Simulation and Statistical Model Checking for Modestly Nondeterministic Models*

Jonathan Bogdoll, Arnd Hartmanns, and Holger Hermanns

Saarland University – Computer Science, Saarbrücken, Germany

Abstract. Modest is a high-level compositional modelling language for stochastic timed systems with a formal semantics in terms of stochastic timed automata. The analysis of Modest models is supported by the Modest Toolset, which includes the discrete-event simulator modes. modes handles arbitrary deterministic models as well as models that include nondeterminism due to concurrency through the use of methods inspired by partial order reduction. In this paper, we present version 1.4 of modes, which includes several enhancements compared to previous prototypical versions, such as support for recursive data structures, interactive simulation and statistical model checking.

1 Introduction

Modest [6] is a high-level compositional modelling language based on a formal semantics in terms of stochastic timed automata (STA) that provides an expressive syntax with features such as recursive processes, user-defined functions and exception handling. STA are a rich semantic model that includes nondeterministic and discrete probabilistic choices as in probabilistic automata (PA, [9]), hard real-time behaviour as in timed automata (TA, [1]) as well as stochastic sampling and delays according to arbitrary probability distributions. In fact, many well-known models, such as Markov chains, PA or TA are special cases of STA, and most are easy to identify on the syntactic level in Modest.

The analysis of models specified in Modest is supported by the Modest Toolset, available at `www.modestchecker.net`, which provides several tools for model-checking different subsets of Modest/STA as well as a discrete-event simulator, modes, that supports almost the entire Modest language. The tools are integrated into a graphical modelling and analysis environment. This paper focuses on modes, which was released in a prototypical version in the first half of 2011 [5], and has been significantly extended since.

The focus of modes is to allow simulation of nondeterministic models in a sound way. While most discrete-event simulators rely on hidden schedulers to resolve nondeterministic choices, which may influence the results in unexpected ways [8], modes uses methods inspired by partial order reduction [2] to decide,

* This work has been supported by the DFG as part of SFB/TR 14 AVACS and by the DFG/NWO bilateral research project ROCKS. It has received funding from the EU FP7 programme as part of the MEALS project, grant agreement n° 295261.

on-the-fly, whether any nondeterminism it encounters can be safely resolved in an arbitrary way, or whether doing so could skew the simulation results.

Since its original presentation, modes has continually been improved and extended in order to make it more robust, applicable to more case studies, and more user-friendly. To aid with model debugging, a new interactive simulation mode has been added. Aside from the original case studies presented in [5]—ARCADE [7] dependability evaluation models and the IEEE 802.3 binary exponential backoff protocol—modes has since been applied to the analysis of wireless sensor networks [3] and to network delay and queueing models as part of the Data Networks course taught at Saarland University in summer 2011. The unprejudiced use by ∼100 students has greatly improved the tool's robustness.

2 Language Enhancements

MODEST is a modelling language combining features from process algebra with convenient constructs from programming languages, with a focus on succinctness and expressivity. The interested reader is invited to refer to [6] for details concerning the language design and setup. To improve the usability and applicability of modes, we added four extensions that are fully supported by modes: Recursive data structures, user-defined functions, binary and broadcast synchronisation of actions and a new **delay** keyword to succinctly specify stochastic delays.

Data Structures and Functions. The original MODEST language definition was abstract w.r.t. the handling of data. Consistent and expressive means for data manipulation, however, are crucial to building complex, but readable models. Prior to the version presented in this paper, the only data types supported by modes were atomic types (like **bool** or **real**), fixed-size arrays thereof and C-like structs whose members had to be atomic types. We have replaced the latter with ML-like data types. For example,

datatype `list` = { **real** `hd`, `list` **option** `tl` };

declares a linked list of real numbers. For an instance `l` of that type, `l.hd` accesses its head and `l.tl` accesses its tail, which is a `list` **option**, i.e. it can be **none** or a `list`. To define operations on these types, but also to perform more complex computations, modes supports user-defined functions; as an example, to compute the length of a list, one could use the following function `len`:

function `len`(`list` **option** `l`) = **if** `l` == **none then** 0 **else** 1 + `len`(`l!.tl`);

These functions can be (mutually) recursive; however, if a function call on a simulation run does not terminate, neither will the run itself. User-defined data types and functions have been used, for example, to model and simulate a network with queues that prioritise packets according to length, using a sorted list.

New Synchronisation Modes. MODEST models are usually specified as the parallel composition of a set of concurrent processes, which then run asynchronously with the possibility of synchronising on certain action labels. As part of an effort to connect MODEST to UPPAAL [4], we have extended MODEST to support

CCS-style binary and UPPAAL-style broadcast communication in addition to the previously available CSP-style multi-way synchronisation. These additions, which greatly simplify certain modelling scenarios and allow more concise value passing via global variables (because the assignments of a sender (`a! {x := 7}`) are performed before the assignments of the receivers (`a? {y := x}`)), are fully supported by modes.

A Shorthand for Stochastic Delays. Delays in a timed model can be specified using guards and deadlines; for example, if c is a clock, the MODEST behaviour **when**$(c \geq 2)$ **urgent**$(c \geq 4)$ `tau` will result in an edge labelled `tau` being available for two time units starting when $c \geq 2$; as soon as c reaches 4, this edge has to be taken. To specify stochastic delays, a value is first sampled from some probability distribution and then used as above:

$$\{= c = 0,\ x = \texttt{Exp}(\lambda) =\};\ \textbf{when}(c \geq x)\ \textbf{urgent}(c \geq x)\ \texttt{tau}$$

causes `tau` to be executed precisely after an amount of time that is exponentially distributed with rate λ. To improve readability and lower the initial learning curve for models using such stochastic or deterministic delays, we have added a **delay** shorthand, so one can simply write **delay**(`Exp`(λ)) `tau` instead.

3 Towards Statistical Model Checking

When preparing the simulation of a model, two important decisions that have to be made are when to terminate a single simulation run and how many simulation runs to perform in order to obtain sufficient confidence in the results computed.

Termination Criteria for Runs. In previous versions, modes could be instructed to terminate a simulation run after a fixed number of steps, after a certain amount of model time, or when a given predicate evaluates to *true* for the first time. When simulating to compute the value of some property, e.g. the expected time until a packet is transmitted, these criteria were problematic: A run would either be aborted prematurely, i.e. when the transmission was not complete and the time was not yet known, or it would continue long after the packet had arrived, wasting simulation time. To avoid these situations, modes now supports terminating simulation runs precisely at the moment when all properties specified in the model can be decided.

Beyond Fixed Batch Sizes. In order to perform a statistical evaluation for a model property, the results of a number of simulation runs have to be collected. The accuracy of the result reported, e.g. specified by a standard deviation and a confidence interval, depends on how rare the event is and how many simulation runs were performed. Like many other simulators, previous versions of modes required the number of runs to be specified a priori. This had the unfortunate consequence that either too many runs were performed for a frequent event, or too few for a rare event—in which case another, larger batch of runs had to be performed, wasting the previous runs. modes version 1.4 offers two approaches to handle this problem:

Keeping the current statistical evaluation based on confidence intervals with a user-selected confidence level (default 95 %), the simulator can now be instructed to keep generating new runs only until the confidence interval becomes smaller than a specified width (an absolute value or relative to the standard deviation) for properties computing a value (e.g. an expected value or a probability), or until the one-sided confidence limit confirms or contradicts the bound in a Boolean property (e.g. "is the expected time to reach a safe state below x?").

For Boolean properties comparing a probability to some bound, modes 1.4 can also use sequential probability ratio testing [10], using an indifference region and risk levels specified by the user. Usage of this test is typically called *statistical model checking*, which has the advantage of requiring only a very low number of runs if the bound is far from the actual probability. For better usability, the risk levels are computed from the global confidence level in a symmetric fashion by default, but may be adjusted if desired.

4 Conclusion

In this paper, we have presented modes 1.4, a discrete-event simulator for the MODEST language that still is, to our knowledge, the only simulation tool that can deal with (some class of) nondeterministic models in a sound way. Compared to previous releases, which we rather consider research prototypes, version 1.4 has seen significant improvements in terms of ease of use, in particular ease of modelling and support for statistical model checking, as well as in robustness.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
2. Baier, C., D'Argenio, P.R., Größer, M.: Partial order reduction for probabilistic branching time. Electr. Notes in Theor. Comput. Sci. 153(2), 97–116 (2006)
3. Baró Graf, H., Hermanns, H., Kulshrestha, J., Peter, J., Vahldiek, A., Vasudevan, A.: A verified wireless safety critical hard real-time design. In: IEEE WoWMoM 2011. IEEE
4. Behrmann, G., David, A., Larsen, K.G.: A tutorial on UPPAAL. In: SFM-RT 2004. LNCS 3185:200–236. Springer
5. Bogdoll, J., Ferrer Fioriti, L.M., Hartmanns, A., Hermanns, H.: Partial order methods for statistical model checking and simulation. In: FMOODS/FORTE 2011. LNCS 6722:59–74. Springer
6. Bohnenkamp, H.C., D'Argenio, P.R., Hermanns, H., Katoen, J.P.: MoDeST: A compositional modeling formalism for hard and softly timed systems. IEEE Transactions on Software Engineering 32(10), 812–830 (2006)
7. Boudali, H., Crouzen, P., Haverkort, B.R., Kuntz, M., Stoelinga, M.: Architectural dependability evaluation with Arcade. In: DSN 2008. pp. 512–521. IEEE CS Press
8. Hartmanns, A.: Model-checking and simulation for stochastic timed systems. In: FMCO 2010. LNCS 6957:372–391. Springer
9. Segala, R.: Modeling and Verification of Randomized Distributed Real-Time Systems. Ph.D. thesis, MIT, Cambridge, MA, USA (1995)
10. Wald, A.: Sequential analysis. Wiley, New York (1959)