Statistical Model Checking of e-Motions Domain-Specific Modeling Languages

Francisco Durán, Antonio Moreno-Delgado, and José M. Álvarez-Palomo

E.T.S.I. Informática
University of Málaga, Spain
{duran,amoreno,alvarezp}@lcc.uma.es

Abstract. Domain experts may use novel tools that allow them to design and model their systems in a notation very close to the domain problem. However, the use of tools for the statistical analysis of stochastic systems requires software engineers to carefully specify such systems in low level and specific languages. In this work we line up both scenarios, specific domain modeling and statistical analysis. Specifically, we have extended the e-Motions system, a framework to develop real-time domain-specific languages where the behavior is specified in a natural way by in-place transformation rules, to support the statistical analysis of systems defined using it. We discuss how restricted e-Motions systems are used to produce Maude corresponding specifications, using a model transformation from e-Motions to Maude, which comply with the restrictions of the VeStA tool, and which can therefore be used to perform statistical analysis on the stochastic systems thus generated. We illustrate our approach with a very simple messaging distributed system.

1 Introduction

Model Driven Engineering advocates the use of models as the key artifacts in all phases of development, artifacts from which whole systems can be derived, analysed and implemented [21]. To be able to define such models in terms 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., [20]). 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.

DSMLs are typically defined by means of its structural aspects (with its 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 or browsers. Typically, to perform some type of analysis or to generate code, such models need to be transformed into formalisms or programming languages with the appropriate tool support. There are many success stories using this approach. 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) [33], with systems such as AToM3 [13], AGG [37] or e-Motions [29] implementing it.

The specification of the explicit behavioral semantics of DSMLs helps in MDE 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., [37] and [31]). CheckVML [34], GROOVE [28] and e-Motions [29] support the model checking of systems whose behavior is specified by graph transformation systems.

This is, however, not enough, since applications become more and more complex, and model checking is a very expensive procedure, both in time and space, being 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 [19], that iteratively compute the exact (or approximate) measure of paths satisfying relevant logical formulas. Although tools like PRISM [22] and UPPAAL [4] 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" are evaluable. YMER [39] and VeStA [36] were pioneering tools implementing these techniques. Latest releases of the well-established tools PRISM and UPPAAL have more recently also included capabilities for statistical model checking (see [23] and [9]).

Statistical methods has 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 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.

In this paper we describe how the e-Motions tool has been extended so the models built conforming to user-defined DSMLs are suitable for statistical model checking. e-Motions models are transformed into Maude specifications satisfying the requirements of the PVeStA tool [3] (an extension and parallelization of VeStA [36]). Such Maude specifications are therefore suitable to be stochastically analyzed using PVeStA. We illustrate the use of e-Motions to model systems and its statistical model checker with a very simple messaging system.

The remaining of the paper is structured as follows. Section 2 presents e-Motions and VeStA/PVeStA, and their underlying Maude system. Section 3 explains how e-Motions specifications are statistically analysed using PVeStA and how the connection between these two systems is established. The way systems are defined in e-Motions and how they can be statistically analysed is illustrated with a case study in Section 4. Section 5 discusses some related work and Section 6 wraps up presenting some conclusions and future work.

2 Preliminaries

In this section, we introduce the e-Motions language and tool, the Maude system and the Maude implementation of e-Motions, and the VeStA/PVeStA tool.

2.1 The e-Motions System

e-Motions [29] is a graphical language and framework that supports the specification, simulation, and formal analysis of real-time systems. It supports the graphical specification of the dynamic behavior of DSMLs using their concrete syntax, making this task very intuitive.[1] The abstract syntax of a DSML is specified as an Ecore metamodel, which defines all relevant concepts—and their relations—in the language. Its concrete syntax is given by a GCS (Graphical Concrete Syntax) model, which attaches an image to each language concept. Then, its behavior is specified with (graphical) in-place model transformations. e-Motions provides a model of time, supporting features like duration, periodicity, etc., and mechanisms to state action properties [29, 30].

In-place transformations are defined by rules, each of which represents a possible *action* of the system. These rules are of the form $[\mathrm{NAC}]^* \times \mathrm{LHS} \to \mathrm{RHS}$, where LHS (left-hand side), RHS (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 conditions for the rule to be applied, whereas the RHS represents the effect of the corresponding action. A LHS may also have positive conditions, which are expressed, as any expression in the RHS, using OCL. Thus, a rule can be applied, i.e., triggered, if a match of the LHS is found in the model, its conditions are satisfied, and none of its NAC patterns occur. If several matches are found, one of them is non-deterministically chosen and applied, giving place to a new model where the matching objects are substituted by the appropriate instantiation of its RHS pattern. The transformation of the model proceeds by applying the rules on sub-models of it in a non-deterministic order, until no further transformation rule is applicable.

[1] e-Motions got an "ease of use" award at the 7th Transformation Tool Contest [26].

In e-Motions, there are two types of rules to specify time-dependent behavior, namely, *atomic* and *on-going* rules. Atomic rules represent atomic actions with a duration, which is specified by an interval of time. Atomic rules with duration zero are called *instantaneous* rules. On the other hand, on-going rules represent continuous actions that may be interrupted at any time.

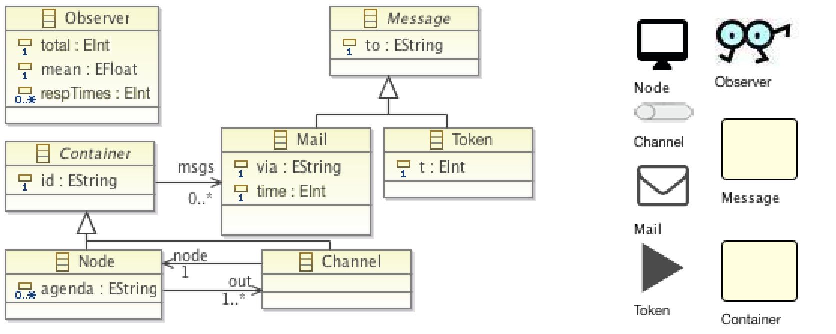
A special kind of object, named *Clock*, represents the current global time elapse. Designers can use it in their rules (using its attribute *time*) to know the amount of time that the system has been working.

Figure 1 shows the metamodel and concrete syntax for a very simple messaging system, where there are nodes interconnected via channels. Each node has an agenda (a set with the identifiers of the other nodes in the net), and may deliver messages to any other node in it. There are two types of messages in the system, namely `Token` and `Message`. Figure 2 shows a sample initial configuration conforming to the metamodel in Figure 1a and using the concrete syntax in Figure 1b. Figure 3 shows the atomic rules defining the possible actions that may happen in such systems. The `NewMessage` rule states that every time a node receives a token message with time zero, a new message is created addressed to another node chosen from the agenda following a uniform distribution, and will be sent through an outgoing channel also chosen probabilistically—`WITH` blocks state positive conditions that have to be hold on a given match of the LHS for the rule to be triggered. `Mail` objects will be moved from nodes to channels and from channels to nodes by rules `Node2Channel` and `Channel2Node`, respectively, both with a duration that follows a normal distribution (see the definition of variable `STime` at the bottom of the rule and its use in the header to establish the duration). The `MessageArrival` and `DecreaseToken` rules model, respectively, the arrival of a message to its destination node, where the observer gathers information about the time taken, and the pass of time for the token messages. `NewMessage` and `MessageArrival` are modelled as instantaneous actions (duration `[0,0]`). `Node2Channel`, `Channel2Node` and `DecreaseToken` have probabilistic durations, whilst the duration of the first two are calculated in the rule itself, for the third one the duration is given by the attribute `t` of the token, whose value was assigned in a previous `NewMessage` rule.

2.2 Maude

Maude [10, 11] is an executable formal specification language based on rewriting logic [24], a logic of change that can naturally deal with states and non-deterministic concurrent computations. A rewrite logic theory is a tuple (Σ, E, R), where (Σ, E) is an *equational theory* that specifies the system states as elements of the initial algebra $\mathcal{T}_{(\Sigma,E)}$, and R is a set of rewrite rules that describe the one-step possible concurrent transitions in the system.

Rewriting will operate on congruence classes of terms modulo E. This of course does not mean that an implementation of rewriting logic must have an E-matching algorithm for each equational theory E that a user might specify. The equations are divided into a set A of structural axioms for which matching algorithms are available and a set E of equations. Then, for having a complete



(a) Metamodel (b) Concrete syntax

Fig. 1: Metamodel and concrete syntax for the messaging system

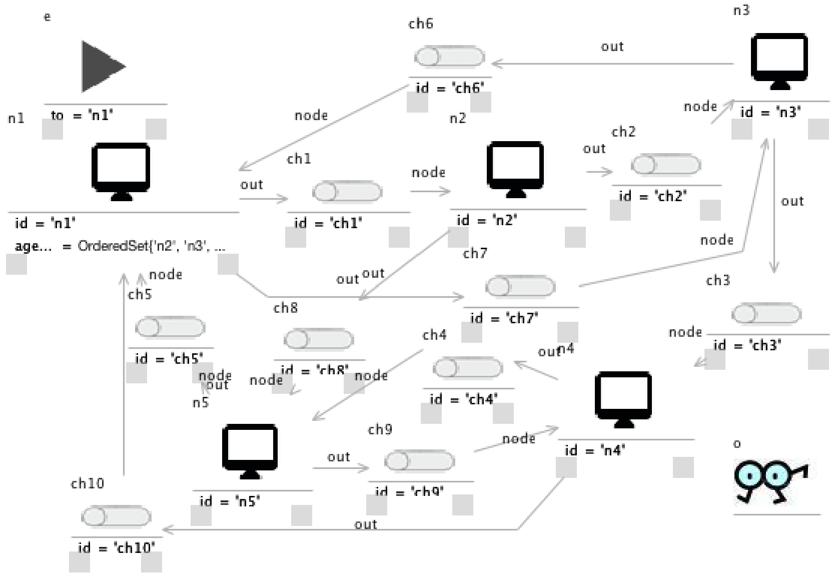


Fig. 2: Messaging system's initial configuration

agreement between the specification's initial algebra and its operational semantics by rewriting, a rewrite theory $(\Sigma, E \cup A, R)$ is assumed to be such that the set E of equations is (ground) Church-Rosser and terminating modulo A, and the rules R are (ground) coherent with the equations E modulo A (see [14, 15]).

In the case of Maude, the equational logic is membership equational logic (MEL) [7], which can be seen as an extension of order-sorted logic with sorts, subsorts, and partial functions, and where atomic sentences include both equations $t = t'$ and memberships $t : s$, stating that term t has sort s. Maude provides support for rewriting modulo associativity, commutativity and identity, which

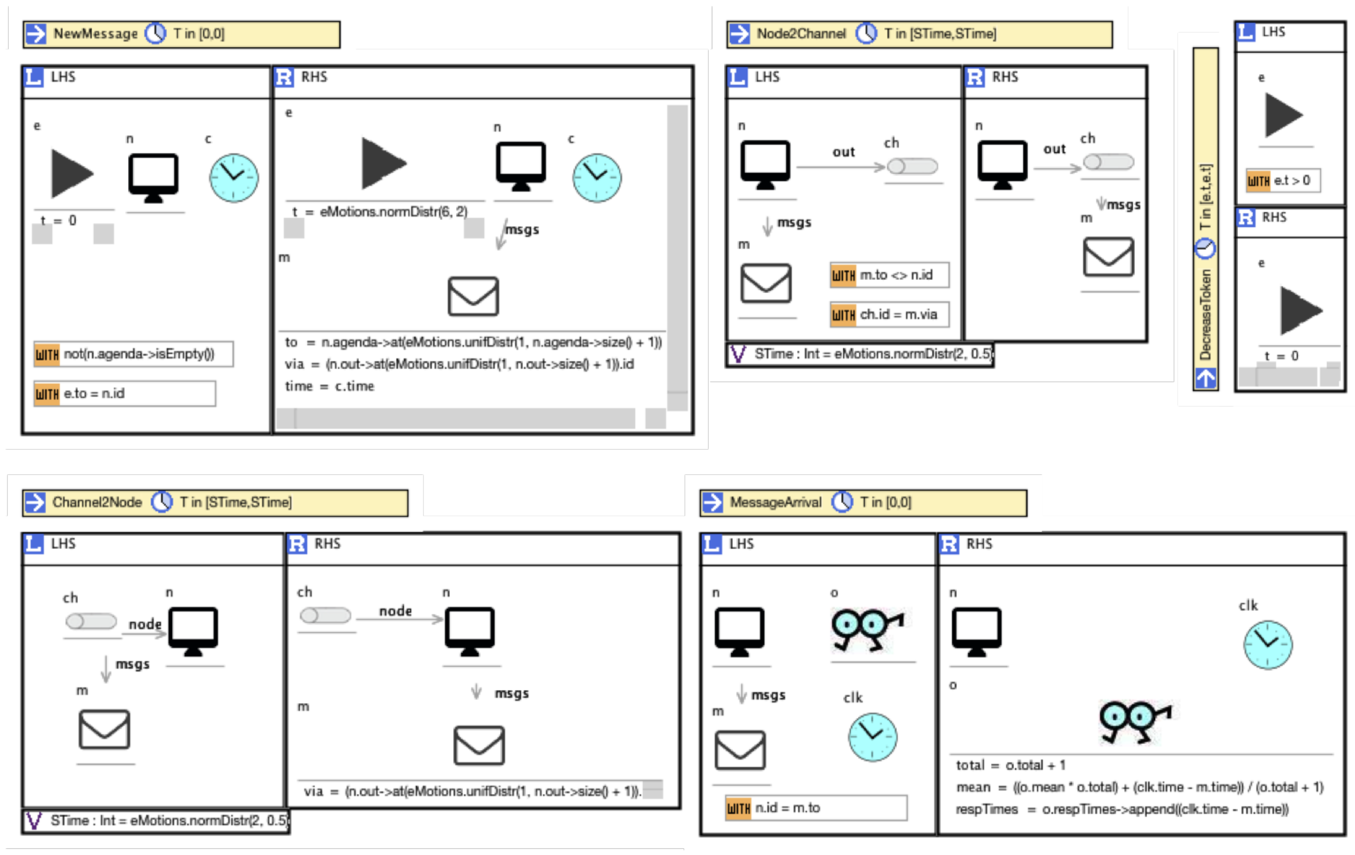

NewMessage T in [0,0]
LHS
e
n
c
t = 0
WITH not(n.agenda->isEmpty())
WITH e.to = n.id
RHS
e
n
c
t = eMotions.normDistr(6, 2)
msgs
m
to = n.agenda->at(eMotions.unifDistr(1, n.agenda->size() + 1))
via = (n.out->at(eMotions.unifDistr(1, n.out->size() + 1)).id
time = c.time
Node2Channel T in [STime,STime]
LHS
n
out
ch
msgs
m
WITH m.to <> n.id
WITH ch.id = m.via
RHS
n
out
ch
msgs
m
STime : Int = eMotions.normDistr(2, 0.5)
DecreaseToken T in [e.t,e.t]
LHS
e
WITH e.t > 0
RHS
e
t = 0
Channel2Node T in [STime,STime]
LHS
ch
node
n
msgs
m
RHS
ch
node
n
msgs
m
via = (n.out->at(eMotions.unifDistr(1, n.out->size() + 1)).
STime : Int = eMotions.normDistr(2, 0.5)
MessageArrival T in [0,0]
LHS
n
o
msgs
clk
m
WITH n.id = m.to
RHS
n
o
clk
total = o.total + 1
mean = ((o.mean * o.total) + (clk.time - m.time)) / (o.total + 1)
respTimes = o.respTimes->append((clk.time - m.time))

Fig. 3: Messaging system's rules

perfectly captures the evolution of models made up of objects linked by references as in graph grammar.

Maude counts with a rich set of validation and verification tools, increasingly used as support to the development of UML, MDA, and OCL tools (see, e.g., [32] for an overview). Furthermore, Maude has demonstrated to be a good environment for rapid prototyping, and also for application development (see [11]).

Among other applications, Maude may be seen as a general framework where to develop model transformations [6]. Maude is used as a formal notation to provide the precise semantics of the corresponding e-Motions specifications (as described in [30]), while at the same time the model transformations between e-Motions and Maude allow the Maude tools to become available in the e-Motions environment. More precisely, the generated Maude specification is a Real-Time Maude specification. Real-Time Maude [27] is a rewriting-logic-based specification language and formal analysis tool that extends the Maude system [11] to support the formal specification and analysis of *real-time systems*. Real-Time Maude provides a sort `Time` to model the time domain, which can be either discrete or dense. Then, pass of time is modelled with *tick rules* like

$$\texttt{crl [}l\texttt{] : \{ } t\texttt{, } T \texttt{ \} => \{ } t'\texttt{, } T+\tau \texttt{ \} if } C \texttt{ .}$$

where t and t' are system states (an evolving model in our case), T is the global time, and τ is a term of sort `Time` that denotes the *duration* of the rewrite, and that affects the *global time elapse*. Since tick rules affect the global time, in Real-Time Maude time elapse is usually modeled by one single tick rule, and the system dynamic behavior by instantaneous transitions [27]. Although there are other sampling strategies, in the most convenient one this single tick rule models time elapse by using two functions: the `delta` function, that defines the effect of time elapse over every model element, and the `mte` (maximal time elapse) function, that defines the maximum amount of time that can elapse before any action is performed. Then, time advances non-deterministically by any amount τ, which must be less or equal than the maximum time elapse of the system.

$$\texttt{crl [tick] : \{ } t\texttt{, } T \texttt{ \} => \{ delta(}t\texttt{, } \tau\texttt{), } T+\tau \texttt{ \} if } \tau \leq \texttt{mte(}t\texttt{)} \wedge C \texttt{ .}$$

2.3 Maude representation of e-Motions models and metamodels

As in [32, 5], the algebraic semantics of an Ecore[2] metamodel *MM* is provided by a MEL theory $\mathit{Spec}_{\mathit{MM}}$ so that a model M conformant with *MM* is an element of the initial algebra $\mathcal{T}_{\mathit{Spec}_{\mathit{MM}}}$. The e-Motions definition of a domain specific language provided by a metamodel *MM* plus a set of transformation rules defining its dynamic semantics, is then represented as a rewrite theory extending $\mathit{Spec}_{\mathit{MM}}$ with some additional definitions and rules specifying such a behavior.

An ATL transformation transforms e-Motions models into Maude executable specifications, which can be used for simulation and analysis. Although a detailed

[2] Ecore is equivalent to the EMOF (Essential MOF) portion of MOF defined in the MOF 2 specification [25].

presentation of this transformation can be found in [30], we give here a general account of it to understand the rest of the paper.

e-Motions' models are represented in Maude as structures of sort `@Model` of the form $mm\{obj_1\ obj_2\ ...\ obj_N\}$, where mm is the name of its metamodel and obj_i are the objects that constitute the model. An object is a record-like structure of the form $< o : c \mid a_1 : v_1\#...\#a_n : v_n >$ (of sort `@Object`), where o is the object identifier (of sort `Oid`), c is the class the object belongs to (of sort `@Class`), and $a_i : v_i$ are attribute-value pairs (of sort `@StructuralFeature-Instance`). Given appropriate definitions for all classes, attributes and references in its corresponding metamodel, a possible valid state could be as follows:

```
@smp-mm@ {
  < n1 : Node | id : "n1" # out : Set{"ch1", "ch2"} # agenda : ... >
  < ch1 : Channel | id : "ch1" # node : "n2" >
  ...
}
```

This code snippet shows part of a model in which there is a node object `"n1"` of class `Node` which is connected to channels `"ch1"` and `"ch2"`, which in turn are connected to nodes `"n2"` and `"n3"`.[3]

Although in e-Motions there are two kinds of rules, namely, atomic and ongoing rules, for the purpose of the work at hand only atomic rules are used. So in what follows, we sketch the Maude specification of the atomic e-Motions rules.

Atomic rules are represented as two Real-Time Maude instantaneous rules, one modeling its *triggering* and another one modeling its actual *realization. Triggering rules* represent the action's preconditions. When a rule precondition is satisfied, the triggering Maude rule is applied and an atomic action execution (AAE) object is created. AAE objects represent atomic rules' executions, each one acting as a countdown to the finalization of the action. AAE objects gather the information needed for its instantiation: the rule's name (l), the identifiers of the elements involved in the action (ρ), and the variables used in it (ν). Initially, the timer (τ) is set to the given duration of the rule.

$$\texttt{crl [}l\texttt{] : } \{t, T\} \texttt{ => } \{t, AAE(l, \rho, \nu, \tau), T\} \texttt{ if } C \texttt{ .}$$

The *realization rule* represents the postcondition of the rule, which can be performed once the action's timer is consumed, and only if none of the action's participants have been deleted by other actions. Then, the subterm matching the LHS is substituted by the corresponding instantiation of the RHS and the attribute values are computed.

$$\texttt{crl [}l\texttt{] : } \{t, AAE(l, \rho, \nu, 0), T\} \texttt{ => } \{t', T\} \texttt{ if } C \texttt{ .}$$

As above explained, time elapse is modeled by using the `delta` and `mte` functions. Both functions need to be defined only over time-dependent elements,

[3] In e-Motions, all structural features are qualified with the name of the class they belong to, and all elements are qualified with the name of the metamodel they are defined in. All these qualifications have been removed to improve readability.

namely the `Clock` instance and AAE objects. The `delta` function decreases AAE timers and increases the clock value. The rest of objects remain unchanged.

Action execution objects AAE gather additional information for dealing with scheduling, periodicity, etc. The interested reader is referred to [30] for a complete account on them and on the representation of on-going rules. From the point of view of executability by rewriting and, in particular, for the discussion on un-quantified non-determinism in the following sections the key idea is that AAE objects are required for the realization of actions.

2.4 The VeStA/PVeStA tool

There are two main approaches for statistical model checking: sequential testing [40], implemented, e.g., in Ymer [39], and black-box testing [35], implemented, e.g., in VeStA [36]. In sequential testing, sample execution paths are generated until its answer can be guaranteed to be correct within the required error bounds. In black-box testing, the system is not controlled to generate specific execution traces. Instead, a quantitative measure of *confidence* is computed for given samples.

VeStA [36] performs discrete-event simulation from a Maude specification by invoking the Maude interpreter. Given a Maude model, an initial state (or configuration) and a temporal logical formula expressed in QuaTeX [2], VeStA is used to perform stochastic analysis. QuaTeX uses real-valued states and path functions to quantitatively specify properties about probabilistic models. Specifically, QuaTeX provides an expressive language for real-valued temporal properties through the combination of recursive function declarations, an if-then-else construct, and a next operator. The reader is referred to [2] for a detailed account on QuaTeX. For the soundness of the analysis carried out in VeStA, the specification to be analyzed has to have absence of *un-quantified non-determinism* [35].

AlTurki et al. have extended the VeStA tool with a parallel implementation, PVeStA [3], which makes the analysis substantially more efficiently. VeStA and PVeStA have been used for the analysis of systems and algorithms by different authors (see, e.g., [1, 16, 8]).

3 PVeStA-compliant representation of e-Motions models

When rewriting a system, there might be different sources of non-determinism. Some of them are part of our specification, due to probabilistic choices and stochastic real-time. However, rewrite engines need to take their own choices. When there are several matches, for a given rule or for several rules, rewrite engines will choose an alternative following some internal criteria. For the statistical analysis used in VeStA/PVeStA to be sound the rewriting logic specification cannot contain un-quantified non-determinism [2].

The thus obtained specification may be used for rewriting in Maude, but other tools in the Maude formal environment, as its model checker or its reachability analysis tool, can also be used on it [31]. In section 3.2 we show how, by meeting its requirements, we can also use the PVeStA tool for carrying on statistical model checking.

3.1 Un-quantified-non-determinism-free e-Motions systems

Writing an arbitrary rewrite specification that meets the *un-quantified-non-determinism* free requirement is non-trivial. We could check whether a specification meets the requirement by performing a critical pair analysis and checking that there are no rules that can be applied simultaneously.[4] However, the checking would not be easy either. And although it may give us some hints on the sources of un-quantified non-determinism, we would still have to change the specification.

To avoid this problem, and to make easier to write a specification free from *un-quantified non-determinism*, Agha et al. propose in [2] the use of the *actor* model. To guarantee that only one rule can be fired at any time, messages are scheduled following a continuous probability distribution. To improve its executability, a centralized scheduler is used in [3], so that only one scheduled message or object is available for execution at any time. With this scheduler-based scheme, having a single message in the initial configuration, rules with one object and a message in its left-hand sides, and no two rules for the same message, are a sufficient condition to meet the requirement. Eckhardt et al. relaxed the requirements on systems in [16] by allowing nested configurations of objects. The basic idea is however the same one, if every rule is going to be fired by a message, this message determines the rule match, and there is only one message out of the scheduler at a time, there is only one rule that may be fired and in one possible way.

Given the direct transformation of e-Motions configurations of objects with references into Maude configurations of objects, we may use the same scheduler scheme with the following changes on the requirements:

- There is a distinguish class `Message` whose objects represent messages.
- Objects communicate through asynchronous message passing, avoiding direct synchronization among them. We allow several objects in the left-hand sides of rules, but only when they are related by a containment relation, and not to model communication between them.
- Message and action execution objects are scheduled so that there is only one of these objects out of the scheduler.
- In the initial configuration there is only one message object, and no action execution object. If there are more than one message objects, they have to be scheduled.
- Rules may be fired either by messages or by action execution objects *AAE*. Each rule has in its left-hand side either a message or an action execution object. There is no rule without one of these objects in its left-hand side.

[4] Critical pair analysis is available in Maude, and has been used for tools like its confluence and coherence checkers (see [15]).

– As in [3], there might be in the right-hand side of a rule any number of message and action execution objects, but only one may be non-scheduled. The rest must be scheduled so that only one remains in the under-execution configuration. When there are several messages in the left-hand side of a rule, the order of the messages is specified in the transformation.
– If there are two rules with the same message or action execution object in its LHS, they cannot be simultaneously applicable. This is a requirement often used in critical pair analysis (cf. [15]): if there is a critical pair between two rules, their conditions should not be simultaneously satisfiable.
– The duration intervals of all atomic rules must be of the form `[n,n]`. Intervals of the form `[n,m]` are a source of un-quantified no-determinism, since the actual duration of the corresponding action might be any value in that interval.

These requirements are a sufficient condition for the specification to meet the un-quantified-non-determinism-free condition. Given the direct transformation between e-Motions rules and Maude rules, these requirements can indeed be checked on the e-Motions model itself.

Our scheduler contains both messages and action execution objects, which are released one by one in every rewriting tick step. As in [3], the elements in the scheduler are ordered according to their scheduled time. Messages are always ahead of actions, as they are ready to be consumed as soon as they are generated by a realization rule. Those objects scheduled for the same time are served in accordance with the time they were inserted in the scheduler (FIFO). The order of *AAE* objects is determined by their timers, being the first action execution object the one with the smallest timer. If several action execution objects have the same countdown, they follow a FIFO order.

Thus, a Maude rule mapped from an e-Motions rule can be triggered by two reasons:

1. there is a message which matches the message of the left-hand side of an instantaneous rule or a triggering rule of a non-instantaneous rule, or
2. there is an *AAE* object whose countdown has reached zero.

Let us check these requirements on the example given in Section 2. The first observation is that there is a single message (`Mail` or `Token`) in the lefthand side of each rule. If we assume that the initial configuration has a single message (a `Token` object in our case), the scheduler will make sure that there is only one message at a time in the running configuration. Notice that `NewMessage` is the only rule that have two messages in its righthand side. In this case, the transformation generating the Maude specification will decide which one goes first in the scheduler. The other important observation is that there will never be two nodes referencing to the same `Mail` object. In those other cases in which there are possible overlaps, indicating that there may be more than one applicable rule, or multiple matches for the same rule, we can check that their conditions are not simultaneously satisfiable. See for example that with a `Token` object in the running configuration, there might be matches for rules `NewMessage` and

`DecreaseToken` at the same time. Notice however that `NewMessage` requires `e.t = 0` and `DecreaseToken` requires `e.t > 0`. There is a similar situation for rules `Node2Channel` and `MessageArrival`, in this case `m.to <> n.id` and `m.to = n.id` cannot be satisfied simultaneously.

3.2 Modifications of Maude rules

The Maude modules supporting the e-Motions infrastructure and the Maude rules mapped from the e-Motions rules have been modified to make use of the scheduler. Regarding the infrastructure modules, a new module defines the scheduler, operations to insert and remove objects from the scheduler, and extensions to the operations `delta` and `mte`. This module is independent from the systems and is added to the resulting Maude specification.

Every Maude rule mapped from an e-Motions atomic instantaneous rule must be modified by wrapping all the messages in its right-hand side with the operator `schedule`, which takes a list of one or more elements and insert them in the scheduler, following equations defined in the infrastructure module. For the Maude rules mapped from e-Motions non-instantaneous rules, there are more modifications. Messages present in the left-hand side of the triggering rule cannot be removed when the rule is executed, they must be available in the system because they are required for the corresponding realization action. However, they cannot stay free in the configuration because they could be chosen again. Therefore, they must be wrapped with the operator `blocked`, allowing to free another scheduled message from the scheduler. *AAE* objects created on the RHSs of rules have to be included within `schedule` operators to be handled by the scheduler. For the realization rules, messages that appear on its LHS must be wrapped with `blocked` operators, since the have to match with those wrapped in the triggering rule. Finally, those messages created in such rules have to be enclosed within `schedule` operators. The e-Motions scheduler releases a new message or *AAE* object if the current state of the system has no free message or *AAE* after a rewriting step.

In e-Motions systems time advances by means of the `tick` rule which, given the current state, computes the minimum among the maximum time elapses (MTEs) of the actions that may be performed on the current state. If that value is greater than zero, it means that there is no action that can be triggered or realized at that time. In that case, the global time is advanced until that value and the countdown values of all the *AAE* objects are updated according to that value. The operation `delta` makes that update. The operations `delta` and `mte` have been modified to take into account the elements contained in the scheduler.

3.3 e-SMC: e-Motions & PVeStA integration

A new extension for the e-Motions framework has been developed to allow automatic modifications of e-Motions specifications for them to hold the restrictions mentioned in Section 3.2. This extension is named e-SMC and it has been implemented as an Eclipse plugin, integrated with the e-Motions tool. e-SMC encapsulates all the process from the mapping from the e-Motions system to the Maude specification to the execution of PVeStA as statistical analyzer, and the presentation of the analysis results. e-SMC also allows the user to specify the QuaTeX query that describes the property to be analyzed. The e-SMC tool, its documentation and some examples are available at `http://maude.lcc.uma.es/esmc`.

In the e-Motions framework, the model described with the DSML passes through a series of transformations to finally generate a Maude specification. The first one is an ATL model-to-model transformation, which generates Maude models conforming the Maude metamodel. The second one is a *Xtend* transformation which generates the final Maude code. e-SMC includes a new model-to-model transformation from the generated Maude models by the ATL transformation to Maude models compliant with the PVeStA restrictions. This new Maude models are, in turn, passed as input to the *Xtend* transformation.

4 Case study: A simple messaging system

We illustrate the kind of statistical model checking we may perform with the very simple messaging system introduced in Section 2. To better illustrate the possible kinds of analysis, we compare the first simple messaging protocol results with a second version of the system in which each node has a routing table to decide which channel the message will be sent through in rules `NewMessage` and `Channel2Node`. In this second version, instead of probabilistically choosing an output channel, the value of the `via` attribute is retrieved from a table storing the best channel for a given destination.

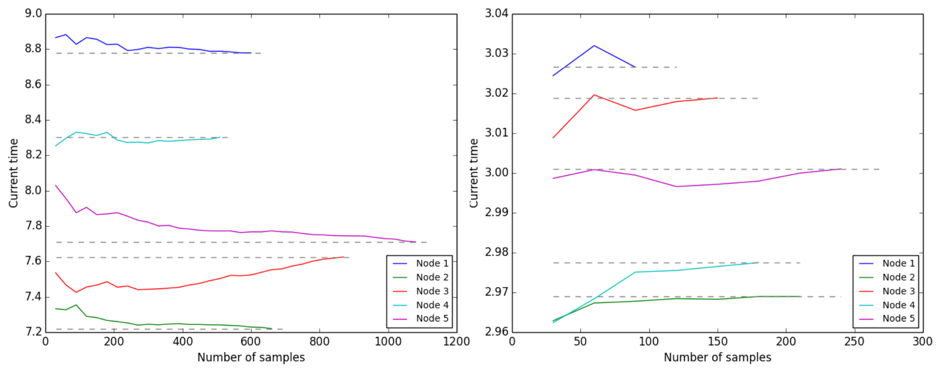
The most interesting property to be analyzed in these simple message passing systems is *how well connected is each node?* Or *how long does it take a message to reach its target?* However, this property has to be statistically analyzed, since it depends on the value of three stochastic parameters: (i) when is the next message going to be sent, (ii) which is the target node, and (iii) which channel chooses a node to send the message through. In terms of statistical model-checking, the property at hand could be expressed as "with a confidence of 99%, which is the mean time a message takes to commute between the source and target nodes?".

We proceed by defining a state expression which retrieves the mean response time collected by the `Observer` object at that time. Our executions have been performed using 8 threads (servers in the PVeStA terminology) and a master (client in PVeStA terms) running batches of 30 samples on each thread. After several iterations PVeStA returns the mean time elapsed between the start node n_i sending the message and the target node receiving it.

For the case of simple message passing with routing tables the time elapsed for messages sent from each node has been drastically reduced. Note that in the first case there may be even messages looping without finding their target. In this case we run batches of 10 samples on each thread, since it takes a smaller amount of values to converge.

Table 1: Execution times and mean time for messages being sent

	Simple Message Passing			*Simple Message Passing Routing*		
n_i	Ex. time	# samples	*mean. time*	Ex. time	# samples	*mean. time*
n_1	$93s$	600	8.7795	$23s$	240	2.9484
n_2	$111s$	660	7.2204	$18s$	160	2.9639
n_3	$96s$	870	7.6253	$16s$	160	2.9984
n_4	$92s$	510	8.3019	$18s$	160	3.0141
n_5	$87s$	1080	7.7106	$25s$	240	3.0233



(a) Simple Message Passing (b) Message Passing with Routing

Fig. 4: Mean message delivery time

Table 1 shows the execution times for each node, the number of samples needed to reach the confidence interval, and finally value of the property under study. Of course, the case study with routing converges with less samples since we are fixing which is the route the message will follow. Graphs in Figure 4 show the evolution of the mean of the arrival times for each of the nodes in the system. Notice how they converge after some number of samples to their respective final mean values, once the confidence interval is reached.

5 Related work

Heckel et al. propose in [17] the modeling and analysis of stochastic graphs transformation systems by defining Continuous Time Markov chains from GTSs with transition matrices representing the probabilities of the application of each rule. They provide some support using GROOVE [28] and PRISM. In later works [38], they handle distributions depending on pairs rule-match and may perform stocastic simulation. They develop GRaSS, with VIATRA as back-end, which can run multiples simulations limited by a time amount or number of steps. GRaSS may then calculate some statistical values with given confidence intervals.

GROOVE [28] supports the modeling of object-oriented systems, with graph transformations as a basis for model transformation and operational semantics. Systems thus defined may then be verified using model checking. CheckVML does something similar, although in this case system specifications into a tool-independent abstract representation of transition systems, from which Promela specifications are generated to model check using Spin.

Several attempts to reduce the complexity of model checking have also been proposed. Isenberg et al. propose in [18] the use of bounded model checking via SMT solving. Yousefian et al. use genetic algorithms in [41] to search specific states in large state spaces.

Based on ideas from Event Scheduling, de Lara et al. propose in [12] an interesting way of adding explicit time to graph transformation rules by scheduling rules in the future. Basically, they schedule all possible matches of rules (what they call *events*) and proceed by handling each of these events. To avoid the explosion in the number of matches, they use a control graph which establishes the possible sequences in which the rules may be applied. This idea could be an alternative way of guaranteeing the absence of un-quantified non-determinism, although at the cost of providing the control graph.

6 Conclusions

We have presented how the e-Motions tool has been extended so that the models built conforming to user-defined DSMLs are suitable for statistical model checking. e-Motions models are transformed into Maude specifications satisfying the requirements of the PVeStA tool [3], making them suitable to be stochastically analyzed. We have illustrated the use of e-Motions to model systems and its statistical model check with a very simple application for message delivery.

With our approach we provide statistical model checking capabilities to user-defined DSMLs in a user-friendly graphical environment. Statistical model checking offers a completely automatic procedure, with the possibility of adjusting the desired confidence interval and error bound.

Although the basic functionality and tooling is already available, much work remains ahead. For example, we would like to automate the check of the satisfaction of the non-quantified-non-determinism requirements. e-Motions features like periodicity, scheduling, or non-degenerate intervals are not yet supported. Moreover, although the response times obtained with PVeStA are acceptable, we would like to explore the possibility of using more powerful model checkers as back-end tools. Finally, we will complete the graphical representations of the obtained distributions of results inside our Eclipse plugin.

Acknowledgements This work was partially supported by Research Project TIN2014-52034-R and by Universidad de Málaga (Campus de Excelencia Internacional Andalucía Tech).

References

1. G. Agha, M. Greenwald, C. A. Gunter, S. Khanna, J. Meseguer, K. Sen, and P. Thati. Formal modeling and analysis of DOS using probabilistic rewrite theories. In *Proc. of FCS*, 2005.
2. G. Agha, J. Meseguer, and K. Sen. PMaude: Rewrite-based specification language for probabilistic object systems. In *Proc. of QAPL*, *ENTCS* 153:213–239, 2006.
3. M. AlTurki and J. Meseguer. PVeStA: A parallel statistical model checking and quantitative analysis tool. In *Proc. of CALCO*, *LNCS* 6859:386–392, 2011.
4. J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL - a tool suite for automatic verification of real-time systems. In *Proc. of DIMACS/SYCON*, *LNCS* 1066:232–243, 1996.
5. A. Boronat and J. Meseguer. An algebraic semantics for MOF. In *Proc. of FASE*, *LNCS* 4961:377–391, 2008.
6. A. Boronat and J. Meseguer. MOMENT2: EMF model transformations in Maude. In *Proc. of JISBD*, pp. 178–179, 2009.
7. A. Bouhoula, J.-P. Jouannaud, and J. Meseguer. Specification and proof in membership equational logic. *Theor. Comput. Sci.*, 236(1-2):35–132, Apr. 2000.
8. R. Bruni, A. Corradini, F. Gadducci, A. Lluch Lafuente, and A. Vandin. Modelling and analyzing adaptive self-assembly strategies with Maude. In *Proc. of WRLA*, *LNCS* 7571:118–138, 2012.
9. P. E. Bulychev, A. David, K. G. Larsen, M. Mikucionis, D. B. Poulsen, A. Legay, and Z. Wang. UPPAAL-SMC: statistical model checking for priced timed automata. In *Proc. of QAPL*, *EPTCS* 85:1–16, 2012.
10. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. F. Quesada. Maude: Specification and programming in rewriting logic. *Theor. Comput. Sci.* 285(2): 187–243, 2001.
11. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *All About Maude*, *LNCS* 4350, 2007.
12. J. de Lara, E. Guerra, A. Boronat, R. Heckel, and P. Torrini. Domain-specific discrete event modelling and simulation using graph transformation. *Software and System Modeling*, 13(1):209–238, 2014.
13. J. de Lara and H. Vangheluwe. AToM3: A tool for multi-formalism and metamodelling. In *Proc. of FASE*, *LNCS* 2306:174–188, 2002.
14. F. Durán, S. Lucas, C. Marché, J. Meseguer, and X. Urbain. Proving operational termination of membership equational programs. *Higher-Order and Symbolic Computation*, 21(1-2):59–88, 2008.
15. F. Durán and J. Meseguer. On the Church-Rosser and coherence properties of conditional order-sorted rewrite theories. *J. Log. Algebr. Program.*, 81(7-8):816–850, 2012.
16. J. Eckhardt, T. Mühlbauer, M. AlTurki, J. Meseguer, and M. Wirsing. Stable availability under denial of service attacks through formal patterns. In *Proc. of FASE*, *LNCS* 7212:78–93, 2012.
17. R. Heckel, G. Lajios, and S. Menge. Stochastic graph transformation systems. In *Proc. of ICGT*, *LNCS* 3256:210–225, 2004.
18. T. Isenberg, D. Steenken, and H. Wehrheim. Bounded model checking of graph transformation systems via SMT solving. In *Proc. of FMOODS/FORTE*, *LNCS* 7892:178–192, 2013.
19. D. N. Jansen, J. Katoen, M. Oldenkamp, M. Stoelinga, and I. S. Zapreev. How fast and fat is your probabilistic model checker? An experimental performance comparison. In *Proc. of HVC*, *LNCS* 4899:69–85, 2007.

20. S. Kelly and J.-P. Tolvanen. *Domain-specific modeling: enabling full code generation*. John Wiley & Sons, 2008.
21. S. Kent. Model driven engineering. In *Proc. of IFM*, *LNCS* 2335:286–298, 2002.
22. M. Z. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. In *Proc. of TACAS*, *LNCS* 2280:52–66, 2002.
23. M. Z. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. of CAV*, *LNCS* 6806:585–591, 2011.
24. J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theor. Comput. Sci.*, 96(1):73–155, Apr. 1992.
25. Meta object facility (MOF) core specification, 2013. Version 2.4.1.
26. A. Moreno-Delgado and F. Durán. The movie database case: A solution using the Maude-based e-Motions tool. In *7th Transformation Tool Contest (TTC)*, vol. 1305 of *CEUR Workshop Proc.*, pp. 116–124, 2014.
27. P. C. Ölveczky and J. Meseguer. Semantics and pragmatics of Real-Time Maude. *Higher-Order and Symbolic Computation*, 20(1-2):161–196, 2007.
28. A. Rensink. The GROOVE simulator: A tool for state space generation. In *Proc. of AGTIVE*, *LNCS* 3062:479–485, 2004.
29. J. E. Rivera, F. Durán, and A. Vallecillo. A graphical approach for modeling time-dependent behavior of dsls. In *Proc. of VL/HCC*, pp. 51–55. IEEE, 2009.
30. J. E. Rivera, F. Durán, and A. Vallecillo. On the behavioral semantics of real-time domain specific visual languages. In *Proc. of WRLA*, *LNCS* 6381:174–190, 2010.
31. J. E. Rivera, A. Vallecillo, and F. Durán. Formal specification and analysis of domain specific languages using Maude. *Simulation*, 85(11/12):778–792, 2009.
32. J. R. Romero, J. E. Rivera, F. Durán, and A. Vallecillo. Formal and tool support for model driven engineering with Maude. *J. of Object Techn.*, 6(9):187–207, 2007.
33. G. Rozenberg, ed.. *Handbook of Graph Grammars and Computing by Graph Transformations, vol. 1: Foundations*. World Scientific, 1997.
34. Á. Schmidt and D. Varró. CheckVML: A tool for model checking visual modeling languages. In *Proc. of UML*, *LNCS* 2863:92–95, 2003.
35. K. Sen, M. Viswanathan, and G. Agha. Statistical model checking of black-box probabilistic systems. In *Proc. of CAV*, *LNCS* 3114:202–215, 2004.
36. K. Sen, M. Viswanathan, and G. A. Agha. VeStA: A statistical model-checker and analyzer for probabilistic systems. In *Proc. of QEST*, pp. 251–252. IEEE, 2005.
37. G. Taentzer. AGG: A graph transformation environment for modeling and validation of software. In *Proc. of AGTIVE*, *LNCS* 3062:446–453, 2004.
38. P. Torrini, R. Heckel, and I. Ráth. Stochastic simulation of graph transformation systems. In *Proc. of FASE*, *LNCS* 6013:154–157, 2010.
39. H. L. S. Younes. Ymer: A statistical model checker. In *Proc. of CAV*, *LNCS* 3576:429–433, 2005.
40. H. L. S. Younes and R. G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *Proc. of CAV*, *LNCS* 2404:223–235, 2002.
41. R. Yousefian, V. Rafe, and M. Rahmani. A heuristic solution for model checking graph transformation systems. *Appl. Soft Comput.*, 24:169–180, 2014.