations). Some of these transformations could become "standard" such as the mapping onto Petri nets with subsequent analyses. Rather counter-intuitively, it is expected that increasing the number of (re-useable) transformations will increase the potential for optimizations, thus increasing the quality of the final product.

References

[1] Special issue on domain-specific modeling with visual languages. *Journal of Visual Languages and Computing*, 15(3-4):207–330, June - August 2004. Edited by J. Gray, M. Rossi, and J.-P. Tolvanen.

[2] A. Agrawal, G. Karsai, and F. Shi. Graph transformations on domain-specific models. Technical Report ISIS-03-403, Vanderbilt University, Institute for Software Integrated Systems, November 2003.

[3] Juan de Lara and Hans Vangheluwe. 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)*, Lecture Notes in Computer Science 2306, pages 174 – 188. Springer-Verlag, April 2002. Grenoble, France.

[4] Juan de Lara and Hans Vangheluwe. Using AToM^3 as a Meta-CASE tool. In *4th International Conference on Enterprise Information Systems (ICEIS)*, pages 642 – 649, April 2002. Ciudad Real, Spain.

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

[6] Juan de Lara Jaramillo, Hans Vangheluwe, and Manuel Alfonseca Moreno. Using meta-modelling and graph grammars to create modelling environments. In Paolo Bottoni and Mark Minas, editors, *Electronic Notes in Theoretical Computer Science*, volume 72. Elsevier, February 2003. 15 pages. `www.elsevier.nl/locate/entcs/volume72.html`.

[7] H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg. *Handbook of Graph Grammars and Computing by Graph Transformation. Vol. 2: Applications, Languages, and Tools*. World Scientific, 1999.

[8] Thomas Huining Feng. Dcharts, a formalism for modeling and simulation based design of reactive software systems. M.Sc. dissertation, School of Computer Science, McGill University, February 2004.

[9] Rony G. Flatscher. Metamodeling in EIA/CDIF meta-metamodel and metamodels. *ACM Transactions on Modeling and Computer Simulation*, 12(4), 2002.

[10] Gabor Karsai, Greg Nordstrom, Akos Ledeczi, and Janos Sztipanovits. Specifying Graphical Modeling Systems Using Constraint-based Metamodels. In *Proceedings of the IEEE International Symposium on Computer Aided Control System Design*, pages 89–94, Anchorage, Alaska, September 2000.

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

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

[13] G. Rozenberg. *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1*. World Scientific, 1997.