Abstract

This article introduces the use of multi-paradigm modelling to facilitate computer assisted modelling of hybrid systems. Multi-Paradigm modelling consists on the combination of meta-modelling, multi-formalism and multiple abstraction levels. The approach allows one to model different parts of a system using different formalisms. Models can be automatically converted between formalisms as the latter are also (meta-)modelled and represented as Abstract Syntax Graphs. Thus, transformations between formalisms can be represented as models in the graph grammars formalism. We present the implementation of these concepts in AToM$^3$, and as an example we model a hybrid system by combining Causal Block Diagrams (for the continuous part) and Statecharts (for the discrete part). Both models are translated into the object-oriented simulation language OOCSMP for simulation.

Keywords: Hybrid Systems, Multi-Paradigm Modelling, Graph Grammars, Causal Block Diagrams, Statecharts, AToM$^3$, OOCSMP.

1 Introduction

Complex systems are characterized by components of different nature which have to be modelled using the most appropriate formalism. This need for multiple modelling formalisms is specially acute in hybrid systems, composed of continuous and discrete elements. Several approaches are possible to deal with these systems:

1. A single super-formalism may be constructed which subsumes all the formalisms needed in the system description. This is neither possible nor meaningful in most cases, as it is difficult to find such a powerful formalism, and sometimes more convenient problem specific formalisms are preferred for each component.

2. Each system component may be modelled using the most appropriate formalism and tool. In this approach, each component is simulated with a formalism-specific simulator (co-simulation). Interaction due to component coupling is resolved at the trajectory (simulation data) level. Questions about the overall system can only be answered at the level of input/output (state trajectory). It is no longer possible to answer symbolic, higher-level questions which could be answered within the individual components’ formalisms.

3. As in co-simulation, each system component may be modelled using the most appropriate formalism and tool. In multi-formalism modelling and simulation however, a single formalism is identified into which each of the component models may be symbolically transformed [16]. The formalism to transform to depends on the question to be answered about the system. It is easily seen how multi-formalism modelling subsumes both the super-formalism and the co-simulation approaches.

In order to make the multi-formalism approach applicable, we have to interconnect a plethora of different modelling tools for different formalisms. We use meta-modelling to solve this problem. Using a higher layer of modelling, we describe the formalisms themselves and using this information we generate customized tools for the described formalisms. The effort required to construct a tool for a modelling formalism thus becomes minimal. If we use a common data structure to internally represent the models, interconnecting the tools becomes easier, as transformation between formalisms is reduced to the transformation of these data structures.

In this article, we describe the use of AToM$^3$ [3] for the modelling of hybrid systems. AToM$^3$ is a tool which implements the multi-paradigm concepts presented above. The tool has a meta-modelling layer in which different formalisms are modelled. From this meta-specification AToM$^3$ generates a tool for the described formalism. Models are represented internally using Abstract Syntax Graphs. As a consequence, transformations between formalisms is reduced to graph rewriting. Thus, the transformations themselves can be expressed as graph grammar rules [7]. Although graph grammars have been used in
very diverse areas such as graphical editors, code optimization, computer architecture, etc. [8], to our knowledge, they have never been applied to multi-paradigm modelling. As an example of the use of the multi-paradigm concepts in AToM³ for the modelling of hybrid systems, we present the classical hybrid system of a temperature and level controlled liquid in a vessel [2]. The liquid temperature and level is modelled using Causal Block Diagrams (CBD), whereas the controller is modelled using Statecharts[11]. For simulation, both components are transformed into the simulation language OOCsmp [1]

The rest of the paper is organized as follows: section 2 describes the multi-paradigm approach for modelling complex systems; section 3 gives a brief overview of AToM³; section 4 presents a meta-model for OOCsmp, similar to UML class diagrams in which methods can be described either in the Statecharts, in the CBD formalism or as OOCsmp textual code; section 5 presents the example of the vessel, which is described with the meta-model described before, and then translated into OOCsmp for simulation; section 6 presents the related work and finally section 7 presents the conclusions.

2 Multi-Paradigm Modelling

Computer Automated Multi-Paradigm Modelling [17] is an emerging field which addresses and integrates three orthogonal directions of research:

1. Multi-Formalism modelling, concerned with the coupling of and transformation between models described in different formalisms. In Figure 1, a part of the “formalism space” is depicted in the form of an FTG. The different formalisms are shown as nodes in the graph. The arrows denote a homomorphic relationship “can be mapped onto”. The mapping consists of transforming a model in the source formalism into a behaviourally equivalent one in the target formalism.

In our approach, we allow the specification of composite systems by coupling heterogeneous components expressed in different formalisms. For the analysis of its properties the composite system must be assessed by looking at the whole multi-formalism system. That is, its components may have to be transformed to a common formalism, which can be found in the FTG. In our approach formalisms are meta-modelled and stored as graphs. Thus, the transformations denoted by the arrows of the FTG can be modelled as graph grammars.

2. Model Abstraction, concerned with the relationship between models at different levels of abstraction.

3. Meta-Modelling, which consists on modelling the formalisms themselves. Formalisms are described as models using meta-formalisms, which are formalisms expressive enough to describe other formalisms’ syntax and semantics. Examples are the Entity Relationship (E-R) formalism or UML class diagrams. A model of a meta-formalism is called a meta-meta-model; a model of a formalism is called a meta-model. Table 1 depicts the levels considered in our meta-modelling approach.

The meta-level formalism may have to be extended with the ability to express constraints. For example, when modelling a Deterministic Finite Automaton, different transitions leaving a given state must have different labels (to ensure determinism). This cannot be expressed within E-R or UML class diagrams alone. Constraints are usually expressed in a textual constraint language, which is added to the (usually graphical) meta-modelling formalism. For this purpose, some systems [15] (including ours) use the Object Constraint Language OCL [12] used in the UML. As AToM³ is implemented in Python [14], arbitrary Python code may also be used.

3 AToM³: an overview

AToM³ is a tool for computer aided multi-paradigm modelling, which is being developed at McGill University in collaboration with the Universidad Autónoma in Madrid. The main idea of the tool is “everything is a model”. In the sense that, during implementation, the AToM³ kernel has been grown from a small initial kernel, models were defined for other parts of it, code was generated and then later incorporated in the kernel. Also, for AToM³ users, it is possible to modify some of this model-defined components, such as the type system, (meta-)formalisms and the user interface.
AToM³’s architecture is shown in Figure 2. Models are represented as white boxes, having on their upper-right corner an indication of the (meta-)+model they were specified with. It must be noted that model manipulations are also models, expressed in the graph grammar formalism.

The main component of AToM³ is the Kernel, which is responsible for loading, saving, creating and manipulating models (at any meta-level, via the Graph Rewriting Processor), as well as for generating code for customised tools. Both meta-models and meta-meta-models can be loaded into AToM³ as shown in Figure 2. The first kind of models allows constructing valid models in a certain formalism, the latter are used to describe the formalisms themselves.

The E-R formalism extended with constraints is available at the meta-meta-level. As stated before, it is perfectly possible to define other meta-formalisms using E-R. Constraints can be specified as OCL or Python expressions, and the designer must specify when (pre- or post- and on which event) the condition must be evaluated. Events can be semantic (such as editing an attribute, connecting two entities, etc.) or graphical (such as dragging, dropping, etc.)

When modelling at the (meta-)¹ level, the entities that may appear in a model must be specified together with their attributes. For example, to define the Petri Net Formalism, it is necessary to define both Places and Transitions. Furthermore, for Places we need to add the attributes name and number of tokens. For Transitions, we need to specify the name.

The (meta-)¹information is used by the AToM³ Kernel to generate some Python files, which, when loaded by the Kernel, allow the processing of models in the defined formalism (see upper-right corner in Figure 2, labeled as “AToM³ (meta-)+models structure”.) One of the components of the generated files is a model of a part of the AToM³ user interface. This User Interface model follows the “Buttons” formalism, and has its own meta-model. Initially, this model represents the necessary buttons to create the entities defined in the formalism’s meta-model, but can be modified to include for example, buttons to execute graph grammars on the current model.

In the meta-model, it is also possible to specify the graphical appearance of each entity of the defined formalism. Objects’ graphical appearance can be icon-like or arrow-like with optional icons decoration in the center, segments and extremes.

4 Hybrid Systems Modelling with AToM³: Meta-Modelling OOCSMP

Our approach for modelling hybrid systems with AToM³ is to use the most appropriate formalism for each component in the system, and then translate them into a common formalism for simulation. In this case, this formalism is OOCSMP [1]. This is an Object Oriented extension of the CSMP Continuous Simulation Language, developed at the Universidad Autónoma de Madrid. Other extensions we have added to OOCSMP allow to handle discrete events, solve Partial Differential Equations or produce distributed simulations. A compiler (called C-OOL) is able to produce Java applets from the OOCSMP models to perform the simulation. One of the main drawbacks of the system is the lack of a graphical modelling environment, models are textual files which must be coded by hand.

In this section we present a meta-model for OOCSMP (built using AToM³). The idea is to use AToM³ to describe OOCSMP models in a formalism similar to UML class diagrams, in which methods can be described using either CBDs, Statecharts or directly in textual OOCSMP. These models are then translated into textual OOCSMP code for simulation. This translation is also a model, expressed in the graph grammar formalism. This process is illustrated in figure 3.

In figure 4 the meta-model for OOCSMP is shown. This meta-model has been built using the E-R formalism. It shows that OOCSMP models are made of Classes, which have a name, a list of Parameters (similar to class attributes) and methods (specified in CBDs, Statecharts or directly in textual OOCSMP). This allows for the specification of multi-formalism systems in appropriate, graphical and well-known formalisms instead of using a lower level textual language. OOCSMP classes can be related by inheritance. From this meta-information, AToM³ is able to generate the tool in figure 5. In the window in the background, an object named Vessel has been declared, this object has a number of parameters (T0, L0, etc), and two methods Behaviour and Controller. The latter is being specified (see top-most window) using the Statecharts formalism.
Figure 2: Meta-Model Modelling in AToM3.

Figure 5: The generated Tool for Hybrid Modelling in OOCSPM. Describing the Temperature and Level Controller of a Vessel using a Statechart
5 An example: A Temperature and Level Controlled Vessel

As an example of applying these concepts and tools, a temperature and level controlled liquid in a vessel is considered. This is a modification of the system described in [2], where structural change is the main issue. On the one hand the liquid can be heated or cooled; on the other hand, liquid can be added or removed (we do not consider phase changes for this simple example). The liquid temperature and level is described by the following Ordinary Differential Equation model:

\[
\frac{dT}{dt} = \frac{1}{L} \left( \frac{W}{c\rho A} - \phi T \right) \quad (1)
\]

\[
\frac{dl}{dt} = \phi \quad \text{if } 0 < l < H \text{ else } 0 \quad (2)
\]

\[is\text{\_low} = (l < l_{low}) \quad (3)\]

\[is\text{\_full} = (l > l_{high}) \quad (4)\]

\[is\text{\_cold} = (T < T_{cold}) \quad (5)\]

\[is\text{\_hot} = (T > T_{hot}) \quad (6)\]

Where the inputs of the model are \(\phi\), the flow rate, and \(W\), the rate at which heat is added or removed. The model’s parameters are \(A\), the cross-sectional surface of the vessel, \(H\) its height, \(c\) the specific heat of the liquid and \(\rho\), the density of the liquid. Equations 3-6 set some threshold output sensors, which are the outputs of this model, and that are sent to the controller.

Equations 1-6 have been included in method Behaviour in class Vessel (see figure 5) using CBDS.

The other part of the system is a controller for the temperature and the level, which tries to maintain both quantities between certain limits. This is implemented as a Statechart and was shown in figure 5. This model was embedded in the method Controller in the Vessel class. The idea is that we will have an orthogonal component for the temperature and another one for the level. Each orthogonal component has three states, which corresponds to the situation in which the quantity to control (level or temperature) is high, medium or low. The possible events (that is, the model’s inputs) which make the system change its state are is\_low, is\_high, is\_cold and is\_hot. As associated actions to these events, \(\phi\) and \(W\) are modified. For example, if the system is in state cold in orthogonal component temperature and receives the event IS\_COLD, the it increases \(W\) and remains in the same state (responding to subsequent IS\_COLD events in the same way) until the event IS\_COLD is not sent by the Behaviour model. \(\phi\) and \(W\) are the outputs of this model, and are fed in the method Behaviour. Finally, the main simulation loop is modelled as textual OOCSMP code and contains the appropriate invocation to methods Behaviour and Controller.

This model is transformed into OOCSMP for simulation by defining a model in the graph grammars formalism. Part of this graph grammar – the one to translate from CBDS into OOCSMP – was presented in [4]. Statecharts can also be converted to OOCSMP because of its discrete event handling possibilities. Once the model has been compiled into Java with COOL, we can execute the simulation. A moment in the simulation is shown in figure 6. In the upper graphic, the variation of temperature and level is shown. It can be seen how, after some transitory effects at the beginning of the simulation, the temperature reaches an oscillatory equilibrium. In the middle graphic the values of IS\_COLD, IS\_HOT, IS\_LOW and IS\_HIGH are shown. It can be observed how IS\_COLD and IS\_HOT are switching from 0 to 1 continuously to maintain the temperature between the limits. In the lower graphic, a listing of some of the variable values is shown. This simulation applet is available at http://www.ii.uam.es/~jlara/investigacion/ecomm/hybrid.html for the reader interested in experimenting with it.

6 Related work

The approach of [13] is similar to ours: they have implemented several editors for continuous (sequential function charts) and discrete formalisms (Statecharts) using the meta-modelling tool DoME [6]. The user builds his composite models with these editors, and they are subsequently translated into the object oriented simulation language MODELICA [9]. In DoME, model manipulations should be expressed either in the Lisp-like language Alter or in Smalltalk; in our approach they can be visually specified by means of graph grammars (combined with Python if desired).

A different approach to the modelling of complex systems is Paul Fishwick’s Multi-Modelling [10]. Basically, his approach deals with multi-formalism (a multi-model can be composed of components described in different formalisms) and multiple abstraction levels. To synchronize the system at its highest level, a coordinator is used to direct the events to the appropriate models (co-simulation). The approach of the Ptolomey [5] environment is another example of the power of this approach. It is noted that its power is partly due to the fact that it is an integrated environment, which does not need to interface with external simulations.

Our approach (Multi-Paradigm Modelling) proposes Meta-Modelling as a method to obtain a tool to model in each formalism, and translates models between formalisms for the purpose of simulation and analysis. This previous step (translation) to simulation makes the simulation process cleaner and permits analysis of properties of the multi-formalism model with tools available in the base formalism. In addition, when a model is translated into a formalism, there are possibilities to apply optimizing transformations. If the translation process goes through multiple formalisms, the one can apply optimizing transformation in each formalism. For example, if all the components of a multi-formalism system are translated, say, to Petri-Nets, then one can use complexity reduction transformations and later reachability analysis to verify that certain system states are possible. In contrast, in the Multi-Modelling approach, one cannot apply these analysis tools, as each component has been described in a different formalism and no trans-
formation to a common formalism is performed. As stated in
the introduction, it is not sufficient to look at properties of the
subcomponents in isolation, one should look at the properties
of the system as a whole.

7 Conclusions

In this paper we have presented an overview of AToM³, a tool
which makes the generation of modelling tools possible by
combining meta-modelling and graph grammars. By means of
meta-modelling it is easy to define the (graphical) syntax of
the kind of models we are interested in. By means of graph
grammars we can express model manipulation (simulation, op-
timization, transformation and code generation). As an exam-
ple, we have presented the generation a visual modelling en-
vironment for OOCSMP in which methods can be expressed
as CBDs, Statecharts or textual OOCSMP code. An example
of the use of the OOCSMP meta-model to describe an hybrid
system has also been presented.

The advantages of using an automated tool for generating
customized model-processing tools are clear: instead of build-
ing the whole application from scratch, it is only necessary to
specify –in a graphical manner– the kind of models we will
deal with. The processing of such models can be expressed by
means of graph grammars, at the meta-level. Our approach is
also highly applicable if we want to work with a slight vari-
ation of some formalism, where we only have to specify the
meta-model for the new formalism and a tranformation into a
“known” formalism (one that already has a simulator available,
for example). We then obtain a tool to model in the new for-
malism, and are able to convert models in this formalism into
the other for further processing.

With respect to the example, the advantages of using a multi-
paradigm approach instead of coding directly in OOCSMP (the
super-formalism approach, see introduction) are clear: we have
used well-know, high level formalisms (CBDs and Statecharts)
to describe the continuous and discrete part of the model. In
contrast, the direct use of textual OOCSMP to model both
parts of the system makes this process harder. In particular the
direct coding of the controller part in OOCSMP is error prone
and much less intuitive, reusable and analyzable than modelling
it with a visual formalism such as Statecharts.

References

[1] Alfonseca, M., Pulido, E., Orosco, R., de Lara,
J. 1997. OOCSMP: An Object-Oriented Simula-
tion Language. Proceedings of the 9th European
Simulation Symposium ESS97, SCS Int., Erlan-
gen. pp. 44–48. See the OOCSMP home page at:

timodels and dynamic structure models: an integration
of DSDE/DEVS and OOPM. In Proceedings of the 1998
Winter Simulation Conference, pp.: 413-419.


