Model checking embedded system designs

Ed Brinksma and Angelika Mader
Faculty of Computer Science, University of Twente
{brinksma,mader}@cs.utwente.nl

Extended abstract [1]

1 Introduction

Model checking has established itself as a successful tool supported technique for the verification and debugging of various hardware and software systems [16]. Not only in academia, but also by industry this technique is increasingly being regarded as a promising and practical proposition, especially in the area of hardware verification [22].

Model checking is also being applied with success to the analysis of the control software in embedded systems [37]. Because such systems are often *mission critical*, in the sense that their failure to operate correctly can be very costly in terms of cost and safety, there is a natural tendency to apply advanced methods to guide and analyse their design.

Because model checkers typically trace the behaviour leading to the explored states they can be used not only analytically, i.e. to verify that given control designs are correct, but also synthetically, i.e. to derive or optimize control designs. Examples of such controller synthesis in the context of timed automata can be found in [31, 7].

In this extended abstract we survey the basic principles behind the application of model checking to controller verification and synthesis. Although the development of model checkers for sophisticated models of system behaviour is a nontrivial task, the conceptual ingredients often remain straightforward.

The major problem besetting model checking, as with many tools dealing with concurrent systems, is the infamous *state space explosion*, caused by the fact that the global state space of a system grows exponentially with the local state spaces of its components. The more sophisticated the model of behaviour, the more difficult to control this phenomenon effectively. This makes abstract modelling and state space search techniques decisive practical ingredients of model checking applications, more than the sophistication of the behavioural model that is supported.

[1] The authors dedicate this paper to their daughter Esther Lilian, whose birth during its writing reduced it to an extended abstract.

A promising development is the area of *guided model checking*, in which the state space search strategy of the model checking algorithm can be influenced to visit more interesting sets of states first. In particular, we will discuss how model checking can be combined with heuristic *cost functions* to guide search strategies.

In the final section of this extended abstract we list a number of current research developments, especially in the area of reachability analysis for optimal control and related issues.

2 Model checking in a nutshell

Model checking is basically a brute force exploration of the states of a given system, checking that a given state s satisfies a property . As a basic model for system behaviour here we use *labelled transition systems* that offer a straightforward way of representing system behaviour in terms of states and state transitions.

Definition 1 *A labelled transition system T is tuple , $t, \rightarrow$ with a nonempty set of system states; t a nonempty set of actions; and $\rightarrow \subseteq \times t \times$ a transition relation. Instead of $(s, , s') \in \rightarrow$ we write $s \rightarrow s'$, and $s \rightarrow s$ if the action is not relevant.*

A simple but important form of state space exploration is *reachability analysis*. This amounts to checking whether states belonging to a GOAL set can be reached following transitions from a given set of initial states INIT (often restricted to a unique initial state $s_0 \in$). This can be seen as calculating the smallest fixpoint * containing INIT of the functional $( ) = \cup st( )$, where $st( ) = \{s' \in | \exists s \in : s \rightarrow s'\}$, and checking that $^* \cap$ GOAL $\neq \emptyset$. This is known as *forward reachability checking*; an abstract algorithm for this is given in Figure 1; *backwards* reachability analysis takes the dual approach by exchanging the roles of INIT and GOAL, and working with $( ) = \cup r( )$, where $r( ) = \{s' \in | \exists s \in : s' \rightarrow s\}$.

If the labelled transition system is finite, i.e. if and L are finite, then the above algorithm will terminate. It assumes that the test $s \in$ GOAL can be decided on the basis of

Proceedings of the Sixth International Workshop on Discrete Event Systems (WODES'02)
0-7695-1683-1/02 $17.00 © 2002 IEEE

PASSED := $\emptyset$
WAITING := INIT
while WAITING$\neq \emptyset$ **do**
  remove some s **from** WAITING
  if $s \in$ GOAL **then return(yes)**
  if $s \notin$ PASSED **then**
    PASSED := PASSED $\cup \{s\}$
    WAITING := WAITING $\cup \{s' | s \rightarrow s'\}$
return(no)

Figure 1. a forward reachability algorithm

local attributes of the state s, such as e.g. the value of variables in state s. A classical example of reachability analysis is deadlock analysis, to see whether a system contains reachable states without outgoing transitions, which can be checked by taking GOAL= $\{s \in \quad | \quad st(s) = \emptyset\}$.

The sort of properties that can be analysed by reachability analysis can be substantially boosted beyond sets that can be characterised by propositional properties of states. To check for regular properties of system traces reachability analysis can be applied to the parallel composition of transition systems. The composition $T_1||T_2$ of two transition systems T_1 and T_2 with action sets L_1 and L_2, respectively, is the product of their respective state spaces with the transition relation defined the SOS inference rules given in Figure 2.

$$\frac{s_1 \rightarrow s_1'}{(s_1, s_2) \rightarrow (s_1', s_2)} \quad \in L_1 - L_2$$

$$\frac{s_2 \rightarrow s_2'}{(s_1, s_2) \rightarrow (s_1, s_2')} \quad \in L_2 - L_1$$

$$\frac{s_1 \rightarrow s_1', s_2 \rightarrow s_2'}{(s_1, s_2) \rightarrow (s_1', s_2')} \quad \in L_1 \cap L_2$$

Figure 2. SOS-rules for composition

Let be a deterministic automaton with actions in *Act*, i.e. a finite labelled transition system $_A, \quad t, \rightarrow_A$ with a set of states SUCCESS$\subseteq \quad _A$ such that for all s and their is at most one s' with $s \rightarrow s'$, accepting a trace language $\subseteq \quad t^*$. To model check property

$$(s) = \quad \{s \in \quad | \exists \sigma \in \quad : s \rightarrow\}$$

on a transition system T we carry out a reachability analysis on $T||$ with

$$\text{GOAL} = \{(s,t) \in \quad \times \quad _A | t \in \text{SUCCESS}\}$$

with INIT= $\{s\}$. In this context the automaton is sometimes referred to as the test automaton, cf. [3].

Much of model checking research has been devoted to checking properties for more powerful logics with LTL [32], CTL [20], and the modal -calculus [28] as important examples. LTL (linear temporal logic) can be used to characterize behaviour in terms of infinite traces. Instead of (test) automata, it uses Büchi automata to encode the acceptance conditions for infinite traces and reachability analysis of the strongly connected components of a transition system. The latter are subsets of system states in which each state is reachable from each other state, and that are closed under transitions, i.e. $st(\ ) \subseteq \quad$, see [38]. CTL (Computational Tree Logic) and the -calculus are so-called *branching-time* logics that characterize behaviour in terms of infinite trees of whose branches represent alternative behaviours and branching points the points of choice. Their model checking procedures essentially compute more sophisticated fixpoints over the state set, driven by recursive descent over the structure of the formulas [15].

We will confine ourselves essentially to reachability analysis, however. This has two reasons: first, most of what we have to say can be phrased in terms of reachability analysis, and obviates the need to deal with the complexities of more refined logical approaches. Second, experience indicates many practical problems can be dealt with effectively in this more restrictive setting. This can be understood informally as follows: a common division of system requirements that must be checked is that into *safety* and *liveness* properties. The former category states that "no bad behaviour will occur". This can often be readily translated into a reachability property: it can be checked whether an approprately chosen set of "bad" states states is reachable. The liveness properties, which express that "eventually some good behaviour will occur", in principle require more than just reachability, but in the setting of embedded systems many important liveness properties belong to the more restricted class of *bounded liveness* properties. This is a statement of the form "event will occur before event (or time t)". Such properties can be tested by reachability analysis, e.g. using test automata that keep track whether can occur without first having seen .

An interesting feature of most model checking algorithms is that in case of a goal state being reached they can produce *witnesses*. These are behaviour traces leading from an initial state to a goal state that it is reached. As the goal states are often 'bad' states that should in principle not be reachable in a safety analysis, such traces are also referred to as *counterexamples*. The capability to produce counterexamples is obtained by refining the algorithm of Figure 2 by storing the states in the set PASSED in a labelled tree data structure where node s is linked with label to node s' when it is established that $s \rightarrow s'$. This refinement of PART makes it correspond to a prefix of the 'unrolling' of the transition system that is being analysed. If a goal state

Proceedings of the Sixth International Workshop on Discrete Event Systems (WODES'02)
0-7695-1683-1/02 $17.00 © 2002 IEEE

is encountered the corresponding witness or counterexample is obtained by tracing back to the (a) root node of the tree. This refinement entails that the set WAITING consists not of states, but of 'detected transitions' $s \rightarrow s'$ from a leaf node s in PASSED. Implementing this set using a queue-like data type that is dequeued by the **remove some** operation in the algorithm obtains a *breadth-first* search strategy for state space exploration, using a stack-like data type similarly implements a *depth-first* strategy. Breath-first strategies are attractive because they always generate a shortest counterexamples, if any exist. Their disadvantage is that they are very expensive, generally storing (many) more transitions than depth-first strategies to find counterexamples.

The practically most limiting factor for model checking are the space requirements for storing the set/tree PASSED of states/transitions that have already been visited. Data types and algorithms to economize such space requirements, such as BDDs, bitstate hashing, partial-order search etc. [13, 25, 33], are therefore of vital interest for the practical application of model checking. Such methods, however, cannot be of any help in cases where there are in principle an unbounded number of states that need to be explored. This can be because the system model includes features such as unbounded message queues, or, more interesting in our context, state parameters of a continuous nature, such as time in real-time systems, or other physical quantities in hybrid systems in general.

Symbolic model checking adresses these issues by looking at finite representations of infinite state systems by lifting the transition relation from individual states to (infinite) sets of states for how to do this depend on the source of unboundedness in the model. For us it will be interesting to get an idea by looking at model checking algorithms for timed automata [4], a very popular model for representing real-time embedded systems. The model consist of a finite state machine or automaton modelling the finite control structure underlying the system at hand; it is extended with *clock variables* from a given set $\mathbb{C}$. The variables 'follow' time in the sense that their values increase with Δt when Δt time units elapse, but can be reset as the result of a control transition. Both control states or *locations* and transitions may be guarded by propositions over the set of clocks taken from a set $(\mathbb{C})$. For locations these propositions are known as *invariants* and state for what clock values the system is allowed to be in a particular location. For transitions they state under which clock conditions a transition may be taken.

Definition 2 *A timed automaton* $\quad$ *is a tuple* $L, \ , I, \ _0$ *with* L *a finite set of (control) locations;* $\subseteq L \times \ (\mathbb{C}) \times \ (\mathbb{C}) \times$ *the set of edges, where an edge contains a source location, a guard, a set of clocks to be reset, and a target location;* $I : L \rightarrow \ (\mathbb{C})$ *assigns* invariants *to locations; and* $_0 \in L$ *the* initial location. *We write* $\xrightarrow{r} \ '$ *instead of* $(\ , g, r, \ ') \in \ .$

It is clear that timed automata have an infinite underlying state space consisting of states of the form $\ , \bar{} \ ,$ where is a location and $\bar{}$ is a so-called clock assignment. A seminal result by Alur and Dill is that if the expressivity of the set of clock expression is carefully restricted (comparisons between natural numbers and clocks x or clock differences $x -$), reachability for timed automata becomes decidable. This is done by partitioning the space of clock assignments into so-called *regions* whose elements affect the control flow through guards and invariants in precisely the same manner. An improvement over the concept of region is that of a zone, which is used in Uppaal, and can be understood as the coarsest partioning that has this property. In Figure 3 there is a reformulation of the basic reachability algorithm of Figure 1 for timed automata with zones. $s, \ \rightsquigarrow \ s', \ '$ expresses states with location s and assignments in zone can reach states with location s' and assignments in $'$ either by letting time pass (delaying) or executing a control transition.

PASSED := $\emptyset$
WAITING := $\{ \ s_0, \ _0 \ \}$
while WAITING$\neq \emptyset$ **do**
  remove some $s, \quad$ **from** WAITING
  if $\exists \ g, \quad \in$ GOAL: $s = g \wedge \ \subseteq$
    then return(yes)
  if $\forall \ s, \ ' \in$ PASSED: $\not\subseteq \ '$
    then
    PASSED := PASSED $\cup \{ \ s, \ \}$
    WAITING := WAITING $\cup$
      $\{ \ s', \ ' \mid s, \ \rightsquigarrow \ s', \ ' \}$
return(no)

Figure 3. Abstract reachability with zones

Reachability analysis can also be applied to hybrid automata, but due to the more complicated nature of the dynamics of the continuous variables decidability cannot be obtained since the corresponding algorithm may not terminate, in contrast to that of Figure 3. Here safety analysis is often carried out using so-called *over-approximation*, by extending $\rightsquigarrow$ to a decidable version that makes more states reachable than are reachable in the actual system. If certain bad states are unreachable in an over-approximation then this result carries over to the original system. Examples can be found in [17, 21].

Proceedings of the Sixth International Workshop on Discrete Event Systems (WODES'02)
0-7695-1683-1/02 $17.00 © 2002 **IEEE**

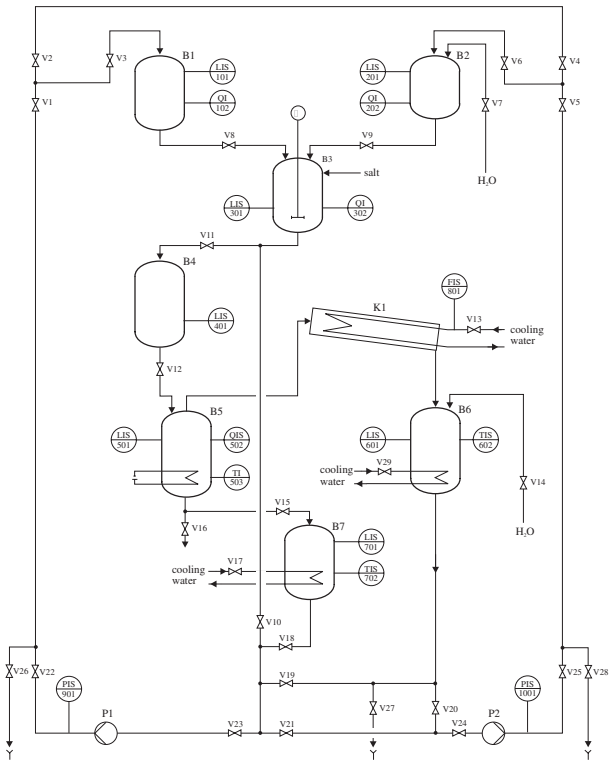

Figure 4. The P/I-Diagram of the Batch Plant

3 Example: verification and optimization of a controller

One of the case studies of the European VHS project was the design and verification of a controller for a simple chemical batch plant, as depicted in Figure 4, originally designed for student exercises. We describe its main features below; a more detailed account can be found in Kowalewski's description of the plant [27]. It turned out to be a very nice example for the application of model checking techniques.

The plant "produces" batches of diluted salt solution from concentrated salt solution (in container B1) and water (in container B2). These ingredients are mixed in container B3 to obtain the diluted solution, which is subsequently transported to container B4 and then further on to B5. In container B5 an evaporation process is started. The evaporated water goes via a condenser to container B6, where it is cooled and pumped back to B2. The remaining hot, concentrated salt solution in B5 is transported to B7, cooled down and then pumped back to B1.

The controlled batch plant is clearly a hybrid system. The discrete element is provided by the control program and the (abstract) states of the valves, mixer, heater and coolers (open/closed, on/off). Continuous aspects are tank filling levels, temperatures, and time. The latter can be dissected into real-time phenomena of the plant on the one hand, such as tank filling, evaporation, mixing, heating and cooling times, and the program execution and reaction times,on the other.

In [30, 11] we report on the use of the model checker SPIN [36] to the verification and optimization of a controller for this plant. SPIN is a state-of-the-art model checker. Its models are essentially labelled transition systems defined in the related modelling language Promela. As Promela does not cater for modelling continuous system parameters an adequate discrete model had to be devised. For volumes and temperatures only critical extremal values were considered; time was discretized and a variable time-advance technique [35] was used to avoid a state space explosion caused by having a huge domain of dicretized time points. This technique makes sure that only essential points in time (when something 'happens') are considered in reachability analysis, and not the uninteresting time states leading up to them.

In [30] we showed how a controller that was designed by careful analysis of the plant could be verified using this model. Because SPIN supports LTL model checking this could be done by verifying a formula that states that a state in which a batch is produced is reached infinitely often. However, because the behaviour of the plant is fully determined by the control actions, reachability analysis would also suffice to establish this by showing that a next batch producing state is reachable from a batch producing state (this involves adding a batch counter to the model to avoid a trivial solution in which such a state is reached by doing nothing).

In [12] it is subsequently shown how the controller may be optimized with respect to time, i.e. to look for a batch production schedule that takes a minimal time for producing a batch. The original controller design was such that it would always schedule a maximal non-conflicting set of events (i.e. not competing for the same resources). Now the controller model was made non-deterministic by allowing all non-empty subsets of non-conflicting events to be scheduled. Then reachability analysis was used to find witnesses for states where N batches had been produced before time T. By increasing N for fixed T each time a witness is found, the last witness trace that can be found contains the scheduling information for optimal batch production. Here, it is crucial to carefully tune the 'time horizon' T, which should be large enough not to prematurely cut off a possibly interesting witness, and should be small enough to allow effective searching of the state space. The latter has become potentially very large by the nondeterministic model of the controller.

Proceedings of the Sixth International Workshop on Discrete Event Systems (WODES'02)
0-7695-1683-1/02 $17.00 © 2002 IEEE

4 Minimal-cost guided model checking

The search for optimal schedules as explained above involves the explicit manipulation of the parameters N and T (and in fact an other parameter to control the degree of nondeterministic branching, see [11]), to control the effectiveness of the search for witnesses. Such guiding of the search strategy can be made more explicit by adding a notion of cost to the selection of alternatives. The process can then be guided by automatically selecting the cheapest alternative.

Figure 5 describes a refinement of the standard (symbolic) reachability algorithm using a notion of cost. It requires that symbolic states come equipped with *cost assignments* that determine the cost (s) for each state $s \in$. Moreover, it is assumed that for each priced symbolic state , the function $mi\ C\ st(\ ,\ ) = i\ f\{\ (s) | s \in \}$ can be determined effectively. Priced symbolic states can be ordered by defining , , if $\subseteq$ and (s) (s) for all $s \in$, which can be paraphrazed informally as , is "as big and cheap" as , .

```
COST :=
PASSED := ∅
WAITING := { s0, 0 }
while WAITING ≠ ∅ do
    remove ,  from WAITING
           with minimal minCost( , )
    if  ∩ GOAL ≠ ∅ ∧ mi C st( ∩ GOAL, )  COST
        then COST:=mi C st( ∩ GOAL, )
    if ∀ , ∈ PASSED: , / ,
        then
        PASSED := PASSED ∪ { , }
        WAITING := WAITING ∪
            { , | , ⇝ , }
return(COST)
```

Figure 5. Symbolic minimal-cost reachability

In [29] an instantiation of the above minimal-cost reachability algorithm is presented for timed automata, the linearly prices timed automata model, which has been implemented in the UPPAAL tool. Here, costs accumulate linearly with the residence time in locations with location-dependent rates, and with transition dependent costs for each transition taken. Minimal-cost reachability in this setting is shown to be decidable. An independent result along these lines can be cound in [5].

In [12] this approach is compared to the SPIN approach for finding optimal schedules. The search strategy that is actually employed uses so-called *heuristics*, a variant in which a separately defined variable is used to determine the priority with which states are removed from the WAITING list. In all except one case optimal schedules were easily detected this way, obviating the need for repeated runs with different parameter values, as with SPIN. It does need some experimentation, however, to find heuristics that induce the right search strategy. Details and applications to other case studies can be found in [21].

5 Scheduling synthesis

Given guided model checking strategies to evaluate promising parts of the reachability tree first, model checking techniques can also be used effectively to synthesize controllers. The simplest case is when the plant to be controlled is deterministic, in the sense that it either moves on its own to a uniquely determined next state, or makes a transition caused by a control action that is uniquely determined by that action. The VHS batch plant case study is an example of this. This kind of scheduling control, and the related field of *planning* is a rich source of problems [19, 26, 14, 1, 2].

In such cases one can do reachability analysis on the uncontrolled plant model, with plant states in which (several) batches have been produced as goal states. Guided search obtains witnesses that contain the necessary scheduling information. Because the uncontrolled plant allows for very many useless control scenarios (e.g. opening a valve under an empty container, opening and immediately closing the same valve, etc.), this usually requires some tweaking of the model and/or the search heuristics to get rid of improductive behaviours.

Using test automata to exclude the selection of already detected scheduling strategies can be used to find more general or alternative schedules. Using minimal-cost reachability schedules can be detected that satisfy some optimality criterion.

General supervisory control for discrete events systems [34] is closely related to reachability analysis on *game automata*. A game automaton is a labelled transition system , $t, \rightarrow$ with an action set $t =$ $\cup$ that is partitioned into *environment* (or uncontrolled) actions from and *controlled* actions from . Given a set of GOAL states one wishes to establish the subset $\subseteq$ of states in which the controller has a *winning strategy*, i.e. from which a state in GOAL can be reached no matter what enabled environment actions are selected on the way. This set can actually determined by *backward* reachability analysis, calculating the smallest fixpoint containing GOAL of

$$(\ ) = \ \cup\ r\ (\ ) \cup\ r\ (\ )$$

where

- $r\ (\ ) = \{s \in\ | \exists s' \in\ ,\ \in\ : s \rightarrow s'\}$
- $r\ (\ ) = \{s \in\ | \forall s' \in\ ,\ \in\ : s \rightarrow s'\}$

Proceedings of the Sixth International Workshop on Discrete Event Systems (WODES'02)
0-7695-1683-1/02 $17.00 © 2002 IEEE

Putting $G = \text{GOAL}$ this fixpoint can be calculated as the limit of the chain

$$G \subseteq \quad (G) \subseteq \ldots \subseteq \quad ^{i}(G) \subseteq \quad ^{i+1}(G) \subseteq \ldots$$

where for finite game automata the limit is obtained by stagnation for some finite index i. This can be done using an algorithm analogous to that of Figure 1.

A classical paper on the application of game theory and winning strategies to controller synthesis for timed automata is the one by Maler, Pnueli and Sifakis [31].

6 Research issues

As shown above there are fruitful applications of model checking techniques, especially reachability analysis, to the design and verification of embedded systems, varying from the analysis of given control systems to the synthesis of new controllers. Important issues that play a role in current research are:

- *Expressive model classes*: embedded systems often have to deal with a combination of extensions of the transition system paradigm along different modelling dimensions, such as real-time behaviour, uncertainty (nondeterminism and stochastic behaviour), and a notion of cost and/or rewards. The challenge is to be able to have models that can be effectively model checked in all useful areas of this multi-dimensional space. These models must also have efficient machine representations in the form of clever symbolic data structures, such as zones, BDDs, clock difference diagrams, etc.

  It is especially challenging to see whether results that have been obtained in the context of (optimal) control and timed automata can be extended to models that include stochastic features. In this respect it is interesting to know that model checking techniques are being developed for the machine-assisted evaluation of performability measures of continuous time Markov chains and decision processes [23, 24]. There is also work being done on the incorporation of probabilistic choice in the UPPAAL framework [18].

- *Compositionality*: to combat the state space explosion effectively it will be necessary to do as much of the calculations on the smaller local state spaces of component processes as possible, instead of on the potentially huge global state space of a system. It is therefore interesting to see to what extent compistional model checking techniques such as CBR (compositional backwards reachability) [8] carry over to the (optimal) reachability problems discussed above.

- *Optimality and guided search*: cost-driven reachability analysis can also be applied to the game theoretic approach control synthesis described above [10]. Because of the huge state spaces of most practical application, guided search techniques are almost always needed to obtain results with a reasonable use space and time resources. A much better understanding is needed of effective search heuristics and how they are best represented.

- *Abstractions*: abstraction fights the state space explosion by replacing a (too) large model with a simpler one that preserves all properties of interest. There is a well-establised body of work in classical model checking that relates abstractions to classes of logical formulas that represent properties of interest. Abstractions can also be developed that preserve winning strategies, also in the presence of cost functions [10].

- *Modelling methodology*: this practically important aspect of applications is often overlooked. As was reported here the VHS batch plant case study showed that good results could be obtained using a non-symbolic model checker (SPIN) on a hybrid control problem. Clever modelling techniques may compensate for less advanced modelling features, and in fact produce more manageable state spaces. There is a lack of systematic understanding of how qualitative features such as ease of modelling, size of model, precision of results, maintainability, verification speed, etc. relate to different modelling classes and styles. The current practice is best described by the slogan "model hacking preceedes model checking".

- *Verification vs. testing*: the drive to obtain models that can be used effectively for analysis via model checking techniques leads to the application of aggressive abstraction techniques to obtain the needed compactification and simplification. This may easily lead to models that no longer preserve all relevant properties of the actual, implemented system. Testing is the validation method that addresses a system at the level of its physical implementation, and that can be used to check whether undue abstractions have been made. It is important to understand how model-based testing methods may be used to systematically complement verification by model checking, and help to validate model cheking models [9].

The IST project AMETIST [6] adresses these and other issues in an effort to develop a full-blown methodology for the design and analysis the of real-time embedded systems with a special focus (optimal) scheduling and resource allocation problems.

Proceedings of the Sixth International Workshop on Discrete Event Systems (WODES'02)
0-7695-1683-1/02 $17.00 © 2002 IEEE

Acknowledgements

Much of the contents of the paper result from joint work and discussions with Kim Larsen. The application of heuristic search methods to the batch plant example was doen in collaboration with Ansgar Fehnker.

References

[1] Y. Abdeddaïm and O. Maler. Job-shop scheduling using timed automata. In *Proceedings CAV 2001*, volume 2102 of *Lecture Notes in Computer Science*, pages 478–492. Springer-Verlag, 2001.

[2] Y. Abdeddaïm and O. Maler. Pre-emptive job-shop scheduling using stopwatch automata. In *Proceedings TACAS 2002*, volume 2280 of *Lecture Notes in Computer Science*, pages 113–126. Springer-Verlag, 2002.

[3] L. Aceto, P. Bouyer, A. B. no, and K. Larsen. The power of reachability testing for timed automata. In *Proceedings FST & TCS'98*, volume 1530 of *Lecture Notes in Computer Science*, pages 245–256. Springer-Verlag, 1998.

[4] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 138:183–335, 1994.

[5] R. Alur, S. L. Torre, and G. Pappas. Optimal paths in weighted timed automata. In *Fourth International Workshop on Hybrid Systems: Computation and Control*, number 2034 in Lecture Notes in Computer Science, pages 49–62, 2001.

[6] AMETIST homepage. `http://ametist.cs.utwente.nl/`.

[7] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. In *Proceedings IFAC Symposium on System Structure and Control*, pages 469–474. Elsevier, 1998.

[8] G. Behrmann, K. Larsen, H. Andersen, H. Hulgaard, and J. Lind-Nielsen. Verification of large state/events systems using compositionslity and dependency analysis. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1579 of *Lecture Notes in Computer Science*, pages 163–177. Springer-Verlag, 1999.

[9] E. Brinksma. Verification is experimentation! *Software Tools for Technology Transfer*, 3(2):107–111, 2001.

[10] E. Brinksma and K. Larsen. From reachability to optimal control, 2002. unpublished manuscript.

[11] E. Brinksma and A. Mader. Verification and optimization of a PLC control schedule. In *Proceedings of SPIN2000*, volume 1885 of *Lecture Notes in Computer Science*. Springer, 2000.

[12] E. Brinksma, A. Mader, and A. Fehnker. Verification and optimization of a plc control schedule. *Software Tools for Technology Transfer*, 2002. accepted for publication.

[13] R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

[14] A. Cimatti, E. Giunchiglia, F. Giunchiglia, and P. Traverso. Planning via model checking: A decision procedure for ar. In *Proceedings 4th European Conference on Planning (ECP-97)*, volume 1348 of *Lecture Notes in Artificial Intelligence*, pages 130–142. Springer-Verlag, 1997.

[15] E. Clarke, E. Emerson, and A. Sistla. Automatic verification of finite-state concurrent systems using trmporal logic specifications. *ACM TOPLAS*, 8(2):244–263, 1986.

[16] E. Clarke and J. Wing. Formal methods:state of the art and future directions. Technical report, Strategic Directions in Computing Research, Formal Methods Working Group, august 1996.

[17] T. Dang and O. Maler. Reachability analysis via face lifting. In *Hybrid Systems: Computation and Control*, volume 1386 of *Lecture Notes in Computer Science*, pages 96–109. Springer-Verlag, 1998.

[18] P. D'Argenio, B. Jeannet, H. Jensen, and K. Larsen. Reachability analysis of probabilistic systems by successive refienements. In *Proceedings PAPM-PROBMIV 2001*, volume 2165 of *Lecture Notes in Computer Science*, pages 29–56. Springer-Verlag, 2001.

[19] H. Dierks, G. Berhrmann, and K. Larsen. Solving planning problems using real-time model-checking (translating pddl3 into timed automata). In *Proceedings AIPS*, 2002. to appear.

[20] E. Emerson and E. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2:241–266, 1982.

[21] A. Fehnker. *Citius, Vilius, Melius*. PhD thesis, University of Nijmegen, 2002.

[22] R. Gerth. Model checking if your life depends on it: a view from intel's trenches. In M. Dwyer, editor, *Model Checking Software, Proceedings 8th International SPIN Workshop*, volume 2057 of *Lecture Notes in Computer Science*, page 15. Springer-Verlag, 2001.

[23] B. Haverkort, L. Cloth, H. Hermanns, J.-P. Katoen, and C. Baier. Model checking performability properties. In *Proceedings DSN 2002*. IEEE CS Press, 2002.

[24] H. Hermanns and J.-P. Katoen. Performance evaluation:=(process algebra + model checking) x markov chains. In *Proceedings CONCUR 2001*, volume 2165 of *Lecture Notes in Computer Science*, pages 59–81. Springer-Verlag, 2001.

[25] G. Holzmann. An improved protocol reachability analysis technique. *Software-Practice and Experience*, 18(2):137–161, 1988.

[26] T. Hune, K. G. Larsen, and P. Pettersson. Guided Synthesis of Control Programs using UPPAAL. *Nordic Journal of Computing*, 8(1):43–64, 2001.

[27] S. Kowalewski. Description of case study CS1 "experimental batch plant". `http://www-verimag.imag.fr/VHS/main.html`, July 1998.

[28] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[29] K. G. Larsen, G. Behrmann, E. Brinksma, A. Fehnker, T. Hune, P. Pettersson, and J. Romijn. As cheap as possible: Efficient cost-optimal reachability for priced timed automat. In G. Berry, H. Comon, and A. Finkel, editors, *Proceedings of CAV 2001*, number 2102 in Lecture Notes in Computer Science, pages 493–505. Springer–Verlag, 2001.

[30] A. Mader, E. Brinksma, H. Wupper, and N. Bauer. Design of a PLC control program for a bach plant, VHS case study 1. *European Journal of Control*, 7(4):416–454, 2001.

Proceedings of the Sixth International Workshop on Discrete Event Systems (WODES'02)
0-7695-1683-1/02 $17.00 © 2002 **IEEE**

[31] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In E. Mayr and C. Puech, editors, *Proceedings STACS '95*, volume 900 of *Lecture Notes in Computer Science*, pages 229–242. Springer-Verlag, 1995.

[32] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.

[33] D. Peled. Combining partial-order reductions with on-the-fly model checking. *Formal Methods in System Design*, 8:39–64, 1996.

[34] P. Ramadge and W. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal of Control and Optimization*, 25(1):206–230, 1987.

[35] G. Shedler. *Regenerative Stochastic Simulation*. Academic Press, 1993.

[36] SPIN homepage. `http://wwwnetlib.bell-labs.com/netlib/spin/whatispin.html`.

[37] J. Tretmans, K. Wijbrans, and M. Chaudron. Development of a storm surge barrier controil system: Revisiting seven myths of formal methods. *Formal Methods in System Design*, 19(2):195–215, 2001.

[38] M. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115:1–37, 1994.

Proceedings of the Sixth International Workshop on Discrete Event Systems (WODES'02)
0-7695-1683-1/02 $17.00 © 2002 **IEEE**