Hybrid UML Components for the Correct Design of Self-optimizing Mechatronic Systems*

Sven Burmester† and Holger Giese
Software Engineering Group
University of Paderborn
Warburger Str. 100
D-33098 Paderborn, Germany
[burmi|hg]@upb.de

Oliver Oberschelp
Mechatronic Laboratory Paderborn
University of Paderborn
Pohlweg 98
D-33098 Paderborn, Germany
Oliver.Oberschelp@mlap.de

January, 2004

*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 International Graduate School of Dynamic Intelligent Systems.

Abstract

Complex technical systems, such as networked mechatronic systems, can share information in the network and exploit the computational power available today to achieve an automatic improvement of the technical system performance at run-time by means of self-optimization. To realize this vision appropriate means for the design of such systems are required. To locally support self-optimization it is not enough just to permit to alter some free parameters of the controllers. Instead, support for the correct reconfiguration of the internal structures of the controllers is required. To design correct complex networked mechatronic systems, an approach for the design and verification of the coordination between the controller structure reconfiguration and the real-time network processing is necessary. We therefore propose hybrid components and a related hybrid statechart extension for the Unified Modeling Language (UML); it is to support the design of self-optimizing mechatronic systems by allowing specification of the necessary flexible reconfiguration of the system as well as of its hybrid subsystems in a modular manner. The interface abstraction of these hybrid components further supports to check for the correct reconfiguration and enables the compositional model checking of the real-time aspects of the system.

Contents

1 Introduction

Mechatronic systems [1] are complex technical systems whose behavior is actively controlled with the help of computer technology. The design of these systems is marked by a combination of technologies used in mechanical and electrical engineering as well as computer science. The focus of the development is always on the technical system whose motion behavior is controlled by software.

The increasing efficiency of microelectronics, particularly in embedded systems, allows the development of mechatronic systems that besides the required control use computational resources to improve their long term performance. In addition, connecting the different entities of complex mechatronic systems via networks enables them to share their experience which is another opportunity for them to improve their performance. These forms of self-optimization allow an automatic improvement of a technical system during operation which increases the operating efficiency of the system and reduces the operating costs.

A generally accepted definition of the term self-optimization is difficult to find. In our view, the core function of self-optimization in technical systems is generally an automatic improvement of the technical system at run-time with respect to defined target criteria. Here the difference to adaptation becomes clear. Adaptation is only an adjustment to varying system- and environmental conditions. Therefore, self-optimization explores design alternatives at run-time while adaptation reacts optimally only within the limits of a given design. Thus, in a self-optimizing design, development decisions are being shifted from the design phase to the system run-time.

There are two opportunities of optimization during runtime. The first is to optimize parameters [2] the second is to optimize the structure. However, alteration of the free parameters of the system will not lead very far because many characteristics, in particular those of the controller, can be altered only in the internal structures and not just by a modification of parameters [3, 4].

At first, it seems easy to apply well-known optimization methods to the controller of the technical system at run-time in order to improve e.g. controller parameters in a real environment online [2]. However, alteration of the free parameters of the system will not lead very far because many characteristics, in particular those of the controller, can be altered only in the internal structures and not just by a modification of parameters [3].

The idea of developing mechatronic systems which can optimize their structure automatically at run-time is far more ambitious. For the design and subsequent realization of self-optimized systems, it is necessary that the methods employed allow the continuous behavior to be adjusted during run-

time on a number of different levels, such as parameters, blocks, and the whole topology of the system. Changes in the topology and complex coordination, which are most appropriately modeled in a discrete manner, have also to be supported. Thus, the methods employed for the development of mechatronic systems with self-optimization have to support hybrid systems as well as flexible reconfiguration. The flexible reconfiguration of a specification can result in an overwhelming additional complexity of the system. Therefore means to ensure that the reconfiguration of the specified system is always done in a consistent manner are required.

While most approaches to hybrid modeling [5, 6, 7] describe how the continuous behavior can be modified when the discrete control state of the system is altered, we need an approach that allows the continuous behavior as well as its topology to be altered whenever this is required for the self-optimization of a system. A suitable concept of mechatronic components should thus permit to alter not only the discrete state, but also its internal topology and its continuous interface with the environment. Therefore, to encapsulate the operation modes within a single component (cf. *information hiding* [8]), a component must have multiple discrete modes and each mode must be able to exhibit a different external continuous interface. Additionally, a suitable approach has to provide appropriate means to verify that the designed systems contains no timing inconsistencies and only performs correct reconfiguration steps.

Our suggestion is to integrate methods used in mechanical engineering and software engineering to support the design of mechatronic systems with self-optimization. We therefore combine component diagrams and state machines as proposed in the forthcoming UML 2.0 [9] with block diagrams [3] usually employed by control engineers. The proposed component-based approach thus allows a decoupling of the domains: A control engineer can develop the continuous controllers as well as their reconfiguration and switching in form of hybrid components. A software engineer on the other hand can integrate these components in his design of the required reliable and correct real-time coordination. The resulting hybrid components and hybrid statecharts permit the required modular hierarchical design of hybrid systems. Additionally, concepts for the verification of the real-time behavior and correct reconfiguration are outlined.

In Section 2 we will examine related work. Section 3 will deal with our application example in detail, including a description of the continuous, the discrete, and the hybrid model. In Section 4, our approach to hybrid modeling with an extension of UML components will be expounded, an approach to ensure correct reconfiguration is outlined, and the integration into a compositional model checking scenario of the overall system is sketched. Section

5 then formalizes the employed concepts and proofs the main results which have been used to assure correct reconfiguration. Thereafter we sum up with a final conclusion and an outlook on future work.

2 Related Work

A couple of modeling languages have been proposed to support the design of hybrid systems [10, 11, 12]. Most of these approaches provide models, like linear hybrid automata [10], that enable the use of efficient formal analysis methods, but lack of methods for structured, modular design, that is indispensable in a practical application [13].

To overcome this limitation, hybrid automata have been extended to hybrid Statecharts in [14]. Hybrid Statecharts reduce the visual complexity of a hybrid automaton through the use of high-level constructs like hierarchy and parallelism, but for more complex systems further concepts for modularization are required.

The hybrid extensions HyROOM [15], HyCharts [16, 17] and Hybrid Sequence Charts [18] of ROOM/UML-RT [19, 20] integrate domain specific modeling techniques. The software's architecture is specified similar to ROOM/UML-RT and the behavior is specified by statecharts whose states are associated with systems of ordinary differential equations and differential constraints (HyCharts) or Matlab/Simulink block diagrams (HyROOM). HyROOM models can be mapped to HyCharts [15]. Through adding tolerances to the continuous behavior this interesting specification technique enables automatic implementation, but support for the modular reconfiguration is not given.

In [21] guiding principles for the design of hybrid systems are sketched. It describes how to apply techniques that are used in automotive engineering, like the combination of statecharts, blockdiagrams and commercial tools. Following this approach hybrid systems need to be decoupled into discrete and continuous systems in the early design phases. Therefore a seamless support and a common model are not provided.

Within the Fresco project the description language Masaccio [22] which permits hierarchical, parallelized, and serially composed discrete and continuous components has been developed. A Masaccio model can be mapped to a Giotto [23] model, that contains sufficient information about tasks, frequencies, etc. to provide an implementation. The project provides a seamless support for modeling, verification and implementation, but our needs for advanced modeling techniques that support dynamic reconfiguration are not addressed.

CHARON [7] is a statechart-like modeling language for hybrid systems and describes the continuous behavior by first-order differential equations. It is possible to model complex systems by means of parallelism and hierarchy. The generation of Java code is actually restricted to simulation purposes and no modular support for reconfiguration for the continuous part is given.

[24] deals with the verification of computer controlled continuous systems. They model the plant by continuous differential equations and the controller by a flat discrete automaton whose modes are associated with (discrete) quasi-continuous difference equations. Model checking is performed by transforming the hybrid model into a finite state machine (FSM) that is verified by the model checker CheckMate. As the model of the controller is flat there is no support for modularization. Due to the lack of high-level constructs the approach does not fit for complex systems.

The OMG published a RFP (Request for Proposal) for *UML for Systems Engineering (UML for SE)* [25]. The idea of UML for SE is to provide a language that supports the system engineer in modeling and analysing software, hardware, logical and physical subsystems, data, personnel, procedures, and facilities. One distinguishing proposal is called *Systems Modelling Language (SysML)* [26] that takes a subset of the UML 2.0 proposal and extends it. Main extensions, related to the design of continuous and hybrid systems are *Structured Classes*, that describe a class' component structure extended by continuous communication links between ports. In *Parametric Diagrams* the *parametric* (arithmetic) *relations* between numerical attributes of instances are specified and the nodes of Activity Diagrams are extended with continuous functions and in- and outputs. The RFP shows that there is need for a UML extension supporting system engineering and including continuous behavior. The SysML proposal provides this support, but for the specification of parametric relations a static structure is assumed. There is no support for modeling structural changes or replacements of the parametric relations.

3 Application Example

We will use the switching between three controller structures as a running example. The concrete example is an active vehicle suspension system with its controller which stems from the *Neue Bahntechnik Paderborn*[1] research project. The project has been initiated and worked upon by several departments of the University of Paderborn and the Heinz Nixdorf Institute. In the project, a modular rail system will be developed; it is to combine mod-

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

ern chassis technology with the advantages of the Transrapid[2] and the use of existing rail tracks. The interaction between information technology and sensor/actuator technology paves the way for an entirely new type of mechatronic rail system. The vehicles designed apply the linear drive technology used in the Transrapid, but travel on existing rail tracks. The use of existing rail tracks will eliminate an essential barrier to the proliferation of new railbound transport systems [27].

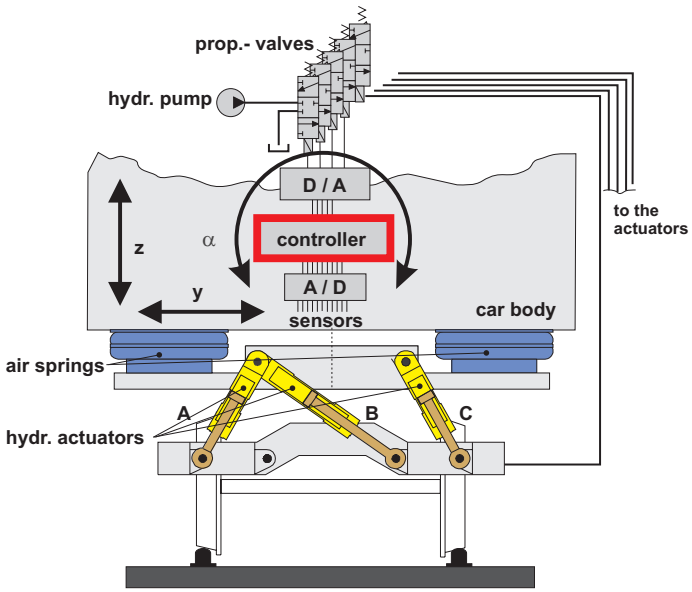

Figure 1: Scheme of the suspension/tilt module

Figure 1 shows a schema of the physical model of the active vehicle suspension system. The suspension system of railway vehicles is based on air springs which can be damped actively by a displacement of their bases. It completely dispenses with passive dampers in parallel to the springs. The result is an excellent decoupling of the coach body from high-frequency excitations of drive or rails. The active control of the spring-based displacement can be restricted to relatively low frequencies around the natural frequency of the car body. All in all, this system can noticeably improve ride comfort especially at higher frequencies.

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

[2]http://www.transrapid.de/en

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 [28]. 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 overall controller structure comprises different feedback controllers, e.g., for controlling the hydraulic cylinder position and the dynamics of the car body.

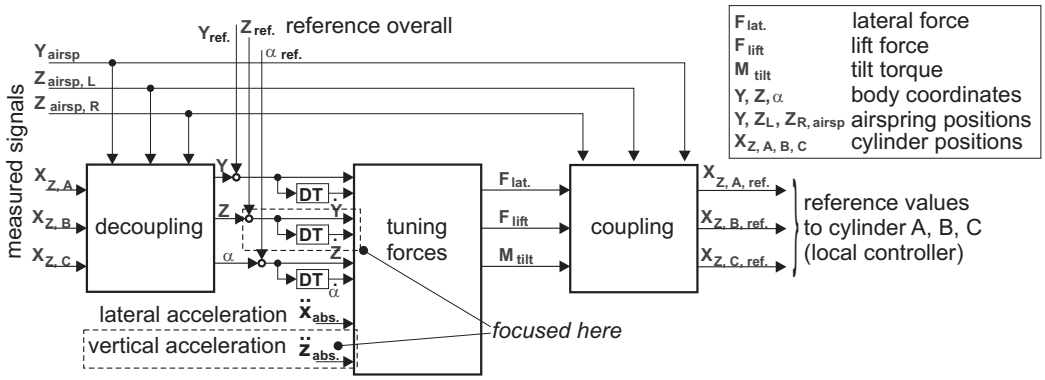

Figure 2: Body control controller structure (reference mode)

The schema of the overall body controller is depicted in Figure 2. The subordinated controller is not displayed here for reasons of clarity. The controller essentially consists of the blocks decoupling, tuning forces, and coupling. In the block decoupling the kinematics of the cylinders is converted into Cartesian coordinates for the description of the coach-body motion. The actual control algorithm is in the block tuning forces. Here the forces are computed which are to affect the mechanical structure. In the block coupling the forces and torques are converted again into reference values for the subordinated cylinder controller [29]. The behavior of the system model and the controllers is described by nonlinear differential equations. The employed CAE tool CAMeL [30] allows a description of the continuous part of the model by strict hierarchical block diagrams and basic blocks with nonlinear differential vectorial state equations in the shape $\dot{\underline{x}} = \underline{f}(\underline{x}, \underline{u}, t)$ and $\underline{y} = \underline{g}(\underline{x}, \underline{u}, t)$, where $\underline{x}$ is the state vector, $\dot{\underline{x}}$ is the first derivation of the state vector, $\underline{y}$ the output vector, $\underline{u}$ the input vector, and t the time.

3.1 Switching and Cross Fading

The controller has two normal modes: Reference and absolute. The controller reference uses a given trajectory that describes the motion of the coach body

$z_{ref} = f(x)$ and the absolute velocity of the coach body $\dot{z}_{abs}$ (derived from $\ddot{z}_{abs}$). The z_{ref} trajectory is given for each single track section. The track section registry communicates this reference trajectory to the vehicle. In case a reference trajectory is not available, another controller which requires only the absolute velocity of the coach body $\dot{z}_{abs}$ for the damping of the coach-body motion has to be used.

Besides the regular modes another controller named robust is required for an emergency; it must be able to operate even if neither the reference trajectory nor the measurement of the coach-body acceleration are available (see Figure 3).

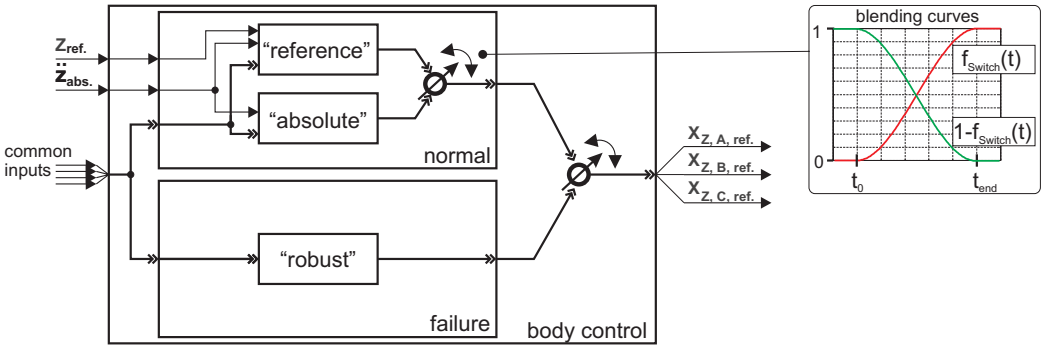

Figure 3: Fading between different continuous control modes

For a 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. To start the new controller up, it is often necessary to initialize its states on the basis of the states of the old controller. In our example, the switching from the normal block to the failure block (see Figure 3) can be processed atomically because the robust controller actually has no state. Different theoretical works deal with the verification of stability in systems with any desired switching processes [31]. In the simple case of a switch to a robust controller, stability can be maintained with the help of condition monitoring [32].

If, however, 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 3, 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.

In an appropriate self-optimizing design the aim must be to use the most

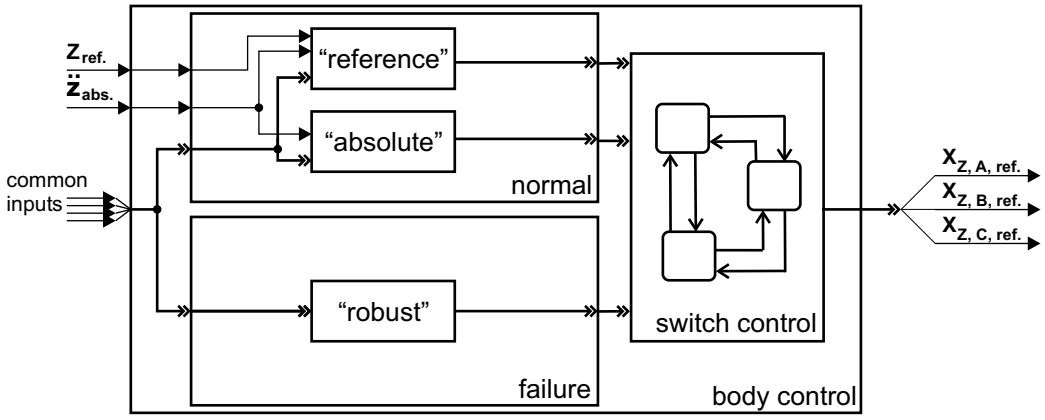

Figure 4: Controlling the fading between different continuous control modes with a statechart

comfortable controller while still maintaining the shuttle's safety. If, for instance, the absolute controller is in use and the related sensor fails, the controller may become instable and cause an accident. Thus there is need for a discrete control monitor which ensures that only safe configurations are allowed. The system outlined above can optimize the system behavior when additional information is available or compensate for a lack of sensor data. In Figure 4, this standard approach for the integration of discrete control elements into block diagrams is depicted. The alternative controller outputs are feed into a discrete block which behavior is described by a statechart. The defacto industry standard employing this concept is MATLAB/Simulink and Stateflow.[3]

3.2 Hybrid Modeling

When the control engineer has to model the fading between the different controllers according to Figure 3, it is reasonable for him to use a hybrid automaton [5] with at least three discrete states for each of the controllers. These are the locations Robust, Absolute and Reference (see Figure 5). The locations' continuous behavior represents the employed controllers which are specified by blocks, that are usually employed in control engineering. The blocks are hierarchical, usually contain complex controller structures, and require different input signals. Black-filled arrows denote the common inputs which are always available and required by all controllers, white-filled arrows denote special inputs that are not always available. The start location

[3] www.mathworks.com

is associated with the robust controller. When for instance the $\ddot{z}_{abs}$ signal becomes available (as indicated by the zAbsOK signal), the location and the controller can be changed to the absolute mode.

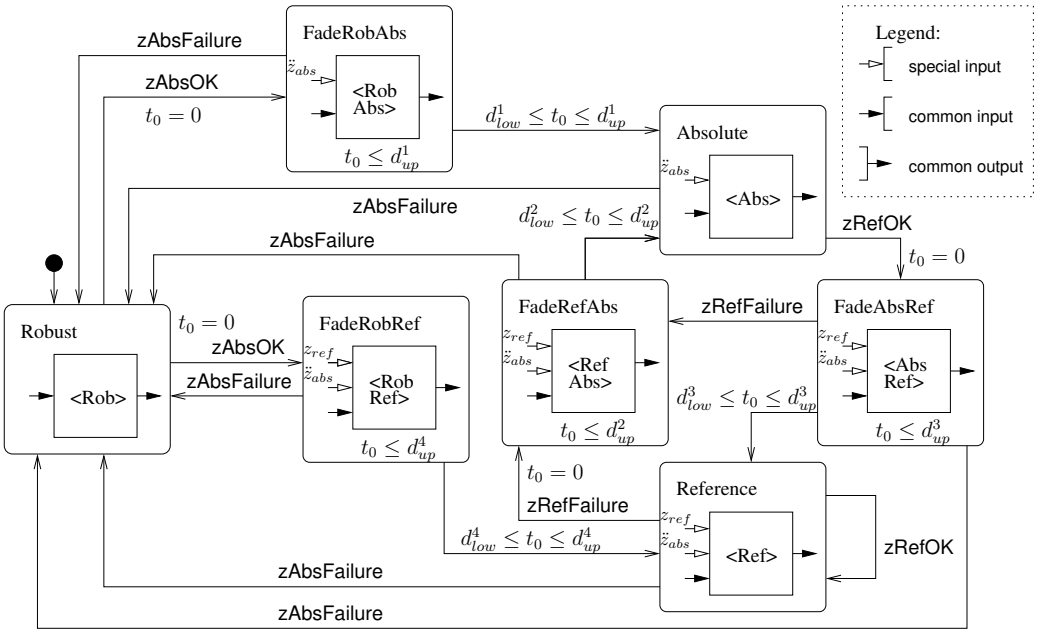

Figure 5: Hybrid view of the body control with additional fading locations

As outlined in Section 3.1, an atomic switch between these controllers can usually not guarantee stability. Thus the Absolute location cannot be entered directly and an additional location FadeRobAbs is entered (see Figure 5). This intermediate location comprises the cross fading activity from the robust to the absolute controller. In order to set the fading's duration to $d_1 = [d^1_{low}, d^1_{up}]$, the state variable (clock) t_0 is set to 0 when entering the FadeRobAbs-location. The invariant $t_0 \leq d^1_{up}$ and the guard $d^1_{low} \leq t_0 \leq d^1_{up}$ ensure the demanded duration. Note that inside the fading locations $\dot{t}_0 = 1$ 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 (cf. Figure 5).

3.3 Discrete Coordination

As shown in the Section 3.1, the different controllers require different input signals. Some of these signals are read by sensors of the mechatronic system

(e.g., $\ddot{z}_{abs}$); others are transmitted by the wireless network and saved in an intermediate storage (z_{ref}). When z_{ref} is needed it is provided by the storage that prevents immediate signal break-offs in case of a network failure. Other signals may not be available for some time in the case of sensor failures. In this case, the real-time control software has to replace the currently active controller by one whose required inputs are available. In order to design the required coordination in our approach, we use UML component diagrams and our real-time extension of UML Statecharts [33].

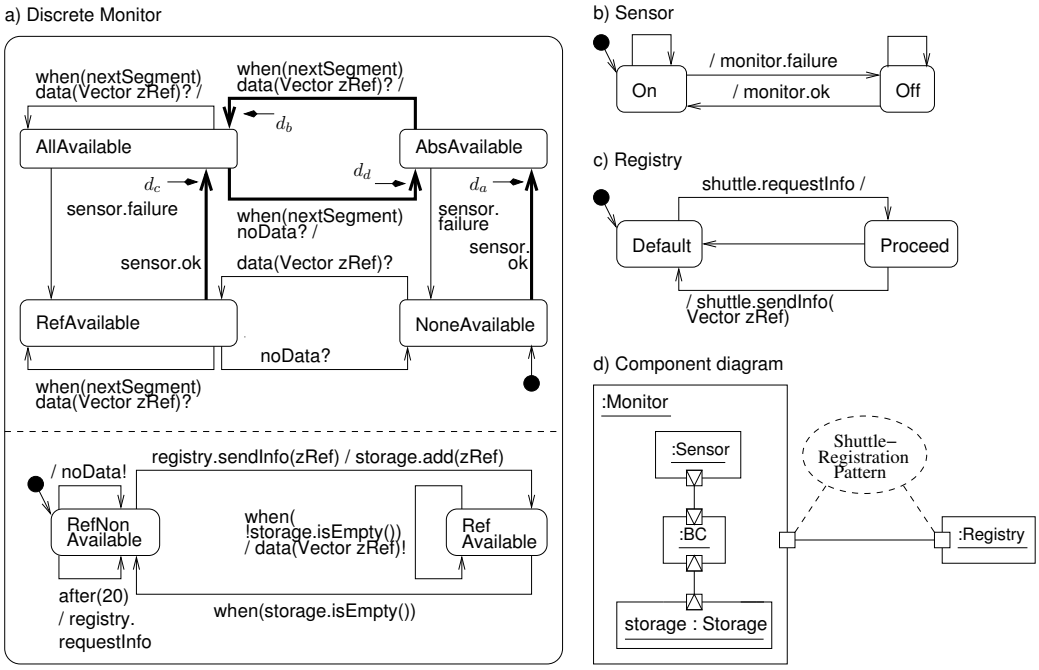

Figure 6: Behavior and structure of the monitor

Figure 6a specifies the behavior of the control monitor software. The upper orthogonal state consists of four locations indicating which of the two signals $\ddot{z}_{abs}$ and z_{ref} are available. Some transitions can fire immediately, others are associated with a deadline interval $d = [d_{low}, d_{up}]$ specifying how much time a transition may take minimal and maximal. These transitions are thicker and marked with an arrow and the intervals (d_a, ..., d_d). The lower branch communicates with the track section registry (Figure 6c) and frequently requests the z_{ref} function. Figure 6b depicts the sensor's behavior, checking if the sensor works and raising events about the availability of the $\ddot{z}_{abs}$-signal. In Figure 6d the overall structural view is presented by means of a UML component diagram. The Monitor component, that embedds three components, communicates with the Registry component through ports

and a communication channel. The Shuttle-Registration pattern specifies the communication protocol between Monitor and Registry [34].

The embedded components communicate continuous signals through so called *continuous ports*, depicted by triangles whose orientation indicates the direction of the signal flow.[4] E.g. the continuous signal $\ddot{z}_{abs}$ is transmitted from the Sensor component to the BC component through their continuous ports.

Staying in the location RefNonAvailable 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 in the storage component (see Figure 6d) and switches to the location RefAvailable. It sends the signal to the upper branch through the synchronized event data! with the required signal as parameter, when the shuttle approaches the next segment. Following the internally stored data is consumed until the storage is empty. The statecharts from Figure 6a and Figure 6c thus describe the possible different modes of the monitor software which result from the set of available sensor inputs on the one hand and the communication protocol between shuttle and registry on the other hand.

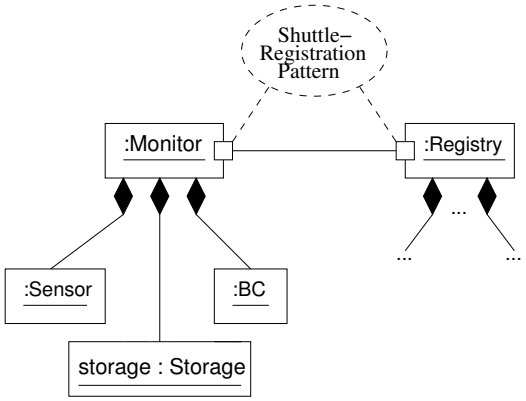

Figure 7: System Hierarchy

Figure 7 the strict hierarchic composition of the monitor component consisting of the three other components. The Registry component may be structered similar. As the interaction between monitor and registry is defined by the Shuttel-Registration pattern and both components have to follow the pattern, the pattern serves as supordinated unit which coordinates the associated components.

[4]Ports in the sense of UML 2.0 are called *discrete ports*.

It is important to note that we always have such a strict hierarchical structure within one subsystem which defines the continuous operating controller for the related sensors and actuators, while at the higher abstraction layers dynamically established structures (modeled using patterns) between the components (shuttles) are used to describe their interaction. In more advanced scenarios, the separated controller present in each shuttle will become connected by establishing their hierarchical combination by means of a supervising pattern to realize the required stable continuous control of shuttle convoys.

4 The Approach

To support the design of complex networked mechatronic systems as outlined in the preceding section, we introduce in the following our notions for hybrid Statecharts in Section 4.1, hybrid Components in Section 4.2, and then their modular composition in Section 4.3. As described in Section 3.3, a strict hierarchy within one shuttle is used to build the hybrid behavior of the controller. We will employ this restriction to simplify the checking of possible reconfigurations for correctness in Section 4.4. The connections between different shuttles via network can also result in non hierarchical component structures. However, at this level of non hierarchical structures no continuous interaction is present and thus we only have to check the real-time processing. Therefore, the domain specific compositional model checking approach outlined in [34] can be employed as outlined in Section 4.5.

4.1 Hybrid Statecharts

Figure 5 presents the example of a hybrid automaton. It contains the locations Robust, Absolute, and Reference, representing the three different controllers, and the locations FadeRobAbs, FadeRobRef, FadeAbsRef, and FadeRefAbs, regulating fading between the controllers. A look at Figure 5 reveals that the explicit fading locations considerably increase the number of visible locations of the automaton and make it unnecessarily difficult to understand.

Thus the proposed extension of UML Statecharts towards hybrid Statecharts provides a short-form to describe the fading. The short-form contains the following parameters: 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. This short-form is displayed in Figure 8. The *fading-transitions* are

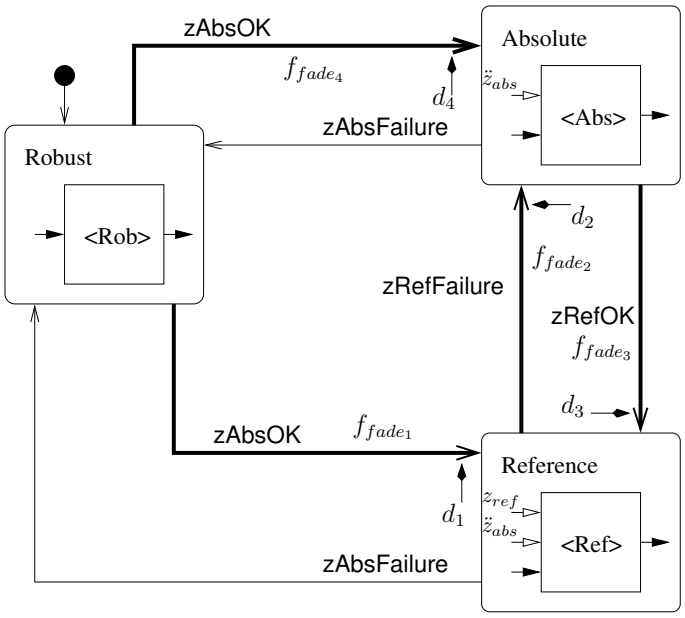

Figure 8: Behavior of the body control component

visualized by thick arrows while atomic switches have the shape of regular arrows. The fading duration interval for atomic transitions is defined as $[0,\ 0]$.

An atomic transition, leaving the source or the target state of a currently active fading transition, interrupts the execution of the fading transition. In case of conflicting atomic transitions of the source and target state, the source state transitions have by default priority.

We employ time-annotated statecharts to describe required real-time behavior and refer the continuous behavior only by embedding appropriate basic quasi-continuous blocks. 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 [34] for the timed case only. Note that due to our simplified hybrid statechart semantics most of the problems w.r.t. compositionallity described in the literature can be avoided (cf. [35]).

4.2 Hybrid Components

One serious limitation of today's approaches for hybrid systems is due to the fact that the continuous part of each location has to have the same

set of required input and provided output variables (continuous interface). To foster the reconfiguration we propose to describe the different externally relevant continuous interfaces as well as the transition between them using hybrid interface Statecharts.

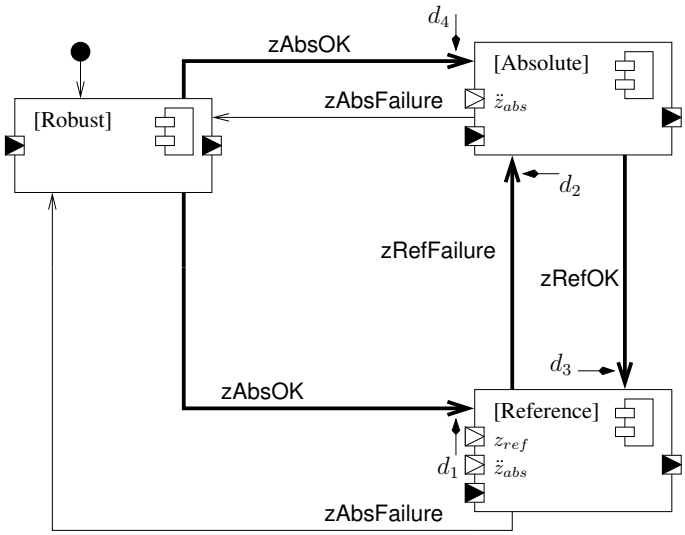

Figure 9: Interface Statechart of the body control component

The related interface automaton of the body control component of Figure 8 is displayed in Figure 9. It shows that the body control component has three possible different interfaces. The (continous) 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. 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.

Interface automata can be employed to abstract from realization details of a component. A crucial prerequisite for this idea is a reasonable notion of refinement which guides interface abstraction. A component in our approach can thus be described as a UML component –with ports with distinct quasi-continuous and discrete signals and events– by

- a hybrid interface automata which is a correct abstraction of the component behavior (see later Definition 12) which determines what signals are active in what state,
- the dependency relation between the output and input signals of a component per state of the interface Statechart, and
- the behavior of the component usually described by a single hybrid Statechart and its aggregated subcomponents.

In our example, the BC component is described by its hybrid interface Statechart presented in Figure 9, the additionally required information on which dependencies between output and input variables exist which is not displayed in Figure 9, and its behavior described by the hybrid Statechart of Figure 8 where the required quasi-continuous behavior is specified by controllers described by quasi-continuous blocks.

4.3 Hierarchical Composition

The in Figure 8 presented form of a hybrid statechart with its support for state-dependent continuous interfaces does, still, not include the case that the employed controllers themselves may show hybrid behavior. Instead, complete separation of discrete and continuous behavior like in [7, 6, 5, 36] is still present.

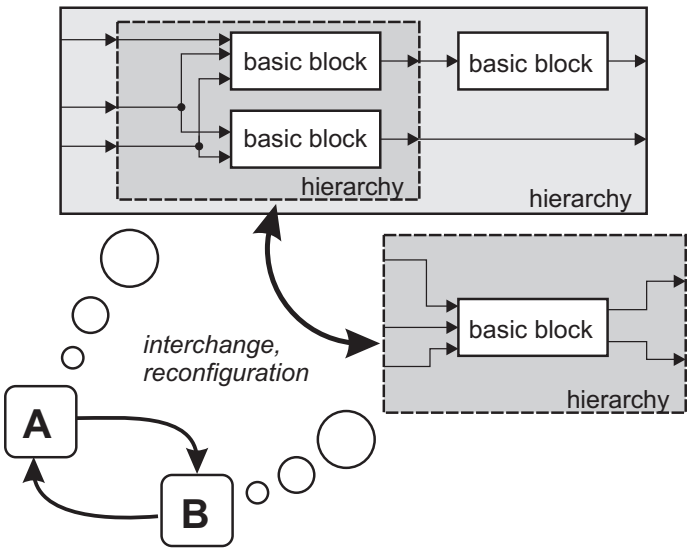

Figure 10: Reconfigurable block diagram

To overcome this limitation, we propose to assign the required configuration of aggregated subcomponents (not only quasi-continuous blocks) to each state of a hybrid Statechart by means of UML instance diagrams. Thus to realize modular reconfiguration we only have to provide a solution to alter the hierarchical elements (cf. Fig. 10). In this manner the required coordination of aggregated components can rather easily be described (see Figure 11), similar to composite structure diagrams and structured classes in UML 2.0.

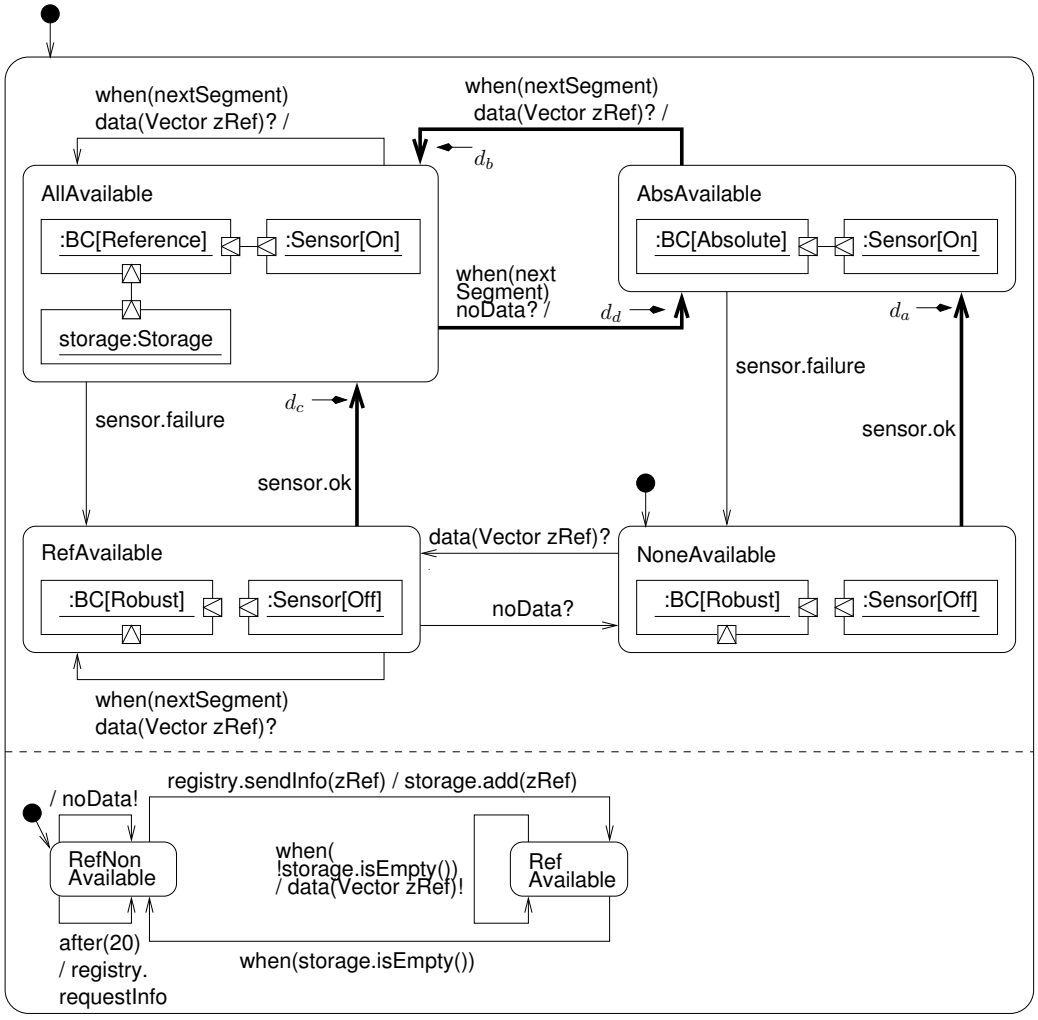

Figure 11: Hierarchical integration of the body control component BC into the monitor

In the structural view described by a UML component diagram, the integration can be reflected by a composition relation between the Monitor component and the BC component as the one presented in Figure 6d. Note that the Monitor component has also a non-hierarchical link to the Registry component which has to be handled by its discrete behavior specified in Figure 6a.

In the behavioral view, every discrete state has to be associated with a configuration of the BC component. In the design of these associations, only the interface description (see Figure 9) is relevant and the inner structure of the BC component can be neglected. Therefore, as shown in Figure 11, we can assign to each location of the upper orthogonal state of the statechart 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 hybrid Statechart of Figure 11 defines a mapping of states to required configurations of the subcomponents. The required synchronization by means of signals between the Statechart and its contained subcomponents is not directly specified. We, however, assume that the externally visible states of each component are controllable (we can via appropriate signals control which state should happen) and therefore can derive the required additional signals the hybrid Statechart has to emit to control its subordinated components.

4.4 Correct Reconfiguration

The presented approach exploits the domain specific restrictions that within one subsystem the continuous operating controller for the related sensor and actuators is always described by a strict hierarchical structure. The required abstraction relation between the interface Statechart and the related component behavior permits us to restrict our considerations here to a compositional scenario, where only the correct coordination between each component and its aggregated subcomponent has to be studied by considering their interface Statecharts (cf. Theorem 1) to ensure the correct reconfiguration of the whole system.

To make sure the specified composition of high-level transitions as, for example, specified by the hybrid Statechart of Figure 11 is indeed correct, for each atomic switch or fading between two states in the hybrid Statechart the corresponding transition of the interface automaton of the embedded subcomponent has to be triggered correctly. Otherwise invalid location combinations cannot be excluded. However, the fading durations (the time the hybrid automaton will stay in corresponding fading locations) must also be compatible. As in the general form of hybrid systems considered here reach-

ability is undecidable [37], we cannot expect to find an automatic solution to this problem. As even for timed automata reachability is only decidable for the restricted case that the clocks are restricted to simple comparisons with rational constants and updates to rational constant values, we have to look for restricted cases that can be supported in an appropriate manner.

Due to the fact that hierarchical composition in contrast to the general parallel composition restricts a potential overlapping of locations, the compatibility can for restricted cases be checked on the syntactical level without consideration of the full state-space of the model. By restricting our considerations here to *simple interface Statecharts* where only the fading locations are characterized by a simple duration restriction and the states are not restricted, we can check that the hybrid Statechart alone is an abstraction of the hybrid Statechart combined with the interface Statecharts of the subcomponents.

Figure 12 depicts a part from the monitor behavior and the BC integration from Figure 11 and a part form the interface Statechart of the BC component from Figure 9. 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 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. As sketched in Figure 12, we have to check that for each transition in the hybrid Statechart and the related state transitions implied by the assigned configurations of the source and target state for the aggregated subcomponents a compatible transition in the interface Statechart of each subcomponent exists (cf. Theorem 2 in Section 5.5).

Another proof becomes possible when we restrict the component behavior as well as the interface Statecharts to timed automata which can be model checked. We can then check simply whether their composition can only reach the specified configurations and that no timing inconsistencies or deadlocks exist.

In our example, a syntactical check of the hierarchical composition is sufficient to prove that the underlying subcomponents cannot invalidate the timing properties ensured by the embedding hybrid Statechart of the monitor.

4.5 Compositional Model Checking

At the higher abstraction levels the behavior of dynamically established structural connections are described by patterns. As depicted in Figure 6d

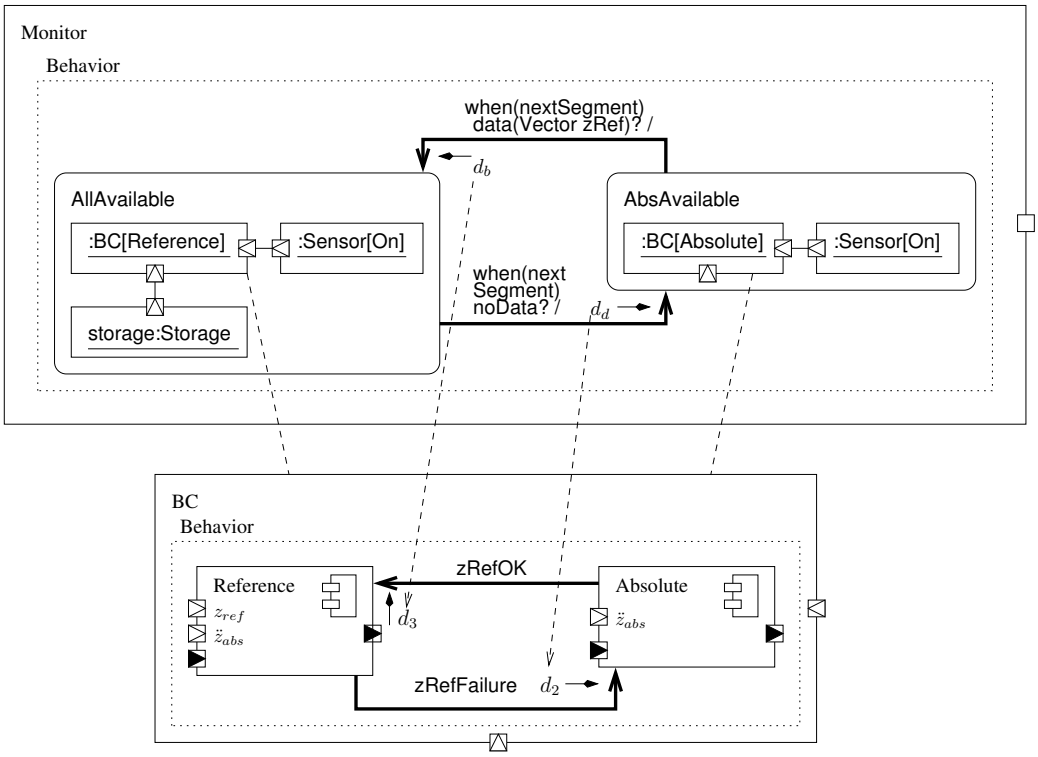

Figure 12: Scheme for the syntactical checking of correct reconfiguration

the communication between the registry and the monitor is described by the ShuttleRegistration Pattern pattern. Such a pattern consists of the communication protocol and the relevant behavior of the components, which is usually a further generalization of the components' and their embedded components' behavior.

To verify the correct coordination between the different components of the network via the pattern, we further exploit the abstraction relation between the subcomponents and its interface Statecharts. This implies that the hybrid Statechart plus the interface Statecharts are also a correct abstraction of the behavior of the component and its subcomponents (cf. Lemma 1). When the hybrid Statechart as well as the interface Statecharts are restricted to timed automata, we can model check them to exclude temporal inconsistencies. [34] shows that it is sufficient to verify correctness of the pattern and of the single components to ensure correctness of the composed system.

In our example we abstract from the quasi-continuous behavior such that the timing behavior of the monitor depicted in Figure 11 together with the interface automata of all embedded components contains no relevant hybrid behavior any more. Therefore, our concepts of refinement and generalization

enable us to use compositional model checking in order to verify the composed system's real-time behavior with feasible effort.

4.6 Pattern-based Verification

If the embedding statechart does only contain timing constraints but no general hybrid constraints, we can exploit the result of Theorem 2 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 embedding Real-Time Statechart of the monitor.

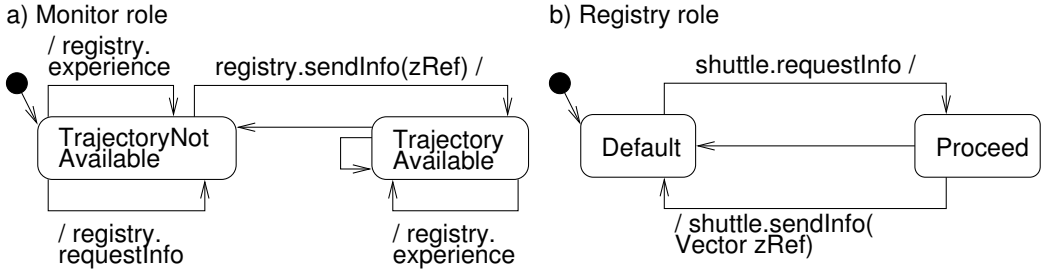

Figure 13: Roles of the Monitor-Registration-pattern

The real-time communication between the registry and the monitor is described by the ShuttleRegistration pattern as depicted in Figure **??**. 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 13b. The related Monitor role which has to be realized by the shuttle software is depicted in Figure 13a. 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 11.

To verify the correct real-time coordination between the monitor and registry the above outlined refinement of the role protocols as well as model checking the pattern has been shown to be sufficient to verify required local safety and liveness properties (cf. [34]).

To verify the correct real-time coordination between the monitor and the registry the above outlined refinement of the role protocols is required.

Further model checking the pattern has been shown to be sufficient to verify the required local safety and liveness properties (cf. [34]).

5 Formalization

We have to formally define the semantics of the employed and extended UML concepts such that the restrictions and requirements of our application domain as outlined in Section 3 are fulfilled. The request for verifiability further requires that the employed concepts have a rigorous foundation. Therefore, we also present formal definitions for the employed notion of hybrid automata, parallel composition, and refinement.

5.1 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 $[A \to B]$ and $[A \rightharpoonup B]$ to denote the set of total resp. partial functions from A to B. Let $f \in [A \to B]$ resp. $f \in [A \rightharpoonup B]$ then $\tilde{f} \in [C \to B]$ (resp. $\tilde{f} \in [C \rightharpoonup B]$) with $C \subseteq A$ is defined as $\tilde{f}(x) = f(x) \forall x \in C$. $EQ(V_l, V_r)$ denotes the set of all equations of the form $v_l = f^i(v_r^1, ..., v_r^n)$ with operations f^i of arity n and left- and right-hand side variables of the equation $v_l \in V_l$, $v_r^1, ..., v_r^n \in V_r$. $COND(V)$ 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 $v_l = \perp_i(v_r^1, ..., v_r^n)$ 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 $[V \to \mathbb{R}]$. All values of a vector of the length n can be represented in a similar fashion as $[[0, n] \to \mathbb{R}]$.

$f \otimes g$ further denotes the composition of the two functions $f : A_1 \to B_1$ and $g : A_2 \to B_2$ 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 $a_1 : [A_1 \to B_1] \to [A_1' \to B_1']$ and $a_2 : [A_2 \to B_2] \to [A_2' \to B_2']$ with disjoint sets $A_1 \cap A_2 = \emptyset$ and $A_1' \cap A_2' = \emptyset$ defined by $(a_1 \oplus a_2)(x \otimes y) := a_1(x) \otimes a_2(y)$.

5.2 Continuous Components

5.2.1 Continuous Behavior

For a subsequent presentation of the required hybrid behavioral model as a direct combination of its continuous and discrete parts, we describe at first an exclusively continuous system formally by means of differential equations for the state and an additional ordinary equation for the output.

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* $V^{\dot{x}}$ *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.*

$F \cup G$ 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 $X = [V^x \to \mathbb{R}]$ which describes all possible assignments for the state variables. A trajectory $\rho_u : [0, \infty] \to [V^x \to \mathbb{R}]$ for the set of differential equations F and input $u : [0, \infty] \to [V^u \to \mathbb{R}]$ 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 $\theta_u : [0, \infty] \to [V^y \to \mathbb{R}]$ using G analogously. The semantics for a continuous model M is given by all possible triples of environment and system trajectories (u, ρ_u, θ_u) denoted by $[\![M]\!]$.

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* $M_1 = (V_1^x, V_1^u, V_1^y, F_1, G_1, C_1)$ *and* $M_2 = (V_2^x, V_2^u, V_2^y, F_2, G_2, C_2)$ *denoted by* $M_1 \| M_2$ *is again a continuous model* $M = (V^x, V^u, V^y, F, G, C)$ *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* $X_0 = \{((l_1, l_2), (x_1 \otimes x_2)) | (l_1, x_1) \in X_1 \wedge (l_2, x_2) \in X_2\}$.

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

Note that we omit the explicit notion of time present in the form employed in CAMeL, because clocks can be easily emulated by means of an additional state variable $v_1 \in V^x$ with $\dot{v}_1 = 1 \in F$. This special case only constant operations are used as denoted by $OP_{const} \subset OP_{det}$.

5.2.2 Refinement of Continuous Behavior

The semantics for a continuous model M is given by all possible triples of environment and system trajectories (u, ρ_u, θ_u) denoted by $[\![M]\!]$. We further write $x \rightarrow_\pi x'$ for a path $\pi = (u|_{[\delta,\delta']}, \theta_u|_{[\delta,\delta']})$ with start $\delta \geq 0$ and end point $\delta' > \delta$ if $(u, \rho_u, \theta_u) \in [\![M]\!]$ and $x = \rho_u(\delta)$ and $x' = \rho_u(\delta')$ to abstract from the internal state ρ_u. The possible interference of the continuous model with its environment for a specific state x is described using $\text{offer}_c(M, x)$ which is defined as all possible values for the first derivation for all input variables which M can handle: $\text{offer}_c(M, x) := \{(du/dt)(0) | \exists (x \rightarrow_{(u,\theta_u)} x') \in [\![M]\!]\}$. A refinement needs to be able to handle at least all the interferences as the unrefined continuous model.

Definition 3 *For two continuous models M_I and M_R holds that M_R is a* refinement *of M_I denoted by $M_R \sqsubseteq_c M_I$ iff*

$$\forall x \in X_R : x \rightarrow_\pi \exists x' \in X_I : x' \rightarrow_\pi \quad \textit{and}$$

$$\textit{offer}_c(M_R, x) \supseteq \textit{offer}_c(M_I, x').$$

It is to be noted that $\text{offer}_c(C_I)$ may simply not consider unexpected cases for the input u. While in principle the realization can thus behave arbitrarily for such input, the current formalization assumes that the refinement does not specify the behavior for such cases either.

5.3 Discrete Components

5.3.1 Discrete Behavior

Timed automata can be employed to formally describe the outlined discrete behavior [33]. Due to lack of space we further will omit the syntactical complexity of the additional statechart concepts, such as hierarchy and history [38] within this paper and define a flat finite automata model. The time aspect will be added later when we extend the model to a hybrid one.

Definition 4 *A* finite automaton *is described by a quintuple (L, I, O, T, L^0) with L a finite set of locations, I a finite set of input signals, O a finite set of output signals, T a finite set of transitions, and $L_0 \subseteq L$ the set of start*

locations. Any transition $t = (l, g, l') \in T$ consists of a source location $l \in L$, a target location $l' \in L$, and an I/O-guard $g \in [\wp(I) \times \wp(O) \rightarrow \{true, false\}]$ which determines for which set of input and output signals the transition can be fired.

An automaton has an inner state, described by a location $l \in L$. The firing of a transition $t = (l, g, l') \in T$ changes the location from l to l' if appropriate input and output signals are available. The semantics for a finite automaton is given by all possible finite and infinite execution sequences $l_0 \rightarrow_{(i_0,o_0)} l_1 \rightarrow_{(i_1,o_1)} \ldots$ denoted by $[\![M]\!]$. The parallel composition of two automata is defined as follows:

Definition 5 *For M_1 and M_2 two automata the* parallel composition *$(M_1 \| M_2)$ is an automaton $M = (L, I, O, T, L^0)$ with $L = L_1 \times L_2$, $I = I_1 \cup I_2$, $O = O_1 \cup O_2$, $L^0 = L_1^0 \times L_2^0$, and $T = \{((l_1, l_2), g, (l_1', l_2')) | (l_1, g_1, l_1') \in T_1 \wedge (l_2, g_2, l_2') \in T_2\}$ where $g(i, o) = g_1(i \cap I_1, o \cap O_1) \wedge g_2(i \cap I_2, o \cap O_2)$.*

While in the continuous model stability is of major concern, in the automata model discrete anomalies like deadlocks or starvation have to be excluded.

5.3.2 Automata Refinement

The semantics for a finite automaton is given by all possible finite and infinite execution sequences $l_0 \rightarrow_{(i_0,o_0)} l_1 \rightarrow_{(i_1,o_1)} \ldots$ denoted by $[\![M]\!]$. We further write for a path $\pi = (i_0, o_0); (i_1, o_1); \ldots; (i_n, o_n)$ and appropriate states $l_0 \rightarrow_\pi l_n$ if $l_0 \rightarrow_{(i_0,o_0)} l_1 \rightarrow_{(i_1,o_1)} \cdots \rightarrow_{(i_n,o_n)} l_n \in [\![M]\!]$. The offered interactions for a state l are further denoted by the set $\text{offer}_a(M, l)$ which is defined as $\{(i, o) | \exists l \rightarrow_{(i,o)} l' \in [\![M]\!]\}$.

An appropriate branching-time notion of refinement for automata, which ensures safe substitution w.r.t. deadlocks,[5] can then be defined as follows:

Definition 6 *For two automata M_I and M_R holds that M_R is a* refinement *of M_I denoted by $M_R \sqsubseteq_a M_I$ iff a relation $\Omega \subseteq L_R \times L_I$ exists with for all $(l, l'') \in \Omega$ holds*

$$\forall l \rightarrow_\pi l' \quad \exists l'' \rightarrow_\pi l''' \quad : (l', l''') \in \Omega \qquad \textit{and}$$

$$\textit{offer}_a(M_R, l) \supseteq \textit{offer}_a(M_I, l'').$$

[5]The proposed solution equals ready simulation. A more general solution which is not restricted to pairs of states has been defined, e.g., for CSP in [39].

5.4 Hybrid Components

5.4.1 Hybrid Automata

The behavior described in Section 3.3, comprising continuous as well as discrete elements, can be formally defined by means of the concept of hybrid automata [5, 6, 7]. In the following, we use a formalization which combines Definitions 1 and standard automata and is especially adjusted to our needs. It provides means to specify continuous behavior, synchronous event handling, and reconfiguration as required to support self-optimization and to integrate the concepts of the mechatronic and software engineering domains.

Definition 7 *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,* $a \in [[V^x \to \mathbb{R}] \to [V^x \to \mathbb{R}]]$ *the continuous update, and* $l' \in L$ *the target-location. For every* $l \in L$ *we require that* $D(l)$ *is well-formed.*

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 $X = [V^x \rightharpoonup \mathbb{R}]$ 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 $x \in [V^x \to \mathbb{R}]$. 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 alters just the control 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 continuous guard is fulfilled $(g(x \otimes u) = true)$ for $u \in [V^u \to \mathbb{R}]$ 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 $(l, x) \to_{(i \cup o)} (l'', a(x))$.

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

$\rho_u : [0, \delta] \to [V^x \to \mathbb{R}]$ be a trajectory for the differential equations $F(l)$ and the external input $u : [0, \delta] \to [V^u - V^y \to \mathbb{R}]$ 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 $\theta_u : [0, \delta] \to [V^y \to \mathbb{R}]$ 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 $(u_0, l_0, \rho^0_{u_0}, \theta^0_{u_0}, \delta_0) \to_{e_0} (u_1, l_1, \rho^1_{u_1}, \theta^1_{u_1}, \delta_1) \ldots$ denoted by $[\![M]\!]_t$ where all $(l_i, \rho^i_{u_i}(\delta_i)) \to_{e_i} (l_{i+1}, \rho^{i+1}_{u_{i+1}}(0))$ 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 [5].

The parallel composition of two hybrid automata is defined as follows:

Definition 8 *For two hybrid automata M_1 and M_2 the* parallel composition *$(M_1 \| M_2)$ results in a hybrid automaton $M = (L, D, I, O, T, S^0)$ with $L = L_1 \times L_2$, $D(l, l') = D_1(l) \| D_2(l')$, $I = I_1 \cup I_2$, $O = O_1 \cup O_2$. The resulting transition relation is $T = \{((l_1, l_2), g_1 \wedge g_2, g^i_1 \cup g^i_2, (a_1 \oplus a_2), (l'_1, l'_2)) | (l_1, g_1, g^i_1, u_1, l'_1) \in T_1 \wedge (l_2, g_2, g^i_2, u_2, l'_2) \in T_2 \wedge g^i_1 \cap (I_2 \cup O_2) = g^i_2 \cap (I_1 \cup O_1)\} \cup \{((l_1, l_2), g_1, g^i_1, u_1, (l'_1, l_2)) | (l_1, g_1, g^i_1, u_1, l'_1) \in T_1 \wedge g^i_1 \cap (I_2 \cup O_2) = \emptyset\} \cup \{((l_1, l_2), g_2, g^i_2, u_2, (l_1, l'_2)) |\ (l_2, g_2, u_2, l'_2) \in T_2 \wedge g^i_2 \cap (I_1 \cup O_1) = \emptyset\}$. S^0 is defined as $S^0_1 \times S^0_2$.*

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 automata is well-defined.

Those complex conditions, which a correct parallel composition has to fulfill, highlight an important fact: Such a composition of two locations with incompatible control laws (e.g., if the input and output variables do not fit) can easily result in undefined behavior. Therefore, state-dependent changes of the set of supported output variables and necessary input variables that are required for a component-based modeling of self-optimizing hybrid behavior have to be subject to subtle cross-component coordination as outlined in Section 5.5. Also more appropriate means of describing the usual case of fading from one control law to another have to be provided, too. To support state-dependent input and output variables, we define a *flexible hybrid automaton*:

Definition 9 *A* flexible 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 $D(l) = (V^x(l), V^u(l), V^y(l), F(l), G(l), C(l), X^0(l))$ 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, $g \in COND(V^x(l) \cup V^u(l))$ 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.

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 taking instead of the location independent V^x etc. always the location dependent notion $V^x(l)$ into account.

The parallel composition also follows directly from the non configurable case. In the case of flexible hybrid 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 non-flexible 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 10 *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 $(l, g, g', a, l_f) \in T$ holds that $(v = 0) \in a$, there is exactly one transition leaving l_f $(|\{(l_f, g, g', a, l')|(l_f, g, g', a, l') \in T\}| = 1)$, and for this transition holds $g \equiv d_{min} \leq v \leq d_{max}$, $g' = true$ and $a = Id$. All non fading locations are* regular locations.

Definition 11 *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 statechart we can ensure that two fading locations are never directly connected.

5.4.2 Hybrid Refinement

We write for a possible execution sequence of states and transitions of a hybrid automaton $M = (L, D, I, O, T, S^0)$ with $(u_0, l_0, \rho^0_{u_0}, \theta^0_{u_0}, \delta_0) \rightarrow_{e_0} (u_1, l_1, \rho^1_{u_1}, \theta^1_{u_1}, \delta_1) \in [\![M]\!]_t$ simply $(l_0, \rho^0_{u_0}(0)) \rightarrow_{(u_0, \rho^0_{u_0}, \theta^0_{u_0}, \delta_0)} (l_0, \rho^0_{u_0}(\delta_0)) \rightarrow_{e_0} (l_1, \rho^1_{u_1}(0)) \rightarrow_{(u_1, l_1, \rho^1_{u_1}, \theta^1_{u_1}, \delta_1)} (l_1, \rho^1_{u_1}(\delta_1))$ to represent the state changes in a more uniform manner. We thus have the concept of a hybrid path $\pi = (u_0, \theta^0_{u_0}, \delta_0); e_0; \ldots; (u_n, l_n, \theta^1_{u_n}, \delta_n); e_n$ such that we write $(l_0, \rho^0_{u_0}(0)) \rightarrow_\pi (l_n, \rho^n_{u_n}(\delta_n))$ iff it holds that $(l_0, \rho^0_{u_0}(0)) \rightarrow_{(u_0, \rho^0_{u_0}, \theta^0_{u_0}, \delta_0)} (l_0, \rho^0_{u_0}(\delta_0)) \rightarrow_{e_0} \ldots (l_n, \rho^n_{u_n}(0)) \rightarrow_{(u_n, l_n, \rho^n_{u_n}, \theta^n_{u_n}, \delta_n)} (l_n, \rho^n_{u_n}(\delta_n))$.

For $e'_i = e_i - (O \cap I)$ the 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 have an abstract path $\pi' = (u_0, \theta^0_{u_0}, \delta_0); e'_0; \ldots; (u_n, \theta^1_{u_n}, \delta_n); e_n; \ldots$ and write $(l_0, \rho^0_{u_0}(0)) \Rightarrow_{\pi'} (l_n, \rho^n_{u_n}(\delta_n))$. 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 $\mathrm{offer}(M, (l, x))$ which is defined as $\{e | \exists((l, x) \Rightarrow_e (l', x)) \in [\![M]\!]\} \cup \{(du/dt)(0) | \exists((l, x) \Rightarrow_{(u, \theta_u, \delta)} (l, x')) \in [\![M]\!]\}$.

An appropriate notion of hybrid refinement can then be defined by combining the continuous notion of refinement from Definition 3 and the automata refinement of Definition 6 as follows:

Definition 12 *For two flexible hybrid 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, so that for every $c \in (L_R \times X_R)$ a $c'' \in (L_I \times X_I)$ exists such that $(c, c'') \in \Omega$ and for all $(c, c'') \in \Omega$ holds*

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

$$\mathit{offer}(M_R, c) \supseteq \mathit{offer}(M_I, c''). \tag{2}$$

We can show refinement for the parallel composition of hybrid automata when one automaton is a refinement.

Lemma 1 *For two hybrid automata M_R and M_I and any third automata M_C holds ($M_R \sqsubseteq M_I \Rightarrow M_R \| M_C \sqsubseteq M_I \| M_C$.)*
Proof: *(sketch) The relation Ω for $M_R \| M_C \sqsubseteq M_I \| M_C$ can be straight forward derived from the one for $M_R \sqsubseteq M_I$.* $\square$

The outlined notion of refinement further implies that a refined behavior will only show an erroneous behavior w.r.t. synchronization such as deadlocks when the refined behavior will do also. A first step to proof this is the following lemma.

Lemma 2 *For two hybrid automata M_R and M_I with $M_R \sqsubseteq M_I$ holds (M_I deadlock $\Rightarrow M_R$ deadlock).*
Proof: *(sketch) For each deadlock a related deadlock can be straight forward derived for M_R using the relation Ω for $M_R \sqsubseteq M_I$.* $\square$

The final result is that refinement preserves deadlock freedom for arbitrary contexts.

Theorem 1 *For two hybrid automata M_R and M_I with $M_R \sqsubseteq M_I$ and any context behavior M_C holds*

$$M_I \| M_C \text{ deadlock free} \Rightarrow M_R \| M_C \text{ deadlock free} .$$

Proof: *(sketch) by combination of the Lemmata 1 and 2.* $\square$

The presented results for the refinement of hybrid behavior is in the following employed to define interface automata and hybrid components.

5.4.3 Hybrid Statecharts and Components

Due to lack of space we further will omit the syntactical complexity of the additional statechart concepts, such as hierarchy and history [38] within this paper and further use the flat hybrid automata model of Definition 7. However, this can be accomplished much like the case of the syntax and semantics of the Real-Time Statechart presented in [33].

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.

For embedding or connecting a hybrid component (cf. Figure 8) we do not need all details of the component realization, but only enough information about its external observable behavior such that compatibility can be analyzed. This externally relevant behavior is described in our approach through an *interface statechart*. This interface statechart describes the external 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 8 is displayed in Figure 9. It shows that the body control 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.

The externally relevant behavior covered by the interface statecharts 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 13 *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 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* of the component described by a single hybrid Statechart which coordinates its aggregated subcomponents.
- an interface in form of an *interface statechart* which is a correct abstraction of the component behavior (cf. Definition 12).

In our example, the BC component is described by its realization by the hybrid statechart of Figure 8 where the required quasi-continuous behavior is specified by controllers in form of quasi-continuous blocks. The interface Statechart presented in Figure 9 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 14 *A* flexible hybrid component C *is a pair* (M_I, M_R) *with* $M_I = (L_I, D_I, I_I, O_I, T_I, S_I^0)$ *an interface automaton and* $M_R = (L_R, D_R, I_R, O_R, T_R, S_R^0)$ *the concrete hybrid realization 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* Ω *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 realization statechart by abstracting from realization details at the syntactical level. The valid refinement between a given interface statechart and a realization in a *top-down* scenario must in contrast be additionally verified at the semantical level.

Definition 15 *For flexible hybrid automata M_S and $M_1, \ldots, M_n$ the* hierarchical parallel composition *$(M_S \| _H (M_1 \| \ldots \| M_n))$ is defined by a restriction $H \subseteq L_S \times (L_1 \times \cdots \times L_n)$ on the hybrid automaton $M = M_S \| M_1 \| \ldots \| M_n$. The restriction holds iff for all $\vec{l} \in L$ reachable in $[\![M]\!]$ 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 in the interface statecharts specified events.

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

Definition 16 *For a hybrid automaton $M = (L, D, I, O, T, S^0)$ the* hiding *of some signals $A \subseteq I \cup O$ denoted by $M \backslash_A$ 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 flexible hybrid statechart specified in Figure 11 equals $M_S \backslash_{I_1 \cup \cdots \cup I_n \cup O_1 \cup \cdots \cup O_n}$. The additional details of M_S describing the coordination with M_1, ..., M_n are omitted in in Figure 11, 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 statechart (as present in Figure 11) makes sure that $|H \cap \{l_1\} \times (L_1 \times \cdots \times L_n)| = 1$. Consequently, l_1 can coexist only with the related regular target locations of $M_1, \ldots, M_n$. For a fading location $l_1 \in L_1$ we can expect that $|H \cap \{l_1\} \times (L_1 \times \cdots \times L_n)| = 2$ such that l_1 can coexist only

with the related regular target locations of $M_1, \ldots, M_n$ or with intermediate fading locations of $M_1, \ldots, M_n$ which eventually lead to this target location. We require that there is no interaction among $M_1, \ldots, M_n$, so that they do not initiate location changes of parallel components.

5.5 Syntactical Refinement

As model checking for the general form of hybrid systems considered here is undecidable [37], checking refinement of general hybrid systems is thus also undecidable. Due to the fact that hierarchical composition in contrast to the general parallel composition restricts a potential overlapping of locations, the compatibility can be checked on the syntactical level without consideration of the full state-space of the model (cf. Figure 12). In these checks fading transitions and their durations play an important role. Formalizinig their semantics leads to *simple interface Statecharts*.

Definition 17 *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.*

The following theorem describes a simple syntactical rule which is sufficient to proof for the above sketched restricted case that a hierarchical parallel product does not have any timing errors. The basic idea is that the timing interval of a hybrid Statechart's fading transitions has to be conform with the one of its embedded simple interface Statechart, as described in Section 4.4.

Theorem 2 *For the hierarchical parallel composition* $M_1 \|_H M_2$ *of two hybrid automata* M_1 *and* M_2 *holds* $M_1 \|_H M_2 \sqsubseteq M_1 \backslash_{I_2 \cup O_2}$ *if*

1. $I(M_1 \|_H M_2) = I(M_1 \backslash_{I_2 \cup O_2})$,
2. *all initial states are also contained in* H: $(\{(l_1, l_2) | (l_1, x) \in S_1^0 \wedge (l_2, y) \in S_2^0\} \subseteq H)$,
3. M_2 *is a simple interface Statechart (cf. Definition 17), and*
4. *for all* $(l_1, l_2) \in H$ *and transition* $t_1 = (l_1, g_1, g_1^i, a_1, l_1') \in T_1$ *holds:*
   (a) *if* l_1' *is not a fading location, then for all* $t_2 = (l_2, g_2, g_2^i, u_2, l_2') \in T_2$ *with* $g_1^i \cap (I_2 \cup O_2) = g_2^i$ *must hold:*
      i. $g_2 = \text{true}$,
      ii. l_2' *is a passive location (cf. Definition 11), and*

iii. $(l'_1, l'_2) \in H$.

In addition at least one such transition in M_2 *must exist.*

(b) if l'_1 *is a fading location we can conclude that exactly one transition* $t'_1 = (l'_1, g'_1, g_1^{i'}, a'_1, l''_1) \in T_1$ *with* $g'_1 \equiv d^1_{min} \leq v \leq d^1_{max}$ *and* $g_1^{i'} = \emptyset$ *exists (see Definition 10). For any* $t_2 = (l_2, g_2, g_2^i, a_2, l'_2) \in T_2$ *with* $g_1^i \cap (I_2 \cup O_2) = g_2^i$ *must hold:*

i. $g_2 = \text{true}$,

ii. l'_2 *is a fading location, and*

iii. $(l'_1, l'_2) \in H$.

For the uniquely determined successor transition $t'_2 = (l'_2, g'_2, g_2^{i'}, a'_2, l''_2) \in T_2$ *with* $g'_2 \equiv d^2_{min} \leq v \leq d^2_{max}$ *must hold:*

iv. l''_2 *is a passive location (cf. Definition 11),*

v. $(l''_1, l''_2) \in H$, *and*

vi. $[d^2_{min}, d^2_{max}] \subseteq [d^1_{min}, d^1_{max}]$ *must be satisfied.*

Again, at least one such pair of transitions in M_2 *must exist.*

Proof: *In order to proof Theorem 2, first an appropriate* Ω *conform to Definition 12 is chosen, so that every state* (l, x) *of* M_1 *is related to an according state of* $M_1 \|_H M_2$. *Then, we will show that for every transition* t_I *of* M_1 *an according transition* t_R *of* $M_1 \|_H M_2$ *exists and that* t_R *fires when* t_I *fires. Then, we can conclude that every path in* $[\![M_1 \|_H M_2]\!]_t$ *corresponds to one of* $[\![M_1]\!]_t$, *which is used to show that equations (1) and (2) from Definition 12 hold. Due to the distinction between continuous steps and discrete steps and due to the distinction between fading and non-fading transitions, the proof consists of 3 different cases:*

Let $X_R := [V_1^x \cup V_2^x \to \mathbb{R}]$, *choose* $\Omega = \{((l_1, l_2), x_R), (l_1, \tilde{x}_R)) | (l_1, l_2) \in H, x_R \in X_R\}$.

Case 1: Discrete non-fading transitions *For all transitions* $t_1 = (l_1, g_1, g_1^i, a_1, l'_1) \in T_1$, *with* l'_1 *is not a fading location, exists one path in the automaton's semantics:* $(l_1, x_1) \to_{g_1^i} (l'_1, a_1(x_1)) \in [\![M_1]\!]_t$. *Due to the requirement (4a) of this theorem, there exists at least one transition* $t_2 = (l_2, g_2, g_2^i, a_1, l'_2) \in T_2$ *with* $g_2 = \text{true}$, $g_2^i = g_1^i \cap (I_2 \cup O_2)$, $C_2(l_2) = \text{true}$ *and* $(l'_1, l'_2) \in H$. *Due to Definition 8* $M_1 \|_H M_2$ *contains* t_1 *a transition*

$$t = ((l_1, l_2), g_1 \wedge g_2, g_1^i \cup g_2^i, (a_1 \oplus a_2), (l'_1, l'_2)) \tag{3}$$

with $g_1^i \cap (I_2 \cup O_2) = g_2^i$. *As* $g_1 \wedge g_2 \overset{(4ai)}{=} g_1 \wedge \text{true} = g_1$ *and as* $g_1^i \cup g_2^i \overset{(4a)}{=} g_1^i \cup (g_1^i \cap (I_2 \cup O_2)) = g_1^i$, t *is equal to*

$$t = ((l_1, l_2), g_1, g_1^i, (a_1 \oplus a_2), (l'_1, l'_2)) \tag{4}$$

Note that firing of t just depends on the triggers g_1 and g_1^i and not on g_2 or g_2^i. Thus, for all $(l_1, l_2) \in H$ holds that an according path from (l_1, l_2) to (l_1', l_2') is in the semantics of $M_1 \|_H M_2$: $((l_1, l_2), x_R) \rightarrow_{g_1^i} ((l_1', l_2'), (a_1 \oplus a_2)(x_R)) \in [\![M_1 \|_H M_2]\!]_t$ and $(l_1', l_2') \in H$.

Therefore, for each $c := ((l_1, l_2), x_R) \in (L_1 \times L_2) \times X_R$ there exists a $c'' := (l_1, \tilde{x}_R)$ so that for $c' := ((l_1', l_2'), x_R') \in (L_1 \times L_2) \times X_R$, and $c''' := (l_1', \tilde{x}_R')$, holds: There exists the execution sequences $c \rightarrow_{g_1^i} c'$ and $c'' \rightarrow_{g_1^i} c'''$ with $(c, c'') \in \Omega$ and $(c', c''') \in \Omega$, which is a requirement for refinement, cf. Definition 12.

Case 2: Discrete fading transitions *For all transitions $t_1 = (l_1, g_1, g_1^i, a_1, l_1') \in T_1$, with l_1' is a fading location, holds that a path is in the semantics of M_1 that represents firing of t_1, residing in l_1' for a specific time δ_0, and leaving l_1' after δ_0:*

$$(l_1, x_1) \rightarrow_{g_1^i} (l_1', \rho_0(0)) \rightarrow_{(u_0, \rho^0_{u_0}, \theta^0_{u_0}, \delta_0)} (l_1', \rho_0(\delta_0)) \rightarrow_{\emptyset} (l_1'', a_1'(\rho_0(\delta_0))) \in [\![M_1]\!]_t \tag{5}$$

with $\delta_0 \in [d^1_{min}, d^2_{max}]$. Due to requirement (4b) of this theorem there exists at least one transition

$$t_2 = (l_2, g_2, g_2^i, a_2, l_2') \in T_2 \tag{6}$$

with $g_2 = \text{true}$, $g_2^i = g_1^i \cap (I_2 \cup O_2)$, and $(l_1', l_2') \in H$. As l_2' is a fading location, there exists a successor transition of t_2 (cf. Definition 10):

$$t_2' = (l_2', g_2', g_2^{i'}, a_2', l_2'') \in T_2 \tag{7}$$

with $g_2' \equiv d^2_{min} \leq v \leq d^2_{max}$, $g_2^{i'} = \emptyset$, and $(l_1'', l_2'') \in H$. Due to Definition 8 $M_1 \|_H M_2$ contains a

$$t = ((l_1, l_2), g_1 \wedge g_2, g_1^i \cup g_2^i, (a_1 \oplus a_2), (l_1', l_2')) \tag{8}$$

with $g_1^i \cap (I_2 \cup O_2) = g_2^i$ and a transition

$$t' = ((l_1', l_2'), g_1' \wedge g_2', g_1^{i'} \cup g_2^{i'}, (a_1' \oplus a_2'), (l_1'', l_2'')) \tag{9}$$

with $g_1^{i'} \cap (I_2 \cup O_2) = g_2^{i'}$. As $g_1 \wedge g_2 \overset{(4bi)}{=} g_1 \wedge \text{true} = g_1$ and as $g_1^i \cup g_2^i \overset{(4b)}{=} g_1^i \cup (g_1^i \cap (I_2 \cup O_2)) = g_1^i$, t is equal to

$$t = ((l_1, l_2), g_1, g_1^i, (a_1 \oplus a_2), (l_1', l_2')) \tag{10}$$

Note that firing again just depends on the triggers g_1 and g_1^i and not on g_2 or g_2^i. Due to requirement (4b) holds: $g_1' \wedge g_2' \equiv (d^1_{min} \leq d^1_{max}) \wedge (d^2_{min} \leq d^2_{max})$. As $[d^2_{min}, d^2_{max}] \subseteq [d^1_{min}, d^1_{max}]$ (cf. requirement (4b vi), it follows that

$$g_1' \wedge g_2' \equiv d^2_{min} \leq v \leq d^2_{max}. \tag{11}$$

As further $g_1^{i'} \cup g_2^{i'} \overset{(4b),(4biv)}{=} \emptyset \cup \emptyset = \emptyset$, t' *is equal to*

$$t' = ((l_1', l_2'), d_{min}^2 \leq v \leq d_{max}^2, \emptyset, (a_1' \oplus a_2'), (l_1'', l_2'')). \tag{12}$$

Note, that firing of t' *just depends on* $d_{min}^2 \leq v \leq d_{max}^2$. t' *fires after the time* $\delta_0 \in [d_{min}^2; d_{max}^2]$.

Thus, for all $(l_1, l_2) \in H$ *holds that every path in* $[\![M_1 \|_H M_2]\!]_t$ *corresponds to one of* $[\![M_1]\!]_t$, *like the one from equation (5):* $((l_1, l_2), x_R) \rightarrow_{g_1^i} ((l_1', l_2'), (a_1 \oplus a_2)(x_R)) \rightarrow_{(u_0, \rho_{u_0}^0, \theta_{u_0}^0, \delta_0)} ((l_1', l_2'), (\rho^0(a_1 \oplus a_2)(x_R))) \rightarrow_{\emptyset} ((l_1'', l_2''), (\rho^0(a_1 \oplus a_2)(x_R))) \in [\![M_1 \|_H M_2]\!]_t$. *Further it holds:* $(l_1', l_2') \in H$ *and* $(l_1'', l_2'') \in H$.

Therefore, for each $c := ((l_1, l_2), x_R) \in (L_1 \times L_2) \times X_R$ *there exists a* $c'' := (l_1, \tilde{x}_R)$ *so that for* $c' := ((l_1'', l_2''), x_R') \in (L_1 \times L_2) \times X_R$, *and* $c''' := (l_1'', \tilde{x}_R')$, *holds: There exists the execution sequences* $c \Rightarrow_\pi c'$ *with* $\pi = g_1^i; (u_0, \theta_{u_0}^0, \delta_0); \emptyset$ *and* $c'' \Rightarrow_\pi c'''$ *with* $(c, c'') \in \Omega$ *and* $(c', c''') \in \Omega$, *which is again the above mentioned requirement for refinement, cf. Definition 12.*

Case 3: Continuous transitions *Let* $(l_1, \tilde{\rho}_{u_0}^0(0)) \rightarrow_{(\tilde{u}_0, \tilde{\rho}_{u_0}^0, \theta_{u_0}^0, \delta_0)} (l_1, \tilde{\rho}_{u_0}^0(\delta_0))$ *be a path from the semantics* $[\![M_1]\!]$. *For all* $(l_1, l_2) \in H$ *exists an according path in the semantics* $[\![M_1 \|_H M_2]\!]_t$: $((l_1, l_2), \rho_{u_0}^0(0)) \rightarrow_{(u_0, \rho_{u_0}^0, \theta_{u_0}^0, \delta_0)} ((l_1, l_2), \rho_{u_0}^0(\delta_0))$.

Therefore, for each $c := ((l_1, l_2), \rho_{u_0}^0(0)) \in (L_1 \times L_2) \times X_R$ *there exists a* $c'' := (l_1, \tilde{\rho_{u_0}^0}(0))$ *so that for* $c' := ((l_1, l_2), \rho_{u_0}^0(\delta_0)) \in (L_1 \times L_2) \times X_R$, *and* $c''' := (l_1', \tilde{\rho}_{u_0}^0(\delta_0))$, *holds: There exists the execution sequences* $c \rightarrow_{(u_0, \rho_{u_0}^0, \theta_{u_0}^0, \delta_0)} c'$ *and* $c'' \rightarrow_{(u_0, \rho_{u_0}^0, \theta_{u_0}^0, \delta_0)} c'''$ *with* $(c, c'') \in \Omega$ *and* $(c', c''') \in \Omega$, *which is again the above mentioned requirement for refinement, cf. Definition 12.*

Hybrid paths *Induction is used to show that the above results are valid for* any *path of discrete or continuous transitions. Therefore* $\forall c = ((l_1, l_2), x_R) \exists c'' = (l_1, \tilde{x}_R)$ *so that* $\forall (c, c'') \in \Omega$ *holds condition 1:*

$$\forall c \Rightarrow_\pi c' \quad \exists c'' \Rightarrow_\pi c''' \quad : (c', c''') \in \Omega \tag{13}$$

Environment interference *Since we showed that each part of a possible path corresponds to a discrete fading or non-fading or to the flow of a location, we conclude* $offer(M_1, (l_1, x)) = \{g_1^i | \exists (l_1, g_1, g_1^i, a_1, l_1') \in T_1\} \cup \{(du/dt)(0) | \exists (l_1, x) \Rightarrow_{(u, \theta_u, \delta)} (l_1, x')\}$.

Further, we conclude $offer(M_1 \|_H M_2, ((l_1, l_2), x)) = \{g_1^i | \exists ((l_1, l_2), g_1, g_1^i, a_1 \oplus a_2, (l_1', l_2')) \in T_{1\|_H 2}\} \cup \{(du/dt)(0) | \exists ((l_1, l_2), x) \Rightarrow_{(u, \theta_u, \delta)} (l, x')\}$ *This is equivalent to* $\{g_1^i | \exists (l_1, g_1, g_1^i, a_1, l_1') \in T_1\} \cup \{(du/dt)(0) | \exists l_1 \in L_1$ *with* $D(l_1) =$

$(V^x(l_1), V^u(l_1), V^y(l_1), F(l_1), G(l_1), C(l_1), X^0(l_1))\}$ *and for which exists trajectories* ρ_u *and* θ_u.

Therefore it holds: $\mathit{offer}(M_R, c) \supseteq \mathit{offer}(M_I, c'')$. *Due to Definition 12 and together with (13) follows that* $M_1 \|_H M_2 \sqsubseteq M_1 \backslash_{I_2 \cup O_2}$. $\square$

Theorem 2 can be extended to the general case of $M_S \|_H (M_1 \| \ldots \| M_n)$ by induction. Due to the syntactical check of Theorems 2, the hierarchical composition by means of the underlying hybrid control software cannot invalidate the timing properties ensured by the embedding hybrid statechart of the monitor.

6 Run-Time Architecture and Tool Support

In order to reach interoperability for mechatronic systems, which are only alterable within certain limits, one can use appropriate middleware solutions like IPANEMA [40], which allows abstraction from hardware details. IPANEMA is a platform concept for distributed real-time execution and simulation to support rapid prototyping. It allows a modular-hierarchical organization of tasks or processes on distributed hardware.

In order to make interoperability possible also for hybrid components, which contain the kinds of alteration capability described above, the support of alteration capability by the middleware must be considerably extended. First it is necessary to generate the models in accordance to their modular-hierarchical structure. This is the basis for a reconfiguration.

In each discrete state of the system $l \in L$, the differential equations $F(l)$ and ordinary equations of $G(l)$ have to be employed to compute the correct continuous behavior. Thus in every location of this kind only the relevant equations have to be evaluated. To reach this aim, the architecture provides means for every component to adjust the set of active equation blocks in such a way that the required reconfiguration of the component system is efficiently managed.

In the modular execution framework outlined, the required execution order results from the local evaluation dependencies within each component as well as from their interconnection. It must thus be determined at deployment-time or run-time.

The continuous nonlinear differential equations are solved by applying suitable numeric solvers. Computation is time-discrete. Incrementation depends on the solver and the dynamics of the continuous system. A time-accurate detection of a continuous condition is not possible if the controller is coupled with a real technical system. Thus, we restrict the urgent reaction

to continuous conditions in the hybrid statecharts to a detection within the desired time slot (cf. [41, 23, 42]).

It is planned to support the presented concepts by both the CAE tool CAMeL [30] and the CASE tool Fujaba [34]. With both tools, the result of every design activity will be a hybrid component. Each of these hybrid components itself can integrate hybrid components. The integration only requires the information given by the component interface. E.g., the code generation for a hybrid statechart will check timing inconsistencies due to invalid hierarchical compositions and will generate code for the embedding component which coordinates the aggregated subcomponents (cf. Definition 15).

7 Conclusion and Future Work

Complex mechatronic systems with self-optimization are hybrid systems which reconfigure themselves at run-time. As outlined in the paper, their modeling can hardly be done by the approaches currently available. Therefore, we propose an extension of UML components and Statecharts towards reconfigurable hybrid systems which supports the modular hierarchical modeling of reconfigurable systems with hybrid components and hybrid Statecharts.

The presented approach permits that the needed discrete coordination can be designed by a software engineer with statecharts extended only with time. (cf. Figure 6a; it manages the necessary coordination with the track-segment registry and keeps track of the available signals.) In parallel, a control engineer can construct the advanced controller component which offers the technical feasible reconfiguration steps. (see Figure 8a; it describes the possible control laws and how and when a switching or fading between them is possible.) These two views can then be integrated as depicted in Figure 11 using only the component interface of the controller component.

A first problem which comes with the ability of the system to reconfigure itself at run-time is incompatible reconfiguration. Here we exploit the domain specific restrictions that within one subsystem the continuous operating controller for the related sensor and actuators is always described by a strict hierarchical structure. Thus only the correct coordination between each component and its aggregated subcomponent has to be studied by considering their abstractions in form of the interface Statecharts to ensure the correct reconfiguration of the whole system.

At the higher abstraction levels the behavior of dynamically established structural connections are described by pattern. When the hybrid Statechart

as well as the interface Statecharts are restricted to timed automata, we can model check them to exclude temporal inconsistencies by using the results of [34]: verifying the correctness of the pattern and of the single components is thus sufficient to ensure the correctness of the composed system.

The presented concepts are planed to be supplemented by an automatic and modular code generation for the hybrid models and run-time support for the reconfiguration. Thus, it will support the modular and component-based development of self-optimizing mechatronic systems from the model level down to the final code.

It is planned to support the presented concepts by both the CAE tool CAMeL [30] and the CASE tool Fujaba [34]. With both tools, the result of every design activity will be a hybrid component. Each of these hybrid components itself can integrate hybrid components. The integration only requires the information given by the component interface. Additional automatic and modular code generation for the hybrid models and run-time support for the reconfiguration will thus result in support for the modular and component-based development of self-optimizing mechatronic systems from the model level down to the final code.

Acknowledgement

We thank Björn Axenath for his comments on earlier versions of the paper.

References

[1] Bradley, D., Seward, D., Dawson, D., Burge, S.: Mechatronics. Stanley Thornes (2000)

[2] Li, P.Y., Horowitz, R.: Self-optimizing control. In: Proc. of the 36th IEEE Conference on Decision and Control (CDC), San Diego, CA, USA (1997) 1228–1233

[3] Föllinger, O., Dörscheid, F., Klittich, M.: Regelungstechnik - Einführung in die Methoden und ihre Anwendung. Hüthig (1994)

[4] Isermann, R., Lachmann, K.H., Matko, D.: Adaptive Control Systems. Prentice Hall (1992)

[5] Henzinger, T.A., Ho, P.H., Wong-Toi, H.: HyTech: The Next Generation. In: Proc. of the 16th IEEE Real-Time Symposium, IEEE Computer Press (1995)

[6] Bender, K., Broy, M., Peter, I., Pretschner, A., Stauner, T.: Model based development of hybrid systems. In: Modelling, Analysis, and Design of Hybrid Systems. Volume 279 of Lecture Notes on Control and Information Sciences. Springer Verlag (2002) 37–52

[7] Alur, R., Dang, T., Esposito, J., Fierro, R., Hur, Y., Ivancic, F., Kumar, V., Lee, I., Mishra, P., Pappas, G., Sokolsky, O.: Hierarchical Hybrid Modeling of Embedded Systems. In: First Workshop on Embedded Software. (2001)

[8] Parnas, D.L.: A Technique for Software Module Specification with Examples. Communications of the ACM **15** (1972) 330–336

[9] Object Management Group: UML 2.0 Superstructure submission V2.0. (2003) Document ad/03-01-02.

[10] Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T., Ho, P.H., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. Theoretical Computer Science **138** (1995)

[11] Lamport, L.: Hybrid Systems in TLA+, Springer-Verlag (1993)

[12] Wieting, R.: Hybrid high-level nets. In: Proceedings of the 1996 Winter Simulation Conference, Coronado, CA, USA (1996) 848–855

[13] Müller, C., Rake, H.: Automatische synthese von steuerungskorrekturen. In: KONDISK-Kolloquium, Berlin (1999)

[14] Kesten, Y., Pnueli, A.: 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)

[15] Stauner, T., Pretschner, A., Péter, I.: 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, Toronto, Canada (2001) 242–257

[16] Grosu, R., Stauner, T., Broy, M.: 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)

[17] Stauner, T.: Systematic Development of Hybrid Systems. PhD thesis, Technische Universität München (2001)

[18] Grosu, R., Krueger, I., Stauner, T.: Hybrid sequence charts. Technical Report TUM-I9914, Technische Universitaet München, München (1999)

[19] Selic, B., Gullekson, G., Ward, P.T.: Real-Time Object-Oriented Modeling. John Wiley and Sons, New York (1994)

[20] ObjecTime Ltd: UML for Real Time. (1999)

[21] Conrad, M., Weber, M., Mueller, O.: Towards a methodology for the design of hybrid systems in automotive electronics. In: Proc. of the International Symposium on Automotive Technology and Automation (ISATA'98). (1998)

[22] Henzinger, T.A.: 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] Henzinger, T.A., Kirsch, C.M., Sanvido, M.A., Pree, W.: 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)

[24] Kapinski, J., Krogh, B.: Verifying switched-mode computer controlled systems. In: Computer Aided Control System Design, 2002, IEEE Control Systems Society (2002)

[25] Object Management Group: UML for System Engineering Request for Proposal. (2003)

[26] SysML: Systems Modeling Language: SysML. (2004)

[27] Lückel, J., Grotstollen, H., Jäker, K.P., Henke, M., Liu, X.: Mechatronic design of a modular railway carriage. In: Proc. of the 1999 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM99), Atlanta, GA, USA (1999)

[28] Hestermeyer, T., Schlautmann, P., Ettingshausen, C.: Active suspension system for railway vehicles-system design and kinematics. In: Proc. of the 2nd IFAC - Confecence on mechatronic systems, Berkeley, California, USA (2002)

[29] Liu-Henke, X., Lückel, J., Jäker, K.P.: Development of an active suspension/tilt system for a mechatronic railway carriage. In: 1st IFAC Conference on Mechatronic Systems (Mechatronics 2000), Darmstadt, Germany (2000)

[30] Richert, J.: 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)

[31] Lygeros, J., Johansson, K.H., Simic´, S.N., Zhang, J., Sastry, S.S.: Dynamical Properties of Hybrid Automata. In: IEEE TRANSACTIONS ON AUTOMATIC CONTROL, VOL. 48, NO. 1, JANUARY 2003. Volume 48. (2003)

[32] Deppe, M., Oberschelp, O.: Real-Time Support For Online Controller Supervision And Optimisation. In: Proc. of DIPES 2000. International IFIP WG 10.3 / WG 10.4 / WG 10.5 Workshop on Distributed and Parallel Embedded Systems, Mechatronics Laboratory Paderborn, Paderborn University. Supported by IFIP WG 10.3 / WG 10.4 /WG 10.5 and Gesellschaft für Informatik. (2000)

[33] Giese, H., Burmester, S.: Real-Time Statechart Semantics. Technical Report tr-ri-03-239, University of Paderborn, Paderborn, Germany (2003)

[34] Giese, H., Tichy, M., Burmester, S., Schäfer, W., Flake, S.: Towards the Compositional Verification of Real-Time UML Designs. In: Proc. of the European Software Engineering Conference (ESEC), Helsinki, Finland. (2003) Copyright 2003 by ACM, Inc.

[35] Lüttgen, G., von der Beeck, M., Cleaveland, R.: 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. (2000) 120–129

[36] Kühl, M., Reichmann, C., Prötel, I., Müller-Glaser, K.D.: From object-oriented modeling to code generation for rapid prototyping of embedded electronic systems. In: Proc. of the 13th IEEE International Workshop on Rapid System Prototyping (RSP'02), Darmstadt, Germany (2002)

[37] Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What's decidable about hybrid automata? Journal of Computer and System Sciences **57**

(1998) 94–124 A preliminary version appeared in the Proceedings of the 27th Annual Symposium on Theory of Computing (STOC), ACM Press, 1995, pp. 373-382.

[38] Harel, D.: STATECHARTS: A Visual Formalism for complex systems. Science of Computer Programming **3** (1987) 231–274

[39] Hoare, C.A.R.: Communicating Sequential Processes. Series in Computer Science. Prentice-Hall International (1985)

[40] Honekamp, U.: IPANEMA - Verteilte Echtzeit-Informationsverarbeitung in mechatronischen Systemen. PhD thesis, Universität Paderborn, Düsseldorf (1998)

[41] Henzinger, T.A., Horowitz, B., Kirsch, C.M.: Giotto: A Time-triggered Language for Embedded Programming. In: Proceedings of the IEEE 91:84-99, 2003. A preliminary version appeared in the Proceedings of the First International Workshop on Embedded Software (EMSOFT), Lecture Notes in Computer Science 2211, Springer-Verlag, pp. 166-184. (2001)

[42] Stauner, T.: Discrete-time refinement of hybrid automata. In Tomlin, C., Greenstreet, M., eds.: Proceedings of the 5th International Workshop on Hybrid Systems: Computation and Control (HSCC 2002). Volume 2289 of Lecture Notes in Computer Science (LNCS)., Stanford, CA, USA (2002) 407ff