State-of-the-art Tools and Techniques for Quantitative Modeling and Analysis of Embedded Systems

Marius Bozga*, Alexandre David†, Arnd Hartmanns§, Holger Hermanns§, Kim G. Larsen†, Axel Legay‡, Jan Tretmans¶†Aalborg*CNRS Verimag‡INRIA/IRISA §Saarland University ¶Embedded Systems Institute and Radboud University

***Abstract*—This paper surveys well-established/recent tools and techniques developed for the design of rigorous embedded systems. We will first survey UPPAAL and MODEST, two tools capable of dealing with both timed and stochastic aspects. Then, we will overview the BIP framework for modular design and code generation. Finally, model-based testing will be discussed.**

I. CONTEXT

The rigorous design of embedded systems radically differs from pure software design in that it should take into account not only functional but also extra-functional specifications regarding the use of resources of the execution platform such as time, memory and energy. Meeting extra-functional specifications is essential for the design of embedded systems. It requires predictability of the impact of design choices on the overall behaviour of the designed system. It also implies a deep understanding of the interaction between application software and the underlying execution platform.
Our objective is to overview some of the well-established/recent tools and techniques developed for the design of rigorous embedded systems. What makes these tools unique is their ability to deal with both timing and stochastic aspects. We will start by introducing the UPPAAL real-time modeling and verification toolset and its underlying theory as well as recent features. Then, we will discuss the MODESTapproach, that is a new unifying modeling paradigm allowing to interconnect stochastic and timed analysis tools in a semantically sound manner. Then, the BIP workflow for component-based design will be introduced. One of the major features of BIP is its ability to generate correct code for component coordination. Finally, model-based testing, that is already used in industry, and its potential integration in existing toolsets will be discussed.

II. THE UPPAAL TOOLSET

UPPAAL [1] is a tool suit supporting modeling, simulation, verification, synthesis and performance analysis of real-time systems described as networks of timed automata [2] communicating by channel synchronisations. It now features an advanced modelling language where the basic timed automata formalism is extended with a C-like imperative language with user-defined types and functions, allowing for readable and compact models with reusable updates of discrete variables. The property language of UPPAAL allows for safety, liveness and time-bounded liveness properties to be expressed. The tool also features a user-friendly graphical interface, and an efficient symbolic model checking engine. In addition, several flavours of the tool have matured in recent years.

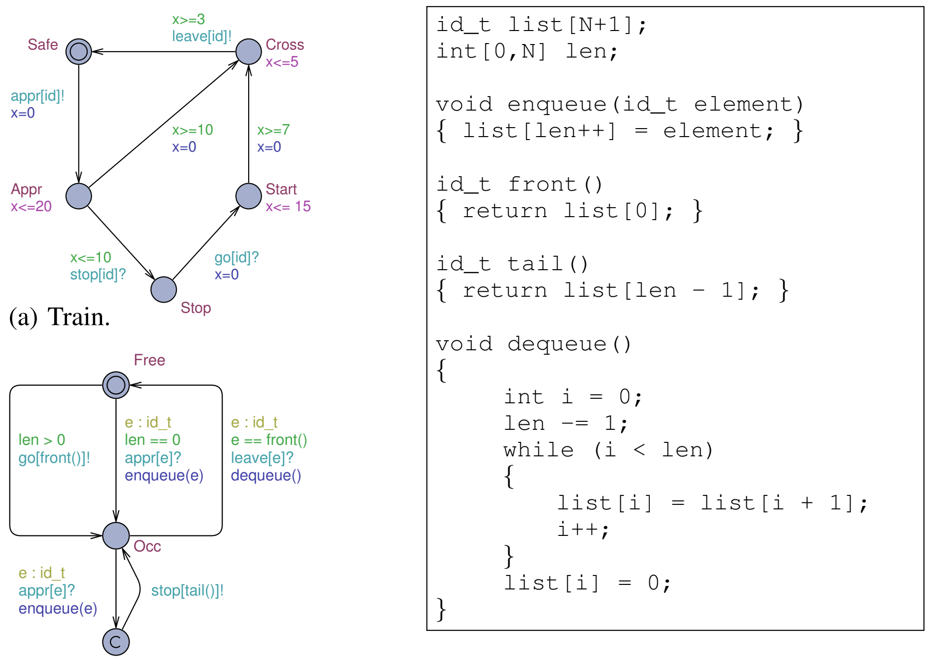
Using a shared modeling and specification formalism, UPPAAL has developed into several domain specific versions:

- UPPAAL-CORA is based on timed automata extended with cost variables [3]. The tool uses a symbolic algorithm for solving minimum cost reachability problems, with several applications to optimization for embedded systems, including Worst-Case Execution Times (WCET) analysis [4].
- UPPAAL-TIGA [5] provides an efficent method for synthesizing winning control strategies from timed *game* automata with respect to safety and liveness objectives. E.g the tool has been applied to controller synthesis of optimal and robust controllers for hydraulic pumps [6].
- UPPAAL-TRON [7] is a testing tool suited for black-box conformance testing of timed systems (see also Section V). UPPAAL-TRON is mainly targeted for embedded software commonly found in various controllers, and applies on-line testing in the sense that tests are derived, executed, and checked during interaction with the system in real-time.
- ECDAR [8] is a variant of UPPAAL supporting compostional development. The tool is designed to check incrementally refinement and consistency between component specifications given as timed automata. Also, the tool allows for structural and logical composition of specifications.
- UPPAAL-SMC [9], [10] is the newest member of the UPPAAL suite coming equipped with a completely new and highly efficient and scalable engine for performing statistical model checking. I.e. based on a stochastic semantics of networks of timed automata, properties are settled with a desired level of confidence based on random simulation runs.

A. A Train Crossing Example

We present an example that we use for model-checking, code synthesis, and performance analysis. We consider a

978-3-9810801-8-6/DATE12/©2012 EDAA



```
id_t list[N+1];
int[0,N] len;

void enqueue(id_t element)
{ list[len++] = element; }

id_t front()
{ return list[0]; }

id_t tail()
{ return list[len - 1]; }

void dequeue()
{
     int i = 0;
     len -= 1;
     while (i < len)
     {
         list[i] = list[i + 1];
         i++;
     }
     list[i] = 0;
}
```

(a) Train.

(b) Controller.

(c) Types and functions of the controller.

Fig. 1. A Uppaal model of a train (a) and a controller (b) with its code (c).

number of trains that are approaching a bridge on their own tracks. The bridge has only one track so a controller decides to stop and restart trains when more than one train is approaching at the same time.

Figure 1.(a) shows a timed automaton template that we use for every train. Trains start in the Safe location and when they are approaching, they synchronize with the controller with appr[id]! and go to the Appr location where they stay for at most 20 time units. Trains may be stopped by the controller before 10 time units (and go to the Stop location) otherwise they will cross and go to the Cross location. When stopped, a train may be restarted with the go[id]? synchronization and then it will cross the bridge between 7 and 15 time units.

Figure 1.(b) shows the controller that maintains a FIFO queue of stopped trains. It has two main locations Free and Occ that keep track of the state of the bridge that is free or occupied. When a train approaches, it is enqueued. It is dequeued when it leaves the bridge. If the bridge is occupied, then it is immediately stopped (bottom committed location marked with `c`).

Figure 1.(c) shows the user-defined code and functions that handle the FIFO queue. The queue is declared as an array and its length is kept track with the `len` integer. The functions `enqueue`, `front`, and `tail` are used to respectively enqueue elements at the end of the queue, read the first element, or read the last element. To dequeue elements we have to shift the values in the array to the left, which is done with a `while` loop.

a) Verification: We are interested in the following properties to ensure correctness of the controller:

- Safety: At most one train at a time will cross the bridge.
  ```
  A[] forall (i:id_t) forall (j:id_t)
  Train(i).Cross && Train(j).Cross imply
  i == j
  ```

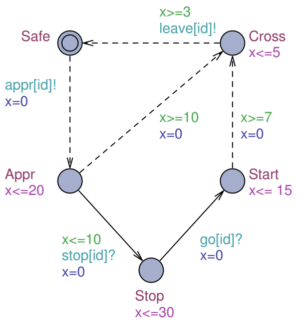


Fig. 2. Timed game automaton for the trains.

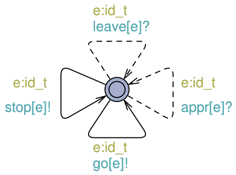


Fig. 3. Unconstrained automaton for the controller.

- Liveness: Whenever a train is approaching then it will eventually cross the bridge.
  ```
  Train(0).Appr --> Train(0).Cross
  ...
  Train(5).Appr --> Train(5).Cross
  ```
  We specify one property per train in this case.
- Absence of deadlock: `A[] not deadlock`.

b) Synthesis: Instead of making a controller by hand we can synthesize it. To do this we use Uppaal-Tiga that is going to solve a two-player timed game where one player, the *environment*, decides non-deterministically when trains arrive and the time it takes to cross, and the second player, the *controller*, decides deterministically when to stop and restart the trains. Figure 2 shows a slightly modified train where

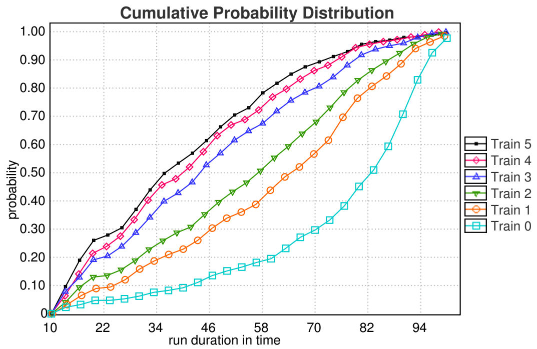


Fig. 4. Cumulative distribution for the trains to cross in function of time.

```
process Channel() {
    clock c;
    put palt {
        :98: {= c = 0 =};
             // transmission delay of
             // up to TD time units
             invariant(c <= TD) get
        : 2: {==} // message lost
    }; Channel()
}
```

Fig. 5. A communication channel with 2 % message loss in MODEST

the environment decides to take the *uncontrollable* transitions (dashed) and the controller the *controllable* transitions. Figure 3 shows an unconstrained automaton that specifies what the controller can do. It is up to UPPAAL-TIGA now to compute a *strategy* that will decide when to take the controllable transitions. We note that this automaton could include some constraints if, for example, it took a certain additional time to send a signal to trains.

c) Performance Analysis: We can study performance of models using UPPAAL-SMC. To do so, we equip our timed automata model with a stochastic semantics. In a nutshell, every train can choose to output `appr[id]!` or `leave[id]!` independently after making a delay according to a distribution. The distribution is exponential when there is no invariant (on the Safe location), or uniform when there is an invariant (on the Cross location). The train picking the shortest delay moves.

We are interested in how fast the trains may cross where we give as rate (of the exponential distribution) `1+id` to differentiate the trains. Figure 4 shows the cumulative distribution over time of trains crossing. To obtain this plot we check the property `Pr[<=100](<> Train(0).Cross)` (for train 0) that we repeat for every train and we supperpose the distributions.

III. MODESTAND THE MODEST TOOLSET

MODEST [11] is a compositional modelling language that combines expressive and powerful syntax-level features with a formal semantics in terms of stochastic timed automata (STA). The STA model includes probabilistic choices based on continuous and discrete probability distributions as well as nondeterministic choices over successor locations and time.

The idea behind MODEST is to provide a *single-formalism, multi-solution* approach to modelling and analysis: Many well-known models are subsumed by STA, such as (probabilistic) timed automata ((P)TA, [12]), and most of them are easy to identify on the syntactical level (for example, a MODEST model is a TA if no probabilistic constructs are used). MODEST can thus be used to build models in a wide range of formalisms, and with an appropriate transformation, existing analysis engines and algorithms that are specifically designed for the formalism at hand can be used for analysis.

This approach is supported by the MODEST TOOLSET, available at `www.modestchecker.net`, which currently provides three ways to analyse MODEST models, with more under development:

- mctau [13] connects UPPAAL to MODEST for TA models, allowing both automated analysis of properties specified in the model as well as export to UPPAAL XML, including automatic layout of the component automata;
- mcpta [14], [15] handles MODEST models of PTA, using the PRISM [16] probabilistic model checker as a backend, but including optimisations specific to MODEST; and
- modes [17] provides discrete-event simulation for arbitrary deterministic MODEST models as well as for models that exhibit nondeterminism due to concurrency.

These tools are seamlessly integrated in mime, a graphical modelling and analysis environment.

A. A Bounded Retransmission Example

Let us consider the Bounded Retransmission Protocol (BRP, [18]), an alternating-bit-based communication protocol with an upper bound on the number of retransmissions. A realistic model of the BRP includes both real-time aspects, namely timeouts and transmission delays [19], and probabilistic choices, namely the loss of messages on the communication channel [20]. To show how these aspects are combined in MODEST in an orthogonal way, the MODEST code modelling the communication channel, which synchronises with the sender and receiver via actions `put` and `get`, is listed in Figure 5. The full model is available as part of the MODEST TOOLSET download.

As a first step to analyse the BRP, we should use mctau. The model contains probabilistic decisions, which UPPAAL cannot handle, but mctau can automatically overapproximate these with nondeterministic decisions. This is particularly useful for model debugging, since a nonprobabilistic analysis is usually significantly faster than a probabilistic one. Table I summarises the results that the different tools obtain on the BRP model; we see that mctau is able to obtain the exact value for the invariant properties T_{A1} (no premature timeouts) and T_{A2} (correct handling of failures) as well the probabilistic reachability properties[1] P_{A} and P_{B} (the probabilities of certain incorrect sender reactions). mctau cannot handle property $\mathrm{E}_{\max}$ (the expected time until a transfer finishes), and cannot safely conclude that the probability for P_1 (no

[1] All properties considered here ask for a *maximum* probability over all resolutions of nondeterminism; minimum probabilities are equally supported.

TABLE I

RESULTS FOR THE BRP MODEL, PARAMETERS $(\mathrm{N}, \mathrm{MAX}, \mathrm{TD}) = (16, 2, 1)$

property	mctau	mcpta	modes
T_{A1}	true	true	true (all 10k runs satisfied T_{A1})
T_{A2}	true	true	true (all 10k runs satisfied T_{A2})
P_A	0	0	0 (no observations in 10k runs)
P_B	0	0	0 (no observations in 10k runs)
P_1	$[0, 1]$	$4.233 \cdot 10^{-4}$	$\mu = 3.0 \cdot 10^{-4}$, $\sigma = 1.7 \cdot 10^{-2}$
P_2	$[0, 1]$	$2.645 \cdot 10^{-5}$	0 (no observations in 10k runs)
D_{max}	$[0, 1]$	$9.996 \cdot 10^{-1}$	$\mu = 9.9 \cdot 10^{-1}$, $\sigma = 1.7 \cdot 10^{-2}$
E_{max}	n/a	33.473	$\mu = 33.473$, $\sigma = 2.136$

success reported), P_2 (uncertainty reported) or the time-bounded property D_{max} (probability of success in time 64) is zero or one.

Once we are confident that the model is correct (in this case, the results for the first four properties match our expectations), we can use mcpta to perform a full probabilistic analysis, which takes noticeably longer than the quick check with mctau (still < 1 min for this small example), but yields precise results for all properties considered.

Again, the very same model can also be simulated with modes; however, the results in Table I show that this particular model is not very well-suited for simulation because we are interested in rather rare events, some of which were never observed in 10000 simulation runs that also took significantly longer than model-checking. In the table, μ is the mean and σ is the standard deviation of the assumed normal distribution for the results. For property E_{max}, which is not a rare event but an expected value, modes performs very well; in general, the advantage of simulation is that it is not subject to the state-space explosion problem and can thus handle arbitrarily large model instances, as well as more complex properties. We also note that simulation, in principle, cannot be used for nondeterministic models; in this case, we explicitly specified a scheduler to resolve nondeterminism, but modes is also able to soundly handle nondeterminism resulting from the interleaving of concurrent behaviour without relying on (implicit or explicit) schedulers [17].

In [14], we also considered two protocols that, in contrast to the BRP, are inherently probabilistic due to the use of randomized schemes to resolve contention, and we explored different modelling approaches, including a straightforward pattern to (mechanically) transform a graphical model given as automata into a (text-based) MODEST model.

IV. COMPONENT-BASED DESIGN OF AUTONOMOUS SYSTEMS

Rigorous system design requires the use of a single component framework, with sound semantics, allowing the representation of the system at different levels of detail, from high-level design to implementation. The use of a single framework is essential to maintain the overall coherency and correctness by relating different models and their properties along the flow.

BIP – Behavior, Interaction, Priority [21] – is a component framework intended to rigorous system design. BIP allows the construction of composite hierarchically structured systems from atomic components characterized by their behavior and their interface. Components are composed by layered application of interactions and of priorities. Interactions express synchronization constraints between actions of the composed components while priorities are used to filter amongst possible interactions and to steer system evolution so as to meet performance requirements e.g. to express scheduling policies. Interactions are described in BIP as the combination of two types of protocols: rendez-vous, to express strong symmetric synchronization and broadcast, to express triggered asymmetric synchronization. The combination of interactions and priorities confers BIP expressiveness not matched by any other existing formalism. It defines a clean and abstract concept of architecture separate from behavior. Architecture in BIP is a first class concept with well-defined semantics that can be analyzed and transformed. BIP relies on rigorous operational semantics that has been implemented by specific execution engines for centralized, distributed and real-time execution.

BIP is used as a unifying semantic model in a rigorous system design flow [22]. Rigorousness is ensured by two kinds of tools: verification tools such as D-Finder [23] for checking safety properties (and deadlock-freedom in particular) and source-to-source transformers [24], [25] that allow progressive refinement of (purely functional) application software towards platform-dependent implementations.

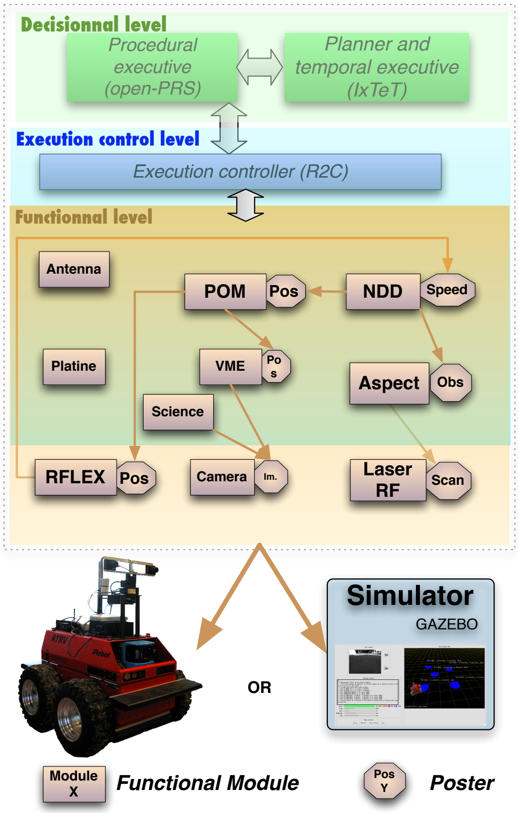


Fig. 6. An overview of the embedded software of DALA

The BIP design flow has been applied for the development of a new version of the functional and execution control level of DALA (see Figure 6), an autonomous rover robot developed at Laas Laboratory, Toulouse. This experiment has been partially described in [26], [27]. The construction involves hierarchical decomposition of the functional level into atomic components, definition of the behavior of each atomic component and description of composition by using interactions and priorities. The BIP model has been formally verified for deadlock-freedom and other safety properties. Moreover, it has been used to synthesize an execution controller that encodes and enforces safety properties by construction, thereby facilitating the development of safe and dependable robotic architectures. Experiments with the code generated automatically from the BIP model demonstrate (via fault injections) that the controller successfully stops the robot from reaching undesired/unsafe states.

V. Model-Based Testing

When models have been used for system design and verification, a logical next step is to obtain a system implementation that faithfully implements the behaviour described in the model. Sometimes, it is possible to generate such an implementation by means of code generation or by stepwise refinement from the model. Whenever such a correctness preserving transformation from model to code is not feasible, e.g., due to a too large abstraction discrepancy between model and implementation, the correctness of the (manually developed) implementation must be checked afterwards. Systematic testing is an important technique for checking the quality of a system implementation and for verifying compliance of its actual behaviour with its specified model behaviour. Traditional testing, however, is often a manual and laborious process, which makes it expensive, error-prone, and time-consuming.

Model-based testing is one of the promising technologies to meet the challenges imposed on testing. In model-based testing, a model serves as a precise and complete specification of what the system shall do, and, consequently, is a good basis for testing the system. Moreover, such a model allows the algorithmic generation of large amounts of test cases, including test oracles, completely automatically from the model. This leads to faster, less error-prone, and better maintainable test generation: millions of test events can be automatically generated, and 'on-the-fly' executed and analysed. And if the model is valid, i.e., expresses precisely what the system under test should do, then all these generated tests are provably valid, too. This makes it possible to automate the testing process well beyond the mere automatic execution of manually crafted test cases, which is the current state of practice.

Model-based testing complements formal verification approaches such as (statistical) model checking. Formal verification intends to show that a system has some desired properties by proving that a model of that system satisfies these properties. Thus, any verification is only as good as the validity of the model on which it is based. Model-based testing starts with a (verified) model, and then aims at showing that the real, physical implementation of the system behaves in compliance with this model. Due to the inherent limitations of testing, such as the limited number of tests that can be performed, testing can never be complete: testing can only show the presence of errors, not their absence.

Model-based testing is more than just the generation of some test cases from a model. A well-defined and sound theory shall underpin model-based testing. Such a theory must support precise reasoning about the objects of model-based testing, such as models, implementations under test (IUT), test cases, test generation, and verdicts, and about their mutual relations: when is an IUT correct with respect to a model, what does it mean for a test case to be valid, and what encompasses a correctness proof for a test generation algorithm. Two important ingredients of such a theory of model-based testing are a testing hypothesis and an implementation relation.

A testing hypothesis, or test assumption, establishes the link between the black-box, real IUT and the world of models. The assumption is made that any real IUT can be modelled by some object in a domain of models. In this way, the testing hypothesis allows reasoning about IUTs as if they were models in this (formal) domain. Building on the testing hypothesis, an implementation relation, also called conformance- or refinement relation, is a formal relation between models of IUTs and specification models. It defines when an IUT is correct with respect to a specification model.

One of the theories for model based testing is the **ioco**-testing theory, where models are expressed as labelled transition systems and compliance is defined with the **ioco**-implementation relation (*I*nput/*O*utput *Co*nformance) [28]. This model-based testing theory, on the one hand, provides a sound and well-defined foundation for transition system testing, having its roots in the theoretical area of testing equivalences and refusal testing. On the other hand, it has proved to be a practical basis for several model-based test generation tools and applications. The **ioco**-testing theory uses labelled transition systems as models for specifications and tests, the testing hypothesis is that implementations behave as if they were input-enabled labelled transition systems, and conformance between IUTs and specification models is defined by **ioco**. An algorithm generates tests from a model, which can be shown to be complete (sound and exhaustive) meaning that the algorithmically generated test cases detect only, and in the limit all, non-**ioco** conforming IUTs.

Tools that implement the **ioco**-approach are, among others, TorX, JTorX, TorXakis, TGV, and Axini Test Manager, which have been applied, for instance, to testing of a highway-tolling system protocol, the EU electronic passport, a software bus, and wireless-sensor-network nodes [29], [30].

An important variant of **ioco** for the embedded systems domain is $\mathbf{rtioco}_e$: environment-relativized timed input/output conformance. It adds testing for real-time properties, and is implemented in the model-based testing tool Uppaal-Tron, a member of the Uppaal family of Timed Automata-based analysis tools.

Benefits of model-based testing compared with manual testing are the generation of large numbers of long, valid tests, allowing testing that is more thorough, faster, cheaper, and more efficient. Moreover, testing is easily repeated after modifications in the model or in the system. Although the construction of a model may seem an extra effort, practice shows that this modeling activity in itself leads to improved understanding of the system, and to earlier detection of imprecise, incomplete, or ambiguous requirements. A model for model-based testing can be more abstract, with less detail, then a model for code generation: testing a system for a particular property is easier than generating a system with that property. Finally, the **ioco**- and $\mathbf{rtioco}_e$-approaches can deal with typical embedded software aspects such as real-time properties, concurrency, nondeterminism, abstraction, under-specified and partial specifications, and assumptions on the (physical) environment,

References

[1] G. Behrmann, A. David, and K. G. Larsen, "A tutorial on uppaal," in *SFM*, ser. Lecture Notes in Computer Science, M. Bernardo and F. Corradini, Eds., vol. 3185. Springer, 2004, pp. 200–236.

[2] R. Alur and D. L. Dill, "Automata for modeling real-time systems," in *ICALP*, ser. Lecture Notes in Computer Science, M. Paterson, Ed., vol. 443. Springer, 1990, pp. 322–335.

[3] G. Behrmann, A. Fehnker, T. Hune, K. G. Larsen, P. Pettersson, J. Romijn, and F. W. Vaandrager, "Minimum-cost reachability for priced timed automata," in *HSCC*, ser. Lecture Notes in Computer Science, M. D. D. Benedetto and A. L. Sangiovanni-Vincentelli, Eds., vol. 2034. Springer, 2001, pp. 147–161.

[4] A. E. Dalsgaard, M. C. Olesen, M. Toft, R. R. Hansen, and K. G. Larsen, "Metamoc: Modular execution time analysis using model checking," in *WCET*, ser. OASICS, B. Lisper, Ed., vol. 15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2010, pp. 113–123.

[5] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. G. Larsen, and D. Lime, "Uppaal-tiga: Time for playing games!" in *CAV*, ser. Lecture Notes in Computer Science, W. Damm and H. Hermanns, Eds., vol. 4590. Springer, 2007, pp. 121–125.

[6] F. Cassez, J. J. Jessen, K. G. Larsen, J.-F. Raskin, and P.-A. Reynier, "Automatic synthesis of robust and optimal controllers - an industrial case study," in *HSCC*, ser. Lecture Notes in Computer Science, R. Majumdar and P. Tabuada, Eds., vol. 5469. Springer, 2009, pp. 90–104.

[7] K. G. Larsen, M. Mikucionis, B. Nielsen, and A. Skou, "Testing real-time embedded software using uppaal-tron: an industrial case study," in *EMSOFT*, W. Wolf, Ed. ACM, 2005, pp. 299–306.

[8] A. David, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski, "Timed i/o automata: a complete specification theory for real-time systems," in *HSCC*, K. H. Johansson and W. Yi, Eds. ACM ACM, 2010, pp. 91–100.

[9] A. David, K. G. Larsen, A. Legay, M. Mikucionis, and Z. Wang, "Time for statistical model checking of real-time systems," in *CAV*, ser. Lecture Notes in Computer Science, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, 2011, pp. 349–355.

[10] A. David, K. G. Larsen, A. Legay, M. Mikucionis, D. B. Poulsen, J. van Vliet, and Z. Wang, "Statistical model checking for networks of priced timed automata," in *FORMATS*, ser. Lecture Notes in Computer Science, U. Fahrenberg and S. Tripakis, Eds., vol. 6919. Springer, 2011, pp. 80–96.

[11] H. C. Bohnenkamp, P. R. D'Argenio, H. Hermanns, and J.-P. Katoen, "MoDeST: A compositional modeling formalism for hard and softly timed systems," *IEEE Transactions on Software Engineering*, vol. 32, no. 10, pp. 812–830, 2006.

[12] M. Z. Kwiatkowska, G. Norman, R. Segala, and J. Sproston, "Automatic verification of real-time systems with discrete probability distributions," *Theor. Comput. Sci.*, vol. 282, no. 1, pp. 101–150, 2002.

[13] J. Bogdoll, A. David, and A. Hartmanns, "mctau: Bridging the gap between Modest and UPPAAL," submitted to TACAS 2012.

[14] A. Hartmanns and H. Hermanns, "A modest approach to checking probabilistic timed automata," in *QEST*. IEEE Computer Society, 2009, pp. 187–196.

[15] A. Hartmanns, "Model-checking and simulation for stochastic timed systems," in *FMCO*, ser. LNCS, vol. 6957. Springer, December 2010.

[16] D. Parker, "Implementation of symbolic model checking for probabilistic systems," Ph.D. dissertation, University of Birmingham, 2002.

[17] J. Bogdoll, L. M. Ferrer Fioriti, A. Hartmanns, and H. Hermanns, "Partial order methods for statistical model checking and simulation," in *FMOODS/FORTE*, ser. LNCS, R. Bruni and J. Dingel, Eds., vol. 6722. Springer, 2011, pp. 59–74.

[18] L. Helmink, M. P. A. Sellink, and F. W. Vaandrager, "Proof-checking a data link protocol," in *TYPES*, ser. LNCS, H. Barendregt and T. Nipkow, Eds., vol. 806. Springer, 1993, pp. 127–165.

[19] P. R. D'Argenio, J.-P. Katoen, T. C. Ruys, and J. Tretmans, "The bounded retransmission protocol must be on time!" in *TACAS*, ser. LNCS, E. Brinksma, Ed., vol. 1217. Springer, 1997, pp. 416–431.

[20] P. R. D'Argenio, B. Jeannet, H. E. Jensen, and K. G. Larsen, "Reachability analysis of probabilistic systems by successive refinements," in *PAPM-PROBMIV*, ser. LNCS, L. de Alfaro and S. Gilmore, Eds., vol. 2165. Springer, 2001, pp. 39–56.

[21] A. Basu, M. Bozga, and J. Sifakis, "Modeling Heterogeneous Real-time Systems in BIP," in *Software Engineering and Formal Methods, SEFM'06 Proceedings*, 2006, pp. 3–12.

[22] A. Basu, S. Bensalem, M. Bozga, J. Combaz, M. Jaber, T.-H. Nguyen, and J. Sifakis, "Rigorous Component-based Design using the BIP Framework," *IEEE Software, Special Edition – Software Components beyond Programming – from Routines to Services*, vol. 28, no. 3, pp. 41–48, 2011.

[23] S. Bensalem, M. Bozga, T.-H. Nguyen, and J. Sifakis, "D-Finder: A Tool for Compositional Deadlock Detection and Verification," in *Computer-Aided Verification, CAV'09 Proceedings*, 2009, pp. 614–619.

[24] M. Bozga, M. Jaber, and J. Sifakis, "Source-to-Source Architecture Transformation for Performance Optimization in BIP," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 4, pp. 708–718, 2010.

[25] B. Bonakdarpour, M. Bozga, M. Jaber, J. Quilbeuf, and J. Sifakis, "From High-Level Component-Based Models to Distributed Implementations," in *Embedded Software, EMSOFT'10 Proceedings*, 2010, pp. 209–218.

[26] A. Basu, M. Gallien, C. Lesire, T.-H. Nguyen, S. Bensalem, F. Ingrand, and J. Sifakis, "Incremental Component-Based Construction and Verification of a Robotic System," in *European Conference on Artificial Intelligence, ECAI'08 Proceedings*, 2008, pp. 631–635.

[27] S. Bensalem, L. de Silva, A. Griesmayer, F. Ingrand, A. Legay, and R. Yan, "A Formal Approach for Incremental Construction with an Application to Autonomous Robotic Systems," in *Software Composition, SC'11 Proceedings*, 2011, pp. 116–132.

[28] J. Tretmans, "Model Based Testing with Labelled Transition Systems," in *Formal Methods and Testing*, ser. Lecture Notes in Computer Science, R. Hierons, J. Bowen, and M. Harman, Eds., vol. 4949. Springer-Verlag, 2008, pp. 1–38.

[29] W. Mostowski, E. Poll, J. Schmaltz, J. Tretmans, and R. Wichers Schreur, "Model-Based Testing of Electronic Passports," in *Formal Methods for Industrial Critical Systems – FMICS 2009*, ser. Lecture Notes in Computer Science, M. Alpuente, B. Cook, and C. Joubert, Eds., vol. 5825. Springer-Verlag, 2009, pp. 207–209.

[30] M. Sijtema, M. Stoelinga, A. Belinfante, and L. Marinelli, "Experiences with Formal Engineering: Model-Based Specification, Implementation and Testing of a Software Bus at Neopost," in *FMICS 2011*, ser. LNCS, G. Salaün and B. Schätz, Eds., vol. 6959, 2011, pp. 117–133.