Syntax and Semantics of Hybrid Components

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

October, 2005
Abstract

Nowadays, the specification of complex systems is done in a component-based manner to obtain a well-structured design. The Unified Modeling Language (UML) became the standard for the component-based software design of non-safety-critical or non-real-time systems. In order to apply component-based UML models even for such critical, technical systems, MECHATRONIC UML has been created as extension of UML. To enable automatic generation of source code and to enable automatic analysis techniques like model checking, a formally defined semantics is required. Therefore, we define the syntax and the formal semantics of hybrid components and hybrid reconfiguration charts from the MECHATRONIC UML in this work.
Contents

1 Introduction 1

2 Application Example 2

3 Modeling with Mechatronic UML 3

4 Flat behavioral models 12
   4.1 Continuous Behavior 13
   4.2 Hybrid Automata 14
   4.3 Hybrid Reconfiguration Automata 16

5 The Component Model 20
   5.1 Static Component Structures 21
   5.2 Variable Component Structures & Reconfiguration 25
   5.3 Syntactical Refinement 28

6 Advanced High-Level Modeling 32
   6.1 Committed Locations 33
   6.2 Asynchronous Communication 34
   6.3 History 34
   6.4 Hierarchy 35
   6.5 exit()- and entry()-methods 35
   6.6 Transition Priorities 37
   6.7 do()-method 39
   6.8 Stop State 39
   6.9 Fading Transitions 40
   6.10 Implied State Changes 40
   6.11 Blockable & Enforceable Transitions 41

7 Related Work 42

8 Conclusion 45

A Prerequisites: Formal Definitions 48
1 Introduction

Recent developments of classical mechanical products show a growing percentage of electrical and especially of software modules. To increase comfort and safety, such mechatronic systems, which combine technologies from mechanical, electrical and software engineering, have to react flexible to a changing environment. Self-optimization also plays an important role in recent developments. If self-optimizing processes lead to new knowledge, the system also has to adapt its behavior in an appropriate manner. Therefore, mechatronic systems are required to adapt their structure and behavior at run-time (reconfiguration). As mechatronic systems usually have additionally real-time requirements and as they often show hybrid behavior, they become more and more complex. Thus, advanced specification techniques are required that reduce the complexity of the models.

In order to obtain a well-structured design, component-based specification techniques are applied like the Unified Modeling Language (UML) [24, 23] which became the standard in software engineering. In order to apply component-based UML models even for the specification of complex, reconfigurable mechatronic systems, MECHATRONIC UML [10] has been invented as extension of UML.

Among others, MECHATRONIC UML defines hybrid components and hybrid reconfiguration charts [12, 9]. Hybrid components integrate discrete behavior specified by extended timed automata models with continuous components (e.g. feedback controllers) that are specified by block diagrams or differential equations. Hybrid reconfiguration charts are used to specify the real-time requirements and the reconfiguration. Due to a tight integration of structure and behavior, hybrid reconfiguration charts and hybrid components lead to models which require reduced effort for analyzes and which thus enable ruling the complexity.

To enable automatic generation of source code and to enable automatic analysis techniques like modelchecking, a formally defined semantics is required. Therefore, we define the syntax and the formal semantics of hybrid components and hybrid reconfiguration charts from the MECHATRONIC UML in this work.

In the next section, we present our application example. In Section 3, it is described how to model reconfiguration with our MECHATRONIC UML approach. Sections 4 and 5 provide the formal definition of syntax and semantics of the flat automata models and the component model applied in Section 3. Syntax and semantics of the high-level constructs applied also in Section 3 are presented in Section 6. Section 7 presents related work and Section 8 draws a conclusion.
2 Application Example

Our application example is taken from the RailCab research project. In this project, a modular rail system is developed consisting of autonomous shuttles which apply the linear drive technology used in the Transrapid, but use existing rail tracks.

The shuttle’s active suspension system and its optimization is one example for a complex mechatronic system we employ in the following. The suspension/tilt module, depicted in Figure 1, is based on air springs which are damped actively by a displacement of their bases and three vertical hydraulic cylinders which move the bases of the air springs via an intermediate frame – the suspension frame. The vital task of the system is to provide the passengers a high comfort and to guarantee safety and stability when controlling the shuttle’s coach body. In order to achieve this goal, multiple feedback controllers are applicable with different capabilities in matters of safety and comfort.

In our controlling component, we apply the three feedback controllers reference, absolute, and robust, providing different levels of comfort and requiring different inputs. The most sophisticated controller reference uses a given trajectory $z_{ref} = f(x)$ that describes the ideal motion of the coach body and the

---

1 http://www-nbp.upb.de/en
2 http://www.transrapid.de/en
absolute acceleration $\ddot{z}_{abs}$ of the coach body. The $z_{ref}$ trajectory is given for each single track section and is communicated by a track section’s registry to the shuttle. In case the reference trajectory is not available, the less comfortable controller absolute which requires only the $\ddot{z}_{abs}$ signal has to be used. In case the sensor that provides the $\ddot{z}_{abs}$ signal fails, the robust controller which provides the fewest comfort, but guarantees stability even when only standard inputs are available, has to be applied.

3 Modeling with Mechatronic UML

One standard approach for modeling the reconfiguration, i.e. for modeling the exchange of the feedback-controllers, is to apply the concept of hybrid automata [15, 1]. Figure 2 depicts such a hybrid automaton. The automaton specifies the behavior of the component which controls the shuttle’s car body – thus it is called BodyControl (BC) component.

The automaton consists of three locations,\(^3\) associated with the aforementioned controllers reference (:Ref), absolute (:Abs), and robust (:Rob). The other four locations are called fading locations as they apply a technique called output cross-fading that is required to avoid discrete jumps in the output signal which can lead to instability when exchanging feedback-controllers. In the fading locations, the output of the previously active feedback controller is faded out, while the output of the new applied feedback controller is faded in. Just when switching to the

\(^3\)In terms of standard automata, a location is a discrete state.
location \textbf{Robust}, fading is not required, as the robust controller has been designed to be so robust that it still guarantees stability when these discrete jumps occur.

In Figure 3, \( t_c \) denotes a clock (a continuous variable with \( t_c = 1 \)) that is reset when firing transitions which lead to the fading locations in order to measure time and to guarantee that the fading location is left after the minimal and before the maximal allowed durations \( d_{\text{low}} \) and \( d_{\text{up}} \). Besides clock resets, the transitions in the example are associated with time guards and with discrete signals (events).

One limitation that can lead to inconsistent configurations is that hybrid automata have static interfaces \([12]\): In each location, the automaton requires all input signals, although they are not used in the according controllers. For example, the signal \( z_{\text{ref}} \) is not required in location \textbf{Robust}. As applying different controllers is motivated by the need to react to unavailable signals (e.g. due to sensor failures), the standard hybrid automata model should not be applied for our application example.

Therefore, we specify BC’s behavior by our notion of \textit{hybrid reconfiguration automata} \([12, 9]\) as depicted in Figure 3. In contrast to standard hybrid automata models (e.g. \([15, 1, 20]\)), the interface of the hybrid automaton is not the union of all input signals of the different controllers. Instead, each location has a different interface. For example, the \textbf{Robust} location has just one input, the \textbf{Reference} location has three inputs. Black ports indicate \textit{permanent} signals which are required in every location, white ports represent \textit{optional} signals.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure3.png}
\caption{Hybrid reconfiguration automaton}
\end{figure}
As mentioned above, the components (feedback-controller) of the fading locations (e.g. :RobAbs) run the controller, associated with the uniquely defined predecessor state and the controller, associated with the uniquely defined successor state, and a further part, that fades the outputs in and out. This leads usually to multiple specifications, implementation, and instantiations of each single component’s behavior which is not necessary. E.g. the robust-behavior is part of :Rob, :RobAbs, and :RobRef.

Figure 4: Structure of BodyControl (BC)

Figure 4 shows the internal structure of the BodyControl (BC) component. BC embeds the four instances rob, abs, ref, and fade of the components Rob, Abs, Ref, and Fade. Figure 4 further shows all possible connectors. Note that the figure depicts all connectors which can be established in any of the component’s states. Usually, they are not all established simultaneous, e.g. some connections which are established within location FadeAbsRef are not established in location Robust.

In a hybrid reconfiguration automaton with a dynamic internal structure that realizes a reconfigurable component, we allow to associate multiple components to each location (see Figure 5). Then, there is the need to specify the connectors between the components and the connectors which interconnect the ports of the superordinated (BC) component with the ports of the embedded components. These connections are depicted in Figure 5 as well. Note that the ports of the superordinated component change when the current location changes (see below). Further note that the component diagram from Figure 4 is like the superposition of the single locations from the hybrid reconfiguration automaton from Figure 5.
Figure 5: Hybrid reconfiguration automaton with dynamic internal structure
The example demonstrates that the fading locations lead to a high visual complexity. Therefore, we introduce several high-level constructs in hybrid reconfiguration charts (cf. Figure 6). We introduce a hierarchical extension called fading transition. Firing such a fading transition is not instantaneous, but it consumes times. The minimal and the maximal duration of the transition is specified by the intervals \( (d_i \text{ in Figure 6}) \). Further, the fading transitions are associated with a fading function, describing how the outputs are faded when firing the corresponding transition.

Figure 6: Hybrid reconfiguration chart with fading transitions

When a component is embedded within a complex context, just its interface and the external visible behavior are required for proper embedding – its internal realization is not important for that. Therefore, we provide the notion of interface automata and interface state charts (see Figure 7), describing the component’s dynamic interface and its external visible real-time behavior. The interface state chart just visualizes the component’s interfaces and under which real-time properties it is possible to switch between them. It abstracts from the internally applied components and from the fading functions. Especially when the component consists of multiple locations with the same interface, the interface state chart provides an abstraction hiding much of the component’s complexity.

The BC component is not a self-contained component as its external ports need to be connected to further component. Thus, BC is embedded in a specific context. The architecture is depicted in Figure 8. The Monitor component coordinates
its embedded components BC, Sensor, and Storage. Further, it communicates via the MonitorRegistration pattern [13] with the Registry. If Registry sends the information about the upcoming track section to Monitor, it stores it in the Storage component. Sensor provides the $\tilde{z}_{abs}$ signal.

Thus, the Monitor component has to fulfill multiple tasks: It has to communicate with the registry, it has to keep track of the available signals, and it has to coordinate its embedded components dependent on the available signals. The behavior is depicted in Figure 9. The lower orthogonal state realizes the communication with the registry. Its behavior is described in detail in [12]. The upper
orthogonal state consists of four states, representing if $z_{ref}$ and $\dot{z}_{abs}$ are available (AllAvailable), if none of them are available (NoneAvailable), or if exactly one input signal is available (AbsAvailable and RefAvailable).

![Hybrid reconfiguration chart](image)

**Figure 9**: Hybrid reconfiguration chart specifying the behavior of Monitor

Dependent on the current location (of the upper orthogonal state), the embedded components have to be reconfigured. Therefore, we propose an integration of the behaviors of Monitor and its subordinated components as depicted in Figure 10 and described in [12, 9].

Each state of Monitor is associated with a configuration of the subordinated components in Figure 10. When Monitor is for example in the location AbsAvailable, the subordinated components Sensor and BC must be connected as depicted in the according associated component instance diagram and BC must be in state Absolute and Sensor must be in state On. When Monitor changes the state for example to AllAvailable, this state change leads to a reconfiguration of the structure as specified by the according component instance diagram. Further, it implies a state change from Absolute to Reference. Thus, the Monitor component reconfig-
Figure 10: Hybrid reconfiguration chart with dynamic internal structure specifying the behavior of Monitor
ures its subordinated components and ensures thereby that the components are in an appropriate state and that all of their ports are connected.

**Correct Reconfiguration** When a component reconfigures its subordinated components, it must be ensured that the specified composition of high-level transitions is indeed correct. Therefore, for each atomic or fading transition, the corresponding transition of the interface automaton of the embedded subcomponents has to be triggered correctly. Otherwise, invalid location combinations cannot be excluded [8]. 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 reachability is undecidable [16], we cannot expect to find an automatic solution for 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 be checked for restricted cases on the syntactical level without consideration of the full state-space of the model. Often, we can check that the hybrid reconfiguration chart alone is an abstraction of the hybrid reconfiguration chart combined with the interface state charts of the subcomponents. 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 state charts to ensure the correct reconfiguration of the whole system.

Figure 11 depicts a part from the monitor behavior and the BC integration from Figure 10 and a part from the interface state chart of the BC component from Figure 7. As described above, 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, \( d_2 \subseteq d_4 \) must hold for the transition to AllAvailable/Reference. As sketched in Figure 11, we have to check that for each transition in the hybrid reconfiguration chart 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 state chart of each subcomponent exists (cf. Theorem 2 in Section 5.3).
Another proof becomes possible when we restrict the component behavior as well as the interface state charts 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 reconfiguration chart of the monitor.

4 Flat behavioral models

In the previous section, we presented our MECHATRONIC UML modeling approach to specify structure and behavior of reconfigurable mechatronic systems. In order to enable automatic code generation and to enable automatic analysis techniques, a formally defined semantics is required. In the remainder of this work, we define syntax and formal semantics of the introduced models.
In this section, the syntax and semantics of hybrid reconfiguration automata are specified. Therefore, we first define the syntax and semantics of a continuous model in Section 4.1. This is used to define hybrid automata (Section 4.2), which will be the base for the definition of hybrid reconfiguration automata (Section 4.3). The definitions are extensions of the ones from [12]. The applied formal mathematical definitions are defined in Appendix A.

4.1 Continuous Behavior

In the example from Section 3, the continuous feedback-controller components reference, absolute, and robust are used. Their behavior is usually specified by means of differential equations in form of block diagrams. Such block diagrams are formalized using the following concept of a continuous block.

4.1.1 Syntax

A continuous block provides a sufficient syntactical structure for the employed concept of differential equations. Its syntax is defined by Definition 1.

**Definition 1.** A continuous block \( M \) is described by a 7-tuple \((V^x, V^u, V^y, F, G, C, X^0)\) with \( V^x \) the state variables, \( V^u \) the input variables, and \( V^y \) the output variables. For the implicitly defined state flow variables \( V^a = V^y \cap V^u \), the set of equations \( F \subseteq \text{EQ}(V^x \cup V^a, V^x \cup V^u \cup V^a) \) describes the flow of the state variables, the set of equations \( G \subseteq \text{EQ}(V^y \cup V^a, V^x \cup V^a \cup V^u) \) determines the output variables, and \( X^0 \subseteq [V^x \rightarrow \mathbb{R}] \) the set of initial states. The invariant \( C \) with \( C \in \text{COND}(V^x) \) is further used to determine the set of valid states.

A block \( M \) is only well-formed when for the system of differential equations \( F \cup G \) holds that there are no cyclic dependencies, no double assignments, all undefined referenced variables are contained in \( V^u - V^y \), and a value is assigned to all state variables \( (V^x) \) and output variables \( (V^y) \).

To define a proper notion of refinement later in Definition 14 in Section 4.3.4, we denote dependencies between input and output variables using \( D(M) \subseteq V^u \times V^y \). The external visible dependencies \( D^v(M) \) are accordingly defined as \( D^v(M) := D(M) \cap ((V^u - V^y) \times (V^y - V^u)) \).

**Definition 2.** The interface \( I(M) \) of a continuous block \( M \) is defined as the external visible input and output variables \( (V^u - V^y, V^y - V^u) \).

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

Before defining the syntax and semantics of a hybrid reconfiguration automaton, like for example the one depicted in Figure 3, we define the syntax and semantics of a standard hybrid automaton with static interfaces similar to hybrid I/O automata [20].

4.2.1 Syntax

The syntax of a hybrid automaton is defined by Definition 4:

Definition 4. 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 \(D(l) = (V^x, V^u, V^y, F(l), G(l), C(l), X^0(l))\) (cf. 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 \(S^0 \subseteq \{(l, x) | l \in L \land x \in X^0(l)\}\). For any transition \((l, g, g', a, l') \in T\) holds that \(l \in L\) is the source-location, \(g \in \text{COND}(V^x \cup V^u)\) the continuous guard, \(g' \in \wp(I \cup O)\) the I/O-guard, \(a \in \wp[V^x \to \mathbb{R}]\) the continuous update, and \(l' \in L\) the target-location. For every \(l \in L\) we require that \(D(l)\) is well-formed.
Definition 5. The interface $I(M)$ of a hybrid automaton $M$ is defined as the external visible event sets and input and output variables $(I - O, O - I, V^u - V^y, V^y - V^u)$.

Note that the presented definition of a hybrid automaton easily permits to still employ the concepts of Real-Time Statecharts such as clocks. We simply have to define clock variables $v_i$ whose values are determined by equations $\dot{v}_i = 1$ in $F$ to encode this feature into a hybrid automaton.

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

Definition 6. For two hybrid automata $M_1$ and $M_2$ the parallel composition $(M_1 \parallel M_2)$ results in a hybrid automaton $M = (L, D, I, O, T, S^0)$ with $L = L_1 \times L_2$, $D(l, l') = D_1(l) \parallel D_2(l')$, $I = I_1 \cup I_2$, $O = O_1 \cup O_2$. The resulting transition relation is $T = \{(l_1, l_2) \in T_1 \times T_2, \exists l' \in T_1 \times T_2 : (l_1, l') \in T_1 \parallel T_2, (l_2, l') \in D(l_1, l')\}$. The composition of hybrid automata is only consistent when the resulting automaton is well-defined.

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

In order to abstract from signals –e.g. signals that are exchanged between automata– we define the hiding of signals as in Definition 7 which is taken from [8].

Definition 7. For a hybrid automaton $M = (L, D, I, O, T, S^0)$ the hiding of some signals $A \subseteq I \cup O$ denoted by $M \setminus A$ is defined as the hybrid automaton $M' = (L, D, I', O', T', S^0)$ with $I' = I - A$, $O' = O - A$, and $T' = \{(l, g, g', u, l) \in T\}$.

4.2.2 Semantics

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

When staying in state $(l, x)$, firing an instantaneous transition $t = (l', g, g', a, l'')$ is done iff

- the transitions source location equals the current location: $l = l'$,
4 FLAT BEHAVIORAL MODELS

- the continuous guard is fulfilled: \( g(x \otimes u) = \text{true} \) for \( u \in [V^u \rightarrow \mathbb{R}] \) the current input variable binding,

- the I/O-guard is true for the chosen input and output signal sets \( i \subseteq I \) and \( o \subseteq O: i \cup o = g^i \), and

- the continuous update still fulfills the invariant of the target location \( a(x) \in C(l'') \).

The resulting state will be \( (l'', a(x)) \) and we note this firing by \( (l, x) \rightarrow_{(i \cup o)} (l'', a(x)) \).

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

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

Other aspects of hybrid behavior, such as zeno behavior and the distinction between urgent and non-urgent transitions, are omitted here. A suitable formalization can be found, e.g., in [15].

4.3 Hybrid Reconfiguration Automata

In this section, syntax and semantics of hybrid reconfiguration automata, like the one from Figure 3 is defined formally.

4.3.1 Syntax

We define the syntax of hybrid reconfiguration charts as in Definition 8.

Definition 8. A hybrid reconfiguration automaton is described by a 6-tuple \( (L, D, I, O, T, S^0) \) with \( L \) a finite set of locations, \( D \) a function over \( L \) which assigns to each \( l \in L \) a continuous model, \( D(l) = (V^x(l), V^u(l), V^y(l), F(l), G(l), C(l), X^0(l)) \) conf. to Definition 1, \( I \) a finite set of input signals, \( O \) a finite set of output signals, \( T \) a finite set of transitions, and \( S^0 \subseteq \{(l, x) | l \in L \land x \in X(l)\} \) the set of initial states. For any transition \( (l, g^i, a, l') \in T \) holds that \( l \in L \) is the source-location, \( g \in \text{COND}(V^x(l) \cup V^u(l)) \) the continuous guard, \( g^i \in \varphi(I \cup O) \) the I/O-guard,
\( a \in \left[ [V^x(l) \to \mathbb{R}] \to [V^x(l') \to \mathbb{R}] \right] \) 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 \( V^v \) are derived analogously. We further use \( V^x(F(l)) \) to denote the state variable set. All assigned output variables are analogously named provided output variable set \( V^v(F(l) \cup G(l)) \) and all input variables employed are named required input variable set \( V^u(F(l) \cup G(l)) \).

**Definition 9.** The (static) interface \( I(M) \) of a hybrid reconfiguration automaton \( M \) is defined as the external visible event sets and input and output variables \( (I - O, O - I, V^u - V^v, V^v - V^u) \).

The parallel composition of two hybrid reconfiguration automata can be defined as follows:

**Definition 10.** For two hybrid reconfiguration automata \( M_1 \) and \( M_2 \) the parallel composition \( (M_1 \parallel M_2) \) results in a hybrid reconfiguration automaton \( M = (L, D, I, O, T, S^0) \) with \( L = L_1 \times L_2 \), \( D(l, l') = D_1(l) \parallel D_2(l') \), \( I = I_1 \cup I_2 \), \( O = O_1 \cup O_2 \). \( T = \{((l_1, l_2), g_1, g_2, (u_1 \oplus u_2), (l'_1, l'_2)) \mid (l_1, g_1, u_1, l'_1) \in T_1 \land (l_2, g_2, u_2, l'_2) \in T_2 \} \cup \{((l_1, l_2), g_1, u_1, (l_1, l'_2)) \mid (l_1, g_1, u_1, l'_1) \in T_1 \} \cup \{((l_1, l_2), g'_2, u_2, (l_1, l'_2)) \mid (l_2, g_2, u_2, l'_2) \in T_2 \} \) is the resulting transition relation where \( g(x, u, i, o) = g_1(x, u, i, o) \land g_2(x, u, i, o), g'_1(x, u, i, o) = g_1(x, u, i, o) \land o \cap I_2 = \emptyset \land i \cap O_2 = \emptyset, \) and \( g'_2(x, u, i, o) = g_2(x, u, i, o) \land o \cap I_1 = \emptyset \land i \cap O_1 = \emptyset \).

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

The parallel composition also follows directly from the non reconfigurable case. In the case of hybrid reconfiguration automata, a correct parallel composition has to ensure that for all reachable \((l, l') \in L \) holds that \( D((l, l')) \) does not contain cyclic dependencies.

**Additional Terms** Further, we define the terms fading location and regular location in Definition 11 and passive location in Definition 12 similar to the ones presented in [8]. The fading locations are inspired by the idea of [20] and represent fading transitions as time consuming intermediate states.

**Definition 11.** For a hybrid reconfiguration automaton \( M = (L, D, I, O, T, S^0) \) a location \( l_f \in L \) with \( D(l_f) = (V^x(l_f), V^u(l_f), V^v(l_f), F(l_f), G(l_f), C(l_f), X^0(l_f)) \) is a fading location iff
4 FLAT BEHAVIORAL MODELS

- the invariant consists of just one inequality with respect to the variable $v$ and an upper bound $d_{\text{max}}$: $C(l_f) \equiv (v \leq d_{\text{max}})$.

- $v$ is a clock: $\exists v \in V^x(l_f)$ with $(\dot{v} = 1) \in F(l_f)$.

- $v$ is reset to zero when entering $l_f$: for all $(l, g, g', a, l_f) \in T$ holds that $(v = 0) \in a$.

- exactly one transition exists leaving $l_f$: $|\{(l_f, g, g', a, l'_f) | (l_f, g, g', a, l'_f) \in T\}| = 1$, and

- for this transition holds $g \equiv d_{\text{min}} \leq v \leq d_{\text{max}}, g' = \text{true}$, and $a = \text{Id}$.

All non fading locations are regular locations.

**Definition 12.** For a hybrid reconfiguration 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.

### 4.3.2 Semantics

The semantics can be derived from the hybrid automata semantics (see Section 4.2.2) by always taking into account the location dependent notion $V^x(l)$ etc. instead of the location independent $V^x$.

### 4.3.3 Interface Automata

While the previously outlined concept of the hybrid automaton (see Section 4.2) permits to model and formalize static structures with static interfaces that describe hybrid behavior, advanced mechatronic systems in contrast integrate complex reconfiguration scenarios which cannot be modeled in an appropriate manner with static interfaces (cf. [12]). If we thus consider dynamic interfaces, we also have to address the resulting problems that receiving and sending of signals as well as reading or writing the continuous variables has to be done in a coordinated fashion as otherwise a consistent behavior cannot be guaranteed. As reading from a currently undefined continuous out variable results in unpredictable and therefore unsafe control behavior, this kind of inconsistency must be excluded for mechatronic systems.

As described in Section 3, we face the problem by the introduction of interface automata. The externally relevant behavior, covered by the interface automaton, only includes the real-time behavior as well as the state-dependent continuous interface. Therefore, the notion of an interface automaton is restricted as follows:
4.3 Hybrid Reconfiguration Automata

**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 $(l, g, g', a, l')$ 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 they are only used to abstract from the evaluation dependencies.

Such an interface automaton can then be employed to enable a safe, strict hierarchical aggregation of sub-components by a super-component (see Figure 10) by providing a more suitable notion of interface than the static interface of a component as defined in Definition 15.

### 4.3.4 Refinement and Abstraction

To study what a correct relation between the realization of a component and its interface automaton is, we write for a possible execution sequence of states and transitions of a hybrid automaton $M = (L, D, I, O, T, S^0)$ with $(u_0, l_0, \rho^0_{u_0}, \theta^0_{u_0}, \delta_0) \rightarrow e_0 (u_1, l_1, \rho^1_{u_1}, \theta^1_{u_1}, \delta_1) \in [M]_t$ 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 $(l_0, \rho^0_{u_0}(0)) \rightarrow_{e} (l_n, \rho^n_{u_n}(\delta_n))$ 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 $\theta^i_{u_i} = \theta^i_{u_i}|_{V^n(l_i) = V^n(x_i)}$ 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 we write $(l_0, \rho^0_{u_0}(0)) \Rightarrow_{e'} (l_n, \rho^n_{u_n}(\delta_n))$. 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 offer($M, (l, x)$) which is defined as $\{e \exists (l, x) \Rightarrow_e (l', x)\} \cup \{(du/dt)(0)\exists (l, x) \Rightarrow (u, \delta_{u_i}) (l, x')\}$.

An appropriate notion of hybrid refinement for the interface is then defined as follows:
Definition 14. For two hybrid reconfiguration automata \( M_I \) and \( M_R \) holds that \( M_R \) is a refinement of \( M_I \) denoted by \( M_R \subseteq M_I \) 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

\[
\forall c \Rightarrow \exists c' \exists c'' : (c', c'') \in \Omega \quad \text{and} \quad \text{(1)}
\]

\[
\text{offer}(M_R, c) \supseteq \text{offer}(M_I, c') \quad \text{and} \quad \text{(2)}
\]

\[
\forall((l_R, x_R), (l_I, x_I)) \in \Omega : D^e(D_R(l_R)) \subseteq D^e(D_L(l_I)). \quad \text{(3)}
\]

Refinement is transitive which is expressed by Theorem 1.

Theorem 1. For three hybrid reconfiguration automata \( M_A, M_B \) and \( M_C \) holds

\[
M_C \subseteq M_B \cap M_B \subseteq M_A \Rightarrow M_C \subseteq M_A. \quad \text{(4)}
\]

Proof. As \( M_C \subseteq M_B \) and \( M_B \subseteq M_A \), there exist \( \Omega_{CB} \subseteq (L_C \times X_C) \times (L_B \times X_B) \) and \( \Omega_{BA} \subseteq (L_B \times X_B) \times (L_A \times X_A) \) cf. Definition 14. We chose \( \Omega_{CA} = \{(c, c'') | (c, \bar{c}) \in \Omega_{CB} \wedge \exists (\bar{c}, c'') \in \Omega_{BA}\} \). The requirement that for every \( c \) an entry \( c'' \) exists in \( \Omega_{CA} \) is fulfilled, as an entry \( \bar{c} \) exists for every \( c \) in \( \Omega_{CB} \) and an according entry \( c'' \) exists for every \( \bar{c} \) in \( \Omega_{BA} \).

- Equation 1 follows directly from the construction of \( \Omega_{CA} \).

- Due to Equation 2 from Definition 14, it holds for all \( c, \bar{c}, \) and \( c'' \) with \((c, \bar{c}) \in \Omega_{CB} \) and \((\bar{c}, c'') \in \Omega_{BA}\): \( \text{offer}(M_C, c) \supseteq \text{offer}(M_B, \bar{c}) \) and \( \text{offer}(M_B, \bar{c}) \supseteq \text{offer}(M_A, c'') \). Thus, \( \text{offer}(M_C, c) \supseteq \text{offer}(M_A, c'') \) is fulfilled.

- Due to Equation 3 from Definition 14, it holds for all \((l_C, x_C), (l_B, x_B), \) and \((l_A, x_A)\) with \((l_C, x_C), (l_B, x_B) \in \Omega_{CB} \) and \((l_B, x_B), (l_A, x_A) \in \Omega_{BA}\): \( D^e(D_C(l_C)) \subseteq D^e(D_B(l_B)) \) and \( D^e(D_B(l_B)) \subseteq D^e(D_A(l_A)) \). Thus, \( D^e(D_C(l_C)) \subseteq D^e(D_A(l_A)) \) is fulfilled.

\[
\square
\]

5 The Component Model

The structure of a complex mechatronic system, as explained in Section 3, is described by UML components. Components are a specialization of UML encapsulated classifiers, which correspond to UML/RT capsules [29] and the ROOM
5.1 Static Component Structures

Components interact with their environment only via one or more signal-based boundary objects called ports. Each port plays a particular role in a collaboration that the component has within its context. Additionally, UML connectors, which correspond to UML/RT connectors and ROOM bindings, are signal-based communication channels that interconnect multiple ports. Their behavior, i.e. how they realize such interactions is described by behavioral models, as described in Section 4.

5.1 Static Component Structures

To model mechatronic systems, we have to extend the UML component model to also support the description of quasi-continuous control behavior. Our straightforward approach to this is to add besides signals also quasi-continuous variables to ports of our MECHATRONIC UML components. While a signal is sent and received at a discrete point in time (cf. SignalEvent in UML), we assume that a quasi-continuous variable has a well-defined value for each point in time.

5.1.1 Syntax

To formalize such a structural description of a MECHATRONIC UML component diagram, we omit that a UML port can have multiple attached UML interfaces denoting the syntactical interface and instead employ only an unstructured set of signals.

**Ports**  A discrete port type $P_d$ is therefore defined by a pair $(I, O)$ with possible incoming signals $I$ and outgoing signals $O$. For a continuous port type $P_c$, we use instead a pair $(V^u, V^y)$ with $V^u$ the input variables and $V^y$ the output variables. Combining both definitions, a hybrid port type $P_h$ is a 4-tuple $(I, O, V^u, V^y)$ which includes both signals as well as variables. For sake of simplicity, we simply use empty signal or variable sets for continuous resp. discrete port types in the following.

In addition, for each port type $P = (I, O, V^u, V^y)$ the inverse port type $\overline{P} := (O, I, V^y, V^u)$ is defined by simply exchanging the in and out signal sets as well as the in and out variable sets. A port declaration is then a pair $(N_i, P_i)$ with $N_i$ a unique port name of the port name set $N_{po}$ and $P_i$ a discrete, continuous, or hybrid

---

4Usually, quasi-continuous variables also may have a quasi-continuous value domain, however, this might not always be the case (e.g., quasi-continuous variables may also be employed to encode boolean flags etc.).
port type. We further refer to a port type $P_i$ of $N_i$ simply using $\text{type}(N_i)$ and refer to the sets $I_i$, $O_i$, $V^a_i$ and $V^y_i$ resp. using $I(N_i)$ etc. In the example of Figure 8, the annotation $\text{mr:MonitorRole}$ declares that a port with name mr and port type MonitorRole exists.

Components The interface of a component is defined by the boundary ports, which interact with the environment. In the example of Figure 8, the position of the port with the annotation $\text{mr:MonitorRole}$ identifies it as a boundary port which is thus part of the component interface. We thus define the static interface of a component simply as a set of port declarations as follows:

Definition 15. The static interface $I$ of a component is a set of port declarations $\{(N_1, P_1), \ldots, (N_n, P_n)\}$ with for all $1 \leq i \leq n$ and $1 \leq j \leq n$ with $i \neq j$ holds $N_i \neq N_j$.

Besides the external connections via ports, a component may also embed other components for which an external interface must be known. Thus, a component includes besides its external port sets also a set $E$ of occurrences of embedded components which is each represented by a pair $(N_j, I_j)$ with $N_j$ the unique occurrence name of the occurrence name set $N_o$ and $I_j$ the interface of that component. To denote a port with port name $N_i$ within the interface of an occurrence with unique name $N_j$, we further use $\text{type}(N_j, N_i)$ and refer to the sets $I_i$, $O_i$, $V^a_i$ and $V^y_i$ resp. using $I(N_j, N_i)$ etc. The sensor:Sensor component with occurrence name sensor and component interface Sensor, which is embedded into the Monitor component as depicted in Figure 8, is an example for such an embedding.

Finally, also the internal behavior of the component may send or receive a given set of signals as well as read or write continuous variables. These signals and variables are defined by the special internal interface $B$.

To describe the mapping between external ports, the ports of embedded components, and ports of the internal behavior as depicted in Figure 4 by means of connector links, a set of links $\text{map} \subseteq (N_{po} \cup (N_o, N_{po})) \times (N_{po} \cup (N_o, N_{po}))$ is used.

Using the above introduced concepts, a static internal component structure can be defined as follows:

Definition 16. The static internal structure or configuration of a component is a tuple $(I, B, E, \text{map})$ with $I$ the component’s static interface, $B$ the interface of the internal behavior, $E = \{(N_1, I_1), \ldots, (N_n, I_n)\}$ the set of embedded occurrences consisting of a unique occurrence name $N_i$ from the occurrence name set $N_o$ and the occurrence’s interface $I_i$, and $\text{map}$ a mapping to describe the connectors between ports of these different interfaces. For all $1 \leq i \leq n$ and $1 \leq j \leq n$ with $i \neq j$ must hold $N_i \neq N_j$. 

22
5.1 Static Component Structures

In some cases, $I$ is equal to $B$, but when the realizations of the embedded occurrences consume signals from $B$, these signals do not appear in $I$. We define consistency for static internal structures as follows:

**Definition 17.** A static internal structure as introduced in Definition 16 is consistent iff

1. map is symmetric ($(a, b) \in \text{map} \Rightarrow (b, a) \in \text{map}$),
2. The mapping map is unique ($(a, b) \in \text{map} \Rightarrow \nexists (a, c) \in \text{map}: c \neq b$),
3. for all $(p, q) \in \text{map}$ holds that $p$ and $q$ refer either to ports of the external port set $I$, ports of the internal behavior $B$, or ports of embedded occurrences,
4. for all $(p, q) \in \text{map}$ holds that $\text{type}(p) = \overline{\text{type}(q)}$, and
5. any ports of the external port set $I$, the internal behavior $B$, or an embedded occurrence is mapped by map to exactly one counterpart.

Note, that item 5 seems to be contradictory to Figure 4, but recall that the BodyControl component from this figure is not described by a static component structure. Variable component structures are described in Section 5.2.

**Behavior** Of course, this description of the structure does not define any behavior formally and precisely. Therefore, we have to integrate the structural description, defined by Definition 16, with the behavioral models from Section 4. Thus, we define the relation between the interface of a behavioral model and a static internal structure:

**Definition 18.** A continuous block $M$ with interface $I(M) = (V^u - V^y, V^y - V^u)$ realizes a given port set $L = \{(N_0, (V^u_0, V^y_0)), \ldots, (N_n, (V^u_n, V^y_n))\}$ of continuous ports iff

- $V^u - V^y = \bigcup_{i=1}^{n} \{N_i.v | v \in V^u_i\} \text{ and }$
- $V^y - V^u = \bigcup_{i=1}^{n} \{N_i.v | v \in V^y_i\}$.

We write $M \vdash L$.

**Definition 19.** A hybrid automaton $M$ with interface $I(M) = (I - O, O - I, V^u - V^y, V^y - V^u)$ realizes a given port set of hybrid ports $L = \{(N_0, (I_0, O_0, V^u_0, V^y_0)), \ldots, (N_n, (I_n, O_n, V^u_n, V^y_n))\}$ iff

- $I - O = \bigcup_{i=1}^{n} \{N_i.a | a \in I_i\}$,
5 THE COMPONENT MODEL

- \( O - I = \bigcup_{i=1}^{n} \{ N_i.a | a \in O_i \} \),
- \( V^u - V^y = \bigcup_{i=1}^{n} \{ N_i.v | v \in V^u_i \}, \text{ and} \)
- \( V^y - V^u = \bigcup_{i=1}^{n} \{ N_i.v | v \in V^y_i \} \).

We write \( M \vdash L \).

Using the introduced concepts for the component interface, internal component structure, and behavior, we can now define what a (hybrid) component looks like.

**Definition 20.** The component realization \( C = (S, M, P, \text{prop}) \) consists of a component structure \( S = (I, B, E, \text{map}) \), an internal behavior \( M = (m_1, \ldots, m_n) \), a set of properties \( P \), and a function \( \text{prop} : \{ M \} \cup (\bigcup_{i=1..n} 2^{m_i}) \rightarrow P \) that assigns a property to the behavior or to some of the behavior’s syntactical elements. It must hold that \( M \vdash B \).

### 5.1.2 Semantics

The semantics of a component realization \( C = (S, M, P, \text{prop}) \) embedding occurrences \( E = \{(N_1, I_1), \ldots, (N_n, I_n)\} \) as defined in Definition 20 is not just defined by the parallel composition of \( M \) with the automata or blocks which realize the embedded occurrences, i.e. the semantics is not just defined as \( M || (M_1 || \ldots || M_n) \) with \( M_i \vdash I_i \) for \( i \in \{1, n\} \). Additionally, the structural connections encoded with the relation \( \text{map} \) have to be reflected in the behavioral model. Therefore, a hybrid automaton behavior \((S, M, P, \text{prop})\) realizing the forwarding behavior implicitly specified within the relation \( \text{map} \) has to be constructed as follows: Let \( S = (I, B, E, \text{map}) \) and \( M = (L, D, I, O, T, S^0) \), with \( D(l) = (V^x, V^u, V^y, F(l), G(l), C(l), X^0(l)) \). behavior\((S, M, P, \text{prop})\) = \( M' = (L, D', I', O', T', S^0) \) with \( D'(l) = (V^x, V^u', V^y', F(l), G'(l), C(l), X^0(l)) \) is defined as follows:

- The set \( L \) of locations and the set of initial states \( S^0 \) is in \( M \) like in \( M' \),
- the set \( V^x \) of continuous state variables, their flow \( F(l) \), the locations’ conditions \( C(l) \) and the initial continuous states \( X^0(l) \) is in \( D \) like in \( D' \),
- the source signals of the connectors become input signals: \( I' = I \cup \bigcup_{(N,N') \in \text{map}} I(N) \),
- the target signals of the connectors become output signals: \( O' = O \cup \bigcup_{(N,N') \in \text{map}} O(N) \),
5.2 Variable Component Structures & Reconfiguration

- additional transitions, each with the same source and target locations, map in every location the source signals of the connectors to its target signals: 
  \[ T' = T \cup \{(l, true, \{N.a, N'.a\}, id, l)|l \in L \land (N, N') \in map \land a \in I(N)\}. \]

- the source variables of the connectors become input variables: 
  \[ V'u = Vu \cup \bigcup_{(N,N') \in map} Vu(N) \]

- the target variables of the connectors become output variables: 
  \[ V'y = Vy \cup \bigcup_{(N,N') \in map} V'y(N) \]

- the connectors are implemented by equations, mapping the source variables to target variables: 
  \[ G'(l) = G(l) \cup \{N.v := N'.v|(N, N') \in map, v \in Vu(N)\}. \]

The behavior of a component realization \( C = (S, M, P, prop) \) with component structure \( S = (I, B, E, map) \) is then given by \( behavior((S, M, P, prop)) \). The behavior of a component instance with instance name \( N \in N_o \) and component behavior \( M \) is further given by \( M|_{+N} \) where \( x|_{+N} \) denotes that all signal and variables of \( x \) are accordingly extended by the prefix \( N \).

5.2 Variable Component Structures & Reconfiguration

The described notion of a component realization with a static internal structure and the definition of its semantics must be extended when the internal structure is variable as in the case of hybrid reconfiguration automata or hybrid reconfiguration charts. Recall that for example the BodyControl component changes its structure dependent on its current location (cf. Figure 5).

5.2.1 Syntax

To ensure that a hybrid component such as the Monitor component depicted in Figure 8 is consistent (cf. Definition 17), the BodyControl component BC would have to ensure that in all internal modes (see Figure 7) the same set of continuous variables is required and offered. However, as depicted in Figure 7, the required inputs are different.

To support the correct design of complex mechatronic systems despite the fact that continuous variables may only be available in a state-dependent manner, we further extend the introduced notion of an interface of a component for embedding

\[ \text{Note that due to item 4 of Definition 17, it holds that } I(N) = O(N'). \]

\[ \text{Note that due to item 4 of Definition 17, it holds that } Vu(N) = V'u(N'). \]
by means of state information to support a safe, modular reconfiguration for the hierarchical embedding of reconfigurable hybrid behavior:

**Definition 21.** A dynamic interface of a component $C$ is a tuple $(M, I, P, \text{prop})$ with $M = (L, D, I, O, T, S^0)$ an interface automaton, $I$ the static interface of the component $C$, $P$ a set of properties, and $\text{prop} : L \rightarrow P$ a function that assigns properties to the interface automaton’s locations. Any $l \in L$ is a valid mode.

Using the notion of dynamic interfaces, we specify a variable structure within a component, as shown in Figures 5 and 10, formally as follows:

**Definition 22.** A dynamic internal structure of a component $C$ with mode set $L$ can be described using a function $S$ which assigns to each mode $l \in L$ an internal structure $S(l) = (I(l), B(l), E(l), \text{map}(l))$, where each $(N, (M, I, P, \text{prop}), l') \in E(l)$ in contrast to the static case contains a dynamic interface (cf. Definition 21) and a required valid mode of that dynamic interface.

Informally, $C$ shows in each of its locations a different (external) interface $I(l)$. $C$’s internal behavior shows also different interfaces, denoted by $B(l)$. Further, multiple component occurrences are embedded into each location. Each of these component realizations is a refinement of its dynamic interface $(M, I, P, \text{prop})$. The valid mode $l'$ of the dynamic interface specifies that the according embedded component is required to be in a location that delivers the interface that is described by $l'$. All input and output signals and all in and out variables of the embedded components and of $C$’s interface $I(l)$ are interconnected by the connectors specified by $\text{map}(l)$.

Such a dynamic internal structure $S$ is structural consistent iff for all modes $l \in L$ holds that $S(l)$ interpreted as a static structure is consistent.

To describe that continuous ports are only available dependent on the current state of a component, we distinguish between permanent ports and optional ports. While the former ones are visualized by white triangle as depicted in Figure 7, for the later ones black triangle are used (cf. Figure 7) to highlight the difference also within component diagrams. For $I^\cup = \bigcup_{l \in L} I_l$ and $I^\cap = \bigcap_{l \in L} I_l$ holds that any $(N, P) \in I^\cap$ is a permanent port, while any $(N, P) \in I^\cup \setminus I^\cap$ is an optional port.

Before defining a reconfigurable component realization, we integrate again the structural definition with the behavioral model. In the reconfigurable case, we integrate the structural description, defined by Definition 22, with the behavioral model of the hybrid reconfiguration chart from Section 4.3:

**Definition 23.** A hybrid reconfiguration automaton $M$ realizes a given interface in form of a set of hybrid port declarations like in the case of a hybrid automaton. If $M$ in mode $l \in L$ realizes a static interface $I(l)$, we write $M, l \vdash I(l)$. If $M$ in addition refines an interface automaton $M^I$ ($M \sqsubseteq M^I$; see Definition 14), it also realizes the dynamic interface $(M^I, I)$ and we write $M \vdash (M^I, I)$. 26
Using the introduced concepts, we define what a (hybrid) component in the case of reconfiguration looks like:

**Definition 24.** The reconfigurable (hybrid) component \( C = (S, M, P, \text{prop}) \) consists of an internal behavior \( M = (L, D, I, O, T, S^0) \), a dynamic component structure \( S \) for \( L \), a set \( P \) of properties, and a function \( \text{prop} : L \rightarrow P \) that assigns properties to the internal behavior’s locations. It must hold that for all \( l \in L \) and \( S(l) = (I(l), B(l), E(l), \text{map}(l)) \) we have \( M, l \vdash B(l) \).

### 5.2 Semantics

An internal behavior \( M \) and a *dynamic internal structure* \( S \) are *behavioral consistent* iff for all \( (l, g, g', a, l') \in T \) and all embedded dynamic interfaces in \( (N, (M, I, P, \text{prop}), l'') \in E(l) \) and \( (N, (M, I, P, \text{prop}), l''') \in E(l') \) hold that a corresponding transition between the required modes \( l'' \) and \( l''' \) exists. A function \( t : L \times L \rightarrow I \cup O \) is assumed which denotes all signals that must be emitted to enforce the mode changes of the embedded interface automata.

We define the semantics of a reconfigurable component by extending its internal behavior, described by a behavioral consistent hybrid reconfiguration automaton \( M = (L, D, I, O, T, S^0) \), with the behavior of the associated component reconfiguration and with the *dynamic internal structure* \( S : L \rightarrow \mathcal{S} \). The former extension is realized similar to the approach from Section 5.1.2 for the static case, the latter one by referring to the above introduced trigger function \( t \):

Let \( S(l) = (I(l), B(l), E(l), \text{map}(l)) \) and \( M = (L, D, I, O, T, S^0) \) with the continuous model \( D(l) = (V^x(I(l)), V^y(l), V^y(l), F(l), G(l), C(l), X^0(l)) \). \( \text{behavior}(S, M, P, \text{prop}) = M' = (L, D', I', O', T', S^0) \) with the continuous model \( D'(l) = (V^x(l), V'^x(l), V'^y(l), F(l), G'(l), C(l), X^0'(l)) \) is defined as follows:

- The set \( L \) of locations and the set of initial states \( S^0 \) is in \( M \) like in \( M' \),
- the set \( V^x \) of continuous state variables, their flow \( F(l) \), the locations’ conditions \( C(l) \) and the initial continuous states \( X^0(l) \) is in \( D \) like in \( D' \),
- the source signals of the connectors become input signals: \( I' = I \cup \bigcup_{l \in L, (N, N') \in \text{map}(l)} I(N) \),
- the target signals of the connectors become output signals: \( O' = O \cup \bigcup_{l \in L, (N, N') \in \text{map}(l)} O(N) \),
- transitions emit additional signals to enforce state changes as denoted by \( t \). Additional transitions, each with the same source and target locations, map in every location the source signals of the connectors to
its target signals: \[ T = \{ (l, g, g', t(l, l'), a, l') \mid (l, g, g', a, t(l, l')) \in T \} \cup \{ (l, true, \{ N.a, N'.a \}, id, l) \mid l \in L \land (N, N') \in map \land a \in I(N) \}. \]

- the source variables of the connectors become input variables: \( V_u'(l) = \bigcup \{ (N, N') \in map(l) \mid V_u(N) \}, \)
- the target variables of the connectors become output variables: \( V_y'(l) = \bigcup \{ (N, N') \in map(l) \mid V_y(N) \}, \) and
- the connectors \( map(l) \) of each location \( l \) are implemented by equations, mapping the source variables to target variables: \( G'(l) = G(l) \cup \{ N.v := N'.v \mid (N, N') \in map(l), v \in V_u(N) \}. \)

The behavior of a reconfigurable component realization \( C = (S, M, P, prop) \) with component structure \( S \) is then given by \( \text{behavior}(C \parallel M|_N) \). The component behavior of a component instance with instance name \( N \in N_o \) and behavior \( M \) of the component realization is then given by \( M|_{+N} \).

### 5.3 Syntactical Refinement

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 in many cases on the syntactical level without consideration of the full state-space of the model (cf. Figure 11). In these checks fading transitions and their durations play an important role. Formalizing their semantics leads to simple interface automata \([8]\).

**Definition 25.** 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.

In order to apply Theorem 2 from \([8]\) (see Theorem 2 below), we define the set \( H \) of possibly reachable state combinations of a reconfigurable hybrid component and its embedded occurrences as follows:

**Definition 26.** Let \( C = (S, M, P, prop) \) be a reconfigurable hybrid component with \( M = (L, D, I, O, T, S_0) \), \( S(l) = (I(l), B(l), E(l), map(l)) \), and \( E(l) = \{ (N_1, (M_1, I_1, P_1, prop_1), t_1), \ldots, (N_n, (M_n, I_n, P_n, prop_n), l_n) \} \). The set of possibly reachable states is \( H = \{ (l, (l_1, \ldots, l_n)) \mid l \in L \} \). We call \( M \parallel_H (M_1 \parallel \ldots \parallel M_n) := \text{behavior}(C \parallel (M_1 \parallel \ldots \parallel M_n)) \) the hierarchical parallel composition of \( M \) and \( M_1 \parallel \ldots \parallel M_n \).

\(^7\)Note that due to item 4 of Definition 17, it holds that \( I(N) = O(N') \).
The following theorem (taken from [8]) describes a simple syntactical rule which is sufficient to prove for the restricted case sketched above that a hierarchical parallel product does not have any timing errors. The basic idea is that the timing interval of a hybrid reconfiguration chart’s fading transitions has to conform with the one of its embedded simple interface state charts, as described in Section 3.

**Theorem 2.** For the hierarchical parallel composition $M_1 \parallel_H M_2$ of two hybrid automata $M_1$ and $M_2$ holds $M_1 \parallel_H M_2 \subseteq M_1 \setminus I_2 \cup O_2$ if

1. $I(M_1 \parallel_H M_2) = I(M_1 \setminus I_2 \cup O_2)$,

2. all initial states are also contained in $H$: $(\{(l_1, l_2)\} | (l_1, x) \in S_1^0 \land (l_2, y) \in S_2^0) \subseteq H$,

3. $M_2$ is a simple interface state chart (cf. Definition 25), and

4. for all $(l_1, l_2) \in H$ and transition $t_1 = (l_1, g_1', a_1, l_1') \in T_1$ holds:

   (a) if $l_1'$ is not a fading location, then for all $t_2 = (l_2, g_2, g_2', a_2, l_2') \in T_2$ with $g_1' \cap (I_2 \cup O_2) = g_2'$ must hold:

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

      ii. $l_2'$ is a passive location (cf. Definition 12), 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 $t_1 = (l_1', g_1', g_1', a_1', l_1'' \in T_1$ with $g_1' \cap (I_2 \cup O_2) = g_1'$ exists (see Definition 11). For any $t_2 = (l_2, g_2, g_2', a_2, l_2') \in T_2$ with $g_1' \cap (I_2 \cup O_2) = g_2'$ must hold:

      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 $t_2' = (l_2', g_2', g_2', a_2', l_2'' \in T_2$ with $g_2' = d_{\text{min}}^2 \leq v \leq d_{\text{max}}^2$ must hold:

      iv. $l_2''$ is a passive location (cf. Definition 12),

      v. $(l_1', l_2'') \in H$, and

      vi. $[d_{\text{min}}^2, d_{\text{max}}^2] \subseteq [d_{\text{min}}^1, d_{\text{max}}^1]$ must be satisfied.

   Again, at least one such pair of transitions in $M_2$ must exist.
5 THE COMPONENT MODEL

Proof. Theorem 2 has been proved in [8]. There, we first chose an appropriate \( \Omega \) conform to Definition 14, so that every state \((l, x)\) of \( M_1 \) is related to an corresponding state of \( M_1\|H M_2 \). Then, we will show that for every transition \( t_l \) of \( M_1 \) an according transition \( t_R \) of \( M_1\|H M_2 \) exists and that \( t_R \) fires when \( t_l \) 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 14 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 \( \Omega = \{((l_1, l_2), (l_1, \tilde{x}_R)) | (l_1, l_2) \in H, x_R \in X_R \} \).

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: \((l_1, x_1) \rightarrow g_1^i ((l_1', a_1(x_1)) \in [M_1]_t \). 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 6 \( M_1\|H M_2 \) contains \( t_1 \) a transition

\[
t = ((l_1, l_2), g_1, g_1^i, (a_1 \oplus a_2), (l_1', l_2'))
\]

with \( g_1^i \cap (I_2 \cup O_2) = g_2^i \). As \( g_1 \land g_2 \stackrel{\text{def}i}{=} g_1 \land \text{true} = g_1 \) and as \( g_1^i \land g_2^i \stackrel{\text{def}a}{=} g_1^i \land g_2^i \land (g_1 \cap (I_2 \cup O_2)) = g_1^i \), \( t \) is equal to

\[
t = ((l_1, l_2), g_1, g_1^i, (a_1 \oplus a_2), (l_1', l_2'))
\]

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 \( M_1\|H M_2 \): \((l_1, l_2), x_1) \rightarrow g_1^i ((l_1', l_2'), (a_1 \oplus a_2)(x_1)) \in [M_1\|H M_2]_t \) 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 \( c' := (l_1', l_2', \tilde{x}_R) \in (L_1 \times L_2) \times X_R \), and \( c'' := (l_1', \tilde{x}_R') \), holds: There exists the execution sequence \( c \rightarrow g_1^i c' \) 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 14.

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 \( l_1 \), residing in \( l_1' \) for a specific time \( \delta_0 \), and leaving \( l_1' \) after \( \delta_0 \):

\[
(l_1, x_1) \rightarrow g_1^i ((l_1', \rho_0(0)) \rightarrow ((w_0, \rho_0, \rho_0(\delta_0)) \rightarrow (l_1', \rho_0(\delta_0)) \rightarrow \theta (l_1, a_1' \rho_0(\delta_0))) \in [M_1]_t
\]
with $\delta_0 \in [d^1_{\text{min}}, d^2_{\text{max}}]$. Due to requirement (4b) of this theorem there exists at least one transition

$$t_2 = (l_2, g_2, g_2^i, a_2, l_2') \in T_2$$

(8)

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 11):

$$t_2' = (l_2', g_2', g_2^i, a_2', l_2'' \in T_2$$

(9)

with $g_2' \equiv d_2^{\text{max}} \leq v \leq d_2^{\text{max}}$, $g_2' = \emptyset$, and $(l_1'', l_2'') \in H$. Due to Definition 6

$M_1 \!\!|^H M_2$ contains a

$$t = ((l_1, l_2), g_1 \land g_2, g_1^i \cup g_2^i, (a_1 \oplus a_2), (l_1', l_2'))$$

(10)

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

$$t' = ((l_1', l_2'), g_1^i \land g_2^i, g_1^i \cup g_2^i, (a_1 \oplus a_2), (l_1'', l_2''))$$

(11)

with $g_1'^i \cap (I_2 \cup O_2) = g_1^i$. As $g_1 \land g_2 (4bi)$ $g_1 \land \text{true} = g_1$ and as $g_1^i \cup g_2^i (4bi)$

$g_1^i \cup (g_1^i \cap (I_2 \cup O_2)) = g_1^i$, $t$ is equal to

$$t = ((l_1, l_2), g_1, (a_1 \oplus a_2), (l_1', l_2'))$$

(12)

Note that firing again just depends on the triggers $g_1$ and $g_2^i$ and not on $g_2$ or $g_2'$. Due to requirement (4b) holds: $g_1^i \land g_2^i \equiv (d_1^{\text{min}} \leq d_1^{\text{max}}) \land (d_2^{\text{min}} \leq d_2^{\text{max}})$. As $[d_1^{\text{min}}, d_1^{\text{max}}] \subseteq [d_1^{\text{min}}, d_1^{\text{max}}]$ (cf. requirement (4b vii), it follows that

$$g_1^i \land g_2^i \equiv d_1^{\text{min}} \leq v \leq d_1^{\text{max}}.$$ 

(13)

As further $g_1^i \cup g_2^i (4bi) \emptyset \cup \emptyset = \emptyset$, $t'$ is equal to

$$t' = ((l_1', l_2'), d_2^{\text{min}} \leq v \leq d_2^{\text{max}}, \emptyset, (a_1' \oplus a_2'), (l_1'', l_2''))$$

(14)

Note, that firing of $t'$ just depends on $d_2^{\text{min}} \leq v \leq d_2^{\text{max}}$. $t'$ fires after the time

$\delta_0 \in [d_2^{\text{min}}, d_2^{\text{max}}]$.

Thus, for all $(l_1, l_2) \in H$ holds that every path in $[M_1 \!\!|^H M_2]$, corresponds to one of $[M_1]$, like the one from equation (7): $((l_1, l_2), x) \rightarrow g_1^i (((l_1', l_2'), (a_1 \oplus a_2)(x_R)) \rightarrow (\omega_0, \rho_0^a, \delta_0)) ((l_1', l_2'), (\rho^0(a_1 \oplus a_2)(x_R))) \rightarrow (l_1'', l_2''), (\rho^0(a_1 \oplus a_2)(x_R))) \in [M_1 \!\!|^H M_2]$. Further it holds: $(l_1', l_2') \in H$ and $(l_1'', l_2'') \in H$.

Therefore, for each $c := ((l_1, l_2), x) \in (L_1 \times L_2) \times X$ there exists a $c'' := (l_1, \tilde{x}_R)$ so that for $c := ((l_1', l_2'), x') \in (L_1 \times L_2) \times X$, and $c'' := ((l_1', \tilde{x}_R)$, holds: There exists the execution sequences $c \Rightarrow c'$ with $\pi = g_1^i; (u_0, \theta^0_u, \delta_0); \emptyset$ and $c'' \Rightarrow c''$ with $(c, c'') \in \Omega$ and $(c', c'') \in \Omega$, which is again the above mentioned requirement for refinement, cf. Definition 14.
Case 3: Continuous transitions  Let \((l_1, \rho_{u_0}^0(0)) \rightarrow (u_0, \rho_{w_0}^0, \theta_{w_0}, \delta_0) (l_1, \rho_{u_0}^0(\delta_0))\) be a path from the semantics \([M_1]\). For all \((l_1, l_2) \in H\) exists an according path in the semantics \([M_1] \rightarrow [M_2]: ((l_1, l_2), \rho_{u_0}^0(0)) \rightarrow (u_0, \rho_{w_0}^0, \theta_{w_0}, \delta_0) ((l_1, l_2), \rho_{u_0}^0(\delta_0))\).

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 \(\rho'' := (l_1, \rho_{u_0}^0(\delta))\) so that for \(\rho' := ((l_1, l_2), \rho_{u_0}^0(\delta)) \in (L_1 \times L_2) \times X_R\), and \(\rho''' := ((l', \rho_{u_0}^0(\delta'))\), holds: There exists the execution sequences \(c \rightarrow (u_0, \rho_{w_0}^0, \theta_{w_0}, \delta_0) \rho'\) and \(\rho'' \rightarrow (u_0, \rho_{w_0}^0, \theta_{w_0}, \delta_0) \rho'''\) with \((c, \rho'') \in \Omega\) and \((\rho', \rho''') \in \Omega\), which is again the above mentioned requirement for refinement, cf. Definition 14.

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

\[
\forall c \Rightarrow \pi c' \wedge \exists \rho'' \Rightarrow \pi c''' : (c', \rho''') \in \Omega
\]  

(15)

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 \(\text{offer}(M_1, (l_1, x)) = \{g_1^i \exists(l_1, g_1, a_1, a_1', l_1') \in T_1\} \cup \{(du/dt)(0) \exists(l_1, x) \Rightarrow (u, a, a, l_1, x')\}.\)

Further, we conclude \(\text{offer}(M_1, (l_1, l_2), x) = \{g_1^i \exists((l_1, l_2), g_1, a_1 \oplus a_2, (l_1', l_2')) \in T_1 \cup \{(du/dt)(0) \exists((l_1, l_2), x) \Rightarrow (u, a, a, l_1', x')\}\) This is equivalent to \(g_1^i \exists(l_1, g_1, a_1, a_1', l_1') \in T_1\} \cup \{(du/dt)(0) \exists(l_1) \in L_1\} with \(D(l_1) = (V^x(l_1), V^u(l_1), V^g(l_1), F(l_1), G(l_1), C(l_1), X^0(l_1))\) and for which exists trajectories \(\rho_u\) and \(\theta_u\).

Therefore it holds: \(\text{offer}(M_R, c) \supseteq \text{offer}(M_1, \rho'')\). Due to Definition 14 and together with (15) follows that \(M_1 \parallel \text{offer}(M_2) \subseteq M_1 \parallel \text{offer}(M_2)\).

Theorem 2 can be extended to the general case of \(M_S \parallel \text{offer}(M_1) \ldots \parallel \text{offer}(M_n)\) 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 Advanced High-Level Modeling

The behavior of the hybrid components from the application example from Section 3 was specified by our high-level automata model, called hybrid reconfiguration chart. In contrast to the hybrid reconfiguration automata model, a hybrid reconfiguration chart allows the use of the high-level constructs:

- committed locations,
6.1 Committed Locations

In this section, we define the semantics of these high-level constructs by mapping them to the formally defined hybrid reconfiguration automata model.

6.1 Committed Locations

To model atomic sequences of actions, e.g. atomic broadcast or multicast, we provide a notion of committed locations in the hybrid reconfiguration chart model. A committed location is a location where no delay is allowed. If an automaton is in a committed location, only transitions starting from such a committed location are allowed. Thus, processes (automata) in committed locations may be interleaved only with processes in a committed location.

\begin{figure}[h!]
\centering
\includegraphics[width=\textwidth]{committed_location.png}
\caption{Committed Location}
\end{figure}

In Figure 12, two orthogonal automata $A_1$ and $A_2$ are depicted. Location $S_1$ is modeled as committed. When $S_1$ is entered, an internal clock $t$ is initialized and a variable $c_j$ is set to true. To ensure that no time is consumed while $S_1$ is active, $S_1$ is associated with the invariant $t \leq 0$. To ensure that no other transition fires at that time all other transitions are labeled with the guard $\bigwedge_i c_i = false$.

For simplification, we will use the notion of Figure 13 in both hybrid reconfiguration charts and hybrid automata to describe atomic steps.
6.2 Asynchronous Communication

Hybrid reconfiguration automata interact via synchronous events. To support asynchronous events in the hybrid reconfiguration chart model, we extend the synchronous communication by adding a queue for each event to the system.

![Figure 14: Scheme for queuing asynchronous events](image)

Figure 14 shows the scheme for an automaton which is used to queue a given set of asynchronous events $e_1, \ldots, e_n$. The automaton consists of the three states, $S_1$, $S_2$ and Error. When the event $\text{send}_e1$ is sent by an automaton, the event is added to the queue. This is realized by the transition $S_1 \rightarrow S_2$, which adds the event to the array $e_1.queue$. If the entire event queue is not full, the automaton will turn back to the initial state (here marked by the double line as location border). Otherwise, the automaton will switch to the Error state. If the automaton is in state $S_1$ consumed events are dispatched by the self-transitions $S_1 \rightarrow S_1$.

6.3 History

Within hierarchical states of hybrid reconfiguration charts, it is possible to define a history flag. Thus, we also introduce a mapping for history.

In Figure 15 (left side), the hierarchical state $S_1$ of a hybrid reconfiguration chart contains the substates $S_1^1$, $S_1^2$ and the history flag (H). The effect of the latter one is that the state which last was active will be occupied again when $S_1$ is entered again. To realize this behavior in the hybrid reconfiguration automata model, a
History location $H$ and a variable $history$ are added for each history flag. On each transition, the variable $history$ is set to a unique value, representing a location, e.g. $history:=1$. When the state $S_1$ is entered for the first time, $history$ is set to $-1$. From the history location $H$, a transition to every other location is added. Each individual outgoing transition from $H$ is marked with a guard $history==i$, where $i$ is the unique id of the target location. In Figure 15 (right side), the transformed automaton is depicted. For flat history, as in the example, the construction is only done for the highest level of the hybrid reconfiguration chart. In case of deep history, the construction also has to be done for all other levels.

6.4 Hierarchy

In Figure 16, the mapping of hierarchy is depicted by means of an orthogonal AND-state. The AND-state $S_1$ of the hybrid reconfiguration chart consists of the two parallel statecharts $A_1, A_2$. Each of them has two states $S_1^1, S_2^1$ resp. $S_3^1, S_4^1$. To map the behavior of the AND-state, three automata are necessary in the hybrid reconfiguration automata model. For each concurrent automaton, an equivalent separate automaton is created. Furthermore, a coordination automaton $C$ which triggers the automata, implementing $A_1$ and $A_2$ respectively, is created.

The mode of operation is as follows: If the AND-state in the hybrid reconfiguration chart is entered, the coordination automaton in the hybrid reconfiguration chart is activated first. Implemented by a committed location, the sub-automata are directly triggered. Whenever a sub-automaton decides to leave the AND-state (modeled by the transition back to the initial location), the coordination automaton ensures that all other sub-automata will leave the AND-state, too.

6.5 exit()- and entry()-methods

To handle the time consuming transitions, the exit()-operation of the source state, the data assignment (action()) and the target state’s entry()-method have to be addressed first. They are mapped to the hybrid reconfiguration automata model by executing them sequentially in the entry action of an additional intermediate state.
Hybrid reconfiguration chart

Hybrid reconfiguration automaton

Figure 16: Mapping of hierarchy
This leads to two different kinds of locations: state locations, representing states, and action locations, in which operations are executed.

Hybrid reconfiguration chart

Hybrid reconfiguration automaton

Figure 17: Mapping of time consuming transitions

Figure 17 shows how a transition is mapped to an additional action location and two transitions of a hybrid reconfiguration automaton. Solid arrows denote urgent, dashed arrows denote non-urgent transitions. If a transition is activated for a specific time interval, an urgent one fires immediately, a non-urgent one fires at any point of time of the interval. The two transitions connect the action location with the source and the target state. The event, guard, time guard and priority of the hybrid reconfiguration chart’s transition are recovered in the urgent transition as these attributes shall trigger the actions.

The deadline is reused in the mapping multiple times. On the one hand, it is used to extend the invariant of the action location. On the other hand, it is used to prevent premature leaving of the action location before the point of time, specified by the lower bound of the deadline. As the last point of time when the action location may be left is specified by the upper bound of the deadline, it is used as the upper bound of the time guard of the non-urgent transition.

### 6.6 Transition Priorities

If a state has more than one outgoing transition and more than one of them is enabled at the same point in time, the transition with the highest priority fires. As priorities are not supported in the hybrid reconfiguration automata model, we have to provide a mapping for this concept. We have to distinguish between two cases: (1) a transition is only triggered by a guard and (2) a transition is triggered by a guard and an event. In the first case, we use the fact that boolean guards can be inverted. In the second case, the problem we have to deal with is that events cannot
be inverted like boolean guards and that two participants running in parallel are synchronized when a transition fires.

\[
\begin{align*}
\text{flag}_{e_1} &> 0 \quad \text{and} \quad \text{prio}_i \geq \max_{j=1,n,j \neq i} \text{flag}_{e_j} \ast \text{prio}_j \\
\text{flag}_{e_n} &> 0 \quad \text{and} \quad \text{prio}_n \geq \max_{i=1,n-1} \text{flag}_{e_i} \ast \text{prio}_i \\
\end{align*}
\]

Figure 18: Transformation of Priority

In Figure 18, the mapping scheme is depicted. A state \( S \) of a hybrid reconfiguration chart and its outgoing transitions \( S \rightarrow S_1 (t_1), \ldots, S \rightarrow S_n (t_n) \) which are synchronize via events \( e_i \square \) with \( \square \in \{?,!\} \) and which are additionally marked with a guard \( \text{guard}_i \) and a priority value \( \text{prio}_i \) are mapped to a hybrid reconfiguration automaton. An additional state \( S' \) associated with a committed flag is added to the hybrid reconfiguration automaton. Due to the fact that \( S' \) is marked as committed, the state must be left immediately again after it was entered. In addition, for every event \( e_i \square \), a variable \( \text{flag}_{e_i} \) is added. The flag characterizes whether the related event is enabled on both the sender and receiver side (\( \text{flag}_{e_i} = 1 \)), whether it is enabled only for one of the participants (\( \text{flag}_{e_i} = 0 \)), or whether it is not enabled for any of the participants (\( \text{flag}_{e_i} = -1 \)). The flag \( \text{flag}_{e_i} \) is tested and updated as follows:\(^8\)

\[
\text{flag}_{e_i} := \text{guard}_i = true?(\text{flag}_{e_i} + 1 : \text{flag}_{e_i})
\]

If the guard \( \text{guard}_i \) evaluates to true, \( \text{flag}_{e_i} \) is incremented. Otherwise, \( \text{flag}_{e_i} \) remains the same. From all outgoing transitions of the state \( S \) the guards are formed as follows:

\[
\text{flag}_{e_i} > 0 \quad \text{and} \quad \text{prio}_i \geq \max_{j=1,n,j \neq i} \text{flag}_{e_j} \ast \text{prio}_j
\]

\( \text{flag}_{e_i} > 0 \) is added to ensure that the original guard \( \text{guard}_i \) is true and both participants are ready to synchronize. To ensure that only the transition with the

\(^8\) \( a?b : c \) is an abbreviation for \( if \ a \ then \ b \ else \ c \).
highest priority fires, we additionally check that there is no other enabled transition with a higher priority than the current one.

\[ \forall i = 1 \ldots n \text{flag}_i := \text{guard}_i \equiv true? (\text{flag}_i - 1 : \text{flag}_i) \]

When a transition fires, the value of all flags \( \text{flag}_i \) has to be refreshed. If a \( \text{flag}_i \) has been incremented before, it is thus accordingly decremented.

### 6.7 do()-method

Figure 19 shows how a state \( S_1 \) and its do-method are mapped to two locations \( S_1 \) and \( S_1' \) in the hybrid reconfiguration automata model: The invariant is not modified. To model the periodic execution of the do()-operation, the new state \( S_1' \) is introduced. The non-urgent transitions \( S_1 \rightarrow S_1' \) and \( S_1' \rightarrow S_1 \) are created. The do()-operation is assigned to the first one. The latter one resets the new introduced clock \( t \) and it is only triggered at \( t \geq p_{low} \). To ensure that the do-operation is executed not later than \( p_{up} \), the invariant \( t \leq p_{up} \) is assigned to \( S_1' \), too. All transitions leading to \( S_1 \) in the hybrid reconfiguration chart are kept in the hybrid reconfiguration automaton. The leaving transitions are doubled so that one of them has its origin in \( S_1 \) and one in \( S_1' \).

![Hybrid reconfiguration chart](image)

Figure 19: Ordinary states with do()-methods

### 6.8 Stop State

The hybrid reconfiguration chart model contains the stop state which denotes the termination of a statechart. In the hybrid reconfiguration automata model, there is no special stop state. This behavior is mapped in a correct manner as depicted in Figure 20. A triggerless self-transition is added to the state so that the automaton stays in this location without causing a deadlock.
6 ADVANCED HIGH-LEVEL MODELING

Figure 20: Stop state

6.9 Fading Transitions

As introduced in Section 3, the fading transitions are a syntactical abbreviation. The mapping that defines the semantics of a fading transition becomes obvious by comparing Figures 5 and 6. The fading transition is mapped to an intermediate location, which we call fading location.

The configuration, associated with this location consists of the source locations configuration, the target locations configuration and one fading component for each output that is available in both, the source and the target configuration. If a component is part of the source and of the target configuration, two different instances of this component occur in the fading location. The fading component obtains the outputs of the source and the target configuration as input. The set $G$ of the fading component consists of the equation $f_{\text{fade}}$ from the fading transition specification. The set $F$ contains one entry $\dot{t}_c = 1$ for a clock $t_c$.

The uniquely defined transition, leading from the source location to the fading location is associated with the event trigger, guard, time guard, and update from the fading transition specification. Further, a reset for a clock $t_c$ is defined for this transition. Let $d = [d_{\text{low}}, d_{\text{up}}]$ be the duration interval of the fading transition. The fading location obtains an invariant $t_c \leq d_{\text{up}}$ and the uniquely defined transition from the fading location to the target location obtains a continuous guard $d_{\text{low}} \leq t_c \leq d_{\text{up}}$.

If the transition is not associated with a fading function, the intermediate fading location is just associated with the configuration of the source location. This can be seen as application of the default fading function $f_{\text{fade}} = 1$

6.10 Implied State Changes

The mapping, explained in Section 6.9, is extended when the superordinated component specifies an implied state change. In order to ensure synchronous state changes in the superordinated and in the subordinated component, additional sig-
nals are added as shown in Figure 21. The signal $i1$ ensures a synchronous switch to the intermediate state, the signal $i2$ ensures a synchronous switch to the target state. Note that we abstracted from the other details like guards etc. in Figure 21.

![Hybrid reconfiguration chart of component C1 embedding component C2](image1)

![Hybrid automaton of C1](image2)

![Hybrid automaton of C2](image3)

Figure 21: Mapping of implied state changes

### 6.11 Blockable & Enforceable Transitions

In some cases it is desired to specify further non-determinism. In [5], an example has been presented in which transitions are specified to be blockable. If a blockable transition is triggered, it may fire or it may be un-triggered (blocked) nonetheless. Opposite behavior is specified by enforceable transitions: An enforceable transition may be fired although it is not triggered due to time guard and I/O-guard. In order to restrict enforced firing of transitions, a time constant called minimal inter enforceable time $T^{MIET}$ is specified. Whenever a transition has been enforced, the next one may not be enforced before this time passed by.

Figure 22 shows the mapping of blockable and enforceable transitions respectively to standard hybrid reconfiguration charts and hierarchical states. Figure 22a shows that a new boolean flag is introduced in the source location of the blockable transition. This flag is non-deterministically set either to true or to false. If the blockable transition’s guard $g$ evaluates to true and if the I/O-guard is fulfilled, the flag determines if the transition fires, as the guard is enriched to $g \land x$ in the mapping.

Mapping the enforceable transition is done similar: In this case, a second transition with the same action is created that fires dependent on the flag $x$. This transition obtains the highest priority. It resets a clock $t_{en,f}$ when it fires. It is only triggered when the valuation of this clock, which has been reset to zero when any transition was enforced the last time, is greater or equal the $T^{MIET}$ time constant.
7 RELATED WORK

Figure 22: a) Blockable and b) enforceable transitions

7 Related Work

To discuss the current state of the art and the limitations of current approaches, we first discuss block diagrams and hybrid automata before turning our attention to concepts which support a decomposition of the models such as hierarchical blocks or components. The related approaches have been discussed in [6]. A more detailed discussion is given in [7].

Block diagrams [27] are the state of the art approach to specify feedback-controllers which is employed in all CAE tools such as Matlab/Simulink. Discrete elements can be used in the block diagram to model reconfiguration of the feedback-controller structure (often using a Statechart like notation such as Stateflow in Matlab/Simulink). A first approach is to embed the discrete control elements into block diagrams in such a manner, that the discrete block, described by a statechart like notation, determines which one of a set of alternative controller outputs is let through. Thus, atomic switching between the output signals of different controllers can be directly modeled. To address output fading, an additional generator for the fading function \( g \) and a weighted output fading element \( y = g(t)u_1 + (1 - g(t))u_2 \) has to be controlled by the statechart. Another option to model reconfiguration are conditionally blocks which are only evaluated if explicitly triggered. Thus, a statechart can be used to only trigger the required elements of the currently active configuration instead of blinding out the results of the not required ones. The more formal hybrid bond graphs approach [21, 22] permits to blind out single components by so called controlled junctions, similar to discrete blocks in block diagrams.

From a visual language perspective, both approaches to describe reconfiguration become problematic if all five identified challenges have to be addressed.

---

9http://www.mathworks.com
All configurations of required block diagrams have to be specified within a single complex diagram. Therefore, identification and comprehension of the different configurations becomes very difficult. The problems become even worse when it comes to the discrete control elements which describe the switching between the different configurations. Either they are rather unsystematically distributed in the block diagram as in the case of hybrid bond graphs or we end up with very complex statecharts describing the reconfiguration for the whole block diagram. The crucial problem here is that a consistent design of the different block diagram configurations and the statechart requires that the designer is able to identify the relation between control states of the statechart and block diagram configurations. However, the provided notations do not support such an understanding on the visual level.

The discussion of block diagrams and hybrid bond graphs showed that there is a strong relation between the system’s current global discrete state and the current configuration. In the mentioned approaches, there is no direct support for a mapping which assigns to each discrete state a related configuration.

Hybrid automata \[15, 1\] overcome this drawback by simply assigning a specific continuous controller to each discrete state, so that each possible configuration is easily derived from the model without complex analysis. Extensions such as Hybrid I/O automata \[20\] further support communicating hybrid automata. Support for concepts such as hierarchic and orthogonal discrete states as known for statecharts have been introduced for Hybrid statecharts as defined in \[17\].

Although these approaches separate the possible configurations and thus overcome one of the drawbacks of block diagrams, the current proposals restrict the possible configurations assigned to a state to read the same inputs and produce the same outputs. Therefore, the interface –especially the information which input signals are required for a safe application of the controller– is not present in these models. Another limitation, these approaches \[15, 1, 20, 17\] have in common is that they are restricted to the specification of behavior while an integration with an architectural description (similar to UML component diagrams or UML classes diagrams) is not provided. Therefore, these models can only be used for the specification of single components (in terms of UML), but a distributed system or a system with a modular, hierarchic architecture cannot be described.

If so called hierarchic blocks are employed in block diagrams to decompose the model, the visual complexity and problems to identify and comprehend the different configurations as well as their relation to discrete control states are less critical. However, this is only true when the interface of each hierarchic block is static and thus reconfiguration is restricted to happen only locally within the blocks. If this is not the case (not all inputs of a block are always required and not all output signals are always produced), the situation becomes even harder as the hierarchic blocks effectively hide their details and thus a designer cannot
7 RELATED WORK

keep track of the configuration effects which can cross the block interfaces. It is to be noted that in order to address challenge (3), we thus either end up in the outlined dilemma that information which is required to comprehend the effects of reconfiguration are hidden or all effects of reconfiguration due to local as well as external effects has to be modeled without a hierarchical decomposition. While in the former case the decomposition becomes a hindrance for comprehension, in the latter case the complexity of the flat model would render any attempt to develop a thorough understanding of the reconfiguration.

Other approaches combining components and hybrid automata concepts such as CHARON [2], HyROOM [30, 3], HyChart [14, 31], HybridUML [4], and Ptomely II [19] provide hierarchical automata models for the specification of behavior and hierarchical architectural models. In UML $^h$ [11], the architecture is specified by extended UML classes diagrams that distinguish between discrete, continuous, and hybrid classes. Also the OMG effort to integrate models from the software engineering domain with models from the control engineering domain [25] falls into this category. The Systems Modeling Language (SysML) [26] is a first proposal to standardize system engineering, which could be integrated with a possible UML 2.0 successor (cf. [18]).

Like the hybrid automata and statecharts, all class- or component-based approaches also assume static interfaces. The main improvement in modeling is the introduction of a hierarchical architectural model. The behavior of the components of the architectural models can then be specified by the (hierarchical) behavioral models. Hybrid behavior is specified by adding continuous components in form of a MATLAB model, a differential equation, or a similar textual description to each discrete state of the component. Thus, the same limitations which have been identified for hierarchic blocks with static interfaces also apply here.

The preceding discussion highlights that current approaches are not sufficient to address the identified five challenges from Section 7: HyROOM and Ptomely II are the only approaches following challenge (1) and allowing the engineers to specify the continuous behavior with the well-known block diagrams. None of the discussed approaches allows an appropriate specification of real-time behavior (conf. challenge (2)) as their semantics –if defined– assume to detect triggered transitions and to fire them without consuming time which is unrealistic and not realizable. Further, they lack of reconfiguration based on local and external events (challenge (3)), as such a reconfiguration usually leads to a change of the interfaces and requires reconfiguration via multiple hierarchical levels, because the feedback-controller components are usually located on the lowest hierarchical level while the real-time coordination with the external world is usually at the higher ones. Current approaches allow just reconfiguration via one hierarchic level without changing the interface. Support for advanced visual constructs for
a simple and intuitive specification of atomic transitions and fading transitions (challenge (4)) is not addressed at all. In addition, none of the approaches permits to handle flexible reconfiguration including compositional adaptation (challenge (5)). Instead, the anticipated reconfiguration steps have to be explicitly modeled right like the reconfiguration between two control algorithms.

8 Conclusion

We presented hybrid components and hybrid reconfiguration charts which are part of Mechatronic UML. After an informal introduction, we defined formally multiple behavioral models. First, we defined some basic behavioral models in order to use them as base for the hybrid reconfiguration automata model. Then, we defined formally hybrid components with static and variable structure. We also support high-level constructs known from statecharts, state machines, or timed automata and we defined new high-level modeling constructs. We defined the semantics of these high-level constructs which belong to the hybrid reconfiguration chart model by a mapping to the hybrid reconfiguration automata model.

The formally defined semantics enables code generation and formal analyzes like modelchecking.

References


REFERENCES


REFERENCES


A Prerequisites: Formal Definitions

This section defines the employed basic mathematical notations:

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 $B$. $EQ(V_l, V_r)$ denotes the set of all equations of the form $v_l = f^i(v^1_l, ..., v^n_l)$ with operations $f^i$ of arity $n$ and left- and right-hand side variables of the equation $v_l \in V_l$, $v^1_l, ..., v^n_l \in V_r$. $COND(V)$ 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 $v_l = \perp_i(v^1_r, ..., v^n_r)$ 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 $(a_1 \oplus a_2)(x \otimes y) := a_1(x) \otimes a_2(y)$.