Meta-Modelling, Graph Transformation and Model Checking for the Analysis of Hybrid Systems

Juan de Lara[1], Esther Guerra[1] and Hans Vangheluwe[2]

[1] Escuela Politécnica Superior
Ingeniería Informática
Universidad Autónoma de Madrid
(Juan.Lara, Esther.Guerra_Sanchez)@ii.uam.es
[2] School of Computer Science
McGill University, Montréal
Québec, Canada
hv@cs.mcgill.ca

Abstract. This paper presents the role of meta-modelling and graph transformation in our approach for the modelling, analysis and simulation of complex systems. These are made of components that should be described using different formalisms. For the analysis (or simulation) of the system as a whole, each component is transformed into a single common formalism having an appropriate solution method. In our approach we make meta-models of the formalisms and express transformations between them as graph transformation. These concepts have been automated in the AToM^3 tool and as an example, we show the analysis of a hybrid system composed of a temperature controlled liquid in a vessel. The liquid is initially described using differential equations whose behaviour can be abstracted and represented as a Statechart. The controller is modelled by means of a Statechart and the temperature as a Petri net. The Statechart models are translated into Petri nets and joined with the temperature model to form a single Petri net, for which its reachability graph is calculated and Model-Checking techniques are used to verify its properties.

Keywords: Graph Rewriting, Meta-Modelling, Multi-Paradigm, Hybrid Systems, Model-Checking.

1 Introduction

Complex systems are characterized by interconnected components of very different nature. Some of these components may have continuous behaviour while the behaviour of other components may be discrete. Systems with both classes of components are called hybrid systems. There are several approaches to deal with the modelling, analysis and simulation of complex systems. Some approaches try to use a formalism general enough (a "*super-formalism*") to express the behaviour of all the components of the system. In general this is neither possible nor adequate. Other approaches let the user model each component of the system using the most appropriate formalism. While in *co-simulation* each component is simulated with a formalism-specific simulator; in *multi-formalism*

modelling a single formalism is identified into which each component is symbolically transformed [10]. In *co-simulation* the simulator interaction due to component coupling is resolved at the trajectory (simulation data) level. With this approach it is no longer possible to answer questions in a symbolic way about the behaviour of the whole system.

In *multi-formalism modelling* however, we can verify properties of the whole system if we choose a formalism with appropriate analysis methods for the transformation. The Formalism Transformation Graph (FTG) [10] can help in identifying a common, appropriate formalism to transform each component. The FTG depicts a part of the "formalism space", in which formalisms are shown as nodes in the graph. The arrows between them denote a homomorphic relationship "can be mapped onto", using symbolic transformations between formalisms. Other arrows (vertical) denote the existence of a simulator for the formalism.

Multi-Paradigm Modelling [10] combines multi-formalism, meta-modelling, and multiple levels of abstraction for the modelling, analysis and simulation of complex systems. Meta-modelling is used to describe different formalisms of the FTG and generate tools for them. In our work, we propose to model both transformation and simulation arrows of the FTG as graph transformation, as meta-models can be represented as attributed, typed graphs. Other model manipulations, for optimisation and code generation can be expressed with graph transformation as well. We have implemented these concepts in the Multi-Paradigm tool AToM^3 [5], which is used in the following section to model and analyse a simple hybrid system.

2 Example: A Temperature-Controlled Liquid in a Vessel

As a very simple example to help clarifying our Multi-Paradigm modelling approach, we show the modelling and analysis of a temperature-controlled liquid in a vessel. The system is composed of a continuous part which represents the liquid behaviour and a discrete part which controls the temperature. The continuous part is described by the following equations:

$$\frac{dT}{dt} = \frac{1}{H}\left[\frac{W}{c\rho A}\right] \tag{1}$$

$$is\_cold = (T < T_{cold}) \tag{2}$$

$$is\_hot = (T > T_{hot}) \tag{3}$$

Where W is the rate at which heat is added or removed, *A* is the cross-section surface of the vessel, *H* is the level of the liquid, *c* is its specific heat and ρ, its density. This system can be observed through two output sensors *is_cold* and *is_hot*, which are set in equations 2 and 3. These sensors allow us to discretize the state-space [10], in such a way that the system can be represented as a finite state automaton (shown to the left of Figure 1). The system's behaviour is governed by equation (1) in each automaton state, while transitions are fired by equations (2) and (3). Though at a much higher level of abstraction, the automaton alone (without the equation) captures the essence of the system's behaviour.

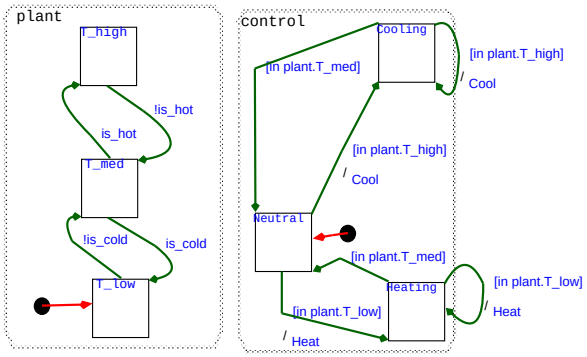

Fig. 1. Statechart Representing the Behaviour of a Temperature-Controlled Liquid.

The controller can also be represented as a state automaton, and is shown to the right of Figure 1. We have combined both automata in a single Statechart with two orthogonal components, named as *plant* and *control*, using the meta-model for Statecharts available in AToM^3. The sensors in equations 2 and 3 have been modelled using a Petri net (shown to the right of Figure 2). This Petri net has a place *Temp* to represent the temperature value. The *Heat* and *Cool* methods invoked by the controller are modelled by transitions which add and remove a token from *Temp*. The number of tokens in *Temp* remains bounded (between 0 and 30 in this example) using a well-known technique for capacity constraining. This technique consists on creating an additional place *Temp'*, in such a way that the number of tokens in both places is always equal to 30. Using the value of both places, we can model the events *is_cold*, *!is_cold*, *is_hot* and *!is_hot* (that can be considered as events produced by *sensors*). In this example we set the intervals [0-10] for cold and [21-30] for hot.

The Statechart can be automatically converted into Petri nets and joined with the component which models the temperature. The transformation from Statecharts into Petri nets was automated with AToM^3 using the graph transformation described in [6]. States and events are converted into places, current states in orthogonal components are represented as tokens. Once the model is expressed in Petri nets, we can apply the available analysis techniques for this formalism. In AToM^3 we have implemented transformations to obtain the reachability graph, code generation for a Petri net tool (PNS), simulation and simplification. The latter can be applied before calculating the reachability graph to help reducing the state-space. The Petri net model, after applying the simplification graph transformation is shown in Figure 2.

The reachability graph is shown in Figure 3 (labels depict tokens in places). Note how the reachability graph calculation is indeed another formalism transformation, from Petri nets into state automata. We have manually set priorities on the Petri net transitions to greatly reduce the state-space. We have set *pr(Tis_cold)>pr(Tis_not_hot)*, *pr(Tis_hot)>pr(Tis_not_cold)*, the probabilities of *Heat* and *Cool* bigger than any other and the probabilities of the *plant* transitions larger than the ones in the *controller*. Once

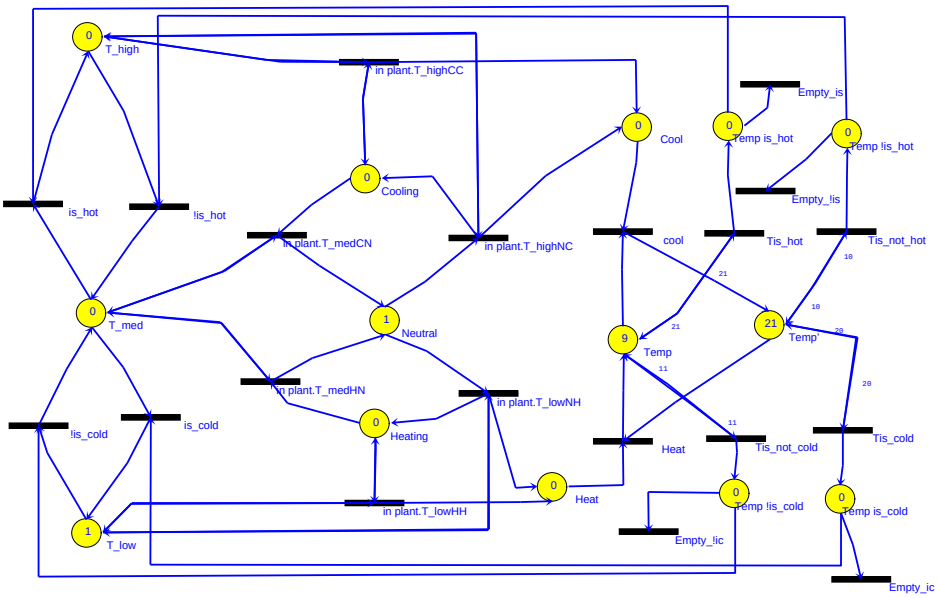

Fig. 2. Petri Net Generated from the Previous Model.

we have the reachability graph, we can formalize the properties we want to check using Computational Tree Logic (CTL) and check the graph using a simple explicit Model-Checking algorithm [4] and a meta-model for CTL we have implemented in AToM^3. The user can specify the CTL formula in two ways: graphically (drawing the graph of the CTL formula using the meta-model), or textually. In the latter case the formula is parsed and translated into an *Abstract Syntax Graph* (ASG). Note how if the user graphically specifies the formula, he is directly building the ASG. In the example, we may ask whether the plant reaches a certain temperature, or the *T_med* state (interval [11-20]), or whether the controller heats the liquid and eventually stops heating. With our initial conditions (liquid at 9 degrees), all these properties are true, given appropriate *fairness* constraints allowing the eventual firing of enabled *plant* and *control* transitions when the sensor transitions are also *enabled*.

3 Related Work

There are similar tools in the graph grammars community. For example, GENGED [2] has a similar approach: the tool allows defining visual notations and manipulate them by means of graph grammars (the graph rewriting engine is based on AGG [1]). Whereas the visual language parsing in the GENGED approach is based on graph grammars, in AToM^3 we rely on constraints checking to verify that the model is correct. The mapping from abstract to concrete syntax is very limited in AToM^3, as there is a "one to one" relationship between graphical icons and entities. On the contrary, this mapping is more

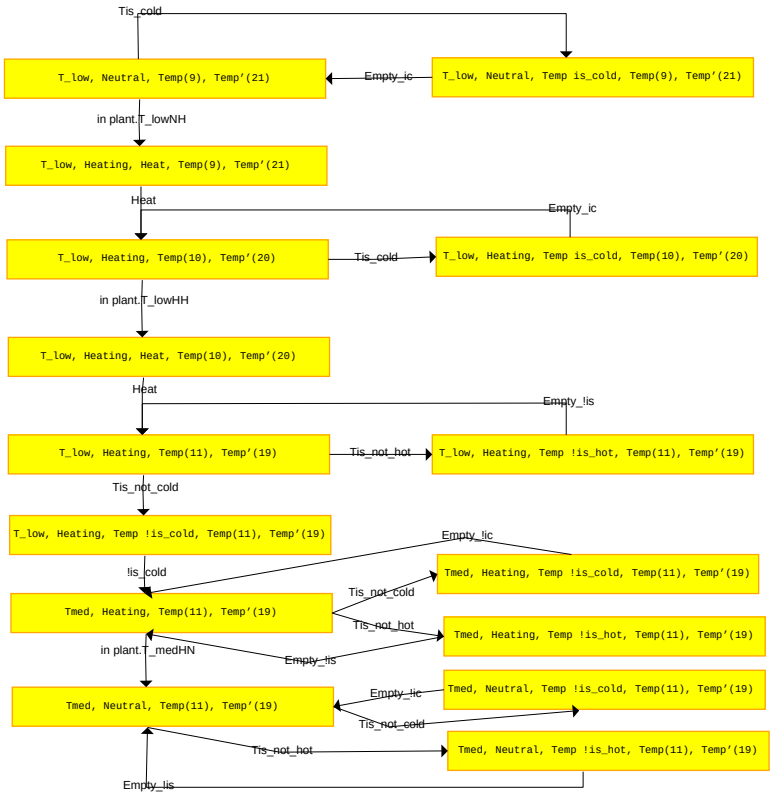

Fig. 3. The Reachability Graph for the Petri Net with Priorities.

flexible in GENGED, which in addition uses a constraint language for the concrete syntax layout. In AToM^3 this layout should be coded as Python expressions. This is lower-lever, but usually more efficient.

Although other tools based on graph grammars (such as DiaGen [8]) use the concept of bootstraping, in AToM^3 there is no structural difference between the generated editors (which could be used to generate other ones!), and the editor which generated them. In fact, one of the main differences of the approach taken in AToM^3 with other similar tools, is the concept that (almost) everything in AToM^3 has been defined by a model (under the rules of some formalism, including graph grammars) and thus the user can change it, obtaining more flexibility.

With respect to the transformation into Petri nets, the approach of [3] is similar, but they create two places for each method in the Statechart (one for calling the method and the other for the return). Other approach for specifying the transformation is the use of triple graph grammars [9].

4 Conclusions

In this paper we have presented our approach (*Multi-Paradigm*) for modelling and analysis of complex systems, based on meta-modelling and graph grammars. We make meta-models of the formalisms we want to work with, and use graph transformation to formally and visually define model manipulations. These include formalism transformation, simulation, optimisation and code generation. To analyse a complex system, we transform each component into a common formalism where the properties of interest can be checked. In the example, we have modelled a simple system, composed of a Statechart and a Petri net component into a single Petri net. Then we have calculated the reachability graph and verified some properties using Model-Checking. It must be noted however that we have made large simplifications for this problem. For example, for more complex systems, an explicit calculation of the reachability graph may not be possible, and other symbolic techniques should be used.

We are currently working in demonstrating properties of the transformations themselves, such as termination, confluence and behaviour preservation. Theory of graph transformation, such as critical pair analysis [7] can be useful for that purpose.

Acknowledgements: We would like to aknowledge the SEGRAVIS network and the Spanish Ministry of Science and Technology (project TIC2002-01948) for partially supporting this work, and the anonymous referees for their useful comments.

References

1. AGG home page: `http://tfs.cs.tu-berlin.de/agg/`
2. Bardohl, R., Ermel, C., Weinhold, I. 2002 *AGG and GenGED: Graph Transformation-Based Specification and Analysis Techniques for Visual Languages* In Proc. GraBaTs 2002, Electronic Notes in Theoretical Computer Science 72(2).
3. Baresi, L., Pezze, M.. *Improving UML with Petri nets*. Electronic Notes in Theoretical Computer Science 44 No. 4 (2001).
4. Clarke, E. M., Grumberg, O., Peled, D. A. 1999. *Model Checking*. MIT Press.
5. de Lara, J., Vangheluwe, H. 2002 *AToM^3: A Tool for Multi-Formalism Modelling and Meta-Modelling.* In ETAPS02/FASE. LNCS 2306, pp.: 174 - 188. See also the AToM^3 home page: `http://atom3.cs.mcgill.ca`.
6. de Lara, J., Vangheluwe, H. 2002 *Computer Aided Multi-Paradigm Modelling to process Petri Nets and Statecharts.* ICGT'2002. LNCS 2505. Pp.: 239-253.
7. Heckel, R., Küster, J. M., Taentzer, G. 2002. *Confluence of Typed Attributed Graph Transformation Systems*. In ICGT'2002. LNCS 2505, pp.: 161-176. Springer.
8. Minas, M. 2003. *Bootstrapping Visual Components of the DiaGen Specification Tool with DiaGen* Proceedings of AGTIVE'03 (Applications of Graph Transformation with Industrial Relevance), Charlottesville, USA, pp.: 391-406. See also the DiaGen home page: `http://www2.informatik.uni-erlangen.de/DiaGen/`.
9. Schürr, A. 1994. *Specification of Graph Translators with Triple Graph Grammars*. LNCS 903, pp.: 151-163. Springer.
10. Vangheluwe, H., de Lara, J., Mosterman, P. 2002. *An Introduction to Multi-Paradigm Modelling and Simulation.* Proc. AIS2002. Pp.: 9-20. SCS International.