

# Modular Design and Verification of Component-Based Mechatronic Systems with Online-Reconfiguration \*

Holger Giese, Sven Burmester,<sup>†</sup> and  
Wilhelm Schäfer  
Software Engineering Group  
University of Paderborn  
[hg|burmi|wilhelm]@upb.de

Oliver Oberschelp  
Mechatronic Laboratory Paderborn  
University of Paderborn  
Oliver.Oberschelp@mlap.de

## ABSTRACT

The development of complex mechatronic systems requires a careful and ideally verifiable design. In addition, engineers from different disciplines, namely mechanical, electrical and software engineering, have to cooperate. The current technology is to use block diagrams including discrete blocks with statecharts for the design and verification of such systems. This does not adequately support the verification of large systems which improve the system behavior at run-time by means of online reconfiguration of its controllers because the system as whole has to be verified. It also does not support cooperative interdisciplinary work because a white-box view on all blocks involved in the online reconfiguration is required. This paper proposes a rigorous component concept based on the notion of UML component diagrams which enables modular composition and decomposition of complex systems with online reconfiguration given by hierarchical hybrid component specifications. The approach enables compatibility checks between components that are often independently developed (across the different disciplines) and supports compositional model checking based on a rigorously defined semantics.

## Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Techniques—*Modules and interfaces, State diagrams*; D.2.4 [Software Engineering]: Software/Program Verification—*Model checking*; I.2.8 [Artificial Intelligence]: Problem Solving, Control Methods, and Search—*Control theory*

## General Terms

Design, Languages, Verification

<sup>†</sup>Supported by the International Graduate School of Dynamic Intelligent Systems. University of Paderborn

\*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.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGSOFT'04/FSE-12, Oct. 31–Nov. 6, 2004, Newport Beach, CA, USA.  
Copyright 2004 ACM 1-58113-855-5/04/0010 ...\$5.00.

## Keywords

Hybrid Systems, Components, Reconfiguration, Unified Modelling Language (UML), Real-Time

## 1. INTRODUCTION

Mechatronic systems combine technologies from mechanical and electrical engineering as well as from computer science [5]. The development of complex mechatronic systems, in particular the software of those systems, has become a major challenge as the tight collaboration which is required between the different disciplines is difficult to achieve. The functionality of these systems is to a large extent defined by their software and its often complex interaction with the hardware.

This interplay between soft- and hardware is in turn mainly defined by the interface between the discrete, event-based software controllers and quasi-continuous feedback controllers such that those systems are usually called hybrid systems [2, 18, 28].

Those hybrid systems are usually time and safety critical. The verification of their models is highly desirable to avoid failures during operation of the system, because testing is not sufficient and often not possible under real environment conditions.

Current standard technology for the design of such systems uses block diagrams which contain blocks with quasi-continuous behavior as well as discrete blocks with statecharts to describe the discrete behavior.

*Online reconfiguration* means that continuous controllers are exchanged at run-time and that the communication structure is changed. It enables an improvement of the system behavior significantly as our example in Section 2 will illustrate.

Modeling reconfigurable systems with block diagrams and statecharts is rather cumbersome. The resulting dependencies easily spread all over the whole model which complicates understanding and formal verification. The behavior of the overall system has to be studied exhaustively, which is usually only possible for small systems with only linear continuous behavior. Approaching reconfigurable systems with block diagrams does therefore not scale and usually results in much manual, ad-hoc work to arrive at a reasonably correct system design.

In contrast to the commonly used separated discrete blocks, hybrid automata/statecharts [13, 4, 3, 2, 18, 28, 17] assign a control law or continuous controller (in form of a block diagram) to each discrete state. They thus result in a tighter integration between the discrete behavior and the continuous control when designing systems with reconfiguration. As the interface between the statecharts and their environment remains static, reconfiguration can, however, only take place locally within each single statechart, which therefore becomes overly complex.

Our approach introduces a rigorously defined concept of a hybrid component, based on the UML 2.0 component diagram notation, which has been informally introduced in [7]. The interface of the hybrid component enables embedded components to be coordinated appropriately by the embedding component without referring to all their implementation details. Within the components, an extended hybrid version of UML Statecharts is employed to enable a clear separation between the discrete system specification by event-based state-transition systems and the continuous system specifications by differential equations and block diagrams.

It supports compositional verification by model checking [9] and thus also scales for complex systems, at least concerning the proof of the correctness of the specified timing properties and consistent reconfiguration. It also supports a rather independent system design by classical engineers who develop the specification of the continuous part and software engineers who develop the specification of the discrete part.

The need for such an approach is underlined by the OMG request for a proposal of UML for Systems Engineering [22]. Current responses to this request are currently being evaluated but to the best of our knowledge they do not address support for compositional verification and online reconfiguration.

We first review the state-of-the-art in modeling for reconfigurable mechatronic systems and related work in Section 2 and additionally define the semantics foundations. Then, the proposed notion for hybrid components is outlined and rigorously defined in Section 3. The hierarchical composition of hybrid components, the modular verification of their correct reconfiguration, and the integration with compositional model checking follows in Section 4. Thereafter, we sum up with a final conclusion.

## 2. STATE OF THE ART MODELING AND RELATED WORK

As a concrete example for a complex mechatronic product we use a version of the software for the RailCab research project.<sup>1</sup> The vision of the RailCab project is an entirely new type of mechatronic rail system, where autonomous shuttles apply the linear drive technology used in the Transrapid, but travel on the existing passive track system of the standard railway.

One particular problem that shows the interconnection between the shuttle's real-time behavior and its feedback control software is the control of the suspension/tilt module. In this paper, we present the design of this control software. The schema of the relevant physical model of our example is shown in Figure 1. The active spring-based displacement is effected by hydraulic cylinders. Three vertical hydraulic cylinders, arranged on a plane, move the bases of the air springs via an intermediate frame, the suspension frame. This arrangement allows a damping of forces in lateral and vertical directions. In addition, it is also possible to regulate the level of the coach and add active tilting of the coach body. Three additional hydraulic cylinders allow a simulation of vertical and lateral rail excitation [16]. The vital task for the control system is to control the dynamical behavior of the coach body. In our example, we will focus only on the vertical dynamic behavior of the coach body. The choice of appropriate feedback controllers for this module is indispensable to provide the passengers a high comfort and it is highly relevant for energy optimizations, safety, and stability.

There exist multiple different controllers applicable to the suspension/tilt module. We focus on 3 controllers with different in- and outputs, providing different comfort: One controller provides sophisticated comfort by referring to a trajectory describing the re-

<sup>1</sup><http://www-nbp.upb.de/en/index.html>

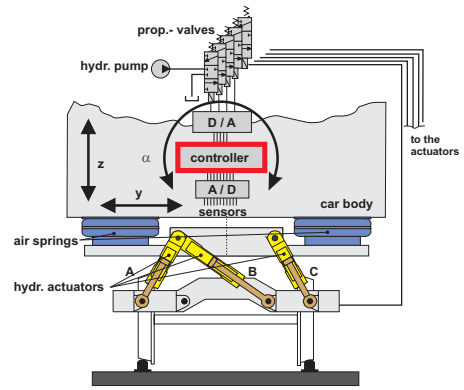


Figure 1: Scheme of the suspension/tilt module

quired motion of the coach body in order to compensate the current track's unevenness, slopes, etc. As a track's optimal reference curve does not change, shuttles can profit from the experiences of former shuttles passing the track. Therefore a shuttle sends its experience to a registry that provides this information to other shuttles.

This leads to the following procedure when a shuttle enters a registry's area: (1) the shuttle requests the trajectory from the local registry, (2) the registry selects the appropriate track reference curve and sends it to the shuttle, (3) the shuttle passes the track using the reference curve, (4) the shuttle sends an experience report to the local registry, (5) the registry uses the shuttle's experience to compute a new, optimized reference curve [21].

To guarantee stability the sophisticated *reference* controller requires –besides the reference curve– the vertical acceleration of the coach body, which is delivered by a sensor. If the reference curve is not available or is not received in time the less comfortable *absolute* controller has to be applied, which requires only the vertical acceleration as input. If the sensor fails, our *robust* controller has to be applied, which provides the lowest comfort, but requires just standard inputs to guarantee stability.

The example shows that the transmission of the reference trajectory has to meet real-time requirements. Further it demonstrates that self-optimization often results in an online-reconfiguration of the feedback controllers.

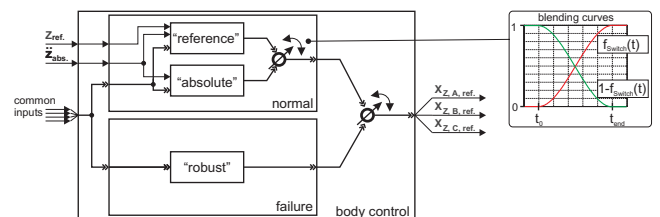


Figure 2: Fading between different continuous control modes

Control engineering systems, such as the control system for the suspension/tilt module, are usually described by means of *block diagrams* [24] as depicted in Figure 2. The body control (BC) component, which is responsible for controlling the suspension/tilt module, consists of the three described controllers. Dependent on the available input signals, the respective controllers are active or not. The reference signal is labeled as  $z_{ref}$  and the absolute acceleration as  $\ddot{z}_{abs}$ . The outputs  $X_{Z,A,ref}, \dots, X_{Z,C,ref}$  denote the positions of the three hydraulic cylinders.

When switching between two controllers one must 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. In our example, the switching from the normal block to the failure block (see Figure 2) can be processed atomically because the robust controller actually has no state.

If the operating points of the controllers are not identical, it will be necessary to cross-fade between the two controllers. This handling is required in the normal block depicted in Figure 2, where a transition between the reference and the absolute controller is realized. The cross-fading itself is specified by a fading function  $f_{switch}(t)$  and an additional parameter which determines the duration of the cross-fading.

To study the limitations of the block diagrams, we review their formal model by means of differential equations. It describes the behavior of a block diagram's single block or of multiple interconnected blocks as follows (see Appendix A for the employed basic mathematical notations):

**DEFINITION 1.** A continuous model  $M$  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  $V^x$  and auxiliary variables  $V^a = V^y \cap V^u$ , the set of equations  $F \subseteq EQ(V^x \cup V^a, V^x \cup V^u \cup V^a)$  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  $X^0 \subseteq [V^x \rightarrow \mathbb{R}]$  the set of initial states. The invariant  $C$  with  $C \in COND(V^x)$  is further used to determine the set of valid states.

$F \cup G$  is only well-formed when there are no cyclic dependencies, no double assignments, and when all undefined referenced variables are contained in  $V^u - V^y$ . A well-formed  $F \cup G$  must also assign a value to all state and output variables present in the definition.

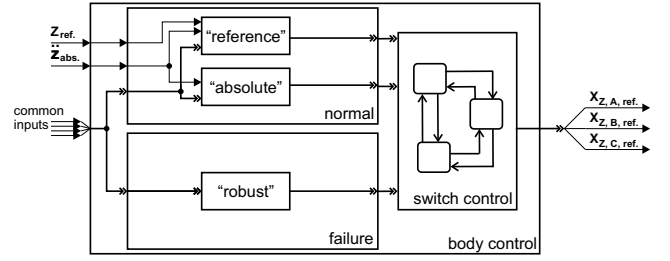
The state space of a continuous behavior is  $X = [V^x \rightarrow \mathbb{R}]$  which describes all possible assignments for the state variables. A trajectory  $\rho_u : [0, \infty) \rightarrow [V^x \rightarrow \mathbb{R}]$  for the set of differential equations  $F$  and input  $u : [0, \infty) \rightarrow [V^u \rightarrow \mathbb{R}]$  with  $\rho_u(0) = x$  for the current continuous state  $x \in X$  and  $\rho_u(t) \in C$  for all  $t \in [0, \infty)$  describes a valid behavior of the continuous system. The output variables  $V^y$  are determined by  $\theta_u : [0, \infty) \rightarrow [V^y \rightarrow \mathbb{R}]$  using  $G$  analogously. The semantics for a continuous model  $M$  is given by all possible triples of environment and system trajectories  $(u, \rho_u, \theta_u)$  denoted by  $\llbracket M \rrbracket$ .

We can compose two continuous models if their variable sets are not overlapping and the resulting sets of equations are well formed as follows:

**DEFINITION 2.** 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  $M_1 \parallel M_2$  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  $X^0 = \{((l_1, l_2), (x_1 \otimes x_2)) | (l_1, x_1) \in X_1^0 \wedge (l_2, x_2) \in X_2^0\}$ .

$M_1 \parallel M_2$  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. A composition is consistent if the resulting continuous model is well-formed.

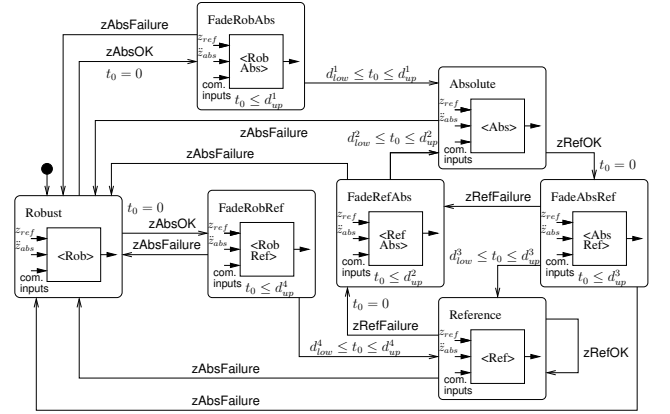
Within the model atomic switching and fading is described by specific continuous blocks. A standard approach to include discrete blocks is to restrict states and signals to a discrete domain. The integration of discrete control elements into block diagrams is depicted



**Figure 3: Controlling the fading between different continuous control modes with a statechart**

in Figure 3. The alternative controller outputs are fed into a discrete block whose behavior is described by a statechart. The de facto industry standard employing this concept is MATLAB/Simulink and Stateflow.<sup>2</sup> Formal verification of MATLAB/Simulink and Stateflow models of moderate size can be accomplished by automatically transforming them to hybrid automata (cf. [1]).

In the standard approach, the continuous control and the discrete statecharts are separated, while we can observe strong relations between specific controller configurations and states for reconfiguration. Hybrid automata [13] bridge this gap by simply assigning a specific continuous controller to each discrete state.



**Figure 4: Hybrid view of the body control with additional fading locations**

Figure 4 shows the body control (BC) component modeled by a hybrid automaton. Among others it consists of the three locations (discrete states) Robust, Absolute, and Reference whose continuous dynamics are specified by continuous models conform to Definition 1. The controller specifications are visualized by blocks with arrows, denoting the in- and outputs.

If for instance the automaton resides in the start location Robust and zAbsOK is raised (to indicate that the  $z_{abs}$  signal is available) the location and the controller changes to the absolute mode. An atomic switch between these controllers can usually not avoid an additional excitation or even guarantee stability. Thus the Absolute location cannot be entered directly and an additional location FadeRobAbs is entered (see Figure 4). This intermediate location comprises the cross fading activity (transition) from the robust to the absolute controller.

Since fading can only guarantee stability if its duration is within

<sup>2</sup><http://www.mathworks.com>

specific bounds, we describe the lower and the upper bound by a *fading duration interval*  $d_1 = [d_{low}^1, d_{up}^1]$ . The duration is modeled through the state variable (clock)  $t_0$  which is set to 0 when entering the *FadeRobAbs*-location. The invariant  $t_0 \leq d_{up}^1$  and the guard  $d_{low}^1 \leq t_0 \leq d_{up}^1$  ensure the specific bounds. Note that inside the fading locations  $t_0 = 1$  holds for any clock variable  $t_0$ . When the fading is completed, the original target location *Absolute* is entered. Specifying the duration of the other fading transitions is done similarly. If the  $\ddot{z}_{abs}$  signal is lost during fading or during the use of the absolute-controller, the default location with its robust control will be entered immediately by the transition which is triggered by the *zAbsFailure* event (cf. Figure 4).

The described behavior comprising continuous as well as discrete elements is formally defined by means of hybrid automata and statecharts [13, 4, 3]. In the following, we use a formalization which extends the notion of a continuous model defined in Definition 1. It provides means to specify continuous behavior and synchronous event handling.

**DEFINITION 3.** A hybrid 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  $D(l) = (V^x, V^u, V^y, F(l), G(l), C(l), X^0(l))$  conf. to Definition 1 with identical variable sets,  $I$  a finite set of input signals,  $O$  a finite set of output signals,  $T$  a finite set of transitions, and a set of initial states  $S^0 \subseteq \{(l, x) | l \in L \wedge x \in X^0(l)\}$ . For any transition  $(l, g, g', a, l') \in T$  holds that  $l \in L$  is the source-location,  $g \in COND(V^x \cup V^u)$  the continuous guard,  $g' \in \wp(I \cup O)$  the I/O-guard,  $a \in [[V^x \rightarrow \mathbb{R}] \rightarrow [V^x \rightarrow \mathbb{R}]]$  the continuous update, and  $l' \in L$  the target-location. For every  $l \in L$  we require that  $D(l)$  is well-formed.

The used interface  $I(M)$  of a hybrid automaton  $M$  is defined as the external visible signal sets and input and output variables  $(I - O, O - I, V^u - V^y, V^y - V^u)$ .

For  $X = [V^x \rightarrow \mathbb{R}]$  the set of possible continuous state variable bindings, the inner state of a hybrid automaton can be described by a pair  $(l, x) \in L \times X$  with  $x \in [V^x \rightarrow \mathbb{R}]$ . 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 alters just the control variables.

When staying in state  $(l, x)$  firing an instantaneous transition  $t = (l', g, g', a, l')$  is done iff  $l = l'$  (the transitions source location equals the current location) and the continuous guard is fulfilled ( $g(x \otimes u) = true$ ) for  $u \in [V^u \rightarrow \mathbb{R}]$  the current input variable binding, 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  $(l, x) \xrightarrow{(i \cup o)} (l'', a(x))$ .

If no instantaneous transition can fire, the hybrid automaton resides in the current location  $l$  for a non-negative and non-zero time delay  $\delta > 0$ . Let  $\rho_u : [0, \delta] \rightarrow [V^x \rightarrow \mathbb{R}]$  be a trajectory for the differential equations  $F(l)$  and the external input  $u : [0, \delta] \rightarrow [V^u - V^y \rightarrow \mathbb{R}]$  with  $\rho_u(0) = x$ . The state for all  $t \in [0, \delta]$  will be  $(l, \rho_u(t))$ . The output variables  $V^y - V^u$  and internal variables  $V^y \cap V^u$  are determined by  $\theta_u : [0, \delta] \rightarrow [V^y \rightarrow \mathbb{R}]$  using  $G(l)$  analogously. We additionally require that for all  $t \in [0, \delta]$  holds that  $\rho_u(t) \in C(l)$ .

The trace semantics is thus given by all possible infinite execution sequences  $(u_0, l_0, \rho_{u_0}^0, \theta_{u_0}^0, \delta_0) \xrightarrow{e_0} (u_1, l_1, \rho_{u_1}^1, \theta_{u_1}^1, \delta_1) \dots$  denoted by  $\llbracket M \rrbracket_t$  where all  $(l_i, \rho_{u_i}^i(\delta_i)) \xrightarrow{e_i} (l_{i+1}, \rho_{u_{i+1}}^{i+1}(0))$  are valid instantaneous transition executions.

Other aspects of hybrid behavior, such as zeno behavior and the distinction between *urgent* and *non-urgent* transitions, are omitted here. A suitable formalization can be found, e.g., in [13]. The parallel composition of two hybrid automata is defined as follows:

**DEFINITION 4.** For two hybrid automata  $M_1$  and  $M_2$  the parallel composition  $(M_1 \parallel M_2)$  results in a hybrid automaton  $M = (L, D, I, O, T, S^0)$  with  $L = L_1 \times L_2$ ,  $D(l, l') = D_1(l) \parallel D_2(l')$ ,  $I = I_1 \cup I_2$ ,  $O = O_1 \cup O_2$ . The resulting transition relation is  $T = \{((l_1, l_2), g_1 \wedge g_2, g_1^i \cup g_2^i, (a_1 \oplus a_2), (l'_1, l'_2)) | (l_1, g_1, g_1^i, u_1, l'_1) \in T_1 \wedge (l_2, g_2, g_2^i, u_2, l'_2) \in T_2 \wedge g_1^i \cap (I_2 \cup O_2) = g_2^i \cap (I_1 \cup O_1)\} \cup \{((l_1, l_2), g_1, g_1^i, u_1, (l'_1, l'_2)) | (l_1, g_1, g_1^i, u_1, l'_1) \in T_1 \wedge g_1^i \cap (I_2 \cup O_2) = \emptyset\} \cup \{((l_1, l_2), g_2, g_2^i, u_2, (l_1, l'_2)) | (l_2, g_2, u_2, l'_2) \in T_2 \wedge g_2^i \cap (I_1 \cup O_1) = \emptyset\}$ .  $S^0$  is defined as  $S_1^0 \times S_2^0$ .

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

Specification of the optimization of the system behavior requires an appropriate coordination for the hierarchical integrated sub-components by the super-component. In our example, some sort of monitor which embeds the body control component has to take action when additional information in form of the reference curve is available or lack of sensor data has to be compensated. This monitor initiates the location switches (e.g. by raising the signals *zAbsOK* etc.) and handles the communication with the Registry.

Discrete blocks as well as hybrid automata support only static (continuous) interfaces (cf. Definition 1 and 3). Thus reconfiguration as well as its coordination is always restricted to occur only within one discrete block or hybrid automaton. The parallel composition of hybrid automata in their standard form can therefore not be employed to decompose behavior which includes reconfiguration.

In our example, the monitor behavior as well as the hybrid automaton describing the behavior of the BC component has to be modeled within one hybrid statechart. In more complex systems, the whole hierarchy and possible cascading controllers have to be specified within one hybrid statechart with parallel states. Therefore, the hybrid statecharts will often become too complex to verify that the continuous model is well-formed for all reachable states.

There is a number of approaches, like Timed and Hybrid Statecharts [17], Charon [3], Masaccio [12], HyCharts [10, 26], HyRoom [27], and Hybrid I/O Automata [20], which address the problem of modeling complex systems by hybrid statecharts. Some of them reduce the visual complexity by means of hierarchy and parallelism. They all fail in providing a component concept which supports a dynamic interface which enables to decompose systems with online reconfiguration into multiple hybrid statecharts.

Consequently 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 statechart. Thus the usually difficult tight cooperation between engineers from different camps is required.

Available compositional reasoning approaches for hybrid systems [18, 15] require large manual effort of inventing auxiliary properties to enable a full verification to decide whether the described reconfiguration is consistent. In contrast, the presented approach will ensure consistency by means of a syntactical check guided by the proposed components and their interfaces.

Further the example shows that even simple examples are – especially due to the fading-locations– so complex, that they become difficult to comprehend. The application of classical high-level constructs, such as hierarchy, parallelism and history, enable some reduction of complexity, but mechatronic systems usually achieve a complexity that requires further advanced concepts such as modularity and a component-based design.

### 3. HYBRID COMPONENTS

To support the design of complex mechatronic systems and to overcome the problems outlined in the last section, we introduce in this section our notion of hybrid components and their rigorously defined semantics. Hybrid components have been introduced informally in [7]. As the Unified Modeling Language (UML) is accepted worldwide as the quasi-standard for modeling, we apply components in the sense of UML 2.0 [23].

#### 3.1 Component Structures

Figure 5 depicts the component structure of our example by means of a UML component diagram: The Monitor component embeds the subordinated Sensor, Storage, and BodyControl (BC) components.

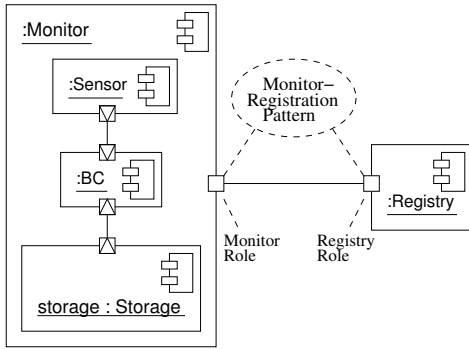


Figure 5: Monitor and its environment

Using the basic UML 2.0 concepts for component diagrams, the hierarchical embedding of the BC component into the Monitor component is modeled using aggregation as presented in Figure 5. 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. E.g. the continuous signal  $\tilde{z}_{abs}$  is transmitted from the Sensor component to the BC component through their continuous ports.

The Monitor-Registry pattern uses a subset of UML 2.0 proposed in [9] and specifies the time-discrete communication between the shuttle’s Monitor component and the Registry. It consists of two roles (MonitorRole and RegistryRole) which specify the protocol for the communication. The role’s behavior, which is considered later in Figure 10 in Section 4.3, is modeled with an extension of statecharts [8].

#### 3.2 Component Realization

The behavior of the hybrid component is specified by means of an extension of UML Statecharts called *hybrid reconfiguration charts*. We employ Real-Time statecharts [8] to describe required real-time behavior and refer the continuous behavior only by embedding ap-

propriate basic quasi-continuous blocks similar to Figure 4 (cf. the BC component behavior in Figure 6).

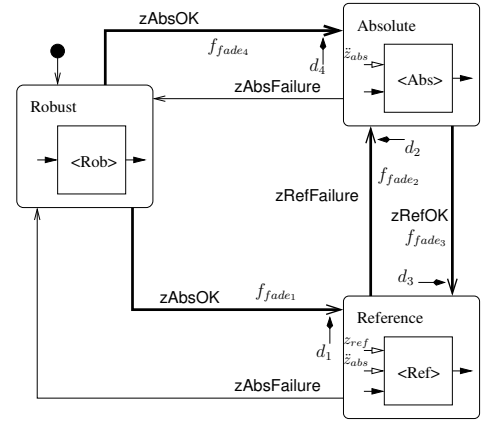


Figure 6: Behavior of the body control component

Within the states of a hybrid reconfiguration chart, the required controller logic with its specific required input and provided output signals is specified (cf. Figure 6), while a hybrid automaton specification requires always the same input and output signals for every location. The continuous ports that are required in each of the three interfaces are filled black, the ones that are only used in a subset of the states are filled white.

To reduce the visual complexity, arising from the fading-states, we additionally support *fading-transitions* in our notion of hybrid reconfiguration charts. The *fading-transitions* are visualized by thick arrows while atomic switches have the shape of regular arrows. The 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.

A comparison with Figure 4 of the related hybrid automaton reveals that besides the locations Robust, Absolute, and Reference, representing the three different controllers, the additional locations FadeRobAbs, FadeRobRef, FadeAbsRef, and FadeRefAbs, regulating fading between the controllers, have been necessary. In Figure 6, the ability to avoid the explicit locations for fading considerably decreases the number of visible locations and thus comprehension is much simpler.

For the considered domain of mechatronic systems, the rather complex micro step semantics of UML statecharts is not necessary. Instead, the quasi-continuous behavior is evaluated constantly and in each state machine cycle only a single transition is fired. Such a semantics has already successfully been employed in [9] for the timed case only. [19] explains that the micro step semantics creates a lot of difficulties concerning the composition of statecharts and their corresponding semantics. Our simplified semantic definition avoids a lot of these problems.

Due to lack of space we further will omit the syntactical complexity of the standard statechart concepts, such as hierarchy and history [11] within this paper.<sup>3</sup> We define a reconfigurable variant of the hybrid automata model presented in Definition 3 which extends the formal concepts of hybrid automata and statecharts [13, 4, 3] to also support the specification of reconfiguration.

<sup>3</sup>This can be accomplished much like the case of the syntax and semantics of the Real-Time Statechart presented in [8].

**DEFINITION 5.** 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 1,  $I$  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,  $g \in \text{COND}(V^x(l) \cup V^u(l))$  the continuous guard,  $g^i \in \wp(I \cup O)$  the I/O-guard,  $a \in [[V^x(l) \rightarrow \mathbb{R}] \rightarrow [V^x(l') \rightarrow \mathbb{R}]]$  the continuous update, and  $l' \in L$  the target-location. For every  $l \in L$  we require that  $D(l)$  is well-formed.

The automaton additionally allows that each location has its own variable sets. We use  $V^x$  to denote the union of all  $V^x(l)$ .  $V^u$  and  $V^y$  are derived analogously. The semantics can be adjusted by always taking into account the location dependent notion  $V^x(l)$  etc. instead of the location independent  $V^x$ .

The parallel composition also follows directly from the non configurable case. In the case of hybrid reconfiguration automata, a correct parallel composition has to ensure that for all reachable  $(l, l') \in L$  holds that  $D((l, l'))$  does not contain cyclic dependencies. In contrast to the case of standard hybrid automata, the added support for changing input and output variables may also result in problems when required inputs are not provided. In our example, the sensor element must be in a state which provides the according data if the BC component is in state Absolute, otherwise BC cannot operate correctly.

Please note that in the presented flat hybrid automata model the higher-level concept of the hybrid statecharts such as fading transitions are represented by means of additional locations at the underlying flat hybrid automaton. The location set is thus partitioned into *regular locations*, which relate to locations of the statechart, and *fading locations*, which result from the fading transitions.

**DEFINITION 6.** For a hybrid automaton  $M = (L, D, I, O, T, S^0)$  a location  $l_f \in L$  with  $D(l_f) = (V^x(l_f), V^u(l_f), V^y(l_f), F(l_f), G(l_f), C(l_f), X^0(l_f))$  is a fading location iff  $C(l_f) \equiv (v \leq d_{max})$ ,  $\exists v \in V^x(l_f)$  with  $(\dot{v} = 1) \in F(l_f)$ , for all  $(l, g, g', a, l') \in T$  holds that  $(v = 0) \in a$ , there is exactly one transitions leaving  $l_f$  ( $|\{(l_f, g, g', a, l') | (l_f, g, g', a, l') \in T\}| = 1$ ), and for this transition holds  $g \equiv d_{min} \leq v \leq d_{max}$ ,  $g' = \text{true}$  and  $a = \text{Id}$ . All non fading locations are regular locations.

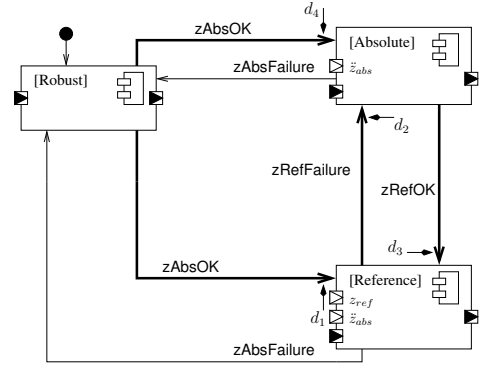
**DEFINITION 7.** For a hybrid automaton  $M = (L, D, I, O, T, S^0)$  a regular location  $l_p \in L$  is a passive location iff the location and all transitions leaving it have no continuous constraints.

Note that for any hybrid reconfiguration chart, we can ensure that two fading locations are never directly connected.

### 3.3 Component Interface

For embedding or connecting a hybrid component (cf. Figure 5) 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. This externally relevant behavior is described in our approach through an *interface state chart*. This interface state chart describes the externally visible states of a component as well as the in- and outputs present in each of these states.

The related interface automaton of the body control component of Figure 6 is displayed in Figure 7. It shows that the body control



**Figure 7: Interface state chart of the body control component**

component has three possible different external relevant states with different continuous interfaces. For all possible state changes, only the externally relevant information, such as possible durations and the signals to initiate and to break the transition, are present.

To study what a correct relation between the realization of a component and its interface automaton is, we write for a possible execution sequence of states and transitions of a hybrid automaton  $M = (L, D, I, O, T, S^0)$  with  $(u_0, l_0, \rho_{u_0}^0, \theta_{u_0}^0, \delta_0) \rightarrow_{e_0} (u_1, l_1, \rho_{u_1}^1, \theta_{u_1}^1, \delta_1) \in \llbracket M \rrbracket t$  simply  $(l_0, \rho_{u_0}^0(0)) \rightarrow_{(u_0, \rho_{u_0}^0, \theta_{u_0}^0, \delta_0)} (l_0, \rho_{u_0}^0(\delta_0)) \rightarrow_{e_0} (l_1, \rho_{u_1}^1(0)) \rightarrow_{(u_1, l_1, \rho_{u_1}^1, \theta_{u_1}^1, \delta_1)} (l_1, \rho_{u_1}^1(\delta_1))$  to represent the state changes in a more uniform manner. We thus have the concept of a hybrid path  $\pi = (u_0, \theta_{u_0}^0, \delta_0); e_0; \dots; (u_n, l_n, \theta_{u_n}^1, \delta_n); e_n$  such that we write  $(l_0, \rho_{u_0}^0(0)) \rightarrow_{\pi} (l_n, \rho_{u_n}^n(\delta_n))$  iff it holds that  $(l_0, \rho_{u_0}^0(0)) \rightarrow_{(u_0, \rho_{u_0}^0, \theta_{u_0}^0, \delta_0)} (l_0, \rho_{u_0}^0(\delta_0)) \rightarrow_{e_0} \dots (l_n, \rho_{u_n}^n(0)) \rightarrow_{(u_n, l_n, \rho_{u_n}^n, \theta_{u_n}^n, \delta_n)} (l_n, \rho_{u_n}^n(\delta_n))$ .

For  $e'_i = e_i - (O \cap I)$  the externally relevant events and  $\theta_{u_i}^i = \theta_{u_i}^i |_{V^y(l_i) - V^u(l_i)}$  the output minus the internal variables, we have an abstract path  $\pi' = (u_0, \theta_{u_0}^0, \delta_0); e'_0; \dots; (u_n, \theta_{u_n}^1, \delta_n); e_n; \dots$  and write  $(l_0, \rho_{u_0}^0(0)) \Rightarrow_{\pi'} (l_n, \rho_{u_n}^n(\delta_n))$ . Note that  $w; e; w'$  with  $e = \emptyset$  is collapsed to  $w; w'$  as no externally relevant events are received or emitted. The offered discrete as well as continuous interactions for a state  $(l, x)$  are further denoted by the set  $\text{offer}(M, (l, x))$  which is defined as  $\{e | \exists (l, x) \Rightarrow_e (l', x)\} \cup \{(du/dt)(0) | \exists (l, x) \Rightarrow_{(u, \theta_u, \delta)} (l, x')\}$ .

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 refinement of  $M_I$  denoted by  $M_R \sqsubseteq M_I$  iff a relation  $\Omega \subseteq (L_R \times X_R) \times (L_I \times X_I)$  exists which contains for every  $c \in (L_R \times X_R)$  a  $c' \in (L_I \times X_I)$  such that  $(c, c') \in \Omega$  and for all  $(c, c'') \in \Omega$  holds

$$\forall c \Rightarrow_{\pi} c' \quad \exists c'' \Rightarrow_{\pi} c'' \quad : (c', c'') \in \Omega \quad \text{and} \quad (1)$$

$$\text{offer}(M_R, c) \supseteq \text{offer}(M_I, c'). \quad (2)$$

As refinement is a precongruence for  $\parallel$  [6], we can exploit its property to preserve time-stopping deadlocks to employ where required the smaller abstraction (interface automaton) rather than the larger refinement (realization).

The externally relevant behavior covered by the interface state charts only includes the real-time behavior as well as the state-dependent continuous interface. Therefore, the notion of an interface automaton is essentially restricted to a timed automaton as follows:

DEFINITION 9. A hybrid automaton  $M = (L, D, I, O, T, S^0)$  is an interface automaton iff for its continuous part  $D$  holds that the set of auxiliary variables is empty ( $V^y \cap V^u = \emptyset$ ), all  $v \in V^x$  are clocks ( $\dot{v} = 1$ ), the update  $a$  for any transition  $(l, g, g^i, a, l')$  is restricted to  $OP_{const}$ , and the continuous input/output behavior for  $V^y$  is not determined ( $G$  is restricted to  $OP_{\perp}$ ).

Note that the concrete operations used in  $G$  do not restrict the possible trajectories and are only used to abstract from the evaluation dependencies.

A further, more restricted variant are *simple interface state charts* where only time constraints on the fading transitions are present.

DEFINITION 10. An interface automaton  $M = (L, D, I, O, T, S^0)$  is simple if it contains only passive and fading locations and two fading locations are never directly connected.

A component in our approach is thus described as a UML component –with ports with distinct quasi-continuous and discrete signals and events– as follows: A *hybrid component* is characterized by

- the *realization* described by a single hybrid reconfiguration chart which coordinates its aggregated subcomponents.
- an interface in form of an *interface state chart* which is a correct abstraction of the realization (cf. Definition 8).

In our example, the BC component is described by its realization by the hybrid reconfiguration chart of Figure 6 where the required quasi-continuous behavior is specified by controllers in form of quasi-continuous blocks. The interface state chart presented in Figure 7 describes the interface.

Using the notion of an interface automaton, we can thus formally define a hybrid component as a realization plus such an abstraction.

DEFINITION 11. A hybrid reconfiguration component  $C$  is a pair  $(M_I, M_R)$  with an interface automaton  $M_I = (L_I, D_I, I_I, O_I, T_I, S_I^0)$  and the concrete hybrid realization  $M_R = (L_R, D_R, I_R, O_R, T_R, S_R^0)$  for which  $M_R \sqsubseteq M_I$  holds. We further require that  $V_I^u = V_R^u - (V_R^u \cap V_R^y)$ ,  $V_I^y = V_R^y - (V_R^u \cap V_R^y)$ ,  $I_I = I_R - (I_R \cap O_R)$ ,  $O_I = O_R - (I_R \cap O_R)$ , and a witness  $\Omega$  for  $M_R \sqsubseteq M_I$  exists such that for any  $(l, l') \in \Omega$  and  $D_I(l) = (V_I^x(l), V_I^u(l), V_I^y(l), F_I(l), G_I(l), C_I(l), X_I^0(l))$  and  $D_R(l') = (V_R^x(l'), V_R^u(l'), V_R^y(l'), F_R(l'), G_R(l'), C_R(l'), X_R^0(l'))$  all dependencies present in  $G_I(l)$  must also be present in  $G_R(l')$ .

The interface automaton abstracts from the continuous behavior, it still contains the information about the input-output dependencies. The notion of a hybrid component thus permits to abstract from all internal variables and signals using the interface automaton.

In a *bottom-up* scenario, the interface of a component can be derived from its hybrid reconfiguration chart by abstracting from realization details at the syntactical level. The valid refinement between a given interface state chart and a realization in a *top-down* scenario must in contrast be additionally verified at the semantical level.

## 4. COMPONENT COMPOSITION

As depicted in Figure 5, our hybrid components can have two different kinds of composition relations: (1) explicit port connections to other components or (2) strict hierarchical aggregation of subcomponents by a super-component.

In the former case, the components are coupled via ports and patterns (cf. Monitor and Registry in Figure 5). As at this level only

the real-time behavior without any continuous elements has to be specified, UML 2.0 components with ports and real-time protocols are sufficient to exclude timing inconsistencies (see Section 4.3).

In the latter case, we have to ensure that the different reconfiguration steps as well as their timing cannot result in an inconsistent situation where the continuous model is not well-formed any more. While in the general case the whole discrete state space of the system has to be explored to exclude this problem, we can exploit the modular structure of complex mechatronic systems modeled as hierarchies of aggregated hybrid components.

We therefore present a concept for the behavioral embedding of the subcomponents within the hybrid reconfiguration charts of a component (Section 4.1), which permits to check consistency w.r.t. reconfiguration at a purely syntactical level (Section 4.2).

### 4.1 Behavioral Embedding

The behavioral embedding of subcomponents is achieved 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. In this manner the required coordination of aggregated components can rather easily be described (see Figure 8), similar to composite structure diagrams and structured classes in UML 2.0. 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 with the outlined behavioral embedding of its subcomponents in Figure 8. We have assigned to each location of the upper orthogonal state of the chart the BC component in the appropriate state. 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 lower orthogonal state of Figure 8 shows the communication with the registry which is considered later in Section 4.3.

The upper orthogonal state consists of the states RefAvailable and AllAvailable which represents whether the required reference curve is available for the *actual* track. The upper state is synchronized by the lower one.

The aggregation of the body control, sensor and storage components by the Monitor component as depicted in Figure 5 is also reflected in Figure 8. The semantics of this behavioral embedding is described by the concurrent execution of the controlled components which is formally described as follows.

DEFINITION 12. For hybrid reconfiguration automata  $M_S$  and  $M_1, \dots, M_n$  the hierarchical parallel composition  $(M_S \parallel_H (M_1 \parallel \dots \parallel M_n))$  is defined by a restriction  $H \subseteq L_S \times (L_1 \times \dots \times L_n)$  on the hybrid automaton  $M = M_S \parallel M_1 \parallel \dots \parallel M_n$ . The restriction holds iff for all  $\vec{l} \in L$  reachable in  $\llbracket M \rrbracket$  holds that  $\vec{l} \in H$ .

This formalization assumes that the reconfiguration of the in- and output events for the different states of the different  $M_i$  described by the behavioral embedding is realized by the coordinating hybrid automaton  $M_S$ .  $M_S$  realizes the specific topology of each state by providing the related signal connections in form of a continuous model which only copies the variables accordingly. In addition, the hybrid automaton  $M_S$  has to trigger the implicitly specified transitions of the subcomponents by emitting the events specified in the interface state charts.

If we abstract from the events exchanged between the automata we can thus assume that  $M_S \parallel_H (M_1 \parallel \dots \parallel M_n)$  is a refinement of  $M_S$ . To formally abstract from these internal events, we additionally define the hiding of events.

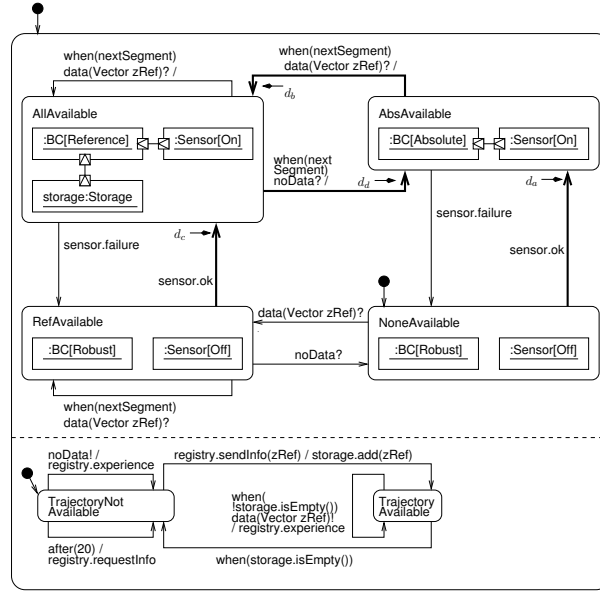


Figure 8: Behavioral embedding in the Monitor realization

DEFINITION 13. For a hybrid automaton  $M = (L, D, I, O, T, S^0)$  the hiding of some signals  $A \subseteq I \cup O$  denoted by  $M \setminus A$  is defined as the hybrid automaton  $M' = (L, D, I', O', T', S^0)$  with  $I' = I - A$ ,  $O' = O - A$ , and  $T' = \{((l, g, g^i - A, u, l) | (l, g, g^i, u, l)) \in T\}$ .

The hybrid reconfiguration chart specified in Figure 8 equals  $M_S \setminus I_1 \cup \dots \cup I_n \cup O_1 \cup \dots \cup O_n$ . The additional details of  $M_S$  describing the coordination with  $M_1, \dots, M_n$  are omitted in Figure 8, as they can be automatically derived.

For any regular location of the corresponding hybrid automaton  $l_1 \in L_1$ , the strict assignment of one state of the contained components to a single location of the hybrid reconfiguration chart (as present in Figure 8) makes sure that  $|H \cap \{l_1\} \times (L_1 \times \dots \times L_n)| = 1$ . Consequently,  $l_1$  can coexist only with the related regular target locations of  $M_1, \dots, M_n$ . For a fading location  $l_1 \in L_1$  we can expect that  $|H \cap \{l_1\} \times (L_1 \times \dots \times L_n)| = 2$  such that  $l_1$  can coexist only with the related regular target locations of  $M_1, \dots, M_n$  or with intermediate fading locations of  $M_1, \dots, M_n$  which eventually lead to this target location. We require that there is no interaction among  $M_1, \dots, M_n$ , so that they do not initiate location changes of parallel components.

## 4.2 Consistency Checking

The complex conditions a correct parallel composition has to fulfill (cf. Definition 4) highlight an important fact: The parallel execution of two components with inconsistent control laws (e.g., if the input and output variables do not fit) can result in undefined behavior.

A single state of a hybrid reconfiguration chart can also result in an incompatible reconfiguration, if the composed dependency relation for the related configuration of subcomponent states contains a cycle. Due to the refinement relation any dependency between an input and output in the component realization  $M_R$  of a component is also present in its interface automaton  $M_I$ . Thus, considering the interface automaton is sufficient to exclude incompatible state configurations.

Additionally, the correct real-time coordination of the fading-durations etc. has to be ensured. By restricting our considerations

here to simple interface state charts where only the fading locations are characterized by a simple duration restriction and the states are not restricted, 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.<sup>4</sup> We have to check for each transition in the hybrid reconfiguration chart and the related state transitions in the interface state chart of the aggregated subcomponents that the behaviors are consistent (cf. Theorem 1).

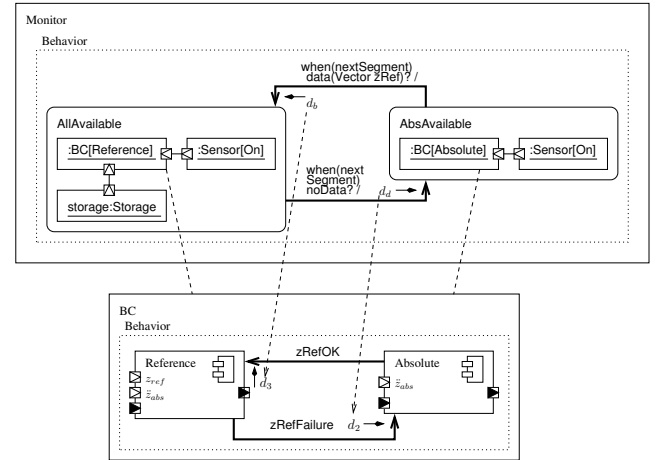


Figure 9: Scheme for the syntactical checking of correct reconfiguration

Figure 9 depicts a part of the monitor behavior and a part of the interface state chart of the embedded BC component (cf. Figures 7 and 8). As described in the previous sections the transition from state AbsAvailable to AllAvailable implies a change of the BC component from state Absolute to Reference. Further the monitor requires

<sup>4</sup>As in the general form of hybrid systems considered here reachability is undecidable [14], we cannot expect to find an automatic solution for the general problem.

this transition to be completed within the timing interval  $d_b$ . As the implied state change of BC will occur within the timing interval  $d_3$ , the overall specification is only correct, if  $d_3 \subseteq d_b$ . Similar,  $d_2 \subseteq d_d$  must hold for the transition to AllAvailable/Reference.

The outlined syntactical rule for the hierarchical parallel composition  $M_1 \parallel_H M_2$  of two hybrid automata  $M_1$  and  $M_2$  ensures that the supervisor cannot be blocked by the supervised automata. The following theorem describes the general syntactical rule which is sufficient to prove for the above sketched restricted case that a hierarchical parallel product does not have any timing errors.

**THEOREM 1.** *For the hierarchical parallel composition  $M_1 \parallel_H M_2$  of two hybrid automata  $M_1$  and  $M_2$  holds  $M_1 \parallel_H M_2 \sqsubseteq M_1 \setminus_{I_2 \cup O_2}$  iff  $I(M_1 \parallel_H M_2) = I(M_1 \setminus_{I_2 \cup O_2})$ , all initial states are also contained in  $H(\{(l_1, l_2) | (l_1, x) \in S_1^0 \wedge (l_2, y) \in S_2^0\} \subseteq H)$ ,  $M_2$  is a simple interface state chart (cf. Definition 10), and for all  $(l_1, l_2) \in H$  and transition  $t_1 = (l_1, g_1, g_1', a_1, l_1') \in T_1$  holds:*

- if  $l_1'$  is not a fading location, then for all  $t_2 = (l_2, g_2, g_2', a_2, l_2') \in T_2$  with  $g_1' \cap (I_2 \cup O_2) = g_2'$  must hold:  $g_2 = \text{true}$ ,  $l_2'$  is a passive location, and  $(l_1', l_2') \in H$ . In addition at least one such transition in  $M_2$  must exist.
- if  $l_1'$  is a fading location we can conclude that exactly one transition  $t_1' = (l_1', g_1', g_1'', a_1', l_1'') \in T_1$  with  $g_1' \equiv d_{min}^1 \leq v \leq d_{max}^1$  exists. For any  $t_2 = (l_2, g_2, g_2', a_2, l_2') \in T_2$  with  $g_1' \cap (I_2 \cup O_2) = g_2'$  must hold:  $g_2 = \text{true}$ ,  $l_2'$  is a fading location, and  $(l_1', l_2') \in H$ . For the uniquely determined successor transition  $t_2' = (l_2', g_2', g_2'', a_2', l_2'') \in T_2$  with  $g_2' \equiv d_{min}^2 \leq v \leq d_{max}^2$  must hold:  $l_2''$  is a passive location,  $(l_1'', l_2'') \in H$ , and  $[d_{min}^2, d_{max}^2] \subseteq [d_{min}^1, d_{max}^1]$  must be satisfied. Again, at least one such pair of transition in  $M_2$  must exist.

PROOF. see [6].  $\square$

Theorem 1 can be extended to the general case of  $M_S \parallel_H (M_1 \parallel \dots \parallel M_n)$  by induction. Due to the syntactical check of Theorems 1, the hierarchical composition by means of the underlying hybrid control software cannot invalidate the timing properties ensured by the embedding hybrid reconfiguration chart of the monitor.

### 4.3 Pattern-based Verification

If the embedding hybrid reconfiguration chart does only contain timing constraints but no general hybrid constraints, we can exploit the result of Theorem 1 to model check the real-time coordination of the overall system.

The syntactical check of the hierarchical composition ensures that the underlying subcomponents cannot invalidate the timing properties ensured by the behavior of the embedding component.

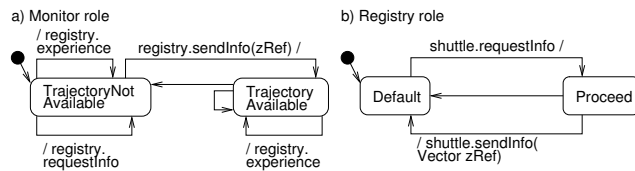


Figure 10: Roles of the Monitor-Registration-pattern

The real-time communication between the registry and the monitor is described by the ShuttleRegistration pattern as depicted in Figure 5. The behavior of the track section's registry which is

frequently contacted by the monitor to obtain the required reference data ( $z_{ref}$ ) is depicted in Figure 10b. The related Monitor role which has to be realized by the shuttle software is depicted in Figure 10a. Besides the lookup of reference data by the monitor, the monitor can also send its gained experience to the registry. Staying in the location TrajectoryNotAvailable the monitor sends requestInfo-requests to the registry. If the registry receives such a request from a shuttle, it may either answer by sending sendInfo back to the shuttle or it may not answer. When the shuttle receives such an answer, it stores it internally and switches to the location TrajectoryAvailable. This role protocol has been refined in the lower orthogonal state of Figure 8.

To verify the correct real-time coordination between the monitor and the registry the refinement of the role protocols outlined above is required. Further model checking of the pattern has been shown to be sufficient to verify the required local safety and liveness properties (cf. [9]).

## 5. CONCLUSION

Today, advanced mechatronic systems are hybrid systems which reconfigure themselves online. As outlined in the paper, the proposed extension of UML components and statecharts supports the required modular hierarchical modeling of reconfigurable systems.

Discrete coordination is designed by a software engineer with timed statecharts, whereas a control engineer may construct the advanced controller component which offers the technical feasible reconfiguration steps, in parallel. Tool support which integrates the models and the automatic code generators of these two parts of the system is under development. The UML CASE tool Fujaba<sup>5</sup> and the CAE tool CAMEL [25] are currently being integrated [7]. These tools will be used to evaluate the presented approach.

A serious problem which comes with the ability of the system to reconfigure itself online is the possibility of an inconsistent reconfiguration. By our notion of components and the exploitation of a domain specific restriction, namely a strictly hierarchical system structure, at least up to the level of components communicating via ports, we only need to check the consistency of a reconfiguration by checking the hybrid reconfiguration chart of a component and the interface state charts of its directly aggregated subcomponents. Finally, a compatibility check at the level of components whose communication is specified by ports and roles, serves to exclude temporal inconsistencies in the discrete specification part of the whole system. Here, compositional model checking can be applied.

## 6. REFERENCES

- [1] A. Agrawal, G. Simon, and G. 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*.
- [2] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(3-34), 1995.
- [3] 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*.
- [4] K. Bender, M. Broy, I. Peter, A. Pretschner, and T. Stauner. Model based development of hybrid systems. In *Modelling*,

<sup>5</sup>www.fujaba.de

*Analysis, and Design of Hybrid Systems*, volume 279 of *Lecture Notes on Control and Information Sciences*, pages 37–52. Springer Verlag, July 2002.

- [5] D. Bradley, D. Seward, D. Dawson, and S. Burge. *Mechatronics*. Stanley Thornes, 2000.
- [6] S. Burmester, H. Giese, and O. Oberschelp. Hybrid UML Components for the Correct Design of Complex Self-optimizing Mechatronic Systems. Technical Report tr-ri-03-246, University of Paderborn, Germany, 2004.
- [7] S. Burmester, H. Giese, and O. 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.
- [8] H. Giese and S. Burmester. Real-Time Statechart Semantics. Technical Report tr-ri-03-239, University of Paderborn, Paderborn, Germany, June 2003.
- [9] H. Giese, M. Tichy, S. Burmester, W. Schäfer, and S. Flake. Towards the Compositional Verification of Real-Time UML Designs. In *Proc. of the European Software Engineering Conference (ESEC)*, Helsinki, Finland. ACM Press, September 2003.
- [10] 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.
- [11] D. Harel. STATECHARTS: A Visual Formalism for complex systems. *Science of Computer Programming*, 3(8):231–274, 1987.
- [12] T. A. Henzinger. Masaccio: A Formal Model for Embedded Components. In *Proceedings of the First IFIP International Conference on Theoretical Computer Science (TCS)*, LNCS 1872, Springer-Verlag, 2000, pp. 549–563., 2000.
- [13] T. 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.
- [14] T. A. Henzinger, P. W. Kopke, A. Puri, and P. 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.
- [15] T. A. Henzinger, M. Minea, and V. 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, LNCS 2034, pages 275–290. Springer Verlag, 2001.
- [16] T. Hestermeyer, P. Schlautmann, and C. Eттingshausen. Active suspension system for railway vehicles-system design and kinematics. In *Proc. of the 2nd IFAC - Conference on mechatronic systems*, Berkeley, California, USA, 9–11 December 2002.
- [17] 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.
- [18] L. Lamport. Hybrid Systems in TLA+. Springer-Verlag, 1993.
- [19] G. Lüttgen, M. von der Beeck, and R. Cleaveland. A compositional approach to statecharts semantics. In *Proceedings of the eighth international symposium on Foundations of software engineering for twenty-first century applications November 6 - 10, 2000, San Diego, CA USA*, pages 120–129, 2000.
- [20] N. Lynch, R. Segala, and F. 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, LNCS 2034, pages 403–417. Springer Verlag, 2001.
- [21] E. Münch, O. Oberschelp, T. Hestermeyer, P. Scheideler, and A. Schmidt. Distributed Optimization of Reference Trajectories for Active Suspension with Multi-Agent Systems. In *18th European Simulation Multiconference (ESM)*, Magdeburg, Germany, 2004.
- [22] Object Management Group. *UML for System Engineering Request for Proposal*, ad/03-03-41, March 2003.
- [23] Object Management Group. UML Superstructure Submission V2.0. OMG Document ad/03-04-01, April 2003. URL: <http://www.omg.org/cgi-bin/doc?ad/2003-04-01>.
- [24] K. Ogata. *Modern Control Engineering*. Prentice Hall, 2002.
- [25] J. Richert. Integration of Mechatronic Design Tools with CAMEL, Exemplified by Vehicle Convoy Control Design. In *Proc. of the IEEE International Symposium on Computer Aided Control System Design*, Dearborn, Michigan, USA, 1996.
- [26] T. Stauner. *Systematic Development of Hybrid Systems*. PhD thesis, Technical University Munich, 2001.
- [27] 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.
- [28] R. Wieting. Hybrid high-level nets. In *Proceedings of the 1996 Winter Simulation Conference*, pages 848–855, Coronado, CA, USA, 1996.

## APPENDIX

### A. ADDITIONAL FORMAL DEFINITIONS

In the appendix we complete the omitted formal prerequisites. We use  $\mathbb{R}$  to denote the set of the real numbers,  $\mathbb{N}_0$  to denote the natural numbers including 0,  $[a, b]$  with  $a, b \in A$  and  $a \leq b$  to denote the interval of all elements  $c \in A$  with  $a \leq c \leq b$ ,  $\wp(A)$  to denote the power set of  $A$ , and  $[A \rightarrow B]$  and  $[A \rightarrow B]$  to denote the set of total resp. partial functions from  $A$  to  $B$ .  $EQ(V_l, V_r)$  denotes the set of all equations of the form  $v_l = f^i(v_r^1, \dots, v_r^n)$  with operations  $f^i$  of arity  $n$  and left- and right-hand side variables of the equation  $v_l \in V_l, v_r^1, \dots, v_r^n \in V_r$ .  $COND(V)$  denotes the set of all conditions over variables of  $V$ . The set of possible operations and constants is named  $OP$ .

As a special case we assume a set of operations  $\{\perp_i\}$  which do not explicitly define for an equation  $v_l = \perp_i(v_r^1, \dots, v_r^n)$  any specific restrictions on the relation between the input and output trajectories. The set of all these operations is denoted by  $OP_\perp$ . The set of only fully deterministic input/output operations are denoted by  $OP_{det}$ .

Other than the vector equations usually employed by control engineers, we employ a set of variables  $V$  to denote each single value and describe the mapping by a function  $[V \rightarrow \mathbb{R}]$ . All values of a vector of the length  $n$  can be represented in a similar fashion as  $[[0, n] \rightarrow \mathbb{R}]$ .

$f \otimes g$  further denotes the composition of the two functions  $f : A_1 \rightarrow B_1$  and  $g : A_2 \rightarrow B_2$  with disjoint definition sets  $A_1 \cap A_2 = \emptyset$  defined by  $(f \otimes g)(x)$  equals  $f(x)$  for  $x \in A_1$  and  $g(x)$  for  $x \in A_2$ . The combination of two updates  $a_1 \oplus a_2$  further denotes the composition of the two functionals  $a_1 : [A_1 \rightarrow B_1] \rightarrow [A'_1 \rightarrow B'_1]$  and  $a_2 : [A_2 \rightarrow B_2] \rightarrow [A'_2 \rightarrow B'_2]$  with disjoint sets  $A_1 \cap A_2 = \emptyset$  and  $A'_1 \cap A'_2 = \emptyset$  defined by  $(a_1 \oplus a_2)(x \otimes y) := a_1(x) \otimes a_2(y)$ .