MODULAR ANALYSIS OF DISCRETE CONTROLLERS FOR DISTRIBUTED HYBRID SYSTEMS

Goran Frehse * **Olaf Stursberg** * **Sebastian Engell** *
Ralf Huuck ** **Ben Lukoschus** **

* *Process Control Lab (CT-AST), University of Dortmund, D-44221 Dortmund (Germany), g.frehse@ct.uni-dortmund.de*
** *Chair of Software Technology, Institute of Comp. Science and Applied Mathematics, University of Kiel, D-24098 Kiel (Germany), {rhu,bls}@informatik.uni-kiel.de*

Abstract: The algorithmic analysis of control systems for large and distributed hybrid systems is considerably restricted by its computational complexity. In order to enable the verification of discrete controllers for such hybrid systems, this contribution proposes an approach that combines decomposition, model checking and deduction. The system under examination is first decomposed into a set of modules represented by communicating linear hybrid automata. The *Assumption/Commitment* method is used to to prove properties of coupled modules and to derive conclusions about the behavior of the entire system. The individual Assumption/Commitment-pairs are proven using established methods for model checking.

Keywords: Abstraction, Assumption/Commitment, Discrete Controllers, Model Checking, Verification

1. INTRODUCTION

To ensure the proper operation of a discrete controller in connection with a hybrid system, a formal investigation, also referred to as verification, can be used to obtain conservative and correct results. First, formal models of both the controller and the plant are developed, then the criteria for a proper operation are described as safety or liveness properties of the models. Various techniques to formally verify such properties for discrete event systems as well as timed and hybrid systems have been developed within the last decade. These techniques can roughly be divided into *theorem proving* and *model checking*. While theorem proving aims at inferring properties by deduction in a mostly manual procedure, the latter performs an algorithmic search through the state space of a transition system in order to detect whether states are reachable, in which a required property is violated. Model checking plays a dominant role in particular for the analysis of hybrid systems, and successful applications have been reported for systems with complex (nonlinear) continuous dynamics, see e. g. (Lynch and Krogh, 2000). However, these applications are restricted to systems with a small number of components and continuous variables due to an inherently large computational effort.

If a large hybrid system exhibits a distributed structure, verification techniques can still be applied using a modular approach as shown in Fig. 1. The system is first decomposed into a set of modules $(M_1, \ldots, M_n)$ that represent physical or functional units. The size of the modules is chosen such that model checking is possible for the composition of a small number of modules, and their behavior is modelled by linear hybrid automata $(S_1, \ldots, S_n)$. The idea is then to show local properties for each module by model checking and to combine the local results by deduction in order to derive a global property of the complete system. The crucial step of this approach is to verify local properties for each module individually using justified assumptions about its environment – here the *Assumption/Commitment* method is employed, as introduced in (Clarke and Emerson, 1982; Queille and Sifakis, 1982). A required property of a single module M_j is first verified by model checking for a likely (or desired) behavior of the module's environment. Hence, the automaton S_j commits itself to the required property (the *commitment* c_j) if the environment behaves according to reasonable restrictions (the *assumptions* a_j). In a second step, it has to be checked whether the environment indeed behaves as assumed, taking into account that the considered module S_j reacts according to the commitment. This step is performed by deductively combining the Assumption/Commitment pairs for all modules. The result is a statement about the global property of the complete system: $S \models (a, c)$.

Different approaches to a decompositional analysis based on the principle of Assumption/Commitment can be found in literature. Depending on the underlying formalism they are called *Rely/Guarantee*, *Assumption/Commitment* or *Assume/Guarantee* (Misra and Chandy, 1981; Jones, 1981; Pnueli, 1984; Abadi and Lamport, 1995). However, most of them were applied to discrete systems and still few applications to real time and hybrid systems have been reported (Hooman, 1997; Chang *et al.*, 1994; Alur and Henzinger, 1997; Henzinger *et al.*, 1998; Henzinger *et al.*, 2000). In difference to those approaches, this contribution aims at developing an analysis strategy that can be used within the design procedure for discrete controllers of distributed hybrid systems. Particularly, manufacturing and processing systems are characterized by a number of interacting processing units with hybrid behavior. Control systems for such processes usually comprise a number of local discrete controllers but also supervisory controllers that coordinate the interaction of different process parts on a more abstract level. The analysis of a controller design for these processes based on the proposed decompositional approach seems promising for two reasons: First, the structure of the control systems as well as the process usually has a modular nature such that the decomposition is obvious. Second, a module often interacts only with a small number of other modules, and thus its environment can be described using a sufficiently small model. Both criteria are essential for a successful application of the proposed approach.

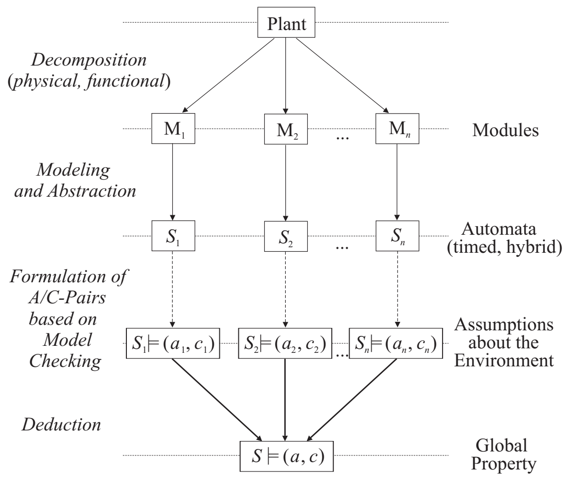

Fig. 1. Analysis Procedure

2. MODELING WITH COMMUNICATING LINEAR HYBRID AUTOMATA

Given a decomposition of the controlled process, an important step is to model the modules such that (i) the communication between the individual modules can be represented, and (ii) that the model can be analyzed by model checking. In (Alur *et al.*, 1995), *Linear Hybrid Automata* (LHA) are defined as a model for linear hybrid systems for which implementations of model checking algorithms are available. A LHA combines a state transition structure with continuous dynamics defined by differential inclusions. The communication between different LHA is modelled by synchronization on common synchronization labels. For our purposes, the LHA model lacks two important features: The communication is not directed, i.e. it is not possible to distinguish between sending and receiving automata. Secondly, all continuous variables are shared, so there is no possibility to specify explicitly that one LHA can assign new variable values which others can only read.

2.1 *The Definition of CLHA*

Therefore *Communicating Linear Hybrid Automata* (CLHA) are defined as an extension of LHA with input and output variables, and labels which distinguish between directed, i.e. sending or receiving, and undirected, i.e. synchronizing, communication: A CLHA is a 6-tuple

$$\mathcal{A} = (Loc, Var, Lab, Edg, Act, Inv)$$

with:

- a finite set of locations $Loc = \{q_1, \ldots, q_p\}$.
- a finite set of real variables $Var = Var_{in} \cup Var_{int}$ comprising two disjoint sets of input and internal variables. A subset $Var_{out} \subseteq Var_{int}$ defines the output variables. A valuation ν (contained in a valuation set V) is a function

which assigns a real value to each variable $x \in Var$, i.e., $\nu(x) \in \mathbb{R}$. A state of $\mathcal{A}$ is a pair (q, ν) of a location and a valuation.

- a finite set $Lab = Lab_{rec} \cup Lab_{send} \cup Lab_{sync}$ that consists of three disjoint sets of symbolic labels, called *receive labels*, *send labels* and *synchronization labels*.
- a finite set Edg of discrete transitions. Each transition $e = (q, l, \rho, q')$ between two locations $q, q' \in Loc$ depends on a label $l \in Lab \cup \{\tau\}$, with τ denoting an internal transition, and an enabling transition relation $\rho \subseteq V \times V$. The transition e is enabled in state (q, ν) iff a valuation ν' with $(\nu, \nu') \in \rho$ exists. ν' denotes the evaluation that results from the transition taken in state (q, v). It is required that $\nu(x) = \nu'(x)$ for all $x \in Var_{in}$, since input variables cannot be changed by discrete transitions. For any location $q \in Loc$ there must be a special internal transition $(q, \tau, \{(\nu, \nu) | \nu \in V\}, q) \in Edg$, called "stutter transition".
- A labeling function $Act : Loc \times Var_{int} \to \mathbb{R}$ that denotes the rate of change of the int. variable x in location q: $Act(q, x) \in [k_1, k_2]$, $k_1, k_2 \in \mathbb{R}$.
- a labeling function $Inv : Loc \to 2^V$ assigning an invariant $Inv(q) \subseteq V$ to each location $q \in Loc$.

Informally, the behavior of $\mathcal{A}$ can be understood as follows: Starting from an initial state (q_0, v_0) the automaton remains in the current location until a transition is taken. The continuous evolution within a location is determined by the activities assigned to the internal variables. A transition must be taken before the invariant is evaluated to be false. The transitions depend on the current values of the input and internal variables, and on the synchronizatin labels in Lab_{rec} and Lab_{sync}: A *run* of $\mathcal{A}$ starting at state (q_0, ν_0) is a finite or infinite sequence

$$(q_0, \nu_0) \xrightarrow{l_0}_{t_0} (q_1, \nu_1) \xrightarrow{l_1}_{t_1} (q_2, \nu_2) \xrightarrow{l_2}_{t_2} \dots$$

of states (q_i, ν_i) with $l_i \in Lab \cup \{\tau\}$, $t_i \in \mathbb{R}_{\geq 0}$, if the following conditions apply:

(1) For all $0 \leq t \leq t_i$ there exists a $\nu \in Inv(q_i)$ such that for all $x \in Var_{int}$, $\nu_i(x) + Act(q_i, x) \cdot t = \nu(x)$.
(2) There exists a valuation $\nu^* \in Inv(q_i)$ with $\nu^*(x) = \nu(x) + Act(q_i, x) \cdot t_i$ for all $x \in Var_{int}$ such that $(q_i, l_i, (\nu^*, \nu_{i+1}), q_{i+1}) \in Edg$.

2.2 *The Composition of CLHAs*

Given two CLHA ($i \in \{1, 2\}$)

$$\mathcal{A}_i = (Loc_i, Var_i, Lab_i, Edg_i, Act_i, Inv_i)$$

with variable sets Var^i_{in}, Var^i_{int}, Var^i_{out} and label sets Lab^i_{rec}, Lab^i_{send}, Lab^i_{sync}, and provided that $Var^1_{int} \cap Var^2_{int} = \emptyset$, and $Lab^i_{send} \cap (Lab^j_{send} \cup Lab^j_{sync}) = \emptyset$, $Lab^i_{rec} \cap Lab^j_{sync} = \emptyset$, for $\{i, j\} = \{1, 2\}$, the *parallel composition* of $\mathcal{A}_1$ and $\mathcal{A}_2$

$$\mathcal{A}_1 || \mathcal{A}_2 = (Loc, Var, Lab, Edg, Act, Inv)$$

is a CLHA with:

- locations $Loc = Loc_1 \times Loc_2$,
- variable sets $Var_{in} = (Var^1_{in} \setminus Var^2_{out}) \cup (Var^2_{in} \setminus Var^1_{out})$, $Var_{out} = Var^1_{out} \cup Var^2_{out}$, $Var_{int} = Var^1_{int} \cup Var^2_{int}$,
- labeling sets $Lab_{rec} = (Lab^1_{rec} \setminus Lab^2_{send}) \cup (Lab^2_{rec} \setminus Lab^1_{send})$, $Lab_{send} = Lab^1_{send} \cup Lab^2_{send}$, $Lab_{sync} = Lab^1_{sync} \cup Lab^2_{sync}$,
- an activity function
  $Act((q_1, q_2), x) = Act_i(q_i, x)$ if $x \in Var^i_{int}$,
- an invariant assignment $Inv : Loc \to 2^V$ with $Inv((q_1, q_2)) = \{\nu \in V | \exists \nu_1 \in Inv_1(q_1), \nu_2 \in Inv_2(q_2) : \nu = \nu_1 \cup \nu_2 \wedge \forall x \in Var_1 \cap Var_2 : \nu_1(x) = \nu_2(x)\}$, and
- $((q_1, q_2), l, \rho, (q'_1, q'_2)) \in Edg$ iff there exist $l_1 \in Lab_1$ and $l_2 \in Lab_2$ such that:
  (1) $(q_i, l_i, \rho_i, q'_i) \in Edg_i$, where ρ_i is the projection of ρ on the variables of $\mathcal{A}_i$, for $i \in \{1, 2\}$, and
  (2) the labels l_1, l_2, and l are one of the following combinations with $\{i, j\} = \{1, 2\}$:
    · $l = l_1 = l_2 = \tau$,
    · $l = l_i \in Lab^i_{rec} \setminus Lab^j_{send}$, $l_j = \tau$,
    · $l = l_1 = l_2 \in Lab_{rec}$,
    · $l = l_i \in Lab^i_{send}$, $l_j = \tau$,
    · $l = l_i = l_j \in Lab^i_{send} \cap Lab^j_{rec}$,
    · $l = l_i \in Lab^i_{sync} \setminus Lab^j_{sync}$, $l_j = \tau$,
    · $l = l_1 = l_2 \in Lab_{sync}$.

3. MODULAR ANALYSIS BASED ON THE ASSUMPTION/COMMITMENT METHOD

Consider the behavior of a module S_i. Let

$$S_i \models (a_i, c_i), \tag{1}$$

denote that S_i commits itself to fulfilling the *commitment* c_i under the *assumption* a_i. The pair (a_i, c_i) is called an *assumption/commitment-pair* (a/c-pair). A/c-pairs must be found for all modules such that their deductive combination guarantees the fulfillment of a global pair (a, c):

$$\frac{\begin{array}{c} S_1 \models (a_1, c_1) \\ \vdots \\ S_n \models (a_n, c_n) \\ B(a_1, \dots, a_n, c_1, \dots, c_n, a, c) \end{array}}{S_1 || S_2 || \dots || S_n \models (a, c)}. \tag{2}$$

In addition to the a/c-pairs, logical conditions B are needed in order to connect assumptions and commitments and to break circularity. While for autonomous systems a is usually required to be *true*, many technical systems depend on outside resources or human interaction which can be represented by an appropriate assumption a. The main difficulty is to find appropriate a/c-pairs for all modules such that their deductive combination guarantees the fulfillment of the global pair (a, c). Choosing the assumption/commitment pairs is the creative work of the modeler and can only partially be automated.

3.1 *Application Using Automata*

The proof of relation (1) can be automated using model checking (Grumberg and Long, 1991). The assumptions a_i are modeled by automata A_i and the commitments c_i are expressed by temporal specifications, e.g., CTL formulae. Alternatively, a test automaton C_i^T can be constructed which includes a fail state that is reachable if and only if c_i is not fulfilled. All automata are specified as communicating linear hybrid automata. Then, the reachability of the fail state is checked to show that:

$$S_i||A_i||C_i^T \models \neg reach(fail) \Rightarrow S_i \models (a_i, c_i). \quad (3)$$

The following two sections propose a/c-pairs on the basis of established proof rules. Both rules are presented on the basis of abstractions, where the desired behavior of a module S_i is specified by an abstraction automaton $\hat{S}_i$. The formula $S_i \preceq \hat{S}_i$ denotes that S_i meets the specification, with $\preceq$ denoting simulation. This means that any behavior of S_i can be matched by a corresponding behavior of $\hat{S}_i$ (but not vice versa). The a/c pair from (3) can be expressed as the following abstraction:

$$S_i||A_i \preceq C_i||A_i, \quad (4)$$

where C_i contains all behaviors of S_i fulfilling c_i.

3.2 *Circular Assumption/Commitment*

The following proof rule, also referred to as Assume/Guarantee rule, has successfully been applied to small real-time and hybrid systems (Henzinger *et al.*, 1998). In order to verify that $S_i||\ldots||S_n$ meets the specifications $\hat{S}_i||\ldots||\hat{S}_n$ the following proof is carried out:

$$\frac{\begin{array}{c} S_1||\hat{S}_2||\ldots||\hat{S}_{n-1}||\hat{S}_n \preceq \hat{S}_1||\hat{S}_2||\ldots||\hat{S}_n \\ \hat{S}_1||S_2||\ldots||\hat{S}_{n-1}||\hat{S}_n \preceq \hat{S}_1||\hat{S}_2||\ldots||\hat{S}_n \\ \vdots \\ \hat{S}_1||\hat{S}_2||\ldots||\hat{S}_{n-1}||S_n \preceq \hat{S}_1||\hat{S}_2||\ldots||\hat{S}_n \\ B(S_1,\ldots,S_n,\hat{S}_1,\ldots,\hat{S}_n) \end{array}}{S_1||S_2||\ldots||S_{n-1}||S_n \preceq \hat{S}_1||\hat{S}_2||\ldots||\hat{S}_n}. \quad (5)$$

Again, additional conditions B are needed to avoid that the composition of the original modules shows a behavior that can't be met by more than one of the abstractions, in which case the proof would fail. With the following definition for A_i and C_i, the constituents of (5) can be obtained from (4):

$$A_i = \hat{S}_1||\ldots||\hat{S}_{i-1}||\hat{S}_{i+1}||\ldots||\hat{S}_n, C_i \preceq \hat{S}_i. \quad (6)$$

3.3 *Chain Rule Assumption/Commitment*

In a chain rule form, the Assumption/Commitment proof becomes simple and requires no further additional logical conditions or explicit deduction:

$$\frac{\begin{array}{c} S_1 \preceq \hat{S}_1 \\ \hat{S}_1||S_2 \preceq \hat{S}_1||\hat{S}_2 \\ \vdots \\ \hat{S}_1||\hat{S}_2||\ldots||\hat{S}_{n-1}||S_n \preceq \hat{S}_1||\hat{S}_2||\ldots||\hat{S}_n \end{array}}{S_1||S_2||\ldots||S_{n-1}||S_n \preceq \hat{S}_1||\hat{S}_2||\ldots||\hat{S}_n}. \quad (7)$$

It can be interpreted in the following way: $\hat{S}_1$ has to capture the behavior of S_1 for all possible inputs. $\hat{S}_2$ has to simulate S_2 with the inputs from $\hat{S}_1$, which is easier than with all possible inputs. For the last module $\hat{S}_n$, only the behavior occurring under the influence of $\hat{S}_1||\ldots||\hat{S}_n$ has to be taken into account. The proof of (7) is straightforward and can be done by iteratively applying the equations to their successors. This rule is simple, but in the following sense, it can't be improved:

- Adding a term $\hat{S}_i$ to both sides of one of the equations will destroy the soundness unless further conditions are included.
- Removing a term $\hat{S}_{i+1}$ will lead to a wider range of inputs that S_i will have to cooperate with.

Let A denote an automaton modeling a global assumption as part of the initial conditions. The automata A_i and C_i become for $i > 1$:

$$A_1 = A, \quad A_i = \hat{S}_1||\ldots||\hat{S}_{i-1}, \quad C_i \preceq \hat{S}_i. \quad (8)$$

In order to reduce the complexity of the proof steps, the assumption can be widened, i.e. for $j < i$ any $\hat{S}_j$ can be dropped from both sides of (8) at any step. This however might lead to an abstraction that is too wide and violates one of the proof steps. If the proof fails because the interactions of the modules cannot be captured by the abstractions in a chain sequence, the assumption should be made more restrictive by adding any S_j, $j > i$, to both sides of (8) at any step. This in turn will increase the complexity.

4. ANALYSIS OF A BUFFER TANK

Consider a tank with an inlet valve and a constant outflow. A controller is responsible for opening the inlet valve while ensuring that the buffer tank neither overflows nor empties completely. The verification task is to check whether the level stays within the limits:

$$0 < x < x_{max} \tag{9}$$

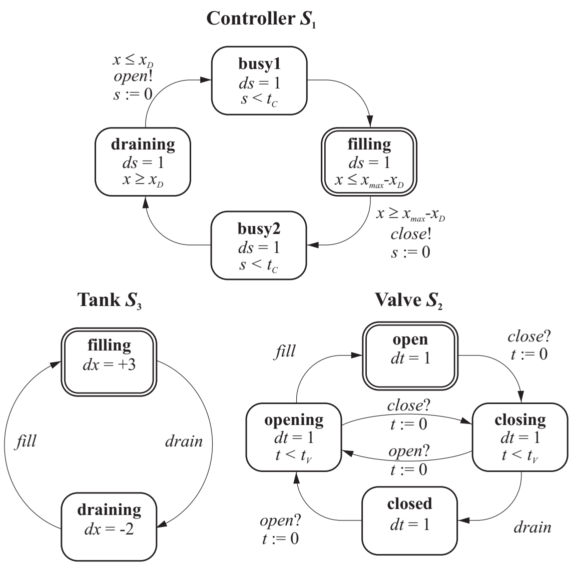

Fig. 2. Automata for controller, valve and tank

Fig. 2 shows the models for the tank, its inlet valve and the controller in the form of CLHA. The graphical representation is as follows: The vertices are labelled with the invariant and the rates of the continuous variables. The transitions are labelled with receive ('?'), send ('!') or synchronization labels (no additional marking), and with assignments for the continuous variables. Due to the constant outflow, the tank level can be modelled with a rate of $r_1 = 3$ m min^{-1} if the inlet valve is open and $r_2 = -2$ m min^{-1} if it is closed. The inlet valve synchronizes with the tank on the labels *drain* and *fill*, and it opens or closes with a delay of t_V after receiving a corresponding signal from the controller. The controller acts when the level is within a distance of x_D to one of the limits. After each action, the controller interacts with a high level controller for at most t_C seconds. This is modelled by including busy states.

Model checking was carried out using the tool HyTech (Henzinger *et al.*, 1997). The CLHA were converted to LHA by adding appropriate loops to represent the behavior of sending and receiving labels. The first step starts with the controller S_1. The expected behavior is modelled as the commitment automaton C_1^T, as shown in Fig. 3. Since the continuous variable x is needed for the analysis, a generator automaton X_1 has been included that comprises all possible rates of x. It changes its sign when the controller sends the *open*! and *close*! commands according to the commitment. If it doesn't, the commitment automaton can reach the fail state. The validity of the generator can

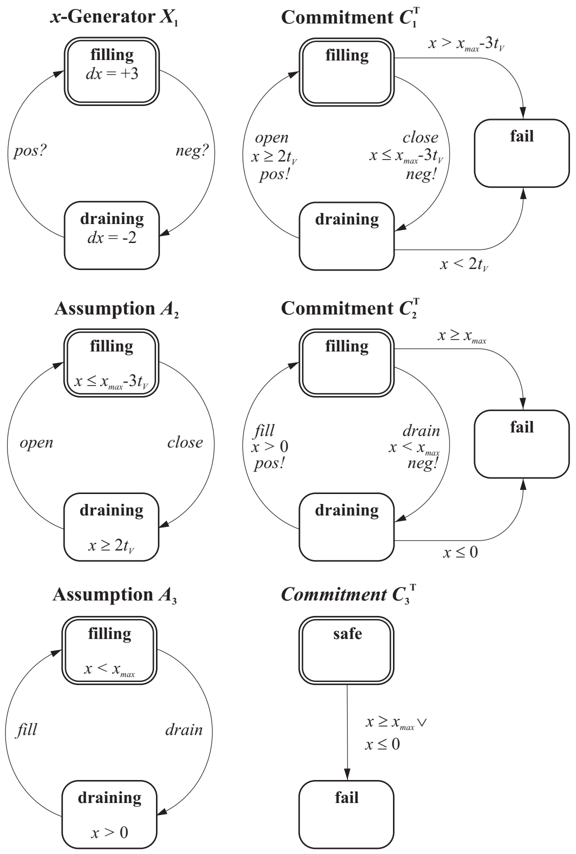

Fig. 3. Assumption and commitment automata

be considered as the global assumption A. Model checking gives the result that the commitment is fulfilled if

$$r_1 t_V \le x_D < x_{max} - r_1 t_C. \tag{10}$$

The second step combines the valve automaton S_2 with the assumption $A_2 = C_1$. The commitment of the valve must be to provide the $fill$ and $drain$ labels in time. Again, the generator automaton for x is needed to provide the possible behaviors of x. Model checking leads to the result that with (10) the commitment is fulfilled. The third step combines the tank automaton S_3 with the assumption $A_3 = C_2$. The commitment is to fulfill (9), which can be tested using the automaton C_3^T or by directly specifying the formula in the model checker. Again, model checking validates the commitment. Finally, it can be deduced in a chain rule fashion as follows:

$$\frac{\begin{array}{c} S_1 \models (a, c_1), \\ S_2 \models (a_2, c_2) \\ S_3 \models (a_3, c_3), \\ a_2 = c_1 \wedge a_3 = c_2 \end{array}}{S_1 || S_2 || S_3 \models (a, c_3)}. \tag{11}$$

The tank level will remain within the limits if the parameters comply with (10).

5. CONCLUSIONS

This contribution presents an approach to verify properties of distributed hybrid systems by applying the Assumption/Commitment method. Instead of performing a single analysis for the complete system, the system is partitioned into small modules, local properties are analyzed by model checking, and global properties are derived by deduction. As demonstrated for an example, the method reduces the costly step of model checking to the composition of relatively simple systems. The successful application of the approach depends on two factors: (i) The system must be decomposed into small modules, preferably such that each module is only affected by a small number of other components. (ii) Appropriate abstractions of each module have to be found. It seems that these requirements can often be met when discrete controllers for processing or manufacturing systems are considered.

The class of CLHA was found to be appropriate to model the behavior of many modules of such systems. The extension of ordinary LHA by different types of variables and synchronization labels facilitates modelling and helps to prevent modelling errors. However, the CLHA are so far mapped into LHA to allow the analysis with existing model checking tools. If the behavior of a module cannot be modelled with sufficient accuracy by CLHA, hybrid automata with more complex continuous dynamics should be used. Techniques to approximate them by LHA are available, see e.g. (Stursberg, 2000). An application to a chemical process has shown promising results (Frehse *et al.*, 2001).

ACKNOWLEDGEMENT

This investigation is financially supported by the German Research Foundation (DFG) under grants KO 1430/6-1 and LA 1012/5-1.

6. REFERENCES

Abadi, M. and L. Lamport (1995). Conjoining specifications. *ACM Trans. Progr. Lang. and Syst.* **17**(3), 507–534.

Alur, R. and T.A. Henzinger (1997). Modularity for timed and hybrid systems. In: *Proc. of the 8th Int. Conf. on Concurrency Theory*. Vol. 1243 of *LNCS*. Springer. pp. 74–88.

Alur, R., C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis and S. Yovine (1995). The algorithmic analysis of hybrid systems. *Theoretical Comp. Science* **138**(1), 3–34.

Chang, E., Z. Manna and A. Pnueli (1994). Compositional verification of real-time systems. In: *Proc. 9th IEEE Symp. Logic in Computer Science*. pp. 458–465.

Clarke, E.M. and E.A. Emerson (1982). Design and synthesis of synchronization skeletons for branching time temporal logic. In: *Logics of Programs Workshop* (Dexter Kozen, Ed.). Vol. 131 of *LNCS*. Springer. pp. 52–71.

Frehse, G., O. Stursberg, S. Engell, R. Huuck and B. Lukoschus (2001). Verification of hybrid controlled processing systems based on decomposition and deduction. In: *Proc. 16th IEEE Int. Symp. Intelligent Control*.

Grumberg, O. and D. Long (1991). Model checking and modular verification. In: *Proc. Int. Conf. on Concurrency Theory*. Vol. 527 of *LNCS*. Springer. pp. 250–265.

Henzinger, T.A., P.-H. Ho and H. Wong-Toi (1997). Hytech: A model checker for hybrid systems. *Software Tools for Technology Transfer* **1**(1/2), 110–122.

Henzinger, T.A., S. Qadeer and S.K. Rajamani (2000). Decomposing refinement proofs using assume-guarantee reasoning. In: *Proc. IEEE Int. Conf. on Computer-Aided Design*. IEEE Comp. Soc. Press. pp. 245–252.

Henzinger, T.A., S. Qadeer, S.K. Rajamani and S. Tasiran (1998). You assume, we guarantee: Methodology and case studies. In: *Proc. 10th Int. Conf. on Computer-Aided Verification*. Vol. 1427 of *LNCS*. Springer. pp. 440–451.

Hooman, J. (1997). Compositional verification of real-time applications. In: *Proc. Compositionality - The Significant Difference*. Vol. 1536 of *LNCS*. Springer. pp. 276–300.

Jones, C.B. (1981). Development methods for computer programs including a notion of interference. PhD thesis. Oxford University Computing Laboratory.

Lynch, N. and Krogh, B. H., (Eds.) (2000). *Hybrid Systems: Computation and Control (HSCC'00)*. Vol. 1790 of *LNCS*. Springer.

Misra, J. and K.M. Chandy (1981). Proofs of networks of processes. *IEEE Trans. on Software Engineering* **7**(7), 417–426.

Pnueli, A. (1984). In transition for global to modular temporal reasoning about programs. In: *Logics and Models of Concurrent Systems*. Vol. 13 of *NATO ASI-F*. Springer.

Queille, J.P. and J. Sifakis (1982). Specification and verification of concurrent systems in CESAR. In: *Proc. 5th Int. Symp. on Progr.*. Vol. 137 of *LNCS*. Springer. pp. 337–351.

Stursberg, O. (2000). Analysis of switched continuous systems based on discrete approximation. In: *Proc. 4th Int. Conf. on Automation of Mixed Processes*. pp. 73–78.