JOT
JOURNAL OF
OBJECT TECHNOLOGY

Journal of Object Technology | **RESEARCH ARTICLE**

Rewriting Logic and Maude for the Formalization and Analysis of DSMLs, and the Prototyping of MDSE Tools

Francisco Durán*
*ITIS Software, University of Málaga, Málaga, Spain

ABSTRACT A few years ago, Antonio Vallecillo and I initiated a long-term collaboration on different matters related to MDSE. In particular, we were interested on the possibilities of analysing models, long before they were implemented. For that, we explored the possibility of graphically defining the behavior of systems and DSMLs. Then, using Maude as back-end tool, and by defining a model transformation between our language for DSML definition and Maude, we were able to execute and analyze DSMLs defined in this way. The simplicity of the setting, and the reduced effort in its implementation allowed us to explore the possibility of considering time-related features, different forms of analysis, including reachability analysis, model checking, and statistical model checking. The same ideas have later been also applied to multi-level modeling in the MultEcore system. This paper summarizes the main contributions by Antonio and myself on these matters.

Dedicated to Antonio Vallecillo in his 60th birthday

KEYWORDS Rewriting logic, Maude, domain-specific modeling languages, timed systems, multi-level modeling, formal verification.

1. Introduction

Model-driven software engineering (MDSE) promotes the use of abstractions and different modelling techniques to tackle the complexity of software by considering models as primary artifacts in each phase of the software engineering life-cycle (Kent 2002). Indeed, MDSE advocates the use of models as the key artifacts in all phases of development. By using appropriate techniques, whole systems can in this way be implemented, even automatically generated, from these models, with a number of advantages (Brambilla et al. 2012). Also, these models can be used to carry on some analysis on them, allowing us to establish certain guarantees on the generated systems at design time. This has in fact been a goal for the software industry for decades, comparing itself with other branches of engineering were models and analysis tools have been available for many years, and are continuously being improved and adapted to everyday's new challenges.

In MDSE, many different possibilities for the analysis of models are available, from syntactic or type checkings to sophisticated theorem proving or model checking. At development time, more low-level facilities, like editors, collaboration tools, typing checks, etc. are the most valuable ones. However, our efforts mean nothing if what is being developed does not meet our requirements, and therefore, the sooner we are able to check properties on our models the better. And precisely in this later type is where we are nowadays short.

To be able to define models, using concepts as close to the problem domain as possible, different technologies for the definition of Domain Specific Modeling Languages (DSMLs) have been proposed (see, e.g., (Kelly & Tolvanen 2008)). The main goal of these DSMLs is to follow the domain abstractions and semantics, allowing modelers to perceive themselves as working directly with domain concepts. Model transformations may then be used to analyze certain aspects of models and then automatically synthesize various types of artifacts, such as source code, simulation inputs, or alternative model representations.

With these ideas in mind, and trying to use *our own medicine*, several years ago Antonio Vallecillo and I initiated a collaboration to explore the possibility of graphically defining the behavior of DSMLs. The original idea was to model change as local transformations of UML's object diagrams, but we soon became aware of graph transformation systems and tools like AGG (Taentzer

JOT reference format:
Francisco Durán. *Rewriting Logic and Maude for the Formalization and Analysis of DSMLs, and the Prototyping of MDSE Tools*. Journal of Object Technology. Vol. 21, No. 4, 2022. http://dx.doi.org/10.5381/jot.2022.21.4.a2

1999) and AToM3 (de Lara & Vangheluwe 2002). The potential of graph and term rewriting took us into the possibility of expanding existing modelizations, and considered the possibility of providing support for the definition and analysis of real-time systems. E. Rivera, who developed the e-Motions language and tool as part of his PhD, was key in this effort. Instead of developing a full-fledged implementation of e-Motions, the system was engineered as a model transformation into the Maude language (Rivera & Vallecillo 2007). Graphical support was developed as an Eclipse plug-in, but underneath there was a formal tool giving access to Maude's capabilities. The approach presented three main advantages: (1) the transformation defined a *formal semantics* in rewriting logic of the e-Motions system; (2) this formalization gave access to Maude's environment of *formal tools*, and (3) the fact that rewriting logic is executable under given conditions, allowed us to also *simulate*, through such a transformation, the DSMLs defined using e-Motions. All were advantages:

- Even though in the MDSE world not much attention is paid to the formal semantics of languages and tools, e-Motions' semantics was made explicit, and the sound use of tools like Maude's reachability checker and its model checker were made directly available.
- The approach opened also the possibility to access other tools in Maude's formal environment (Durán et al. 2011b,a), like its theorem prover (Clavel et al. 2006), its termination (Durán, Lucas, & Meseguer 2008; Durán et al. 2009; Durán, Lucas, Marché, et al. 2008), Church-Rosser and coherence checkers (Durán & Meseguer 2012; Durán, Meseguer, & Rocha 2020), or its symbolic analysis capabilities (Durán, Eker, et al. 2020). We have not had the time nor the resources to include these in the e-Motions' tool, or we may say that more interesting challenges got in our way.
- Even though the transformation into Maude only provided a prototypical implementation, it presented a number of advantages, in addition to the obvious disadvantages:
  - First and foremost, even though the efficiency of term rewriting modulo associativity, commutativity and identity (ACU rewriting) available in Maude, it is easy to imagine a faster implementation written in C++ o Java.
  - Furthermore, although efficiency was in our minds, maintainability and correctness of the specification/implementation was more important to us. In fact, our main goal was to keep the specification at the highest possible level. The level was so high, that the distance between the textual representation of the e-Motions transformation rules and the Maude rules was close to zero. The main advantage of this is that the transformations between e-Motions DSMLs and Maude specifications, and back, were straightforward. Most of the effort went into engineering the formal semantics, the key part, and the graphical interface.
  - Since the Maude specification of e-Motions satisfied its executability requirements, the specification was automatically executable, and we got access to simulation, reachability analysis, and model checking *for free*. The only price to pay was on the graphical representation of the outputs of the tools back in the Eclipse plug-in.
  - The low effort in its development made it very attractive to experiment with new features.

The approach opened several doors related to the development of other tools, mainly related to:

- Access to new analysis techniques: statistical model checking was made available to e-Motions in (Durán et al. 2016).
- Multi-level modeling (MLM): the same techniques used for e-Motions were applied to the development of an implementation for the MultEcore system (Macías et al. 2016, 2017, 2018), a system for the development of MLM DSMLs. In this case, a categorical semantics was provided in (Macías et al. 2018; Wolter et al. 2020) and its full-fledged development using Maude was presented in (Rodríguez, Durán, et al. 2019; Macías et al. 2019; Rodríguez et al. 2022).
- Composition of DSMLs: the composition of DSMLs using e-Motions and MultEcore, or formalized using related technologies, was explored in (Durán, Orejas, & Zschaler 2012; Durán, Zschaler, & Troya 2012; Moreno-Delgado et al. 2014; Zschaler & Durán 2017; Durán et al. 2017; Rodríguez, Rutle, et al. 2019; Zschaler & Durán 2021).
- Several challenging case studies were developed.
  - Using e-Motions:
    - The real-time and composition capabilities of e-Motions were exploited to model different non-functional properties with which to analyze systems in (Durán, Zschaler, & Troya 2012; Troya et al. 2013).
    - A significant part of Palladio was modeled in e-Motions in (Moreno-Delgado et al. 2014).
    - Self-adaptive systems were modeled and analyzed using e-Motions in (de Oliveira et al. 2017, 2021).
    - A. Moreno-Delgado won the "ease of use" award in the 14th Transformation Tool Contest with his specification of the Movie Database Case (Moreno-Delgado & Durán 2014).
    - The tool and several other case studies are available from Atenea's web site at https://atenea.lcc.uma.es/projects/E-motions.html.
  - Using MultEcore
    - Coloured and other types of Petri nets were modeled in (Rodríguez et al. 2018; Durán & Rodríguez 2021; Rodríguez et al. 2022).
    - F. Macías and A. Rodríguez have participated in several editions of the MULTI Challenge (Macías et al. 2017; Rodríguez & Macías 2019).
    - The tool and several other case studies are available at https://ict.hvl.no/multecore/.

The rest of the paper is structured as follows. Section 2 summarizes the main features of the e-Motions system and its approach for the modeling of time-dependent behavior of DSMLs. Section 3 presents the main ideas behind the support in

MultEcore for multi-level modeling and its transformation into Maude to provide execution and analysis facilities. Section 4 discusses the main ideas on the support for formal analysis provided to e-Motions and MultEcore DSMLs. Section 5 widens the discussion to some related works and concludes.

2. The graphical modeling of time-dependent behavior of DSMLs with e-Motions

DSMLs are typically defined by means of their structural aspects—with their corresponding abstract and, in some cases, concrete syntaxes. These definitions allow the rapid development of languages and some of their associated tools, such as editors, browsers, etc. Then, typically, to perform some type of analysis on or to generate code from these models, they are transformed into formalisms or programming languages with the appropriate tool support. However, the semantics of such DSMLs is embedded in the model transformations, and provided by the target formalism, what constrains the rapid definition of such languages. To overcome this situation, different authors have proposed different ways of providing an operational semantics as part of the definition of DSMLs, possibly being the most successful one the one using graph transformation systems (GTS) (Rozenberg 1997), with systems such as AToM3 (de Lara & Vangheluwe 2002), AGG (Taentzer 1999) or e-Motions (Rivera et al. 2009b) implementing it.

The specification of the explicit behavioral semantics of DSMLs helps in MDSE activities such as quick prototyping, simulation, or analysis. Ensuring semantic properties of models is important because any error in a model can easily become a systemic error in the system under development. E.g., AGG and e-Motions provide support for the simulation of models defined conforming to user-defined DSMLs. These and other languages provide support for different kinds of analysis as well, like termination checks, critical pair analysis, or reachability analysis (see, e.g., (Taentzer 1999) and (Rivera et al. 2009a)). Tools like CheckVML (Schmidt & Varró 2003), GROOVE (Rensink 2003) and e-Motions (Rivera et al. 2009b) support the model checking of systems whose behavior is specified by graph transformation systems. The e-Motions tool also provides support for statistical model checking of user-defined DSMLs (Durán et al. 2016).

One way of specifying the dynamic behavior of a DSL is by describing the evolution of the modeled artifacts along some time model. In MDSE, this can be naturally done using model transformations supporting in-place update (Czarnecki & Helsen 2003). The behavior of the DSL is then specified in terms of the permitted actions, which are in turn modeled by the model transformation rules. In these transformation languages, source and target models are always instances of the same metamodel (i.e., they are *endogenous* transformations), and rules are used to (1) describe the preconditions of the actions to be triggered, and (2) represent the effects of these actions in the model. If these transformations use the concrete syntax associated to a DSML, they allow designers to work using only domain specific concepts (de Lara & Vangheluwe 2008), thus raising the level of abstraction and making behavioral specifications more intuitive and natural both to specify and understand.

In critical domains, such as real-time and embedded systems, timeouts, timing constraints and delays are essential concepts. Therefore, these concepts should be explicitly modeled in their behavioral specification to enable a proper analysis and simulation. However, only a few of the current approaches deal with time-dependent behavior. Besides, current approaches do not allow users to model action-based properties, making them inexpressible or forcing unnatural changes to the system specification (Meseguer 2008).

In e-Motions, standard in-place rules were extended so that time and action statements could be included in the behavioral specifications of a DSML. Features like on-going actions, periodicity, and different execution modes, namely eager and lazy, turned to be essential aspects when trying to capture some critical properties of real-time systems. The e-Motions system provided a graphical framework aimed at defining behavioral specification models, amenable for their integration in MDSE processes. Its precise behavioral semantics was given by mapping it into Real-Time Maude (Ölveczky & Meseguer 2007).

Figures 1-2 show the basic elements of the definition of a DSML using e-Motions, specifically a DSML for production systems. Its metamodel is shown in Figure 1. A production system is composed of different kinds of machines: generators, transfers, assemblers and containers. Generators produce items, transfer machines move them through different trays, assemblers consume them to create new ones, and containers store the assembled items. Machines take their inputs from and put their results in trays which contain items up to their capacity. Figure 2 shows then a production model for hammers. It is composed of generators of heads and handles, a transfer machine that moves the generated heads and handles to the input tray of an assembler, an assembler that combines heads and handles to form hammers, and a container that stores (an unlimited number of) hammers.

In our Maude representation of e-Motions, models are represented by sets of objects: Nodes are represented as objects, and node attributes by object attributes. Edges are represented by object attributes too, each one representing the reference (by means of object identifiers) to the target node of the edge. Thus, models are structures of the form $mm\{obj_1\, obj_2 \ldots obj_N\}$, where mm is the name of the metamodel, and the obj_i are the objects that represent the corresponding nodes. Then, given the appropriate definitions for all classes, attributes and references in its corresponding metamodel, Figure 3 shows the Maude term representing the hammers production model depicted in Figure 2.

The dynamic behavior of the system is then specified as an in-place model transformation, which is composed of a set of rules. Each one of these rules represents a possible *action* of the system. For example, the PLS system has rules modeling each of the possible actions that may occur: generation of a head, generation of a handle, carrying a part along a conveyor, etc. Figure 4 shows the Carry rule. In general, these rules are of the form $l : [\text{NAC}]^* \times \text{LHS} \rightarrow \text{RHS}$, where l is the rule's label (its name, Carry in this case); LHS (its left-hand side), RHS (its right-hand side), and NAC (negative application conditions) are model patterns that represent certain (sub-)states of the system. The LHS and NAC patterns express the precondition for the rule to be applied, whereas the RHS one represents its postcondition, i.e., the effect

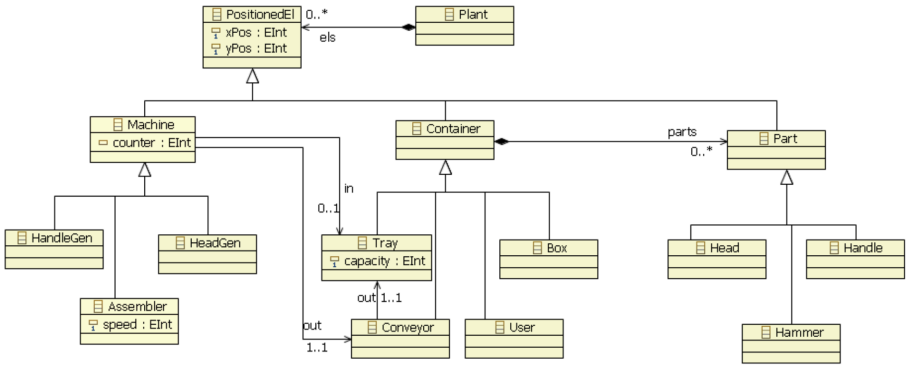

Figure 1 A production system metamodel.

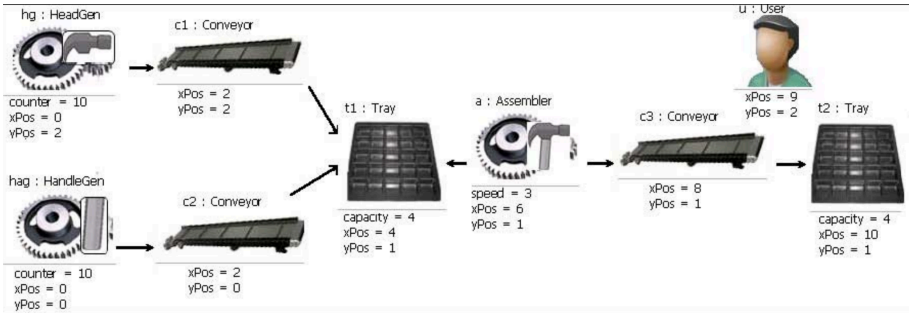

Figure 2 A Production System Model.

```
ProductionSystem {
 < 'hg : HeadGen | counter : 10, xPos : 0, yPos : 2, in : empty, out : 'c1 >
 < 'hag : HandleGen | counter : 10, xPos : 0, yPos : 0, in : empty, out : 'c2 >
 < 'c1 : Conveyor | items : empty, xPos : 2, yPos : 2, in : 'hg, out : 't1 >
 < 'c2 : Conveyor | items : empty, xPos : 2, yPos : 0, in : 'hag, out : 't1 >
 < 't1 : Tray | capacity : 4, items : empty, xPos : 4, yPos : 1, in : ('c1, 'c2), out : 'a >
 < 'a : Assembler | speed : 3, xPos : 6, yPos : 1, in : 't1, out : 'c3 >
 < 'c3 : Conveyor | items : empty, xPos : 8, yPos : 1, in : 'a, out : 't2 >
 < 't2 : Tray | capacity : 4, items : empty, xPos : 10, yPos : 1, in : 'c3, out : null > } .
```

Figure 3 Maude term representing the hammers production model in Figure 2.

of the corresponding action. In the case of the Carry rule, we can observe how, in its left-hand side, there is a part in a conveyor preceding a tray. In its right-hand side, we observe how the part is now in the tray. Thus, a rule can be applied, i.e., *triggered*, if an occurrence (or match) of the LHS is found in the model and none of its NAC patterns occurs. Thus, the Carry rule models the action of moving a part from a conveyor to its subsequent tray. Notice that the rule includes information, not only on how the interconnections between objects may change, or whether objects are created or destroyed, but also how their attributes may change as a result of the application of a rule. The header of the rule may also have information on its timed application, including whether it is instantaneous or not, and in such a case its duration, whether it is periodic or not, and if so what is its period, etc.

The simulation of the system begins with some initial configuration—as the one in Figure 2—by consecutively applying the different rules. In fact, on a given model, different rules may be applied—on the same objects or on differnet ones—and the same rules may be applied on different ways, that is, there might be different matches of the left-hand side of the rule with different parts of the current model. Using the default strategy, rules are applied non-deterministically, and if several matches are found for some of the rules, one of them is non-deterministically selected and applied, producing a new model where the match is substituted by the appropriate instantiation of its RHS pattern (the rule's *realization*). The model transformation proceeds by applying the rules in a non-deterministic order, until none is applicable—although this behavior can be usually modified by some execution control mechanism. The possibility of defining execution strategies, perhaps taking advantage of Maude's strategy language, seems an interesting line of future research (Clavel, Durán, et al. 2007).

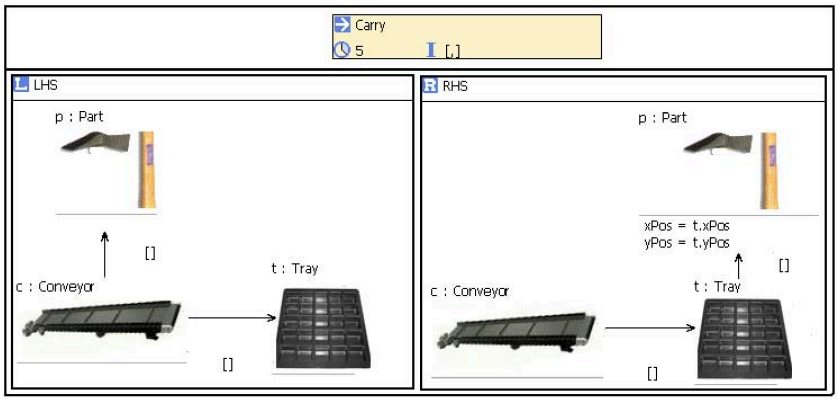

Figure 4 The Carry rule.

3. A Rewriting Logic Semantics for MultEcore

Most of the existing approaches for MDSE are based on the Object Management Group's 4-layer architecture (Meta Object Facility 2016). In these approaches, the number of levels that designers can use to specify modelling languages is restricted: one for (meta)models and one for their instances. However, it has extensively been argued that in many situations the two-level restriction may lead to different problems, including model convolution, accidental complexity, and mixing concepts belonging to different domains—see, e.g., (de Lara et al. 2014; Atkinson & Kühne 2008) for discussions on these issues. By using multilevel modelling (MLM), modellers are no longer forced to fit their modelling language specifications within two levels of abstraction. Instead, in MLM, models can be organised into multiple levels (Atkinson & Kühne 2001). MLM techniques are excellent for the creation of DSMLs, especially when focusing on languages with dynamic behaviour, since behaviour is usually defined at the metamodel level, while it is executed (at least) two levels below at the instance level (de Lara & Guerra 2010b; Atkinson et al. 2015).

Although there exist diverse approaches for MLM—see e.g., (de Lara & Guerra 2010a; Atkinson & Gerbig 2016; Macías et al. 2017)—we focus here on the approach proposed by the tool MultEcore (Macías et al. 2017), formally specified in (Macías et al. 2018, 2019). Specifically, MultEcore is a language and tool for the definition of DSMLs following the MLM approach. In MultEcore, the structure of systems is captured by a hierarchy of models, where each element of a model is typed by an element in a model at an upper level. In MultEcore, one can also explore the behavioural dimension of the specified multilevel hierarchies. As Groove (Rensink 2003) or e-Motions, MultEcore also proposes the definition and simulation of behavioural models based on reusable model transformations. However, in this case, instead of relying on traditional two-level modelling hierarchies, it does it for model hierarchies of any size.

Following an approach similar to the one discussed in Section 2, in (Macías et al. 2019; Rodríguez, Durán, et al. 2019; Rodríguez et al. 2022), the possibility of using Maude for providing a rewriting-logic semantics to MultEcore's MLM DSMLs and using it for execution and analysis was researched. Given an MLM definition of a DSML using the MultEcore facilities, a corresponding Maude (Clavel, Durán, et al. 2007) specification can be automatically generated. In this way, the MultEcore editing facilities become a complete development environment in which we can, not only edit our MLM models, but also experiment with them through their simulation and execution, and analyse them by giving access to Maude's verification tools. Also, as for e-Motions, the syntactical facilities of Maude allowed us to use a representation of MLM hierarchies and MCMT rules very close to that of MultEcore. Indeed, again, this minimal representation distance facilitated the automation of the bidirectional transformation between them.

For illustration purposes, let us use an MLM version of the PLS example. The PLS hierarchy, generated from the real implementation of the PLS multilevel hierarchy created with the tool MultEcore, is shown in Figure 5. Details on the definition of the example can be found in (Macías et al. 2019) or in MultEcore's web site. But let us try to identify the main elements in this hierarchy here.

The hierarchy displays a top model, called generic_plant (Figure 5a), where we define abstract concepts related to product lines that manufacture physical objects. At the top of any hierarchy, at level zero, we assume Ecore, with types EClass and EReference that may be used to type elements in lower levels. In the generic_plant model, Machine defines any device that can create, modify or combine objects, which are represented by the concept Part. In the first case, we indicate the relation from a generator-like machine to the part it generates with the creates relation. In order for parts to be transported between machines or to be stored, they can be inside Containers, and this relation is expressed by the contains relation. All machines may have containers to take parts from or to leave manufactured ones in. These two relations are identified with the in and out relations, respectively.

The second level of the hierarchy contains two models, that define concepts related to specific types of plants. On the left, Figure 5b, we can see the hammer_plant model, where the final product Hammer is created by combining one Handle and one Head. Both hasHandle and hasHead relations express this fact. The type of these two relations is EReference (from Ecore), since there is no relation defined for parts in generic_plant, because the concept of assembling parts is too specific to be

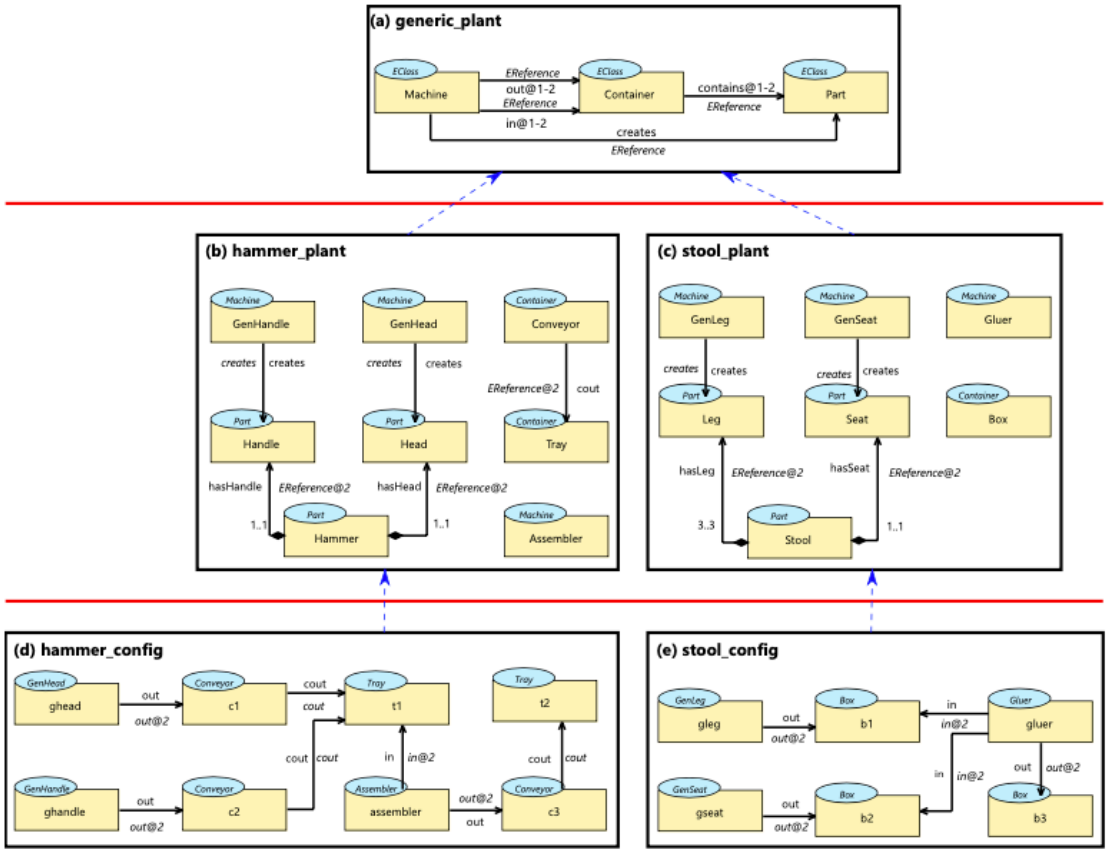

Figure 5 Hierarchy for the PLS case study

located in generic_plant. In the hammer_plant model we also define three types of machines. First, GenHandle and GenHead, that create the corresponding parts, indicated by the two creates arrows. And secondly, Assembler, that generates hammers by assembling the corresponding parts. Finally, the model contains two specific instances of Container, namely Conveyor and Tray. The cout arrow between them indicates that a Conveyor must always be connected to a Tray.

The other model in the second level, depicted in Figure 5c, contains another specification of a manufacturing plant, in this case for stools. In this model, the relations between Stool, Leg and Seat resembles those of Hammer, Handle and Head. The multiplicity of the arrow between the first two indicates that a Stool must have exactly three Legs. Two different types of machine, GenLeg and GenSeat, manufacture Legs and Seats, respectively. The remaining one, Gluer, takes three Legs and a Seat and creates a Stool out of them. Finally, the only container defined for this kind of plant is Box.

The two models at the bottom of the hierarchy, in Figures 5d and 5e, represent specific configurations of a hammer PLS (hammer_config) and a stool PLS (stool_config). They contain specific instances of the concepts defined in the level above, organized to specify correct product lines, in which parts get transferred from generator machines to machines that combine them to obtain the final manufactured products.

In MultEcore, the operational semantics of the system is captured by using the multilevel language so-called Multilevel Coupled Model Transformations (MCMTs), which were formally introduced in (Macías et al. 2018), and which extends traditional model transformation rules to multilevel models. Indeed, the rewriting logic semantics of MLM hierarchies and MCMTs used to specify such languages presented some interesting challenges. However, once such semantics was available, we were able to provide support for some experimental features, namely the parametrization of the tool with different languages to specify attribute values and conditions—the use of OCL and Standard ML was reported in (Durán & Rodríguez 2021)—and the use of nested and cross-level boxes for the specification of patterns in the rules.

As for e-Motions, the operational semantics of a DSML is given by a set of rules, MCMTs in this case, modeling the different actions that may happen in the system. As an example, Figure 6 shows the *SendPartOut* rule, used for moving a created part from its generator into the output container of this machine. This rule displays a META block in which we identify elements from two different models, separated with a red line. At the top level, we mirror part of the generic_plant model, defining elements like out and contains, that are used directly as types in the FROM and TO blocks.

A Maude specification of multilevel hierarchies and MCMTs provides a formal semantics of MultEcore models in rewriting logic. Based on such formalisation, the transformation MultEcore $\longleftrightarrow$ Maude is automated. The Maude specification obtained from MultEcore models using the above transformation

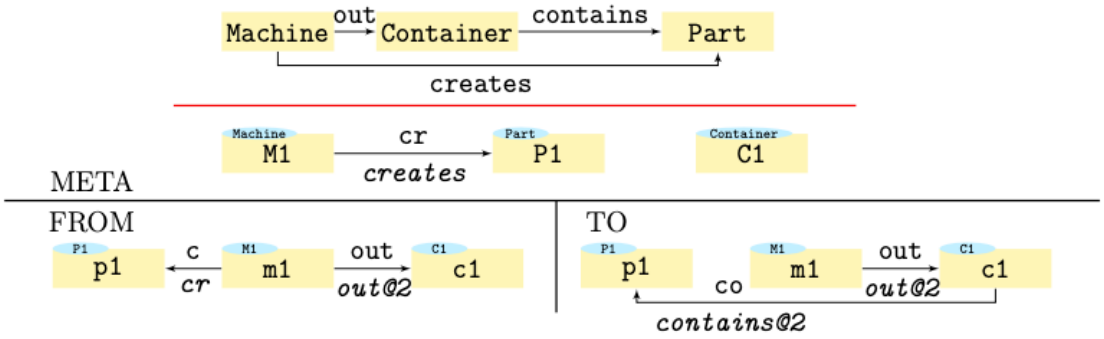

Figure 6 Rule *SendPartOut*: a part leaves the machine that created it

is executable, and it can therefore be used to simulate MultEcore models in Maude. Once the rewriting logic specification of a MultEcore model is available, we can use the tools in Maude's formal environment to analyse it. The constructed infrastructure that connects MultEcore with Maude is detailed in (Rodríguez, Durán, et al. 2019; Rodríguez et al. 2022).

We do not provide here a full description of the Maude semantics, but we would like to point out a few key issues:

- Each MultEcore object (including both a hierarchy and its MCMTs) is mapped into a corresponding Maude object.
- References and conditions are handled in exactly the same way, by using references as names, and using the same set of expressions (types and operators) for conditions.
- One of the major challenges are in the handling of boxing and the performing of the rewriting on multilevel hierarchies. A two-steps process has been developed to handle them. For each MCMT, a rule without the boxes is generated. When Maude finds a match for such a rule, it gets enough information to process the cardinalities of the most-external boxes. Then, using the metaprogramming capabilities of Maude, a second rule with the corresponding number of replicas of the boxes is generated on the fly, which is used to take the corresponding rewriting step using the original partial substitution. Nested boxes are processed one level at a time, recursively unfolding the boxes, and expanding the matching until no further boxes are left. Lets and conditions inside boxes are processed at each step.
- There might be boxes crossing multiple META levels, so that specific instantiations must be handled consistently.
- The support for OCL is based on the Maude semantics of OCL proposed in (Roldán & Durán 2011).

Additional information on MultEcore is available at https://ict.hvl.no/multecore/.

4. Formal analysis of e-Motions and MultEcore DSMLs

The specification of the explicit behavioral semantics of DSMLs helps in MDSE activities such as quick prototyping, simulation, or analysis. Ensuring semantic properties of models is important because any error in a model can easily become a systemic error in the system under development. As mentioned in the previous sections, and explained in details in some of the papers we have already referred to, the formalization of both e-Motions and MultEcore have allowed us to provide access to some of the tools offered in the Maude environment (Durán et al. 2011a,b; Clavel, Durán, et al. 2007). Specifically, reachability analysis and model checking have been made available for both e-Motions (Rivera et al. 2009a) and MultEcore (Rodríguez et al. 2022). For MultEcore, the use of equational abstraction (Meseguer et al. 2008) was exemplified in (Rodríguez et al. 2022) to model check systems with infinite state spaces.

This is, however, not enough, other tools are available to check or analyze diverse properties on systems, or to be able to handle more complex systems not satisfying the requirements of these tools. For example, since applications become more and more complex, and model checking is a very expensive procedure, both in time and space, its use is infeasible in many cases. A very important class of systems that falls out of the scope of classical model checkers are real-time stochastic systems. The methods used to verify quantitative properties of stochastic systems are typically based on numerical methods (Jansen et al. 2007), that iteratively compute the exact (or approximate) measure of paths satisfying relevant logical formulas. Although tools like PRISM (Hinton et al. 2006) and UPPAAL (Bengtsson et al. 1995) have shown very successful in the analysis of this kind of systems, explicitly constructing the corresponding probabilistic model is infeasible in many cases. An alternative method that solves this problem is based on statistical methods, similar to Monte Carlo simulations. By testing our hypothesis on many executions of a system, we may infer statistical evidence on the satisfaction or violation of the specification. Thus, properties like "the probability of completing task X with Y units of energy is greater than 0.3" or "the average amount of energy required to complete task X with confidence interval α and error bound β" can be checked. YMER (Younes 2005) and VeStA (Sen et al. 2005) were pioneering tools implementing these techniques. Well-established tools PRISM and UPPAAL have more recently also included capabilities for statistical model checking—see (Kwiatkowska et al. 2011) and (Bulychev et al. 2012).

Statistical methods have another advantage in the context of DSMLs: are "easy" to use and "cheap". As other model-checking methods, statistical model checking is completely automatic, and can be used where other methods fail. But they can also be used for "normal" systems with a shorter

computation time. Since statistical model checking assumes the existence of inaccuracy in its results, answers are calculated provided a confidence interval and an error bound. As may be expected, these requirements have an impact on the number of samples to be processed, and therefore on the evaluation time.

To be able to use the PVeStA tool (AlTurki & Meseguer 2011) (an extension and parallelization of VeStA), the transformation of e-Motions models was modified so that the generated Maude specifications satisfy the requirements of the tool (Durán et al. 2016). Such Maude specifications are therefore suitable to be stochastically analyzed using PVeStA.

5. Related Works and Wrap up

As pointed out from the beginning of the paper, the purpose of this work was to present some of the works that came up as result of the initial exploration that Antonio Vallecillo and myself initiated several years ago. Actually, it was not exhaustive in all the ideas and publications that came up from that collaboration, nor has made a completed overview of the works that we developed jointly or with other collaborators.

Some of the original ideas our work was based on are due to other authors. For example, one way to specify the semantics of a language is to define a translation from expressions in that language into expressions in another language with well defined semantics (Harel & Rumpe 2004). These *semantic mappings* between semantic domains are very useful, not only to provide metamodels with semantics, but also to be able to simulate, analyze or reason about them using the logical and semantical framework available in the target domain (Cuccuru et al. 2007).

For example, some works specify the behavioral semantics of models by using visual rules (de Lara & Vangheluwe 2004; Ehrig et al. 2006), which prescribe the preconditions of the actions to be triggered and the effects of such actions. These pre- and postconditions are given visually as models that use the concrete syntax of the DSL. This kind of representation is quite intuitive, because it allows designers to work with domain specific concepts and their concrete syntax for describing the rules (de Lara & Vangheluwe 2008). Graph transformation is one of these rule-based approaches. The most common formalization of graph transformation is the so-called algebraic approach, which uses category theory to express the rewriting (Ehrig et al. 2006). This approach supports a number of interesting analysis techniques, such as detecting rule dependencies (Ehrig et al. 2006) or calculating critical pairs (minimal context of pairs of conflicting rules) (Heckel et al. 2002).

However, graph transformation offers limited support for other kinds of analyses, such as reachability analysis, model checking, etc. This is why some authors define semantic mappings between graph transformation and other semantic domains, and then back-annotate the analysis results to the source notation (Cabot et al. 2008; de Lara & Vangheluwe 2004, 2008; Heckel et al. 2002). This possibility allows one to use the techniques specific to the target semantic domain for analyzing the source models. For example, in (Baresi & Spoletini 2006) rules are translated into Alloy in order to study the applicability of sequences of rules and the reachability of models; in (Baldan et al. 2001) rules are translated into Petri nets to check safety properties; in (Varró 2004) they are transformed into Promela for model-checking; and in (Büttner & Gogolla 2006; Cabot et al. 2008) rules are transformed into OCL pre- and postconditions for rule analysis (e.g., conflict detection) using standard OCL tools.

One of the problems of these approaches is due to the fact that they require, from the DSL designer, deep knowledge of the target language in order to specify the transformations. However, this problem can be partially overcome if the transformations can be automated, using, e.g., model transformation techniques. For example, in (de Lara & Vangheluwe 2008) the authors are able to generate the transformations from rule-based Domain-Specific Visual Languages (DSVLs) into semantic domains with an explicit notion of transition, such as Place-Transition Petri Nets. The generated transformation is expressed in the form of operational triple graph grammar rules that transform the static information (initial model) and the dynamics (source rules and their execution control structure). Similarly, in (Cabot et al. 2008) the authors describe how graph transformation rules can be mapped into OCL pre- and postconditions for rule analysis. However, as the own authors describe in their paper, not all kinds of graph transformations can be automatically transformed into OCL, and the analyses that can be conducted are somehow limited.

Other approaches use UML behavioral models to represent the system dynamics. For example, in (Engels et al. 2000) operational semantics are represented using UML collaboration diagrams, which are then formalized into graph transformation rules. In (Fischer et al. 1998), *Story Diagrams* are presented as a new graph rewrite language based on UML and Java. More precisely, the authors propose the use of UML together with pieces of Java code to express a graph rewrite language mixed with object-oriented data concepts. The whole specification (including dynamic behavior) is then transformed into Java classes and methods, although not all kinds of story patterns can be translated automatically. Again, the kind of analysis is limited in these approaches.

Kermeta (Muller et al. 2005) is an extension of EMOF for specifying operational semantics. It enriches the EMOF metamodel with an action specification metamodel, introducing another new language to express specifications of algorithms. Simulation and execution possibilities are available for this approach.

Other works propose model transformation languages to specify the semantics of DSLs. For example, in (Markovic & Baar 2008), QVT is proposed to specify the semantics of OCL. Then, QVT rules are specified between models conforming to the same metamodel, representing in such a way the metamodel behavior, thus acquiring in-place transformations semantics (as graph transformations have). Analysis capabilities are not provided in this case. The MOMENT-QVT tool (Boronat et al. 2006) is a model transformation engine that provides partial support for the QVT Relations language. It is based on an algebraically defined operator, ModelGen, which permits the definition of directed declarative transformations.

Another interesting approach for defining semantic mappings in order to specify the semantics of a language is the semantic anchoring method (Chen et al. 2005). *Semantic anchoring* relies on the use of well-defined "semantic units" of simple,

well-understood constructs and on the use of model transformations that map higher level modeling constructs into configured semantic units. This approach uses Abstract State Machines as a semantic framework, and Microsoft's Abstract State Machine Language (AsmL) and associated tools for programming, simulating and model checking ASM models (Chen et al. 2005). One problem with this kind of approaches is that they normally force the introduction of yet another notation for specifying the mappings (Narayanan & Karsai 2006), or require the mappings to be proved to be correct. By using a single semantic domain (such as our proposal with Maude), we avoid the need of defining the so-called semantic mappings.

Related to these proposals, other works also try formalize the concepts of concrete metamodeling languages, such as UML or OCL, in Maude. In fact, Maude offers several options to represent and manipulate object-oriented systems, depending on the way in which objects, attributes and references are represented; whether reflection is used or not, etc. For example, MOVA (Clavel, Egea, & da Silva 2007) is another Maude-based modeling framework for UML, which provides support for OCL constraint validation, OCL query evaluation, and OCL-based metrication. In MOVA, both UML class and object diagrams are formalized as MEL theories. MOMENT (Boronat et al. 2005) is a generic model management framework which uses Maude modules to automatically serialize software artifacts. It supports OCL queries and is also integrated in Eclipse.

In (Romero et al. 2007), Romero et al. presented a proposal for representing and manipulating models based on the use of Maude, which not only was expressive enough for these purposes, but also offered good tool support for operating with models. Specifically, it showed how some basic operations on models, such as model subtyping, type inference, and metric evaluation, can be easily specified and implemented in Maude, and made available in development environments such as Eclipse.

There is a huge amount of work ahead. We believe that all these works, from which we have inspired, and on which we have built our own contributions, represent a good basis for the development of tools, but also for the understanding of the field, the different techniques to apply, and its targets.

Acknowledgments. The author would like to thank A. Vallecillo for his friendship and long-term collaboration on the matters of this work and other related and no-so-related issues. I would also like to thank the reviewers for carefully reading the manuscript; their comments have been of great help in improving its quality and clarity. F. Durán has been partially supported by projects PGC2018-094905-B-100 and UMA18-FEDERJA-180, and by Universidad de Málaga, Campus de Excelencia Internacional Andalucía Tech.

References

AlTurki, M., & Meseguer, J. (2011). PVeStA: A parallel statistical model checking and quantitative analysis tool. In A. Corradini, B. Klin, & C. Cîrstea (Eds.), *Algebra and coalgebra in computer science - 4th international conference, CALCO 2011. proceedings* (Vol. 6859, pp. 386–392). Springer. doi: 10.1007/978-3-642-22944-2_28

Atkinson, C., & Gerbig, R. (2016). Flexible deep modeling with Melanee. In S. Betz & U. Reimer (Eds.), *Modellierung 2016, 2.-4. märz 2016, karlsruhe - workshopband* (Vol. P-255, pp. 117–122). GI. Retrieved from https://dl.gi.de/20.500.12116/843

Atkinson, C., Gerbig, R., & Metzger, N. (2015). On the execution of deep models. In T. Mayerhofer, P. Langer, E. Seidewitz, & J. Gray (Eds.), *Proceedings of the 1st international workshop on executable modeling co-located with ACM/IEEE 18th international conference on model driven engineering languages and systems (MODELS 2015)* (Vol. 1560, pp. 28–33). CEUR-WS.org. Retrieved from http://ceur-ws.org/Vol-1560/paper5.pdf

Atkinson, C., & Kühne, T. (2001). Processes and products in a multi-level metamodeling architecture. *Int. J. Softw. Eng. Knowl. Eng.*, *11*(6), 761–783. doi: 10.1142/S0218194001000724

Atkinson, C., & Kühne, T. (2008). Reducing accidental complexity in domain models. *Softw. Syst. Model.*, *7*(3), 345–359. doi: 10.1007/s10270-007-0061-0

Baldan, P., Corradini, A., & König, B. (2001). A static analysis technique for graph transformation systems. In *CONCUR 2001 - concurrency theory, 12th international conference, proceedings* (Vol. 2154, pp. 381–395). Springer. doi: 10.1007/3-540-44685-0_26

Baresi, L., & Spoletini, P. (2006). On the use of Alloy to analyze graph transformation systems. In *Graph transformations, third international conference, ICGT 2006, proceedings* (Vol. 4178, pp. 306–320). Springer. doi: 10.1007/11841883_22

Bengtsson, J., Larsen, K. G., Larsson, F., Pettersson, P., & Yi, W. (1995). UPPAAL - a tool suite for automatic verification of real-time systems. In *Hybrid systems* (Vol. 1066, pp. 232–243). Springer.

Boronat, A., Carsí, J. A., & Ramos, I. (2005). Automatic support for traceability in a generic model management framework. In *Model driven architecture - foundations and applications, 1st european conference, ECMDA-FA 2005, proceedings* (Vol. 3748, pp. 316–330). Springer. doi: 10.1007/11581741_23

Boronat, A., Carsí, J. A., & Ramos, I. (2006). Algebraic specification of a model transformation engine. In *Fundamental approaches to software engineering, 9th international conference, FASE 2006, proceedings* (Vol. 3922, pp. 262–277). Springer. doi: 10.1007/11693017_20

Brambilla, M., Cabot, J., & Wimmer, M. (2012). *Model-driven software engineering in practice*. Morgan & Claypool Publishers. doi: 10.2200/S00441ED1V01Y201208SWE001

Bulychev, P. E., David, A., Larsen, K. G., Mikucionis, M., Poulsen, D. B., Legay, A., & Wang, Z. (2012). UPPAAL-SMC: statistical model checking for priced timed automata. In *Proceedings 10th workshop on quantitative aspects of programming languages and systems, QAPL 2012* (Vol. 85, pp. 1–16). doi: 10.4204/EPTCS.85.1

Büttner, F., & Gogolla, M. (2006). Realizing graph transformations by pre- and postconditions and command sequences. In *Graph transformations, third international conference, ICGT 2006, proceedings* (Vol. 4178, pp. 398–413). Springer. doi: 10.1007/11841883_28

Cabot, J., Clarisó, R., Guerra, E., & de Lara, J. (2008). Analysing graph transformation rules through OCL. In *Theory and practice of model transformations - 1st international conference, icmt@tools 2008, proceedings* (Vol. 5063, pp. 229–244). Springer. doi: 10.1007/978-3-540-69927-9_16

Chen, K., Sztipanovits, J., Abdelwahed, S., & Jackson, E. K. (2005). Semantic anchoring with model transformations. In *Model driven architecture - foundations and applications, 1st european conference, ECMDA-FA 2005, proceedings* (Vol. 3748, pp. 115–129). Springer. doi: 10.1007/11581741_10

Clavel, M., Durán, F., et al. (Eds.). (2007). *All about Maude - A high-performance logical framework, how to specify, program and verify systems in rewriting logic* (Vol. 4350). Springer.

Clavel, M., Egea, M., & da Silva, V. T. (2007). The MOVA tool: A rewriting-based UML modeling, measuring, and validation tool. In *XII jornadas de ingeniería del software y bases de datos (JISBD 2007), actas* (pp. 393–394). Thomson Editorial.

Clavel, M., Palomino, M., & Riesco, A. (2006). Introducing the ITP tool: a tutorial. *J. Univers. Comput. Sci.*, *12*(11), 1618–1650.

Cuccuru, A., Mraidha, C., Terrier, F., & Gérard, S. (2007). Enhancing UML extensions with operational semantics. In *Model driven engineering languages and systems, 10th international conference, models 2007, proceedings* (Vol. 4735, pp. 271–285). Springer. doi: 10.1007/978-3-540-75209-7_19

Czarnecki, K., & Helsen, S. (2003). *Classification of model transformation approaches.*

de Lara, J., & Guerra, E. (2010a). Deep meta-modelling with MetaDepth. In J. Vitek (Ed.), *Objects, models, components, patterns, 48th international conference, TOOLS 2010, proceedings* (Vol. 6141, pp. 1–20). Springer. doi: 10.1007/978-3-642-13953-6_1

de Lara, J., & Guerra, E. (2010b). Generic meta-modelling with concepts, templates and mixin layers. In D. C. Petriu, N. Rouquette, & Ø. Haugen (Eds.), *Model driven engineering languages and systems - 13th international conference, MODELS 2010, proceedings, part I* (Vol. 6394, pp. 16–30). Springer. doi: 10.1007/978-3-642-16145-2_2

de Lara, J., Guerra, E., & Cuadrado, J. S. (2014). When and how to use multilevel modelling. *ACM Trans. Softw. Eng. Methodol.*, *24*(2), 12:1–12:46. doi: 10.1145/2685615

de Lara, J., & Vangheluwe, H. (2002). AToM^3: A tool for multi-formalism and meta-modelling. In R. Kutsche & H. Weber (Eds.), *Fundamental approaches to software engineering, 5th international conference, FASE 2002, proceedings* (Vol. 2306, pp. 174–188). Springer. doi: 10.1007/3-540-45923-5_12

de Lara, J., & Vangheluwe, H. (2004). Defining visual notations and their manipulation through meta-modelling and graph transformation. *J. Vis. Lang. Comput.*, *15*(3-4), 309–330. doi: 10.1016/j.jvlc.2004.01.005

de Lara, J., & Vangheluwe, H. (2008). Translating model simulators to analysis models. In J. L. Fiadeiro & P. Inverardi (Eds.), *Fundamental approaches to software engineering, 11th international conference, FASE 2008, proceedings* (Vol. 4961, pp. 77–92). Springer. doi: 10.1007/978-3-540-78743-3_6

de Oliveira, P. A., Durán, F., & Pimentel, E. (2021). A procedural and flexible approach for specification, modeling, definition, and analysis for self-adaptive systems. *Softw. Pract. Exp.*, *51*(6), 1387–1415.

de Oliveira, P. A., Moreno-Delgado, A., Durán, F., & Pimentel, E. (2017). Towards the predictive analysis of cloud systems with e-Motions. In *Cibse* (pp. 169–182). Curran Associates.

Durán, F., Eker, S., Escobar, S., Martí-Oliet, N., Meseguer, J., Rubio, R., & Talcott, C. L. (2020). Programming and symbolic computation in Maude. *J. Log. Algebraic Methods Program.*, *110*.

Durán, F., Lucas, S., Marché, C., Meseguer, J., & Urbain, X. (2008). Proving operational termination of membership equational programs. *High. Order Symb. Comput.*, *21*(1-2), 59–88.

Durán, F., Lucas, S., & Meseguer, J. (2008). MTT: the Maude termination tool (system description). In *IJCAR* (Vol. 5195, pp. 313–319). Springer.

Durán, F., Lucas, S., & Meseguer, J. (2009). Termination modulo combinations of equational theories. In *Frocos* (Vol. 5749, pp. 246–262). Springer.

Durán, F., & Meseguer, J. (2012). On the Church-Rosser and coherence properties of conditional order-sorted rewrite theories. *J. Log. Algebraic Methods Program.*, *81*(7-8), 816–850.

Durán, F., Meseguer, J., & Rocha, C. (2020). Ground confluence of order-sorted conditional specifications modulo axioms. *J. Log. Algebraic Methods Program.*, *111*, 100513.

Durán, F., Moreno-Delgado, A., & Álvarez-Palomo, J. M. (2016). Statistical model checking of e-Motions domain-specific modeling languages. In P. Stevens & A. Wasowski (Eds.), *Fundamental approaches to software engineering - 19th international conference, FASE 2016, proceedings* (Vol. 9633, pp. 305–322). Springer. doi: 10.1007/978-3-662-49665-7_18

Durán, F., Moreno-Delgado, A., Orejas, F., & Zschaler, S. (2017). Amalgamation of domain specific languages with behaviour. *J. Log. Algebraic Methods Program.*, *86*(1), 208–235. doi: 10.1016/j.jlamp.2015.09.005

Durán, F., Orejas, F., & Zschaler, S. (2012). Behaviour protection in modular rule-based system specifications. In N. Martí-Oliet & M. Palomino (Eds.), *Recent trends in algebraic development techniques, 21st international workshop, WADT 2012, revised selected papers* (Vol. 7841, pp. 24–49). Springer. doi: 10 .1007/978-3-642-37635-1_2

Durán, F., Rocha, C., & Álvarez, J. M. (2011a). Tool interoperability in the Maude formal environment. In *CALCO* (Vol. 6859, pp. 400–406). Springer.

Durán, F., Rocha, C., & Álvarez, J. M. (2011b). Towards a Maude Formal Environment. In *Formal modeling: Actors, open systems, biological systems* (Vol. 7000, pp. 329–351). Springer.

Durán, F., & Rodríguez, A. (2021). Towards a Maude-based implementation of MultEcore multilevel modelling languages. In *Prole2021*. SISTEDES. Retrieved from http://hdl.handle.net/11705/PROLE/2021/020

Durán, F., Zschaler, S., & Troya, J. (2012). On the reusable specification of non-functional properties in DSLs. In K. Czarnecki & G. Hedin (Eds.), *Software language engineering, 5th international conference, SLE 2012, revised selected papers* (Vol. 7745, pp. 332–351). Springer. doi: 10.1007/978-3-642-36089-3_19

Ehrig, H., Ehrig, K., Prange, U., & Taentzer, G. (2006). *Fundamentals of algebraic graph transformation*. Springer. doi: 10.1007/3-540-31188-2

Engels, G., Hausmann, J. H., Heckel, R., & Sauer, S. (2000). Dynamic meta modeling: A graphical approach to the operational semantics of behavioral diagrams in UML. In *«uml» 2000 - the unified modeling language, advancing the standard, third international conference, proceedings* (Vol. 1939, pp. 323–337). Springer. doi: 10.1007/3-540-40011-7_23

Fischer, T., Niere, J., Torunski, L., & Zündorf, A. (1998). Story diagrams: A new graph rewrite language based on the unified modeling language and Java. In *Theory and application of graph transformations, 6th international workshop, tagt'98, selected papers* (Vol. 1764, pp. 296–309). Springer. doi: 10.1007/978-3-540-46464-8_21

Harel, D., & Rumpe, B. (2004). Meaningful modeling: What's the semantics of "semantics"? *Computer*, *37*(10), 64–72. doi: 10.1109/MC.2004.172

Heckel, R., Küster, J. M., & Taentzer, G. (2002). Confluence of typed attributed graph transformation systems. In *Graph transformation, first international conference, ICGT 2002, proceedings* (Vol. 2505, pp. 161–176). Springer. doi: 10.1007/3-540-45832-8_14

Hinton, A., Kwiatkowska, M. Z., Norman, G., & Parker, D. (2006). PRISM: A tool for automatic verification of probabilistic systems. In *TACAS* (Vol. 3920, pp. 441–444). Springer.

Jansen, D. N., Katoen, J., Oldenkamp, M., Stoelinga, M., & Zapreev, I. S. (2007). How fast and fat is your probabilistic model checker? an experimental performance comparison. In *Haifa verification conference* (Vol. 4899, pp. 69–85). Springer.

Kelly, S., & Tolvanen, J. (2008). *Domain-specific modeling - enabling full code generation*. Wiley.

Kent, S. (2002). Model driven engineering. In M. J. Butler, L. Petre, & K. Sere (Eds.), *Integrated formal methods, third international conference, IFM 2002, proceedings* (Vol. 2335, pp. 286–298). Springer. doi: 10.1007/3-540-47884-1_16

Kwiatkowska, M. Z., Norman, G., & Parker, D. (2011). PRISM 4.0: Verification of probabilistic real-time systems. In *Computer aided verification - 23rd international conference, CAV 2011, proceedings* (Vol. 6806, pp. 585–591). Springer. doi: 10.1007/978-3-642-22110-1_47

Macías, F., Rutle, A., & Stolz, V. (2016). Multecore: Combining the best of fixed-level and multilevel metamodelling. In *Multi@models* (Vol. 1722, pp. 66–75). CEUR-WS.org.

Macías, F., Rutle, A., & Stolz, V. (2017). Multilevel modelling with multecore: A contribution to the MULTI 2017 challenge. In *Proceedings of MODELS 2017 satellite events* (Vol. 2019, pp. 269–273). CEUR-WS.org. Retrieved from http://ceur-ws.org/Vol-2019/multi_9.pdf

Macías, F., Rutle, A., Stolz, V., Rodríguez-Echeverría, R., & Wolter, U. (2018). An approach to flexible multilevel modelling. *Enterp. Model. Inf. Syst. Archit. Int. J. Concept. Model.*, *13*, 10:1–10:35.

Macías, F., Wolter, U., Rutle, A., Durán, F., & Rodríguez-Echeverría, R. (2019). Multilevel coupled model transformations for precise and reusable definition of model behaviour. *J. Log. Algebraic Methods Program.*, *106*, 167–195.

Markovic, S., & Baar, T. (2008). Semantics of OCL specified with QVT. *Softw. Syst. Model.*, *7*(4), 399–422. doi: 10.1007/s10270-008-0083-2

Meseguer, J. (2008). The temporal logic of rewriting: A gentle introduction. In *Concurrency, graphs and models* (Vol. 5065, pp. 354–382). Springer.

Meseguer, J., Palomino, M., & Martí-Oliet, N. (2008). Equational abstractions. *Theor. Comput. Sci.*, *403*(2-3), 239–264.

Meta Object Facility (MOF) spec. 2.5.1. (2016, October). https://www.omg.org/spec/MOF.

Moreno-Delgado, A., & Durán, F. (2014). The movie database case: A solution using the maude-based e-Motions tool. In *Ttc@staf* (Vol. 1305, pp. 116–124). CEUR-WS.org.

Moreno-Delgado, A., Durán, F., Zschaler, S., & Troya, J. (2014). Modular DSLs for flexible analysis: An e-Motions reimplementation of Palladio. In *Modelling foundations and applications - 10th european conference, ecmfa@staf 2014, proceedings* (Vol. 8569, pp. 132–147). Springer. doi: 10.1007/978-3-319-09195-2_9

Muller, P., Fleurey, F., & Jézéquel, J. (2005). Weaving executability into object-oriented meta-languages. In *Model driven engineering languages and systems, 8th international conference, models 2005, proceedings* (Vol. 3713, pp. 264–278). Springer. doi: 10.1007/11557432_19

Narayanan, A., & Karsai, G. (2006). Using semantic anchoring to verify behavior preservation in graph transformations. *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.*, *4*. doi: 10.14279/tuj.eceasst.4.22

Ölveczky, P. C., & Meseguer, J. (2007). Semantics and pragmatics of Real-Time Maude. *High. Order Symb. Comput.*, *20*(1-2), 161–196.

Rensink, A. (2003). The GROOVE simulator: A tool for state space generation. In J. L. Pfaltz, M. Nagl, & B. Böhlen (Eds.), *Applications of graph transformations with industrial relevance, second international workshop, AGTIVE 2003, revised selected and invited papers* (Vol. 3062, pp. 479–485). Springer. doi: 10.1007/978-3-540-25959-6_40

Rivera, J. E., Durán, F., & Vallecillo, A. (2009a). Formal specification and analysis of domain specific models using Maude. *Simul.*, *85*(11-12), 778–792. doi: 10.1177/0037549709341635

Rivera, J. E., Durán, F., & Vallecillo, A. (2009b). A graphical approach for modeling time-dependent behavior of DSLs. In *IEEE symposium on visual languages and human-centric computing, VL/HCC 2009, proceedings* (pp. 51–55). IEEE Computer Society. doi: 10.1109/VLHCC.2009.5295300

Rivera, J. E., & Vallecillo, A. (2007). Adding behavior to models. In *EDOC* (pp. 169–180). IEEE Computer Society.

Rodríguez, A., Durán, F., & Kristensen, L. M. (2022). Simulation and analysis of MultEcore multilevel models based on rewriting logic. *Softw. Syst. Model.*, *21*(2), 561–586.

Rodríguez, A., Durán, F., Rutle, A., & Kristensen, L. M. (2019). Executing multilevel domain-specific models in Maude. *J. Object Technol.*, *18*(2), 4:1–21.

Rodríguez, A., & Macías, F. (2019). Multilevel modelling with MultEcore: A contribution to the MULTI process challenge. In *Models (companion)* (pp. 152–163). IEEE.

Rodríguez, A., Rutle, A., Durán, F., Kristensen, L. M., & Macías, F. (2018). Multilevel modelling of coloured Petri nets. In *Models (workshops)* (Vol. 2245, pp. 663–672). CEUR-WS.org.

Rodríguez, A., Rutle, A., Kristensen, L. M., & Durán, F. (2019). A foundation for the composition of multilevel domain-specific languages. In *22nd ACM/IEEE international conference on model driven engineering languages and systems companion, MODELS companion 2019* (pp. 88–97). IEEE. doi: 10.1109/MODELS-C.2019.00018

Roldán, M., & Durán, F. (2011). Dynamic validation of OCL constraints with mOdCL. *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.*, *44*.

Romero, J. R., Rivera, J. E., Durán, F., & Vallecillo, A. (2007). Formal and tool support for model driven engineering with Maude. *J. Object Technol.*, *6*(9), 187–207.

Rozenberg, G. (Ed.). (1997). *Handbook of graph grammars and computing by graph transformations, volume 1: Foundations*. World Scientific.

Schmidt, Á., & Varró, D. (2003). CheckVML: A tool for model checking visual modeling languages. In P. Stevens, J. Whittle, & G. Booch (Eds.), *«uml» 2003 - the unified modeling language, modeling languages and applications, 6th international conference, proceedings* (Vol. 2863, pp. 92–95). Springer. doi: 10.1007/978-3-540-45221-8_8

Sen, K., Viswanathan, M., & Agha, G. A. (2005). VESTA: A statistical model-checker and analyzer for probabilistic systems. In *Second international conference on the quantitative evaluaiton of systems (QEST 2005)* (pp. 251–252). IEEE Computer Society. doi: 10.1109/QEST.2005.42

Taentzer, G. (1999). AGG: A tool environment for algebraic graph transformation. In M. Nagl, A. Schürr, & M. Münch (Eds.), *Applications of graph transformations with industrial relevance, international workshop, agtive'99, proceedings* (Vol. 1779, pp. 481–488). Springer. doi: 10.1007/3-540-45104-8_41

Troya, J., Vallecillo, A., Durán, F., & Zschaler, S. (2013). Model-driven performance analysis of rule-based domain specific visual models. *Inf. Softw. Technol.*, *55*(1), 88–110.

Varró, D. (2004). Automated formal verification of visual modeling languages by model checking. *Softw. Syst. Model.*, *3*(2), 85–113. doi: 10.1007/s10270-003-0050-x

Wolter, U., Macías, F., & Rutle, A. (2020). Multilevel typed graph transformations. In *ICGT* (Vol. 12150, pp. 163–182). Springer.

Younes, H. L. S. (2005). Ymer: A statistical model checker. In K. Etessami & S. K. Rajamani (Eds.), *Computer aided verification, 17th international conference, CAV 2005, proceedings* (Vol. 3576, pp. 429–433). Springer. doi: 10.1007/11513988_43

Zschaler, S., & Durán, F. (2017). GTS families for the flexible composition of graph transformation systems. In *Fundamental approaches to software engineering - 20th international conference, FASE 2017, proceedings* (Vol. 10202, pp. 208–225). Springer. doi: 10.1007/978-3-662-54494-5_12

Zschaler, S., & Durán, F. (2021). GTSMorpher: Safely composing behavioural analyses using structured operational semantics. In R. Heinrich, F. Durán, C. L. Talcott, & S. Zschaler (Eds.), *Composing model-based analysis tools* (pp. 189–215). Springer. doi: 10.1007/978-3-030-81915-6_9

About the authors

Francisco Durán is a Full Professor at the Department of Computer Science of the University of Málaga, Spain. Before joining the University of Málaga he was an International Fellow at SRI International, CA, USA. He is one of the developers of the Maude system, and his research interests deal with the verification of systems, the termination, confluence and coherence of rewrite and graph systems, module and model composition, application of formal methods to software engineering, including topics, such as cloud systems, model-driven engineering, component-based software development, open distributed programming, reflection and meta-programming. He is the co-author of about 100 publications in international conferences and journals.