Modular Verification of Safe Online-Reconfiguration for Proactive Components in Mechatronic UML[⋆]

Holger Giese and Martin Hirsch[⋆⋆]

Software Engineering Group, University of Paderborn,
Warburger Str. 100, D-33098 Paderborn, Germany
[hg|mahirsch]@uni-paderborn.de

Abstract. While traditionally the environment considered by an autonomous mechatronic systems only consist of the measurable, surrounding physical world, today advanced mechatronic systems also include the context established by the information technology. This trend makes mechatronic systems possible which consist of cooperating agents which optimize the system behavior by adjusting their local behavior and cooperation structure to better serve their current goals depending on the experienced mechanical and information environment. The MECHATRONIC UML approach enables the component-wise development of such self-optimizing mechatronic systems by providing a notion for hybrid components and support for modular verification of the safe online-reconfiguration. In this paper, we present an extension to the formerly presented solution which overcomes the restriction that only purely reactive behavior with restricted time constraints can be verified. We present how model checking can be employed to also verify the safe modular reconfiguration for systems which include components with complex time constraints and proactive behavior.

1 Introduction

To realize advanced mechatronic systems such as intelligent cooperating vehicles (cf. [1]), the engineers from the different disciplines mechanical engineering, electrical engineering, and software engineering have to cooperate successfully. The development of such systems becomes even more challenging, if the mechatronic systems should be able to adjust their behavior and structure at run-time (cf. self-adaptation and self-optimization [2–4]).

The environment of autonomous mechatronic subsystems today no longer consist only of the physical world. In addition, the context built by the interconnection of the system via information technology such as local bus systems or wireless networking technology has to be taken into account. Therefore, today more flexible mechatronic systems are developed which require complex online reconfiguration schemes for the control algorithms which can effect not only a single component but whole hierarchies of connected components.

[⋆] This work was developed in the course of the Special Research Initiative 614 - Self-optimizing Concepts and Structures in Mechanical Engineering - University of Paderborn, and was published on its behalf and funded by the Deutsche Forschungsgemeinschaft.

[⋆⋆] Supported by the University of Paderborn.

The MECHATRONIC UML approach provides an approach for the component-wise development of such complex self-optimizing mechatronic systems. It supports a component notion and the specification of required online reconfiguration activities which go across multiple levels of a component hierarchy [5–7]. Therefore, online-reconfiguration can be employed also on the higher levels of the system control but also across the boundaries traditionally given by the involved disciplines. The MECHATRONIC UML approach also supports model checking techniques for real-time processing at the higher levels. It addresses the scalability problem of these techniques by supporting a compositional proceeding for modeling and compositional verification of the real-time software when using the UML 2.0 component model and the corresponding definition of ports and connectors as well as patterns [8, 9].

In [6], an approach to combine such a compositional approach with techniques to ensure the proper modular reconfiguration has been presented which require a rather restricted purely reactive behavior and rather restricted local timing constraints for the subordinated components. More complex timing constraints including clock invariants which enforce certain reconfiguration sequences or proactive behavior in the sense that the subordinated component autonomously decides that a reconfiguration is required is currently not supported. In this paper, we present how the verification of the safe reconfiguration can be accomplished with model checking to also cover these two cases.

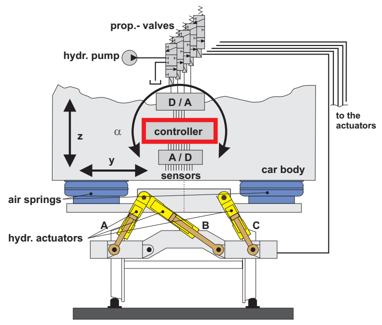
The paper proceeds with an informal introduction on modeling with the MECHATRONIC UML approach by means of an example given in Section 2. Then, the simple syntactical check for purely reactive components with only local externally relevant timing constraints for the reconfiguration is outlined in Section 3. In Section 4, the example is extended with proactive behavior and non complex timing constraints which effect the reconfiguration. The concepts and first evaluation results for verifying the safe reconfiguration follow in Section 5. We close the paper with a discussion of related work and a final conclusion including an outlook on planned future work in Section 6.

2 Modeling

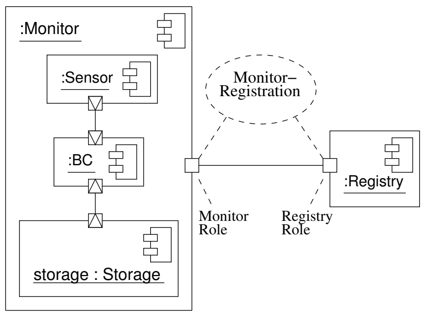
In this section we introduce our MECHATRONIC UML approach focusing on modeling with hybrid components (cf. [10]).

To outline our approach, we employ an example which stems from the RailCab[1] research project at the University of Paderborn. Autonomous shuttles are developed which operate individually and make independent and decentralized operational decisions. The shuttle's active suspension system and its optimization is one example for a complex mechatronic system whose control software we design in the following. The schema of the relevant physical model of our example is shown in Figure 1(a). The suspension module is based on air springs which are damped actively by the displacement of their bases and three vertical hydraulic cylinders which move the bases of the air springs via an intermediate frame – the suspension frame. The vital task of the system is to provide the passengers a high comfort and to guarantee safety when controlling the shuttle's coach body. In order to achieve this goal, multiple feedback controllers are used with different capabilities in matters of safety and comfort [11].

[1] http://www-nbp.upb.de/en/index.html



(a)



(b)

Fig. 1. 1(a)Scheme of the suspension module / 1(b)Monitor and its environment

We focus on 3 controllers which provide the shuttle different comfort: The Reference controller provides sophisticated comfort by referring to a trajectory describing the required motion of the coach body in order to compensate the current track's unevenness, slopes, etc. To guarantee stability, all sensors have to deliver correct values. In case of e.g. incorrect values the less comfortable Absolute controller has to be applied, which requires only the vertical acceleration as input. If this sensor fails, our Robust controller, which provides the lowest comfort, but requires just standard inputs to guarantee stability, has to be applied. We have to distinguish between two different cases: *atomic switching* and *cross fading*. In the case of atomic switching, the change can take place between two computation steps. If the operating points of the controllers are not identical, it will be necessary to cross-fade between the two controllers.

The architecture of the suspension module is depicted in Figure 1(b). The Monitor component coordinates its embedded components BC, Sensor, and Storage. Further, it communicates via the MonitorRegistration pattern with the Registry. If Registry sends the information about the upcoming track section to Monitor, the Monitor stores it in the Storage component. Sensor provides the signals. To model the hierarchical embedding of the BC component into the Monitor component, aggregation for UML 2.0 components is used. The non-hierarchical link of the Monitor component to the Registry component is described by two ports (as defined in the UML 2.0 as unfilled boxes) and a connector.

To additionally model the quasi-continuous aspects of the model in form of communication via continuous signals, we extend the UML by *continuous ports*, depicted by framed triangles whose orientation indicates the direction of the signal flow. The behavior of the hybrid component is specified by means of an extension of UML State Machines called *hybrid reconfiguration charts*. We employ Real-Time Statecharts [12] to describe required real-time behavior and refer the continuous behavior by embedding appropriate basic quasi-continuous block configurations (cf. the BC component behavior in Figure 2(a)).

While a common hybrid automaton specification requires always the same input and output signals for every location, the required controller logic with its specific required input and provided output signals is specified within each state of a hybrid reconfiguration chart (cf. Figure 2(a)). The continuous ports that are required in each of the

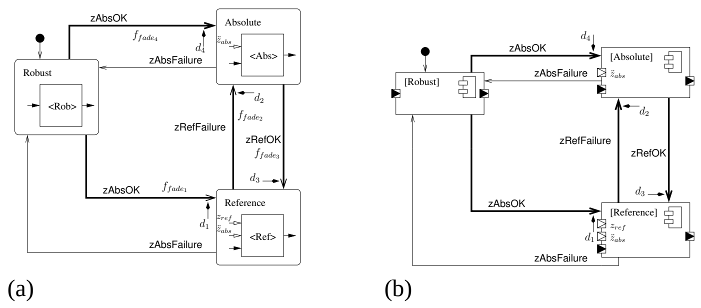


Fig. 2. Behavior description of the BodyControl (2(a)) and the interface state chart (2(b))

three interfaces are filled black, the ones that are only used in a subset of the states are filled white. In our notion of hybrid reconfiguration charts we introduce additional *fading-transitions*, which are visualized by thick arrows, while atomic switches have the shape of regular arrows. Parameters of a transition are: A source- and a target-location, a guard and an event trigger, information on whether or not it is an atomic switch, and, in the latter case, a fading strategy (f_{fade}) and the required fading duration interval $d = [d_{low},\ d_{up}]$ specifying the minimum and maximum duration of fading.

For embedding or connecting a hybrid component we do not need all details of the component realization, but only enough information about its externally observable behavior such that compatibility can be analyzed. In Figure 2(b) the related *interface state chart* of the BC component is displayed. The interface automaton abstracts from the continuous behavior, it still contains the information about the input-output dependencies and permits us to abstract from all internal variables and signals.

Therefor we present in the following a concept for the behavioral embedding of the subcomponents within the hybrid reconfiguration charts of a component (Section 2), which permits to check consistency w.r.t. reconfiguration at a purely syntactical level (Section 3).

By assigning a configuration of aggregated subcomponents (not only quasi-continuous blocks) to each state of a hybrid reconfiguration chart by means of UML instance diagrams, the behavioral embedding of subcomponents is realized (see Figure 3). A switch between the locations of the monitor chart then *implies* a switch between locations of the interface state charts of the embedded components.

The behavior of the Monitor component is specified by a hybrid reconfiguration chart (cf. Figure 3). We have assigned the BC component in the appropriate state to each location of the upper orthogonal state of the chart. E.g., the BC component instance in state Reference has been (via a visual embedding) assigned to the location AllAvailable of the monitor where z_{ref} as well as $\ddot{z}_{abs}$ are available. The communicaton with the Registry is described in the lower othogonal state of Figure 3 (cf. [6]). The upper orthogonal state consists of the states RefAvailable and AllAvailable which represents whether the required reference curve is available for the *current* track. The upper state is synchronized by the lower one.

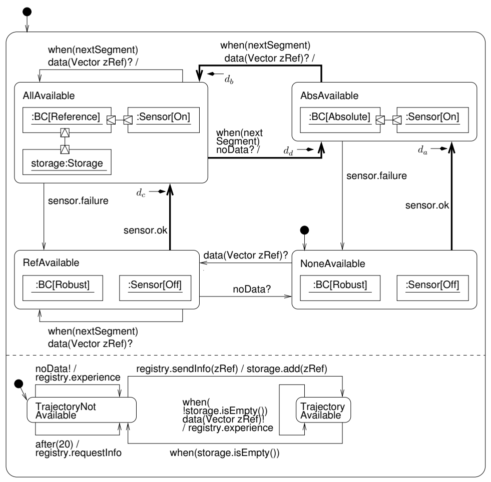


Fig. 3. Behavioral embedding in the Monitor realization

3 Safe Reconfiguration for Reactive Subcomponents

For the outlined MECHATRONIC UML approach, two specific verification tasks for the resulting systems are supported. At first the real-time coordination of the distributed software, which is modeled with UML 2.0 components and connectors and are only interconnected by verified coordination patterns, can be verified using a compositional model checking approach [8, 9]. Secondly, a restricted subset of the outlined hierarchical component structures for modeling of discrete and continuous control behavior can be checked for the consistent reconfiguration and proper real-time synchronization w.r.t reconfiguration [6]. In addition, the second approach can be embedded into the first one.

3.1 Modular Reasoning

In the following, we outline the modular reasoning concept which underlies the second verification task. A hierarchical component structure is described by a set of component instances $C_1, \dots, C_n$ and functions $sub, sub^* : \{1, \dots, n\} \rightarrow \wp(\{1, \dots, n\})$ where $sub(i)$ is the index set of all directly aggregated subcomponents of C_i and $sub^*(i)$ the transitive closure of sub including the input index i. For each component instance C_i we assume a related automaton M_i representing the hybrid reconfiguration chart of the component instances and an automaton M_i^I for the interface state chart. Each automaton M has an interface $I(M)$ consisting of its input and output signals and variables. Two automata can be composed in parallel (||) and the interface of a single automaton M can be restricted by hiding all synchronization signals and variables not included in an interface I denoted by $M|_I$ (cf. Appendix A for some basic definitions for the employed automata and refinement notions).

Starting with a leaf component C_j where $sub(j) = \emptyset$, we require that for the component behavior M_j described by a hybrid reconfiguration chart and the more abstract interface behavior M_j^I described by an interface state chart holds *refinement*:

$$M_j \sqsubseteq_{HY} M_j^I. \tag{1}$$

For a non leaf component C_k, we require that for M_k^I and M_k plus the interface automata of all subordinated components holds

$$(M_k \| \left(\|_{l \in sub(k)} M_l^I\right))|_{I(M_k^I)} \sqsubseteq_{HY} M_k^I, \tag{2}$$

We can ensure that condition 2 holds by proving that between the component behavior and the interface automata *refinement* holds ($M_k|_{I(M_k^I)} \sqsubseteq_{HY} M_k^I$) and that the interference between M_k and its subcomponent behaviors M_l with $l \in sub(k)$ results always in correct *embedding* ($(M_k \| \left(\|_{l \in sub(k)} M_l^I\right))|_{I(M_k)} \sqsubseteq_{HY} M_k$).

Condition 1 and 2 each provide a local abstraction condition for each component C_k and its behavior M_k as well as its interface automata M_k^I. By induction over the tree structure we can thus reason in a modular manner that each component behavior as well as interface automaton is refined by whole tree consisting of its direct and indirect subcomponents including the component itself.

$$\left(\|_{l \in sub^*(k)} M_l\right)|_{I(M_k)} \sqsubseteq_{HY} M_k \qquad \wedge \qquad \left(\|_{l \in sub^*(k)} M_l\right)|_{I(M_k^I)} \sqsubseteq_{HY} M_k^I \tag{3}$$

To apply the outlined modular reasoning concept in practice, we require effective and efficient procedures to check refinement and embedding. For more general hybrid systems than rectangular automata, where all analog variables follow trajectories within piecewise-linear envelopes and are reinitialized whenever the envelope changes, reachability is undecidable [13], and even for systems where these restrictions apply, the model checking is in practice restricted to rather small examples. Therefore, we restrict our attention here to the real-time processing and real-time configuration and the question whether only consistent configurations with well-formed continuous equations can be reached.

In a first step, we can simply abstract from the continuous behavior of an automaton M by only taking into account the clocks such that the reconfiguration charts and interface state charts can be mapped to Real-Time Statecharts. Then, the transformation rules from Real-Time Statecharts to timed automata introduced in [14] can be employed to check refinement as well as embedding. Note that for the timed automata instead of $\sqsubseteq_{HY}$ only $\sqsubseteq_{RT}$ plus the condition 8 for the variable dependencies have to be checked.

It is to be noted that such an abstraction might be rather coarse if subtle timing effects only result from the continuous behavior and related guards of the transitions. In such cases, however, the correct reconfiguration of a system depends on the detailed continuous control behavior which is usually subject to environment disturbance. Therefore, we think this restriction is acceptable.

3.2 Checking Refinement

In our preceding work [6] we drastically restricted the timing constraints that are allowed in the interface state charts to those ones which only restrict the duration of transitions (or better the duration for the corresponding states of the timed automaton).

Definition 1. *A location l' is a* fading *location if a clock c and constants a' and b' exists such that only a single transition $t \in T$ to leave l' exists with $t = (l', a' \leq t \leq b')$, for all transition $(l'', g, S, R, l') \in T$ holds that $c \in R$, and $C(l') = c \leq b'$. A location l is* passive *iff $C^I(l) =$ true and for all transitions $(l, g, S, R, l') \in T$ with l' a fading location holds $g =$ true. An automaton $M = (L, D, I, O, T, S^0)$ is* simple *if sets of passive locations L_p and fading locations L_f exists with $L = L_p \uplus L_f$.*

Due to this simplification, we could employ a simple syntactical checking procedure to determine whether a simple interface automaton $M^I = (L^I, D^I, I^I, O^I, T^I, {S^0}^I)$ for a given simple interface state chart is correctly refined by a given component behavior $M = (L, D, I, O, T, S^0)$. For $L_f \subseteq L$ and $L^I = L^I_p \uplus L^I_f$ with L^I_p the passive locations of L^I, L^I_f and L_f the fading locations, and a given mapping $map : L^I_p \to \wp(L)$ between the passive states of the interface automaton and the related states in the underlying realization of the component in form of a hybrid reconfiguration automaton, we can check whether the real-time refinement holds as follows:

1. For all states $l_i \in L^I_p$, $l \in map(l_i)$, and $l' \in L_f$ we check that for each pair of transitions between passive locations and intermediate fading location holds that the refinement provides only the same external synchronization and operates strictly within the time frame offered by the interface automaton:

$$\forall (l, g, S, R, l'), (l', g', S', R', l'') \in T, \exists (l_i, g'', S, R, l'_i), (l'_i, g''', S', R', l''_i) \in T^I :$$
$$g' = \text{true} \wedge g'' = a \leq t \leq b \wedge g''' = a^I \leq t \leq b^I \wedge c(l') = t \leq b \wedge c(l'_i) = t \leq b^I \wedge$$
$$a^I \geq a \wedge b \geq b^I \wedge l' \in L_f \wedge l'_i \in L^I_f \wedge g = \text{true} \wedge R = R' = \{t\} \wedge l'' \in map(l''_i)$$

$$\forall (l_i, g, S, R, l'_i) \in T^I : g = \text{true} \wedge \exists (l, g', S, R', l') \in T \wedge \bigvee_{(l, g', S, R', l') \in T} g' = \text{true}$$

2. For all states $l, l' \in L \setminus L_f$ check that all transitions between them (which relate to atomic one at the state chart level) are either covered by related transitions of the interface automaton or preserve the mapping:

$$\forall (l, g, S, R, l') \in T :$$
$$(S = \emptyset \wedge \forall l_i \in L^I : (l \in map(l_i) \Rightarrow l' \in map(l_i))) \vee$$
$$(g = \text{true} \wedge \exists (l_i, \text{true}, S, R', l'_i) \in T^I) : l' \in map(l'_i).$$

3. For all states $l \in L_i$ check that $l \notin L_f$ and that they are covered by assigned initial locations of the interface automatona: $\exists l_i \in L^I_i : l \in map(l_i)$.

In addition, for all $l \in L$ and $l_i \in L_i$ with $l \in map(l_i)$ must hold that any dependency between an input and output variable in the realization $D(l)$ is also present in the related interface automaton $D^I(l_i)$.

3.3 Checking the Embedding

To ensure a correct embedding, the correct real-time coordination of the fading-durations etc. have to be checked first. By restricting our considerations here to *simple* interface state charts, we can check that the hybrid reconfiguration chart alone is an abstraction of the hybrid reconfiguration chart combined with the interface state charts of the subcomponents (see condition 2). Again, as reachability is undecidable (or at least a hard problem) for most practical hybrid systems, we proposed in [6] to avoid the analysis of the complete state space and instead use a static check which only operates on the state and transition sets of the hybrid reconfiguration automaton M_i and the interface automaton M_l^I with $l \in sub(i)$.

Given a function $mode : L \times sub(i) \to \bigcup_{j \in sub(i)} L_j^I$ such that for all $l \in L$ and $j \in sub(i)$ holds $mode(l, j) \in L_j^I$. We further assume that all local transitions are labeled with the I/O set $\emptyset$ and that $L_f \subseteq L$ and $L_j^I = L_{j,p}^I \uplus L_{j,f}^I$ with $L_{j,p}^I$ and L_p all passive locations and $L_{j,f}^I$ and L_f are all fading locations.

1. For all states $l \in L \setminus L_f$ and $l' \in L_f$ we check that for each pair of transitions between passive locations and intermediate fading location holds that each aggregated component $j \in sub(i)$ holds that a pair of transitions exists which operates strictly within the time frame offered by M:

$$\forall (l, g, S, R, l'), (l', g', S', R', l'') \in T, \exists (l_j, g_j, S_j, R_j, l'_j), (l'_j, g'_j, S'_j, R'_j, l''_j) \in T_j^I :$$
$$g' = a \leq t \leq b \wedge g'_j = a_j^I \leq t \leq b_j^I \wedge c(l') = t \leq b \wedge c(l'_j) = t \leq b_j^I \wedge a_j^I \geq a \wedge b \geq b_j^I \wedge$$
$$l' \in L_f \wedge l'_j \in L_{i,f}^I \wedge g' = \text{true} \wedge R' = \{t\} \subseteq R \wedge l_j = mode(l, j) \wedge l''_j \; mode(l'', j)$$

2. For all states $l, l' \in L \setminus L_f$, each aggregated component $j \in sub(i)$, and all transitions $(l, g, S, R, l') \in T$, check that the atomic transitions are covered or do not result in any mode change:

$$(mode(l, j) = mode(l', j)) \vee (\exists (l_j, \text{true}, S_j, R_j, l'_j) \in T^I) : l'_j = mode(l', j).$$

3. For all initial states $l \in L_i$ and all aggregated components $j \in sub(i)$ with $l_j = mode(l, j)$, check that all initial locations are covered : $l_j \in L_{j,i}^I$.

We have proven in [6, 15] that this check guarantees for the restricted case of simple interface state charts that condition 2 is fulfilled.

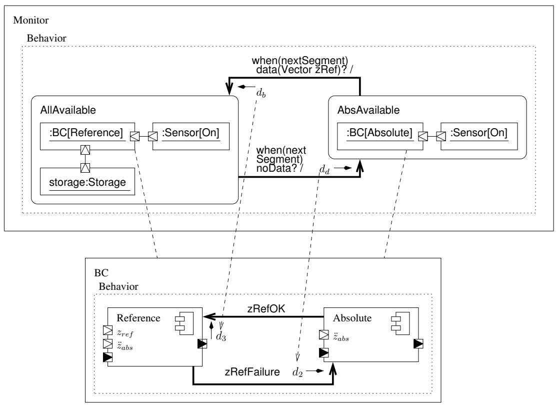


Fig. 4. Scheme for the syntactical checking of correct reconfiguration

In Figure 4, the part of the monitor behavior and a part of the interface state chart of the embedded BC component (cf. Figures 2(b) and 3) are depicted. The semantics of the reconfiguration chart requires that a transition from state AbsAvailable to AllAvailable results in a transition of the BC component from state Absolute to Reference. While the monitor component requires that this transition is completed within the timing interval d_b, for the implied state change in the component BC holds that is must occur within the timing interval d_3. Therefore, a consistent parallel execution of both transitions requires that $d_3 \subseteq d_b$ holds. For the transition to AllAvailable and the resulting transition to state Reference in the BC component holds that $d_2 \subseteq d_d$ must be true.

For a specific state of a hybrid reconfiguration chart holds that it can only result in an incompatible reconfiguration, if the composed dependency relation for the related combination of states of the subcomponent contains a cycle. As the employed refinement relation ensures that for any dependency between an input and output in the realization M also one is present in the related interface automaton M^I, it is sufficient to consider the interface automaton and check all combinations encoded in $mode$ to exclude incompatible state configurations.

3.4 Limitations

The presented results enable the systematic development of complex mechatronic systems with safe online-reconfiguration, as in the current practice of mechatronic systems the strict hierarchical approach with strict top-down command order is standard. However, a number of limitations result which seem to unduly restrict the design space for more advanced modular designs of mechatronic systems in the future.

One severe restriction is that interface state charts can only specify the duration of transitions but not their time-triggered execution. Examples where an extension to more general time constraints in the interface state chart seem beneficial are, for example, restrictions on the frequency of mode switches. In our example, the engineer could more easily ensure the stability of the underlying system if the interface state chart restricted that after a switch from state Reference to state Absolute a certain time threshold should elapse before the BC component permits to switch back to the more comfortable Reference state.

Another limitation is the strict top-down command order. While the processing of the sensor error already indicates that we have to consider errors of the subcomponents, the current form of interface state charts does not permit to encode a required reaction time or switching of the mode within the interface. Thus, whether a required reaction of the embedding component results or not is currently not included in the interface. Therefore, besides only emitting warnings, error reports, or wishes to the embedding component, true *proactive* behavior in the interface state charts seems favorable such that when sending an error report the interface state chart can also initiate that within a given time frame the current state has to be left. As example for proactive behavior, consider the case that BC component detects that the operation with the reference data results in unexpected problems and wants to report this to the embedding Monitor component, the BC component may in addition specify in the interface state chart that therefore the reference mode has to be left within a given deadline to ensure that the observed behavior is not critical.

We can characterize these cases where more expressive notion of interface automaton are required as follows:

Definition 2. *An interface automaton M is* complex *if it is not simple but still deterministic. An interface automaton M is* proactive *if it autonomously decides that a reconfiguration is required which results in a non-deterministic behavior.*

4 Modeling with Complex and Proactive Subcomponents

In this section we give an example on modeling of hybrid systems with proactive behavior. Therefore we extend the example from Section 2. We proceed as follows. First we describe the new behavior informally. Thereafter we describe the new component behavior and the behavioral embedding and their influence on the whole system.

In Section 2 the suspension module was introduced and thereon the control software was modeled. One characteristic of the control software was the top-down command order. It was not possible for the BC component to influence the Monitor component via direct events. E.g. if any error occurs when the BC component is in the Reference state, the BC component has to switch to the Robust state and has to inform the superordinate Monitor component to react in an adequate way. Furthermore we want to avoid perpetual uncontrolled switching between Absolute and Reference.

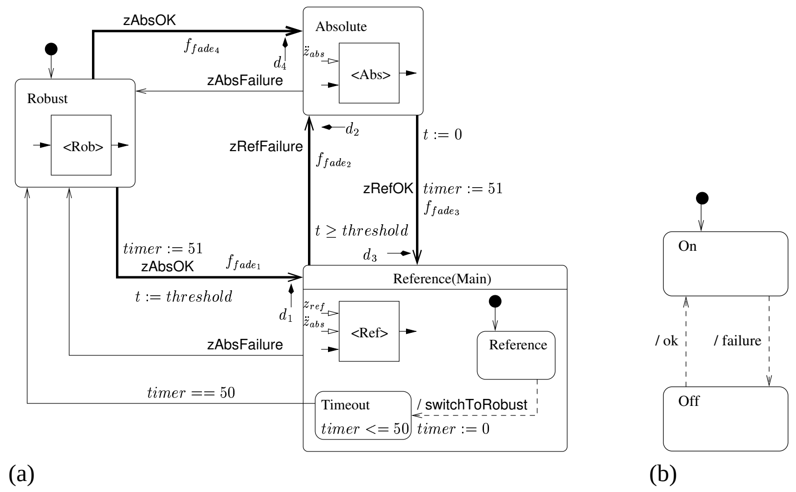


Fig. 5. Behavior description of the BC component (5(a)) and Sensor component (5(b))

In Figure 5(a), the redesigned BC component is depicted. The behavior of the former BC component is extended by proactive bahavior. When the BC component is in the Reference state, the component is now able to decide autonomous to switch to the Robust state. Due this is modeled by a non-urgent transitions (dashed line), non-determinism is introduced. This is modeled as follows: While staying in the Reference state, the body control sends a message switchToRobust to the superordinate Monitor component. This done, the BC component pauses in a Timeout state. If the timeout is reached, the BC component switches to Robust. To control the switching between Absolute and Reference, a timer t is introduced. Everytime the Reference state is entered from the Absolute state, a clock t is set to zero. To avoid an immediate backspace, a guard, representing a threshold, t>=threshold, is added to the transition. All other incoming transitions to the Reference state get an additional assignment t:=threshold, thus the threshold is "omitted". In Figure 5(b) the interface state chart of the sensor component is depicted. The state chart consists of two states, on and off.

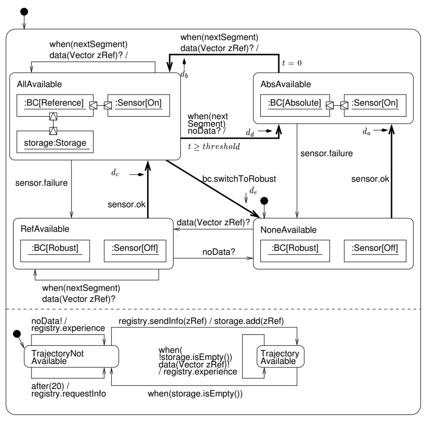


Fig. 6. Behavioral embedding in the Monitor realization

Similar to Figure 3, the behavior of the Monitor component is depicted in Figure 6. In addition to the old Monitor component, we have to take into account the proactive and timing behavior of the subordinated components. Since the body control sends now the switchToRobust message the monitor component has to consume this message.

5 Checking Complex and Proactive Subcomponents

To adjust our modular reasoning approach to the extensions outlined in the last section, we have to provide checks for refinement and correct embedding which support the introduced more expressive interface automata.

5.1 Checking Refinement

Complex and proactive components do not result in a *simple* interface state chart (cf. Definition 2) and thus the checking procedure for ensuring that the interface state chart corresponds to the component behavior outlined in Section 3 is not applicable . In [16], an approach for checking the employed notion of refinement ($M \sqsubseteq_{RT} M^I$) has been presented which requires that M^I is deterministic. If the interface state chart M^I is complex but not proactive, we can thus employ this approach. For a deterministic M^I we have to derive a corresponding test automaton M_t^I as described in [16] and then check $M \| M_t^I$ for time stopping deadlocks.

If, however, the interface state chart M^I is proactive and thus not deterministic, we have to look for alternatives, as we cannot derive a deterministic timed automaton for each non-determinsitic one (cf. [17]). Analyzing the limitation of the approach outlined in [16], we can conclude that the branching within the on-the-fly traversed cross-product simply requires that a unique mapping to a state in the refined model exists which is guaranteed in [16] by the deterministic character of M^I.

We propose to exploit the mapping $map : L_p^I \to L$ between the passive states of the interface automaton and related states in the underlying realization to achieve a feasible solution for this case. For a mapping map which assigns to each realization state exactly one state of the interface automaton ($\forall l \in L : |\{l' | l' \in map(l)\}| = 1$ and thus map^{-1} is a function and we write $l' = map^{-1}(l)$) and the case that no two transitions with the same source location, label, and target location exist ($\forall l, l' \in L, s \subseteq I \cup O : |\{(l, g, S, R, l') \in T\}| \leq 1$), we can build syntactically the cross-product $M' = M^I \times_{map} M$ with $M' = (L', D', I', O', T', S'^0)$, $M^I = (L^I, D^I, I^I, O^I, T^I, S^{I,0})$, and $M = (L, D, I, O, T, S^0)$ and L_f^I and L_f are all fading states of L^I resp. L as follows:

- $L' = \{(l, l') \in L^I \times L\} \cup \texttt{error}$,
- $D'(l, l') = D^I(l) \| D(l')$ and $D'(\texttt{error})$ is empty,
- $I' = I^I \cup I, O' = O^I \cup O$.
- $T' = \bigcup_{(l,l') \in L' \setminus \{\texttt{error}\}, S \subseteq I' \cup O'} T'_{l,l',S}$ with $T'_{l,l',S} = T'^r_{l,l',S} \cup T'^{e1}_{l,l',S} \cup T'^{e2}_{l,l',S}$ the union of related normal (n) and erroneous transition sets ($e1$, $e2$).
- $S'^0 = S'^I \times S^0$

The normal transition set $T'^n_{l,l',S}$ is build by simply composing all pairs of transitions which matching s resulting in the set $\{((l, l'), g \wedge g', S, R \cup R', (l'', l''')) | (l, g, S, R, l'') \in T^I \wedge (l', g', S, R', l''') \in T \wedge l' = map^{-1}(l)\}$.

The first set of erroneous transitions $T'^{e1}_{l,l',S}$ for $l \notin L_f^I$ or $l' \notin L_f$ considers behavior where the interface state chart can fire but not the realization: $T'^{e1}_{l,l',S} = \{((l, l'), g \wedge \neg g'', S, \emptyset, \texttt{error}) | (l, g, S, R, l'') \in T^I \wedge g'' = \bigvee_{(l', g', S, R', l''') \in T} g'\}$.

Analogously the second set of erroneous transitions $T'^{e2}_{l,l',S}$ for $l \notin L_f^I$ or $l' \notin L_f$ considers behavior where the realization can fire but not the interface state chart: $T'^{e2}_{l,l',S} = \{((l, l'), g' \wedge \neg g'', S, \emptyset, \texttt{error}) | (l', g', S, R', l''') \in T \wedge g'' = \bigvee_{(l, g, S, R, l'') \in T^I} g\}$.

For $l \in L_f^I$ and $l' \notin L_f$ in contrast we can assume that the guards always have the form $a' \leq t \leq b'$ and thus have $T'^{e1}_{l,l',S}$ is $\{((l, l'), a \leq t \leq a^I, S, \emptyset, \texttt{error}) | (l, a^I \leq t \leq b^I, S, R, l'') \in T^I \wedge (l, a \leq t \leq b, S, R, l'') \in T\}$ and $T'^{e2}_{l,l',S}$ is $\{((l, l'), b^I \leq t \leq b, S, \emptyset, \texttt{error}) | (l, a^I \leq t \leq b^I, S, R, l'') \in T^I \wedge (l, a \leq t \leq b, S, R, l'') \in T\}$ to encode incompatible time constraints. In the first case, an earlier termination of the fading is checked while in the second a to late termination is detected.

We can then simply check whether a time stopping deadlock or the state `error` can be reached in M' and conclude that refinement holds if no such violation has been detected. Due to the restriction that no two transitions with the same source location, label, and target location exist in T^I, it holds that for each $t' = (l', g', S, R', l''') \in T$ at most one related $t = (l, g, S, R, l'') \in T^I$ exists which have been composed in $T'^n_{l,l',S}$ due to the constraint $l' = map^{-1}(l)$.

A mapping map which assigns to a realization state more than one state of the interface automaton does not result in the intended reduced state space for M^I w.r.t. M and therefore this restriction is reasonable for our specific settings. In the case that two transitions $(l, g, S, R, l') \in T^I$ and $(l, g', S, R', l') \in T^I$ with the same source location, label, and target location exist, we have to distinguish two cases. If $R = R'$ we can simply unite them in a single rule $(l, g \vee g', S, R, l')$ without altering the behavior. Otherwise we can for an additional state l'' and an additional clock replace the first rule $(l, g, S, R, l') \in T^I$ simply by the two rules $(l, g, S, R \cup \{t\}, l'')$ and $(l'', \text{true}, \emptyset, \emptyset, l')$ and setting the invariant $C(l'')$ to $t \leq 0$ to fulfill the assumption.

To verify if the hybrid reconfiguration automaton of the BC component is a correct refinement of the BC component interface state chart, we have to built a timed automata model as explaind above. This done, we can used the model checker UPPAAL [18] for the verification and check the constraints, formulated in TCTL, A[] not deadlock and E<> BodyControl.Error ensuring the correct refinement.

5.2 Checking the Embedding

To realize the dynamic checking for the prove of correct reconfiguration behavior, the hybrid reconfiguration chart and the interface state chart have to be transformed to an appropriate model which can be handeled by a model checker. In [14] transformation rules from Real-Time Statecharts to timed automata were introduced. In the following, we reuse and extend theses transformation rules.

In hybrid reconfiguration charts component instances are embedded in locations. During the transformation, the instances are omitted because the associated interface state charts are also transformed. Due to this fact, for the transformation of locations, we can apply the same one as for Real-Time Statecharts.

In addition to the locations, the transitions have to be transformed. In contrast to Real-Time Statecharts, a transition is associated with a fading function. Since the fading function does not affect the real-time behavior, it is omitted, too. Hence the same transformation as for Real-Time Statecharts is used. As mentioned before, the interface state charts of the embedded component instances have to be transformed. It has to be guaranteed that the embedded component instance leaves an internal location iff the superordinate component leaves a location. For example, when the monitor component leaves the location NoneAvailable, the embedded component instance BC has to leave the internal location Robust. To achive this behavior, we use the synchronization semantics from the timed automata model. The superordinate component has the meaning of a sender (!), whereas the embedded compontent instances have the meaning of a receiver (?). In the case of more than one embedded component instance we use a chain of committed locations (cf. [18]) for sychronization. In Figure 7 an example is depicted.

For the evaluation we use the real-time model checker UPPAAL [18]. In Figures 8 and 9, the transformed timed automata of the interface state chart of the BodyControl and the reconfiguartion state chart of the Monitor component are depicted. For clarity we do not depict the timed automaton of the Sensor (the transformation is rather trival). For the verification the automata are executed in parallel. We check the property A[] not deadlock.

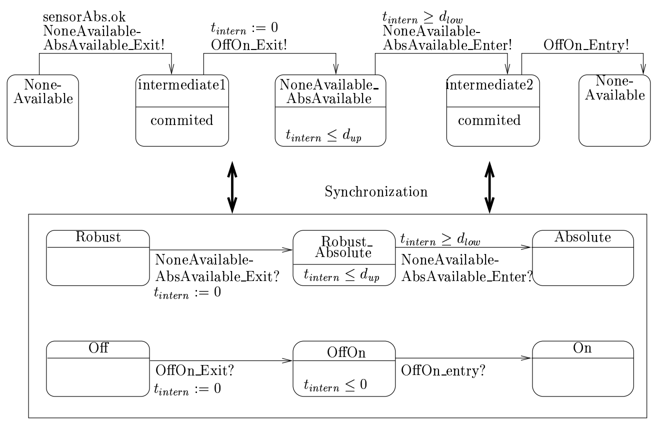


Fig. 7. Synchronization between Monitor, Sensor, and BodyControl

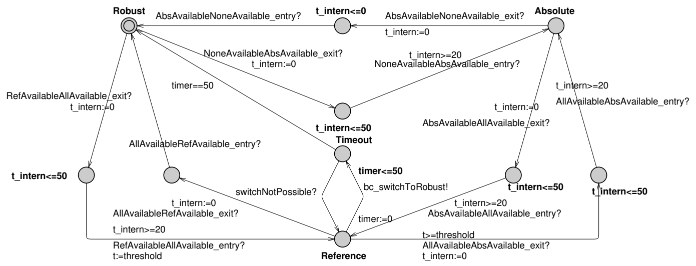

Robust
AbsAvailableNoneAvailable_entry?
t_intern<=0
AbsAvailableNoneAvailable_exit?
t_intern:=0
Absolute
NoneAvailableAbsAvailable_exit?
t_intern:=0
t_intern>=20
NoneAvailableAbsAvailable_entry?
timer==50
RefAvailableAllAvailable_exit?
t_intern:=0
t_intern<=50
Timeout
t_intern:=0
AbsAvailableAllAvailable_exit?
t_intern>=20
AllAvailableAbsAvailable_entry?
AllAvailableRefAvailable_entry?
timer<=50
bc_switchToRobust!
t_intern<=50
switchNotPossible?
t_intern<=50
t_intern<=50
t_intern:=0
AllAvailableRefAvailable_exit?
timer:=0
t_intern>=20
AbsAvailableAllAvailable_entry?
t_intern>=20
RefAvailableAllAvailable_entry?
t:=threshold
Reference
t>=threshold
AllAvailableAbsAvailable_exit?
t_intern:=0

Fig. 8. Timed Automaton of the interface state chart BodyControl

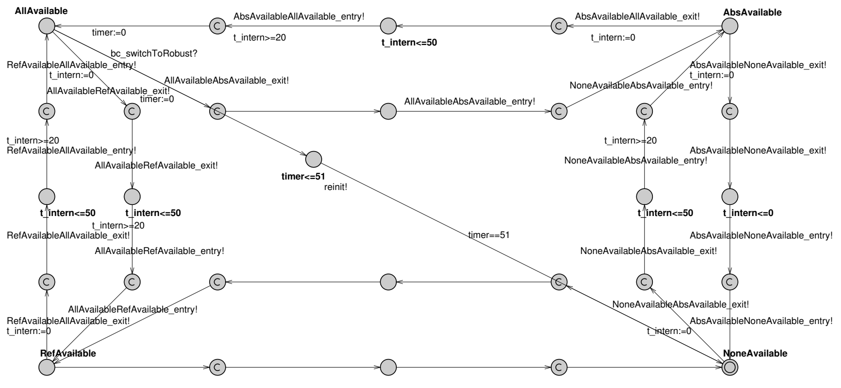

AllAvailable
AbsAvailableAllAvailable_entry!
t_intern>=20
t_intern<=50
AbsAvailableAllAvailable_exit!
t_intern:=0
AbsAvailable
timer:=0
bc_switchToRobust?
RefAvailableAllAvailable_entry!
t_intern:=0
AllAvailableRefAvailable_exit!
timer:=0
AllAvailableAbsAvailable_exit!
AllAvailableAbsAvailable_entry!
NoneAvailableAbsAvailable_entry!
AbsAvailableNoneAvailable_exit!
t_intern:=0
t_intern>=20
RefAvailableAllAvailable_entry!
AllAvailableRefAvailable_exit!
timer<=51
reinit!
t_intern>=20
NoneAvailableAbsAvailable_entry!
AbsAvailableNoneAvailable_exit!
t_intern<=50
t_intern<=50
t_intern<=50
t_intern<=0
RefAvailableAllAvailable_exit!
t_intern>=20
AllAvailableRefAvailable_entry!
timer==51
NoneAvailableAbsAvailable_exit!
AbsAvailableNoneAvailable_entry!
AllAvailableRefAvailable_entry!
RefAvailableAllAvailable_exit!
t_intern:=0
NoneAvailableAbsAvailable_exit!
AbsAvailableNoneAvailable_entry!
t_intern:=0
RefAvailable
NoneAvailable

Fig. 9. Timed Automaton of the Monitor state chart

As result of the verification we get that the system is deadlock free. In particular this means we have a correct embedding of all components. The verification took about 0.31 seconds and at maxium about 2092 KB were allocated by the verifier[2].

6 Related Work

The *de facto* industry standard for modeling of mechatronic systems with hybrid behavior is MATLAB/Simulink and Stateflow[3]. Formal verification of MATLAB/Simulink and Stateflow models of moderate size can be accomplished by automatically transforming them to hybrid automata (cf. [19]).

Besides MATLAB/Simulink and Stateflow, there are also a number of approaches, like Timed and Hybrid Statecharts [20], Charon [21], Masaccio [22], HyCharts & HyRoom [23–25], and Hybrid I/O Automata [26], which address the problem of modeling complex systems by some form of hybrid state charts. Some of them support hierarchy and parallelism as well as a notion of component and some of them even support formal verification.

All existing approaches fail in providing a component concept which supports a dynamic interface which enables to decompose systems with online reconfiguration into multiple hybrid state charts. Thus a usually not feasible check of the whole system is required to ensure that a system with complex reconfiguration such as the presented example is correct w.r.t. the reconfiguration and real-time behavior.

Another consequence is that in these approaches the control engineering know-how for the continuous control and the software engineering know-how for the real-time coordination have to be specified both within a single hybrid component and thus a tight cooperation between engineers from different camps is required. In our approach, in contrast, an interface between the control engineering specific details and the more software engineering oriented distributed real-time processing is possible which support more realistic loosely coupled development processes.

Available compositional reasoning approaches for hybrid systems [27–29] require high manual effort of inventing auxiliary properties to enable a full verification to decide whether the described reconfiguration is consistent. The presented approach in contrast will ensure consistency by means of a syntactical check or modular model checking of the separate components and their interfaces.

We employ in our approach the algorithm for checking the refinement relation between timed automata as proposed in [16]. As outlined in the paper we can only address refinement for complex interface automata with this approach. For proactive and thus non-deterministic interface automata this approach is not applicable.

A more general notion of refinement in the context of clocked CSP is proposed in [30], which is defined for traces in contrast to the state tuple based refinement employed in our approach. However, their approach is limited by the fact that specific clock updates and resets are visible events and thus the required checking of a refinement relation which abstracts from internal clocks as required for our approach is not possible.

[2] The verification was done on a Pentium 4, 2.4 GHz, 1 GB memory, OS Linux Redhat.

[3] http://www.mathworks.com

7 Conclusion & Future Work

Within this paper we presented an incremental improvement of our modular verification approach for checking that the online-reconfiguration of MECHATRONIC UML models is safe. In our earlier proposal [6], severe restrictions to the expressiveness of the supported component interface are employed to ensure that efficient checks can be used which do not have to consider the whole state space. In this paper we present support for more expressive interfaces which include complex timing constraints and proactive behavior employing model checking. The underlying concepts are outlined and formally defined. In addition, first experimental results are reported.

While modeling of hybrid systems with the MECHATRONIC UML approach is already supported by the FUJABA Real-Time Tool Suite[4], the described refinement checks and the check for the correct embedding are currently under development.

In future work, we plan to further extend our approach w.r.t. modeling and verification such that the full hybrid behavior of the components is covered.

References

1. D. Dawson, D. Seward, D.A. Bradley, and S. Burge. *Mechatronics and the Design of Intelligent Machines and Systems*. Nelson Thornes, November 2000.
2. Janos Sztipanovits, Gabor Karsai, and Ted Bapty. Self-adaptive software for signal processing. *Commun. ACM*, 41(5):66–73, 1998.
3. David J. Musliner, Robert P. Goldman, Michael J. Pelican, and Kurt D. Krebsbach. Self-Adaptive Software for Hard Real-Time Environments. *IEEE Inteligent Systems*, 14(4), July/August 1999.
4. Peyman Oreizy, Michael M. Gorlick, Richard N. Taylor, Dennis Heimbigner, Gregory Johnson, Nenad Medvidovic, Alex Quilici, David S. Rosenblum, and Alexander L. Wolf. An Architecture-Based Approach to Self-Adaptive Software. *IEEE Intelligent Systems*, 14(3):54–62, May/June 1999.
5. Sven Burmester, Holger Giese, and Oliver Oberschelp. Hybrid UML Components for the Design of Complex Self-optimizing Mechatronic Systems. In Helder Araujo, Alves Vieira, Jose Braz, Bruno Encarnacao, and Marina Carvalho, editors, *Proc. of 1st International Conference on Informatics in Control, Automation and Robotics (ICINCO 2004), Setubal, Portugal*, pages 222–229. INSTICC Press, August 2004.
6. Holger Giese, Sven Burmester, Wilhelm Schäfer, and Oliver Oberschelp. Modular Design and Verification of Component-Based Mechatronic Systems with Online-Reconfiguration. In *Proc. of 12th ACM SIGSOFT Foundations of Software Engineering 2004 (FSE 2004), Newport Beach, USA*, pages 179–188. ACM Press, November 2004.
7. Sven Burmester, Holger Giese, and Oliver Oberschelp. Hybrid UML Components for the Design of Complex Self-optimizing Mechatronic Systems. In *Informatics in Control, Automation and Robotics*. Kluwer Academic Publishers, 2005. to appear.
8. Holger Giese. A Formal Calculus for the Compositional Pattern-Based Design of Correct Real-Time Systems. Technical Report tr-ri-03-240, Lehrstuhl für Softwaretechnik, Universität Paderborn, Paderborn, Deutschland, July 2003.
9. Holger Giese, Matthias Tichy, Sven Burmester, Wilhelm Schäfer, and Stephan Flake. Towards the Compositional Verification of Real-Time UML Designs. In *Proc. of the European Software Engineering Conference (ESEC), Helsinki, Finland*, pages 38–47. ACM Press, September 2003.

[4] http://www.fujaba.de

10. Sven Burmester, Holger Giese, and Oliver Oberschelp. Hybrid UML Components for the Design of Complex Self-optimizing Mechatronic Systems. In *Proc. of the Eighth International Conference on Informatics in Control, Automation and Robotics (ICINCO) , Setubal, Portugal*. IEEE Press, 2004.
11. T. Hestermeyer, P. Schlautmann, and C. Ettingshausen. Active suspension system for railway vehicles-system design and kinematics. In *Proc. of the 2nd IFAC - Confecence on mechatronic systems*, Berkeley, California, USA, 9-11December 2002.
12. Sven Burmester, Holger Giese, and Wilhelm Schäfer. Model-driven architecture for hard real-time systems: From platform independent models to code. In *Proc. of the European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA'05), Nürnberg, Germany*, Lecture Notes in Computer Science (LNCS), pages 1–15. Springer Verlag, November 2005.
13. Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57:94–124, 1998. A preliminary version appeared in the Proceedings of the 27th Annual Symposium on Theory of Computing (STOC), ACM Press, 1995, pp. 373-382.
14. Sven Burmester, Holger Giese, Martin Hirsch, and Daniela Schilling. Incremental design and formal verification withUML/RT in the FUJABA real-time tool suite. In *Proceedings of the International Workshop on Specification and vaildation of UML models for Real Time and embedded Systems, SVERTS2004, Satellite Event of the 7th International Conference on the Unified Modeling Language, UML2004*, October 2004.
15. Sven Burmester, Holger Giese, and Oliver Oberschelp. Hybrid uml components for the correct design of self-optimizing mechatronic systems. Technical Report tr-ri-03-246, University of Paderborn, Paderborn, Germany, January 2004.
16. Henrik Ejersbo Jensen, Kim Guldstr, Kim Guldstr, and Arne Skou. Scaling up Uppaal Automatic Verification of Real-Time Systems using Compositionality and Abstraction. In *Proceedings of the 6th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT 2000)*, volume 1926 of *Lecture Notes in Computer Science*, Pune, India, September 2000.
17. Stavros Tripakis. Folk theorems on the determinization and minimization of timed automata. In *Formal Modeling and Analysis of Timed Systems: First International Workshop, FORMATS 2003, Marseille, France, September 6-7, 2003. Revised Papers*, volume 2791, pages 182 – 188, 2004.
18. Gerd Behrmann, Alexandre David, and Kim G. Larsen. A tutorial on UPPAAL. In Marco Bernardo and Flavio Corradini, editors, *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004*, number 3185 in LNCS, pages 200–236. Springer–Verlag, September 2004.
19. Aditya Agrawal, Gyula Simon, and Gabor Karsai. Semantic Translation of Simulink/Stateflow models to Hybrid Automata using Graph Transformations. In *International Workshop on Graph Transformation and Visual Modeling Techniques, Barcelona, Spain*, 2004.
20. Y. Kesten and A. Pnueli. Timed and hybrid statecharts and their textual representation. In *Proc. Formal Techniques in Real-Time and Fault-Tolerant Systems, 2nd International Symposium, LNCS 571*. Springer-Verlag, 1992.
21. R. Alur, T. Dang, J. Esposito, R. Fierro, Y. Hur, F. Ivancic, V. Kumar, I. Lee, P. Mishra, G. Pappas, and O. Sokolsky. Hierarchical Hybrid Modeling of Embedded Systems. In *First Workshop on Embedded Software*, 2001.
22. Thomas A. Henzinger. Masaccio: A Formal Model for Embedded Components. In *Proceedings of the First IFIP International Conference on Theoretical Computer Science (TCS), Lecture Notes in Computer Science 1872, Springer-Verlag, 2000, pp. 549-563.*, 2000.

23. R. Grosu, T. Stauner, and M. Broy. A modular visual model for hybrid systems. In *Proc. of Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'98), LNCS 1486*. Springer-Verlag, 1998.
24. Thomas Stauner. *Systematic Development of Hybrid Systems*. PhD thesis, Technical University Munich, 2001.
25. T. Stauner, A. Pretschner, and I. Péter. Approaching a Discrete-Continuous UML: Tool Support and Formalization. In *Proc. UML'2001 workshop on Practical UML-Based Rigorous Development Methods – Countering or Integrating the eXtremists*, pages 242–257, Toronto, Canada, October 2001.
26. Nancy Lynch, Roberto Segala, and Frits Vaandrager. Hybrid I/O Automata Revisited. In *Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control (HSCC 2001), Rome, Italy, March 28-30, 2001*, volume 2034 of *Lecture Notes in Computer Science*, pages 403–417. Springer Verlag, 2001.
27. L. Lamport. Hybrid Systems in TLA+. Springer-Verlag, 1993.
28. Thomas A. Henzinger, Marius Minea, and Vinayak Prabhu. Assume-Guarantee Reasoning for Hierarchical Hybrid Systems. In *Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control (HSCC 2001), Rome, Italy, March 28-30, 2001*, volume 2034 of *Lecture Notes in Computer Science*, pages 275–290. Springer Verlag, 2001.
29. Thomas A. Henzinger, Christoph M. Kirsch, Marco A.A. Sanvido, and Wolfgang Pree. From Control Models to Real-Time Code Using Giotto. In *IEEE Control Systems Magazine 23(1):50-64, 2003. A preliminary report on this work appeared in C.M. Kirsch, M.A.A. Sanvido, T.A. Henzinger, and W. Pree, A Giotto-based helicopter control system, Proceedings of the Second International Workshop on Embedded Software (EMSOFT), Lecture Notes in Computer Science 2491, Springer-Verlag, 2002, pp. 46-60.*, 2002.
30. Stefano Cattani and Marta Kwiatkowska. A refinement-based process algebra for timed automata. *Formal Aspects of Computing*, 17(2):138 – 159, August 2005.
31. Thomas A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: The Next Generation. In *Proc. of the 16th IEEE Real-Time Symposium*. IEEE Computer Press, December 1995.

A Prerequisites

In this appendix the formal foundations of the employed underlying hybrid and real-time automata model are outlined starting with continuous blocks and hybrid reconfiguration automata:

Definition 3. *A* continuous block M *is described by a 7-tuple* $(V^x, V^u, V^y, F, G, C, X^0)$ *with* V^x *the state variables,* V^u *the input variables, and* V^y *the output variables. For the implicitly defined state flow variables* $V^{\dot{x}}$ *and auxiliary variables* $V^a = V^y \cap V^u$*, the set of equations* $F \subseteq EQ(V^{\dot{x}} \cup V^a,\ V^x \cup V^u \cup V^a)$ *describes the flow of the state variables, the set of equations* $G \subseteq EQ(V^y \cup V^a,\ V^x \cup V^u \cup V^a)$ *determines the output variables, and* $X^0 \subseteq [V^x \to \mathbb{R}]$ *the set of initial states. The* invariant C *with* $C \in COND(V^x)$ *is further used to determine the set of valid states. The block* M *is* well-formed *if the system of differential equations* $F \cup G$ *contains no cyclic dependencies, no double assignments, all undefined referenced variables are contained in* $V^u - V^y$*, and a value is assigned to all state variables (*$V^{\dot{x}}$*) and output variables (*V^y*).*

Definition 4. *A* hybrid reconfiguration automaton *is described by a 6-tuple* (L, D, I, O, T, S^0) *with* L *a finite set of locations,* D *a function over* L *which assigns to each* $l \in L$ *a continuous model* $D(l) = (V^x(l), V^u(l), V^y(l), F(l), G(l), C(l), X^0(l))$ *conf. to Definition 3,* I *a finite set of input signals,* O *a finite set of output signals,* T *a finite set of transitions, and* $S^0 \subseteq \{(l, x) | l \in L \wedge x \in X(l)\}$ *the set of initial states. For any transition* $(l, g, g^i, a, l') \in T$ *holds that* $l \in L$ *is the source-location,* $g \in COND(V^x(l) \cup V^u(l))$ *the continuous guard,* $g^i \in \wp(I \cup O)$ *the I/O-guard,* $a \in [[V^x(l) \to \mathbb{R}] \to [V^x(l') \to \mathbb{R}]]$ *the continuous update, and* $l' \in L$ *the target-location. For every* $l \in L$ *we require that* $D(l)$ *is well-formed.*

We can further compose continuous as well as hybrid models as follows:

Definition 5. *The* composition *of two continuous models $M_1 = (V_1^x, V_1^u, V_1^y, F_1, G_1, C_1, X_1^0)$ and $M_2 = (V_2^x, V_2^u, V_2^y, F_2, G_2, C_2, X_2^0)$ denoted by $M_1 \| M_2$ is again a continuous model $M = (V^x, V^u, V^y, F, G, C, X^0)$ with $V_1^x := V_1^x \cup V_2^x$, $V_1^u := V_1^u \cup V_2^u$, $V_1^y := V_1^y \cup V_2^y$, $F := F_1 \cup F_2$, $G := G_1 \cup G_2$, C is derived from C_1 and C_2 as $C = \{(x_1 \otimes x_2) | x_1 \in C_1 \wedge x_2 \in C_2\}$, and the set of initial states is $X^0 = \{((l_1,\, l_2),\, (x_1 \otimes x_2)) | (l_1,\, x_1) \in X_1^0 \wedge (l_2,\, x_2) \in X_2^0\}$. $M_1 \| M_2$ is only well-formed when $V_1^x \cap V_2^x = \emptyset$, $V_1^u \cap V_2^u = \emptyset$, $V_1^y \cap V_2^y = \emptyset$, and F and G are well-formed.*

Definition 6. *For two hybrid reconfiguration automata M_1 and M_2 the* parallel composition *($M_1 \| M_2$) results in a hybrid reconfiguration automaton $M = (L, D, I, O, T, S^0)$ with $L = L_1 \times L_2$, $D(l, l') = D_1(l) \| D_2(l')$, $I = I_1 \cup I_2$, $O = O_1 \cup O_2$. $T = \{((l_1,\, l_2),\, g,\, ,g',\, (a_1 \oplus a_2),\, (l_1^{'},\, l_2^{'})) | (l_1,\, g_1, g_1',\, u_1,\, l_1^{'}) \in T_1 \wedge (l_2,\, g_2,\, g_2',\, u_2,\, l_2^{'}) \in T_2\} \cup \{((l_1,\, l_2),\, g_1',\, u_1,\, (l_1^{'},\, l_2)) | (l_1,\, g_1, u_1,\, l_1^{'}) \in T_1\} \cup \{((l_1,\, l_2),\, g_2',\, u_2,\, (l_1,\, l_2^{'})) | (l_2,\, g_2,\, u_2,\, l_2^{'}) \in T_2\}$ is the resulting transition relation where $g(x,\, u,\, i,\, o) = g_1(x,\, u,\, i,\, o) \wedge g_2(x,\, u,\, i,\, o)$, $g_1'(x, u, i, o) = g_1(x, u, i, o) \wedge o \cap I_2 = \emptyset \wedge i \cap O_2 = \emptyset$, and $g_2'(x, u, i, o) = g_2(x, u, i, o) \wedge o \cap I_1 = \emptyset \wedge i \cap O_1 = \emptyset$.*

The automaton M is only defined when for all reachable $(l, l') \in L$ holds that $D((l, l'))$ is defined and the internal signal sets are disjoint ($((O_1 \cap I_1) \cap (O_2 \cap I_2) = \emptyset$).

It is to be noted that the presented definition of hybrid reconfiguration automata includes timed automata. We simply have to define clock variables v_i which values are determined by equations $\dot{v}_i = 1$ in F to encode this feature of a timed automaton into a hybrid one and ensure the the continuous interface is empty.

For $X_c = [V^c \rightharpoonup \mathbb{R}]$ the set of possible clock variable bindings, the inner state of a timed automaton can then be described by a pair $(l,\, x) \in L \times X_c$ with $x \in [V^c \to \mathbb{R}]$. There are two possible ways of state modifications: Either by firing an instantaneous transition $t \in T$ changing the location as well as the state variables or by residing in the current location which consumes time and just increases the clock variables. When staying in state $(l,\, x)$ firing an instantaneous transition $t = (l',\, g,\, g^i,\, a,\, l'')$ is done iff $l = l'$ (the transitions source location equals the current location) and the clock guard is fulfilled ($g(x) = true$), the I/O-guard is true for the chosen input and output signal sets $i \subseteq I$ and $o \subseteq O$ ($i \cup o = g^i$), and $a(x) \in C(l'')$. The resulting state will be $(l'',\, a(x))$ and we note this firing by $(l,\, x) \to_{(i \cup o)} (l'',\, a(x))$.

If no instantaneous transition can fire, the timed automaton resides in the current location l for a non-negative and non-zero time delay $\delta > 0$. In state $(l,\, x)$ such time consuming transition may result in $(l,\, x \oplus \delta)$ if all $\delta' \in [0, \delta]$ holds $C(x \oplus \delta)$.

The trace semantics is thus given by all possible infinite execution sequences $(l_0, x_0, \delta_0) \to_{e_0} (l_1, x_1, \delta_1) \ldots$ denoted by $[\![M]\!]_t$ where all $(l_i, x_i \oplus delta_i) \to_{e_i} (l_{i+1}, x_{i+1})$ are valid instantaneous transition executions and $(l_i, x_i \oplus delta_i')$ are valid for all $\delta_i' \in [0, \delta_i]$. We use the timed path $\pi = (s_0, t_0) \to_{\delta_0} (s_0, t_0 \oplus \delta_0) \to_{a_0} (s_1, t_1) \to_{\delta_1} (s_1, t_1 \oplus \delta_1) \ldots$ to denote the externally visible real-time behavior. The offered interactions for a state (s, c) are further denoted by the set offer$(M, (s, t))$ which is defined as $\{a | \exists (s, t) \to_a (s', t') \in [\![M]\!]_t\}$. An appropriate notion of real-time refinement (also named timed ready simulation in [16]) can then be defined as follows:

Definition 7. *For two real-time automata M_I and M_R holds that M_R is a* refinement *of M_I denoted by $M_R \sqsubseteq_{RT} M_I$ iff a relation $\Omega \subseteq (S_R \times T_R) \times (S_I \times T_I)$ exists which contains for every $c \in (S_R \times T_R)$ a $c'' \in (S_I \times T_I)$ such that $(c, c'') \in \Omega$ and for all $(c, c'') \in \Omega$ holds*

$$\forall c \Rightarrow_\pi c' \quad \exists c'' \Rightarrow_\pi c''' \quad : (c', c''') \in \Omega \qquad \wedge \tag{4}$$

$$\textit{offer}(M_R, c) \supseteq \textit{offer}(M_I, c''). \tag{5}$$

As this refinement is a precongruence for $\|$ [5, 16], we can exclude time-stopping deadlocks etc. by looking only into more abstract behavioral models instead of their refinements.

To also describe hybrid behavior, we first consider continuous behavior. We denote dependencies between input and output variables using $D(M_i) \subseteq V_i^u \times V_i^y$ and in addition require that the equations are *well-formed*. The external visible dependencies $D^e(M_i)$ are accordingly defined as $D^e(M_i) := D(M_i) \cap (V_i^u - V_i^y) \times (V_i^y - V_i^u)$. The state space of a continuous behavior is $X_i = [V_i^x \rightarrow \mathbb{R}]$ which describes all possible assignments for the state variables.

For $X_i = [V_i^x \rightharpoonup \mathbb{R}]$ the set of possible continuous state variable bindings of a hybrid automaton, the inner state of a hybrid automaton can be described by a pair $(s, x) \in S_i \times X_i$. There are two possible ways of state modifications: Either by firing an instantaneous transition changing the state as well as the state variables or by residing in the current location which consumes time and alters just the control variables.

A trajectory $\rho_u : [0, \infty] \rightarrow [V_i^x \rightarrow \mathbb{R}]$ for the set of differential equations and input $u : [0, \infty] \rightarrow [V_i^u \rightarrow \mathbb{R}]$ with $\rho_u(0) = x$ for the current continuous state $x \in X_i$ describes a valid continuous behavior as long as no transition is fired. The output variables V_i^y are determined by $\theta_u : [0, \infty] \rightarrow [V_i^y \rightarrow \mathbb{R}]$ using the additional equations for these variables analogously.

The trace semantics for a hybrid automaton is thus given by all possible infinite execution sequences $(u_0, l_0, \rho^0_{u_0}, \theta^0_{u_0}, \delta_0) \rightarrow_{e_0} (u_1, l_1, \rho^1_{u_1}, \theta^1_{u_1}, \delta_1) \rightarrow_{e_1} (u_2, l_2, \rho^2_{u_2}, \theta^2_{u_2}, \delta_2) \ldots$ denoted by $[\![M]\!]_t$ where all $(l_i, \rho^i_{u_i}(\delta_i)) \rightarrow_{e_i} (l_{i+1}, \rho^{i+1}_{u_{i+1}}(0))$ are valid instantaneous transition executions.[5]

We further use the concept of a hybrid path $\pi = (u_0, \theta^0_{u_0}, \delta_0); a_0; \ldots; (u_n, l_n, \theta^1_{u_n}, \delta_n); a_n$ such that we write $(l_0, \rho^0_{u_0}(0)) \rightarrow_\pi (l_n, \rho^n_{u_n}(\delta_n))$ iff $(l_0, \rho^0_{u_0}(0)) \rightarrow_{(u_0, \rho^0_{u_0}, \theta^0_{u_0}, \delta_0)} (l_0, \rho^0_{u_0}(\delta_0)) \rightarrow_{a_0} \ldots (l_n, \rho^n_{u_n}(0)) \rightarrow_{(u_n, l_n, \rho^n_{u_n}, \theta^n_{u_n}, \delta_n)} (l_n, \rho^n_{u_n}(\delta_n))$. For A the set of externally relevant events and $\theta^i_{u_i} = \theta^i_{u_i}|_{V^y(l_i) - V^u(l_i)}$ the output minus the internal variables, we derive an abstract path simply by omitting events $a_i \notin A$ and restricting the input and output variables accordingly. The offered discrete as well as continuous interactions for a state (s, x) are further denoted by the set offer$(M, (s, x))$ which is defined as $\{a \in A | \exists (s, x) \Rightarrow_a (s', x')\} \cup \{((du/dt)(0), (d\theta_u/dt)(0)) | \exists (s, x) \Rightarrow_{(u, \theta_u, \delta)} (s, x')\}$. An appropriate notion of hybrid refinement for the interface can then be defined as follows:

Definition 8. *For two hybrid reconfiguration automata M_I and M_R holds that M_R is a* hybrid refinement *of M_I denoted by $M_R \sqsubseteq_{HY} M_I$ iff a relation $\Omega \subseteq (S_R \times X_R) \times (S_I \times X_I)$ exists which contains for every $c \in (S_R \times X_R)$ a $c'' \in (S_I \times X_I)$ such that $(c, c'') \in \Omega$ and for all $(c, c'') \in \Omega$ holds*

$$\forall c \Rightarrow_\pi c' \quad \exists c'' \Rightarrow_\pi c''' \quad : (c', c''') \in \Omega \qquad \wedge \tag{6}$$

$$\textit{offer}(M_R, c) \supseteq \textit{offer}(M_I, c'') \qquad \wedge \tag{7}$$

$$D^e(M_R, c) \subseteq D^e(M_I, c''). \tag{8}$$

[5] Other aspects of hybrid behavior, such as Zeno behavior and *urgent* resp. *non-urgent* transitions, are omitted here. A suitable formalization can be found, e.g., in [31].