Component Behavior Synthesis for Critical Systems*,**

Tobias Eckardt and Stefan Henkler

Software Engineering Group, Heinz Nixdorf Institute, University of Paderborn, Warburger Str. 100, Paderborn, Germany
{tobie,shenkler}@uni-paderborn.de

Abstract. Component-based architectures are widely used in embedded systems. For managing complexity and improving quality separation of concerns is one of the most important principles. For one component, separation of concerns is realized by defining the overall component functionality by separated protocol behaviors. One of the main challenges of applying separation of concerns is the later automatic composition of the separated, maybe interdependent concerns which is not supported by current component-based approaches. Moreover, the complexity of real-time distributed embedded systems requires to consider safety requirements for the composition of the separated concerns. We present an approach which addresses these problems by a well-defined automatic composition of protocol behaviors with respect to interdependent concerns. The composition is performed by taking a proper refinement relation into account so that the analysis results of the separated concerns are preserved which is essential for safety critical systems.

1 Introduction

Component-based architectures are widely used in the domain of embedded real-time systems. The main benefits of using components are their support for information hiding and reuse. The interface of a component is well defined by structural elements and a collaboration of protocols (cf. [1]). The overall component behavior is defined by the (parallelly executed) protocol behaviors. Dependencies between components are reduced to the knowledge of interfaces or ports. Thus, a component can be exchanged if the specified port remains fulfilled.

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

** This work was developed in the project "ENTIME: Entwurfstechnik Intelligente Mechatronik" (Design Methods for Intelligent Mechatronic Systems). The project ENTIME is funded by the state of North Rhine-Westphalia (NRW), Germany and the EUROPEAN UNION, European Regional Development Fund, "Investing in your future".

H. Giese (Ed.): ISARCS 2010, LNCS 6150, pp. 52–71, 2010.

The port and interface definitions of architectural components therefore facilitate the construction of complex functionality by the composition of components and protocols.

For managing complexity and improving quality of systems, separation of concerns [2] is one of the most important principles. It enables primary software engineering goals like adaptability, maintainability, extendability and reusability. Accordingly, advanced applications of separation of concerns have gained popularity like aspect-oriented programming (AOP) [3], for example. For one component, separation of concerns is realized by defining the overall component functionality by separated protocol behaviors [4].

One of the main challenges of applying separation of concerns is the later (application specific) composition of the separated, maybe interdependent concerns [5]. In general, we can distinguish between structural, data, and behavioral composition. In the area of structural composition, approaches exist for example, that consider the software architecture as well as architectural patterns [6,7]. For data composition approaches like [8] support the generation of suitable translators. In [9,4] approaches for the behavioral composition are presented. The overwhelming complexity of embedded real-time systems, however, requires to also consider safety and bounded liveness requirements for the composition which is not included in these approaches. On the other hand, component-based approaches for embedded real-time systems (e. g. [10,11]) suffer the support for interdependent concerns for the well-defined composition.

In this paper, we present an approach which addresses these problems by a well-defined automatic composition of protocol behaviors with respect to interdependent concerns specified as *composition rules*. The defined composition rules preserve timed safety properties which is inherently important for safety critical systems. The composition is performed by taking a proper refinement relation into account, which we call *role conformance*. This way also untimed liveness properties are preserved which is equally essential for safety critical systems. This work extends the fundamental work of [4] to the domain of critical systems (cf. Section 7).

In contrast to approaches which integrate interdependent behavior by an additional observer automaton (e.g., our former work as presented in [12]), our approach enables the explicit specification of interdependent concerns and the synthesis algorithm integrates the specified concerns automatically.

In general, the observer based approach is difficult to apply and error prone. Owned by the implicit specification of composition rules by the observer automata, the developer did not know if the composition rule in mind is really correctly implemented by the observer automata. To forbid for example that two protocol behaviors are at the same time in a specific state, all "relevant" events (timing and messages) have to be observed which lead to the forbidden states. After a corresponding observer automaton has been specified the developer did not know if all relevant events are observed, if too much behavior is observed (forbidden) or if timed safety properties and untimed liveness properties of the protocol behaviors are violated.

Additionally, the developer of the observer automaton has to instrument the protocol behaviors to enable the observation. This is not intended, however, as this may cause malfunctions originating from mistakes of the developer. Altogether the observer based approach is not well suited for safety critical systems.

For our synthesis approach, we extend our modeling approach MECHATRONIC UML which addresses the development of complex embedded real-time systems. MECHATRONIC UML supports the compositional specification and verification of real-time coordination by applying component-based development and pattern-based specification [12]. Furthermore, it also supports the integrated description and modular verification of discrete behavior and continuous control of components [13].

We evaluate our approach by the RailCab project of the University of Paderborn[1]. The vision of the RailCab project is a mechatronic rail system where autonomous vehicles, called RailCabs, apply a linear drive technology, as used by the Transrapid system[2], for example. In contrast to the Transrapid, RailCabs travel on the existing track system of a standard railway system and only require passive switches. One particular problem (previously presented in [12]) is the convoy coordination of RailCabs. RailCabs drive in convoys in order to reduce energy consumption caused by air resistance and to achieve a higher system throughput. Convoys are established on-demand and require small distances between RailCabs. These small distances cause the real-time coordination between the speed control units of the RailCabs to be safety critical which results in a number of constraints that have to be addressed when developing the RailCabs control software.

In the following section, we present the relevant parts of MECHATRONIC UML and give an overview of our synthesis approach. For the formalization of the approach, we give fundamental definitions for the input behavioral specifications in Section 3. In Section 4, we present the concept of composition rules which formalize interdependent concerns. These composition rules are applied within the automatic composition of protocol behavior, as defined by the synthesis algorithm in Section 5. As the effect of the application of a set of composition rules cannot be anticipated, the result of the synthesis can violate properties of the protocol behavior. Therefore, we present the check for role conformance in Section 6. Related work is discussed in Section 7 and at last we conclude with a summary and future work in Section 8.

2 Approach

In MECHATRONIC UML separation of concerns is realized by applying *component-based development* and in accordance with that by rigorously separating inter-component from intra-component behavior. Following this concept, the system is decomposed into participating components and *real-time coordination patterns* [12], which define how components interact with each other.

[1] http://www-nbp.uni-paderborn.de/index.php?id=2&L=1
[2] http://www.transrapid.de/cgi-tdb/en/basics.prg

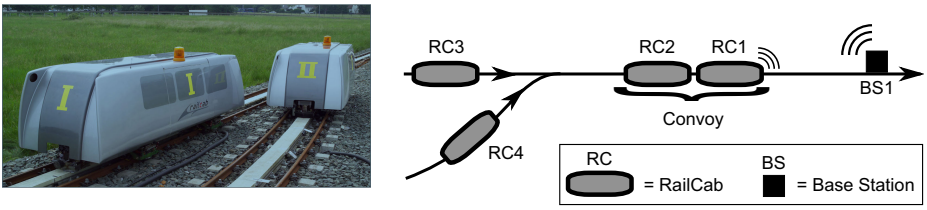

Fig. 1. RailCab example

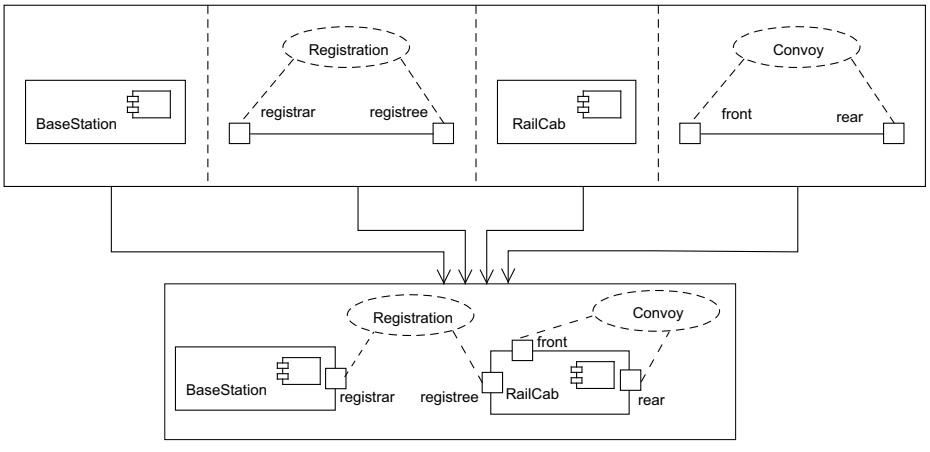

Fig. 2. Combining Separate Specifications in Mechatronic UML

To exemplify this, we use our RailCab case study. In Figure 1, we depict a situation of RailCabs driving in a convoy. The figure on the left shows this situation in the real test bed and on the right an abstraction is shown. In addition to the RailCabs, we depict a base station which is responsible for the power supply of the RailCabs and the management of track information for a specified section. The track information includes the data of all RailCabs in this section. RailCabs use this information to be aware of other RailCabs in their section in order to avoid crashes and possibly build convoys.

We specify two components BaseStation and RailCab (Figure 2) and two coordination patterns Registration and Convoy, which define the before described communication behavior between RailCabs and base stations.

In real-time coordination patterns, *roles* are used to abstract from the actual components participating in one coordination pattern. This way, it is possible to specify and verify coordination patterns independently from other coordination patterns and component definitions and therefore to reduce complexity. In Figure 2 the participating roles of the Registration pattern are registrar and registree; the roles of the Convoy pattern are front and rear. Each role behavior is

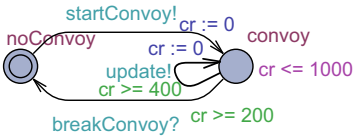

Fig. 3. Simplified Rear Role Timed Automaton

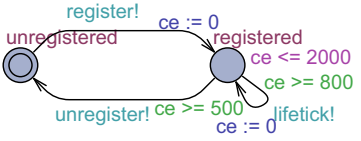

Fig. 4. Simplified Registree Role Timed Automaton

specified by one timed automaton[3] [15,16]. The automata of the rear role and the registree role are depicted in Figure 3 and Figure 4. The automata for the front role and the registrar role only form corresponding counterparts and are therefore not depicted. We present only a simplified version of the behavior in order to present the complete approach by an example.

Initially, the rear role is in state noConvoy and sends a startConvoy event. The clock cr is set to zero before entering the convoy state. In the interval of 200 to 1000 time units the breakConvoy event has to be received as the location invariant of state convoy is $cr \leq 1000$ and the time guard of the transition is $cr \geq 200$ or in the interval of 400 to 1000 time units, periodically an update event is sent. The registree role is initially in the unregistered state, sends a register event and resets the clock. In the registered state in the interval of 800 to 2000 time units, periodically the lifetick event is sent or in the interval of 500 to 2000 time units the unregistered event is sent. The decision of sending the lifetick or unregistered event is at this point of nondeterministic choice.

To obtain an overall system specification later in the development process, the separated components and coordination patterns have to be combined again (Figure 2). The problem which inherently arises at this point is that separate parts of the system were specified as independent from each other when they are in fact not. This means that during the process of combining the separate parts of the system, additional dependencies between the particular specifications have to be integrated. At the same time, the externally visible behavior of the particular behavioral specifications may not be changed in order to preserve verification results [12].

In the overall system view of the RailCab example (Figure 2), the RailCab component takes part in both, the Registration and the Convoy pattern. While those patterns have been specified independently from each other, a system requirement states:

> *In convoy operation mode, each participating RailCab has to be registered to a base station.*

Accordingly, a dependency between both patterns exists, when applied by the RailCab component. As a result, the behavior of the registree role and the

[3] In MECHATRONIC UML, realtime statecharts [14] are used to describe role behavior. Realtime statecharts, however, are based on timed automata. Therefore, we define the complete synthesis procedure on the basis of timed automata in order to make the approach as general as possible.

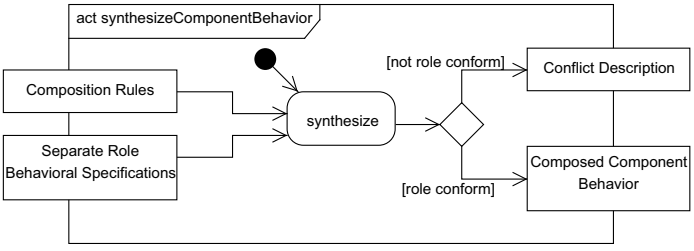

Fig. 5. Activity Diagram Illustrating the Basic Synthesis Approach

behavior of the rear role have to be refined and synchronized with each other when applied by the RailCab component in order to fulfill the system requirements. Still, it has to be regarded that the externally visible behavior of the RailCab component does not change. If this process of refinement and synchronization is performed manually, it is a time consuming and error-prone process. Consequently, this implies the necessity for automation in order to guarantee the required quality of safety critical systems.

In the proposed approach, we formalize the specification of inter-role dependencies and further separate this specification from the specification of pattern role behaviors in order to perform an automatic synthesis for the overall component behavior. Once the synthesis is performed, it is checked if the synthesized component behavior refines each of the particular pattern role behaviors properly.

The approach requires (1) the definition of a suitable refinement relation for (real) dense time systems and (2) the employment of a suitable and efficient abstraction of the timed behavioral models which is needed to perform the refinement check. The result is a fully automatic synthesis algorithm where dependencies between separate behavioral specifications are specified explicitly by so-called *composition rules* (cf. [5]). Accordingly, the input for the algorithm are composition rules and separate behavioral specifications (Figure 5) in the form of timed automata. If the synthesis is possible without violating the externally visible behavior of any of the input specifications, the output is one parallelly composed component behavior which combines all of the input behavioral specifications as well as the composition rules. If the synthesis is not possible, the algorithm returns a conflict description indicating the reason for the impossibility.

We continue with the basic definitions for the input behavioral specifications in the form of timed automata.

3 Prerequisites

For the verification of real-time coordination patterns, Mechatronic UML employs the model checker UPPAAL[4]. UPPAAL uses timed safety automata [16] as the input model [17]. Consequently, we also employ the concept of timed

[4] http://www.uppaal.com/

safety automata for the entire approach and refer to them as timed automata in the following.

Within a timed automaton, we use clock constraints to make the behavior of the automaton dependent on the values of certain clocks of the automaton. A *general clock constraint* is a Boolean formula joining a set of equations and inequations describing the lower and upper bounds for clocks and clock differences.

Definition 1 (General Clock Constraint). *For a set C of clocks, the set $\Phi(C)$ of* general clock constraints *is inductively defined by the grammar $\varphi ::= x \sim n \mid x - y \sim n \mid \varphi \wedge \varphi \mid true \mid false$, where $x, y \in C$, $\sim \in \{\leq, <, =, >, \geq\}$, $n \in \mathbb{N}$.*

We further define *downwards closed clock constraints* as those constraints, which only define upper bounds for clock values. The lower bound of all clocks in a downwards closed clock constraint, consequently, is always zero.

Definition 2 (Downwards Closed Clock Constraint). *For a set C of clocks, the set $\Phi_{dc}(C) \subset \Phi(C)$ of* downwards closed clock constraints *is inductively defined by the grammar $\varphi ::= x \sim n \mid x - y \sim n \mid \varphi \wedge \varphi \mid true$, where $x, y \in C$, $\sim \in \{\leq, <\}$, $n \in \mathbb{N}$.*

With the definitions of clock constraints we can proceed with the definition of the syntax of a timed automaton. Note that this definition corresponds to the one given in [18], which is employed in UPPAAL.

Definition 3 (Timed Automaton). *A* Timed Automaton *A is a tuple $(L, l^0, \Sigma, C, I, T)$ where L is the set of locations, $l^0 \in L$ is the initial location, Σ is the finite set of events where the symbol τ is used for internal events (silent transitions), $I : L \rightarrow \Phi_{dc}(C)$ assigns each location a location invariant as a downwards closed clock constraint, C is the finite set of clocks, and $T \subseteq L \times \Sigma \times \Phi(C) \times 2^C \times L$ is the finite set of transitions $t = (l, e, g, r, l') \in T$ with $l \in L$ the source location, $e \in \Sigma$ the related event, $g \in \Phi(C)$ the time guard as a general clock constraint, $r \subseteq C$ a set of clocks to be reset, and $l' \in L$ the target location.*

Examples of timed automata are depicted in Figure 3 and Figure 4 describing the communication behavior of the rear role and the registree role of the Convoy and the Registration real-time coordination pattern as described in Section 2. On the basis of the above given definitions, we formally define inter-role dependency specifications in the form of composition rules in the next section.

4 Composition Rules

With composition rules, interdependent concerns for the separate role behaviors can be specified as system properties which synchronize parts of the separated role behavioral models.

We divide composition rules into two distinct formalisms that are state composition rules and event composition automata. With *state composition rules*

we are able to synchronize the role behavior with respect to certain state combinations of the particular role automata. *Event composition automata*, on the other hand, provide the possibility to synchronize the role automata on the basis of events and event sequences. Both formalisms also include the specification of timing information for synchronization referring to the clocks of the role automata.

Generally speaking, system properties can be specified in terms of safety and liveness properties for a given behavioral specification [19,20]. *Safety properties* state that something bad will never happen during the execution of a program. *Liveness properties* state that something good will happen eventually. Transferring this to the context of automata synchronizations, these properties always concern two or more automata. Consequently, a *safety property for synchronization* states that something bad will never happen, when executing the corresponding automata in parallel, while a *liveness property for synchronization* expresses that something good will eventually happen during this parallel execution.

Transferring these properties to composition rules, we are able to specify both safety and liveness properties. Safety properties can be specified (1) by means of *state composition rules* in terms of forbidden state combinations of the parallel execution and (2) by means of *event composition automata* by adding further time constraints to time guards of selected transitions. Liveness properties in turn can be specified through state composition rules and event composition automata by adding further time constraints to location invariants of location combinations of the parallel execution.

State composition rules define forbidden state combinations, including timing information, in the parallel execution of the role automata. In order to make statements about forbidden state combinations of a component behavior, we need to define which clock values are forbidden in which automaton location. As the location invariant of an automaton location must be downwards closed (see Definition 3), the forbidden clock valuations can only be described by an *upwards closed clock constraint*. This is then used in a *location predicate* to connect the forbidden clock valuations to a certain location.

Definition 4 (Upwards Closed Clock Constraint). *For a set C of clocks, the set $\Phi_{uc}(C) \subset \Phi(C)$ of* upwards closed clock constraints *is inductively defined by the grammar $\varphi ::= x \sim n \mid x-y \sim n \mid \varphi \wedge \varphi \mid true$, where $x, y \in C$, $\sim \in \{\geq, >\}$, $n \in \mathbb{N}$.*

Definition 5 (Location Predicate). *For a timed automaton $A = (L, l^0, \Sigma, C, I, T)$, a location $l \in L$ and an upwards closed clock constraint $\varphi \in \Phi_{uc}(C)$ the set $\Gamma(A)$ of location predicates $\gamma = (l, \varphi)$ is defined by $\Gamma(A) = L \times \Phi_{uc}(C)$.*

With *state composition rules*, we want to restrict certain state combinations of the concerned role automata. Consequently, we define these state combinations by connecting location predicates by Boolean joins and meets in order to express which timed location combinations are not allowed in the composed component.

Definition 6 (State Composition Rule). *For two timed automata $A_1 = (L_1, l_1^0, \Sigma_1, C_1, I_1, T_1)$ and $A_2 = (L_2, l_2^0, \Sigma_2, C_2, I_2, T_2)$ the set $R^S(A_1, A_2)$ of*

state composition rules ρ *is defined by the grammar* $\rho ::= \neg\rho_\gamma$, $\rho_\gamma ::= \rho_\gamma \wedge \rho_\gamma \mid \rho_\gamma \vee \rho_\gamma \mid \gamma$, *where* $\gamma \in \Gamma(A_1) \cup \Gamma(A_2)$.

An example of a state composition rule is the rule r_1, given with:

$$r_1 = \neg((unregistered, true) \wedge (convoy, true)).$$

The state composition rule r_1 formalizes the pattern overlapping system requirement explained in Section 2. Correspondingly, it defines that a RailCab is not allowed to rest in states $(unregistered, true)$ and $(convoy, true)$ at the same time, where the clock constraint *true* denotes that all clock values of the corresponding automata are concerned.

Event composition automata synchronize the parallelly executed role automata on the basis of events and event sequence by adding further timing constraints to the parallel execution.

For event composition automata, we also apply the syntax of timed automata themselves, as event composition automata are also used to describe possible event sequences of the component behavior. In contrast to pattern role automata, event composition automata do not add any further event occurrences, which means that they do not consume or provide any signals from the channels of the corresponding role automata. In other words, event composition automata are only monitoring event occurrences for a given set of role automata while they do not distinguish between sending or receiving events. They do, however, allow to add further timing constraints to the monitored event occurrences, also in terms of location invariants for the locations between the monitored events. This way, safety and liveness properties for the synchronization of several role automata can be specified. Formally, an event composition automaton is defined as follows.

Definition 7 (Event Composition Automaton). *Let* $A_1 = (L_1, l_1^0, \Sigma_1, C_1, I_1, T_1)$ *and* $A_2 = (L_2, l_2^0, \Sigma_2, C_2, I_2, T_2)$ *be two timed automata. An* event composition automaton $A_E \in R^A(A_1, A_2)$ *is again a timed automaton as a tuple* $(L_E, l_E^0, \Sigma_E, C_E, I_E, T_E)$ *, where* L_E *is a finite non empty set of locations,* $l_E^0 \subseteq L$ *is the initial location,* $\Sigma_E \subseteq \Sigma_1 \cup \Sigma_2$ *is the finite set of events to be observed,* $I : L \rightarrow \Phi_{dc}(C_E)$ *assigns each location a downwards closed clock constraint,* C_E *is a finite set of clocks, with* $C_E \cap (C_1 \cup C_2) = \varnothing$ $T_E \subseteq L_E \times \Sigma_E \times \Phi(C_E) \times 2^{C_E} \times L_E$ *is a finite set of transitions* $t = (l, e, g, r, l') \in T_E$, $l \in L_E$ *is the source location,* $e \in \Sigma_E$ *is the observed event,* $g \in \Phi(C_E)$ *is the time guard,* $r \subseteq C_E$ *is a set of clocks to be reset, and* $l' \in L_E$ *is the target location.*

Semantically, an event composition automaton only observes event occurrences of the given role automata. Consequently, only those events can be used in an event composition automaton, as others can never be observed. Additionally, the set of clocks of the event composition automaton is restricted to be disjoint to the set of clocks of the role automata. This way, it is guaranteed that the event composition automaton cannot widen the time intervals of event sequences of the automata to be synchronized. This in turn guarantees that all verified deadlines of the role automata can still be met and, therefore, that all verified safety properties of the role automata are preserved (see section 6).

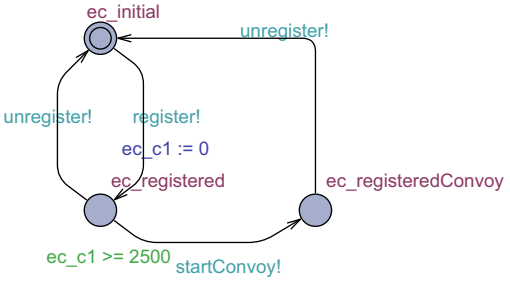

Fig. 6. Event Composition Automaton eca_1

To give an example for the pattern role automata of the rear role and the registree role (Figure 3 and Figure 4), assume a further pattern overlapping system requirement stating that a RailCab has to be registered to a base station for at least 2500 time units before starting a convoy. Observe that this requirement cannot be implemented using a state composition rule, as it is based on the occurrence of the startConvoy! event of the rear role automaton. Accordingly, we specify the event composition automaton eca_1 (Figure 6) to implement this requirement.

For implementing the requirement, the event composition automaton eca_1 monitors the register! event of the registree role automaton. Along with the occurence of this event the clock ec_c1 is reset. The time interval, in which the first following startConvoy! event my occur is then restricted by the time guard ec_c1 >= 2500. This means that a startConvoy! may not occur earlier than 2500 time units after the register! event which realizes that the RailCab has to be registered for at least this time to be able to start a convoy.

Once in ec_registeredConvoy, eca_1 changes its location only on the occurrence of the event unregister!, as in this situation the monitoring has to be started once more from the initial location. In all other situations, the component does not change its state of being registered and therefore this event composition rule does not have to add any further constraints.

With composition rules, we defined a suitable formalism to describe inter-role dependencies. We proceed with the definition of the synthesis algorithm in the next section, which includes the application of composition rules.

5 Synthesis Algorithm

The synthesis algorithm is divided into four distinct steps (see Figure 7). First, the parallel composition of the role automata is computed, which forms an explicit model for the parallel execution of the pattern role automata. On this parallelly composed timed automaton the composition rules are applied, by removing the forbidden system states specified by the state composition rules and by including the specified event composition automata in the parallelly composed

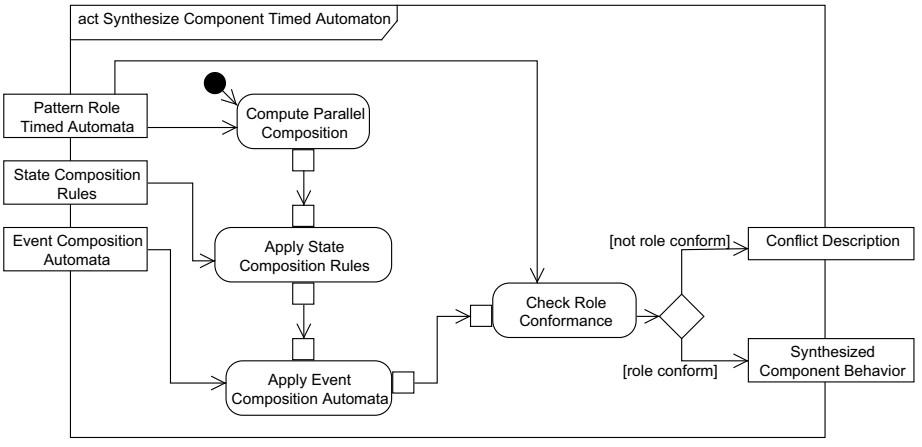

Fig. 7. Synthesis Algorithm for Timed Automata

automaton. In the last step, it is verified that the externally visible behavior of the particular role automata is preserved, as the changes made on the parallelly composed automaton by means of the application of composition rules might lead to violations of properties of the original role behaviors. Note that the overall procedure can also be applied iteratively in the development process.

The parallel composition applied in our approach is derived from the parallel composition operator of the process algebra *Calculus of Communicating Systems (CCS)* [9] as it has also been applied in *networks of timed automata* in [21,17]. In these approaches, the parallel composition allows for both synchronization and interleaving of events. The pattern role automata applied to one MECHATRONIC UML component, however, are defined such that they are independent from each other, in order to allow for compositional model checking. Consequently, we do not need to consider synchronizations in the parallel composition defined here. The parallel composition of the example automata of the rear role and the registree role (Figure 3 and Figure 4) is depicted in Figure 8.

Definition 8 (Parallel Composition). *Let* $A_1 = (L_1, l_1^0, \Sigma_1, C_1, I_1, T_1)$ *and* $A_2 = (L_2, l_2^0, \Sigma_2, C_2, I_2, T_2)$ *be two timed automata with* $C_1 \cap C_2 = \varnothing$ *and* $\Sigma_1 \cap \Sigma_2 = \varnothing$. *We define the parallel composition* $A_1 \parallel A_2$ *as a product automaton* $A_P = (L_P, l_P^0, \Sigma_P, C_P, I_P, T_P)$, *where* $L_P = L_1 \times L_2$, $l_P^0 = (l_1^0, l_2^0)$, $\Sigma_P = \Sigma_1 \cup \Sigma_2$, $I_P : L_P \rightarrow \Phi(C_1) \cup \Phi(C_2)$ *with* $I_P((l_1, l_2)) = I_1(l_1) \wedge I_2(l_2)$, $C_P = C_1 \cup C_2$, $T_P \subseteq L_P \times \Sigma_P \times \Phi(C_P) \times 2^{C_P} \times L_P$, *with* $((l_1, l_2), e_1, g_1, r_1, (l_1{}', l_2)) \in T_P \Rightarrow (l_1, e_1, g_1, r_1, l_1{}') \in T_1$, *and* $((l_1, l_2), e_2, g_2, r_2, (l_1, l_2{}')) \in T_P \Rightarrow (l_2, e_2, g_2, r_2, l_2{}') \in T_2$.

The application of a state composition rule requires to evaluate each location predicate of that rule for a given parallelly composed automaton location.

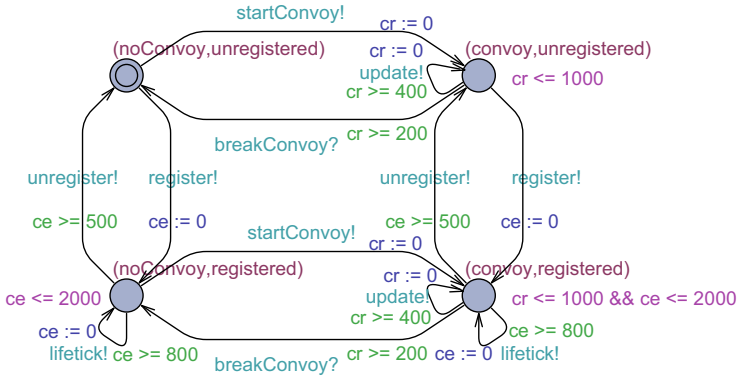

Fig. 8. Parallelly Composed Timed Automaton

Definition 9 (Location Predicate Evaluation). *Given two timed automata* $A_1 = (L_1, l_1^0, \Sigma_1, C_1, I_1, T_1)$, $A_2 = (L_2, l_2^0, \Sigma_2, C_2, I_2, T_2)$, *their parallelly composition* $A_P = A_1 \parallel A_2 = (L_P, l_P^0, \Sigma_P, C_P, I_P, T_P)$, *a corresponding parallelly composed location* $l_p = (l_1, l_2)$, *and a location predicate* $\gamma = (l, \varphi)$ *with* $l \in L_1 \cup L_2$ *and* $\varphi \in \Phi_{uc}(C_1) \cup \Phi_{uc}(C_2)$ *the* location predicate evaluation *is a function* $\gamma : L_P \rightarrow \Phi_{uc}(C_P) \cup \{false\}$ *defined with*

$$\gamma(l_p) = \begin{cases} \varphi, & \textit{iff } (l = l_1) \vee (l = l_2), \\ false, & \textit{else}. \end{cases}$$

On the basis of the evaluation of each location predicate of a state composition rule, the entire composition rule can be applied to a parallelly composed automaton location as defined by the *state composition rule evaluation*.

Definition 10 (State Composition Rule Evaluation). *Given two timed automata* $A_1 = (L_1, l_1^0, \Sigma_1, C_1, I_1, T_1)$, $A_2 = (L_2, l_2^0, \Sigma_2, C_2, I_2, T_2)$, *their parallel composition* $A_P = A_1 \parallel A_2 = (L_P, l_P^0, \Sigma_P, C_P, I_P, T_P)$, *a corresponding parallelly composed location* $l_p = (l_1, l_2)$, *and a state composition rule* $\rho \in R^S(A_1, A_2)$ *the* state composition rule evaluation *is a function* $\rho : L_P \rightarrow \Phi_{dc}(C_P) \cup \{false\}$ *defined with*

$$\rho(l_p) = \begin{cases} \neg\rho_1(l_p), & \textit{iff } \rho \textit{ is of the form } \neg\rho_1, \\ \rho_1(l_p) \wedge \rho_2(l_p), & \textit{iff } \rho_\gamma \textit{ is of the form } \rho_1 \wedge \rho_2, \\ \rho_1(l_p) \vee \rho_2(l_p), & \textit{iff } \rho_\gamma \textit{ is of the form } \rho_1 \vee \rho_2, \\ \gamma(l_p), & \textit{iff } \rho_\gamma \textit{ is the literal } \gamma. \end{cases}$$

where $\gamma \in \Gamma(A_1) \cup \Gamma(A_2)$.

The application of a state composition rule results in a *state composition conform* timed automaton. This automaton originates from the parallel composition

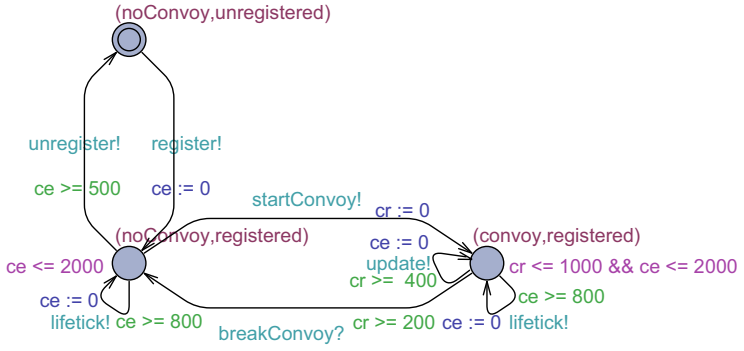

Fig. 9. Synthesized Component Behavior of the RailCab Component

but is modified such that the corresponding state composition rule has been applied to each automaton location. Automaton locations whose invariant is $false$ are further removed from the automaton. The example state composition rule r_1 applied to the parallel composition of the rear and registree role automata (Figure 8) results in the automaton depicted in Figure 9.

Definition 11 (State Composition Conformance). *Let* $A_P = A_1 \parallel A_2 = (L_P, l_P^0, \Sigma_P, C_P, I_P, T_P)$ *be the parallel composition of the timed automata* A_1 *and* A_2*. Further let* $R_1^S \subseteq R^S(A_1, A_2)$ *be a set of state composition rules specified over* A_1 *and* A_2*. The* state composition conform*, parallelly composed timed automaton* $A_{SC} = (L_{SC}, l_{SC}^0, \Sigma_{SC}, C_{SC}, I_{SC}, T_{SC})$ *is defined with* $L_{SC} = L_P \setminus L_R$*, where* $L_R = \{l_p \mid l_p \in L_P \text{ and } \forall\ \rho_1, \ldots, \rho_n \in R_1^S : I(l_p) \wedge \rho_1(l_p) \wedge \ldots \wedge \rho_n(l_p) = false\}$*,* $l_{SC}^0 = l_P^0 \Leftrightarrow l_P^0 \in L_{SC}$*,* $\Sigma_{SC} = \Sigma_P$*,* $I_{SC} : L_{SC} \rightarrow \Phi(C_{SC})$ *with* $I_{SC}(l_p) = I_P(l_p) \wedge \rho_1(l_p) \wedge \ldots \wedge \rho_n(l_p), \forall\ \rho_1, \ldots, \rho_n \in R_1^S$*,* $C_{SC} = C_P$*,* $T_{SC} \subseteq L_{SC} \times \Sigma_{SC} \times \Phi(C_{SC}) \times 2^{C_{SC}} \times L_{SC}$*, with* $(l_p, e, g, r, l_p{}') \in T_{SC} \Leftrightarrow (l_p, e, g, r, l_p{}') \in T_P \wedge l_p, l_p{}' \in L_{SC}$*.*

Similar to the parallel composition used for the parallel execution of the role automata, applying event composition automata can also be compared to the parallel composition operator of the process algebra *Calculus of Communicating Systems (CCS)* [9] or the *networks of timed automata* formalism defined in [21]. Here, the resulting automaton is a composition of the event composition automaton and the parallel composition of the role automata.

The fundamental difference is that for the event composition automaton application only synchronization of events is taken into account, as event composition automata do not define any new event occurrences for the parallel execution. Furthermore, these synchronizations do not take the channel concept into account, which means that a sending event is synchronized with a sending event and also results in a sending event. This also holds for receiving events and originates from the fact that the event composition automaton only observes the event

occurrences of the parallel execution. We call this type of synchronization *silent synchronization*.

In the resulting automaton the additional time guards, clock resets and location invariants of the event composition automaton are added to the composed locations and synchronized transitions as defined in the following.

Definition 12 (Event Composition Conformance). *Let* $A_{SC} = (L_{SC}, l^0_{SC}, \Sigma_{SC}, C_{SC}, I_{SC}, T_{SC})$ *be a state composition conform, parallelly composed timed automaton originating from the timed automata* $A_1 = (L_1, l^0_1, \Sigma_1, C_1, I_1, T_1)$ *and* $A_2 = (L_2, l^0_2, \Sigma_2, C_2, I_2, T_2)$ *with* $C_1 \cap C_2 = \varnothing$ *and* $\Sigma_1 \cap \Sigma_2 = \varnothing$. *Furthermore, let* $A_E = (L_E, l^0_E, \Sigma_E, C_E, I_E, T_E) \in R^A(A_1, A_2)$ *be an event composition automaton for* A_1 *and* A_2. *We define the* event composition conform *and state composition conform, parallelly composed timed automaton* $A_{EC} = (L_{EC}, l^0_{EC}, \Sigma_{EC}, C_{EC}, I_{EC}, T_{EC})$ *with* $L_{EC} \subseteq L_1 \times L_2 \times L_E$, *with* $(l_1, l_2, l_e) \in L_{EC}$ *iff* $(l_1, l_2) \in L_{SC}$ *and* $I_{SC}((l_1, l_2)) \wedge I_E(l_e) \neq false$) *and* (l_1, l_2, l_e) *is reachable trough* T_{EC}, $l^0_{EC} = (l^0_1, l^0_2, l^0_e)$, *iff* $(l^0_1, l^0_2, l^0_e) \in L_{EC}$, $\Sigma_{EC} = \Sigma_1 \cup \Sigma_2$, $I_{EC} : L_{EC} \rightarrow \Phi(C_1) \cup \Phi(C_2) \cup \Phi(C_E)$ *with* $I_{EC}((l_1, l_2, l_e)) = I_{SC}((l_1, l_2)) \wedge I_E(l_e)$, $C_{EC} = C_1 \cup C_2 \cup C_E$, $T_{EC} \subseteq L_{EC} \times \Sigma_{EC} \times \Phi(C_{EC}) \times 2^{C_{EC}} \times L_{EC}$, *with* $((l_1, l_2, l_e), e_1, g_1, r_1, (l_1{}', l_2, l_e)) \in T_{EC} \Leftrightarrow ((l_1, l_2), e_1, g_1, r_1, (l_1{}', l_2)) \in T_{SC} \wedge \forall\ l_e{}' \in L_E : (l_e, e_1, g_e, r_e, l_e{}') \notin T_E$, $((l_1, l_2, l_e), e_2, g_2, r_2, (l_1, l_2{}', l_e)) \in T_{EC} \Leftrightarrow ((l_1, l_2), e_2, g_2, r_2, (l_1, l_2{}')) \in T_{SC} \wedge \forall\ l_e{}' \in L_E : (l_e, e_2, g_e, r_e, l_e{}') \notin T_E$, $((l_1, l_2, l_e), e_1, g_1 \wedge g_e, r_1 \cup r_e, (l_1{}', l_2, l_e{}')) \in T_{EC} \Leftrightarrow ((l_1, l_2), e_1, g_1, r_1, (l_1{}', l_2)) \in T_{SC} \wedge (l_e, e_1, g_e, r_e, l_e{}') \in T_E$, $((l_1, l_2, l_e), e_1, g_1 \wedge g_e, r_1 \cup r_e, (l_1, l_2{}', l_e{}')) \in T_{EC} \Leftrightarrow ((l_1, l_2), e_2, g_2, r_2, (l_1, l_2{}') \in T_{SC} \wedge (l_e, e_2, g_e, r_e, l_e{}') \in T_E$.

To exemplify this, we apply the event composition automaton eca_2 to the parallel composition of the simplified rear role and registree role automaton, where the state composition rule r_1 has already been applied (Figure 9). This results in the timed automaton depicted in figure 10. Note that every location of this automaton refers to both the locations of the role automata as well as the locations of the event composition automaton eca_1. Furthermore, observe that those composed locations which are not reachable from the initial composed location $(noConvoy, unregistered, ec\_initial)$ are omitted.

In the resulting automaton, the clock reset ec_c1 := 0 and the time guard ec_c1 >= 2500 originating from the event composition automaton is added to the register! and to the startConvoy! transition respectively. Furthermore, it is now distinguished between the (noConvoy,registered,...) locations where the RailCab has just been registered (noConvoy,registered,ec_registered) and where the RailCab has already been in a convoy without being unregistered in-between (noConvoy,registered,ec_registeredConvoy)).

The resulting automaton describes the synthesized component behavior of the RailCab component. We have not yet ensured, however, that the externally visible behavior of each of the role automata is preserved. This is described in the next section.

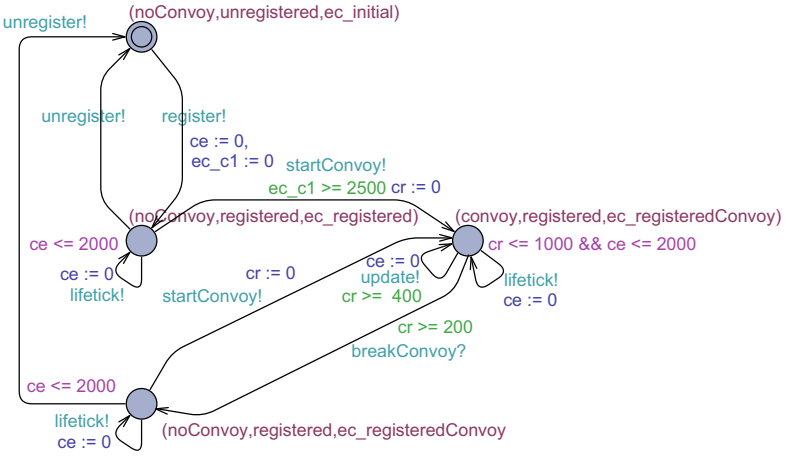

Fig. 10. Event Composition Rule eca_2 Applied to the Timed Automaton Depicted in Figure 9

6 Preserving Role Behavior

After composition rules have been applied to the parallelly composed timed automaton, it is not ensured anymore that the visible behavior of each of the particular role automata is still preserved. Assume for example, the application of an additional state composition rule $r_2 = \neg((registered, true) \wedge (convoy, cr > 100))$ to the composed timed automaton given in Figure 9. This results in a new location invariant (cr<=100 && ce<=2000) for the location (convoy,registered). As a consequence, the outgoing breakConvoy? transition can never be enabled, as its time guard cr>=200 can never evaluate to true. Accordingly, the relevant behavior of the convoy role is not anymore included in the composition conform automaton. Furthermore, note that this is not always trivial to see when specifying composition rules, as some relevant behavior is removed not before two or more rules are applied. The rule r_2 applied on the original parallel composition (see Figure 8), for example, would not remove the executability of the breakConvoy? transition, as the automaton could switch to (unregistered,convoy) to execute the breakConvoy? transition.

In order to preserve the relevant role behavior, we need to ensure that in the refined component behavior all timed safety properties and all untimed liveness properties are preserved. This would imply that no deadlines of the original role automata are violated while still all events of the original automata are (in the correct order) visible within the original time interval. If both of these properties are preserved, we say that the refined component behavior is *role conform*. In the following, we give a sketch of a proof for role conform component behavior. A detailed proof is discussed in [22].

For *preserving timed safety properties*, we have defined the composition rule formalism exactly the way that neither any time interval can be widened nor

can additional events be added to the refined behavior. This means that composition rules can only restrict the time intervals of existing behavior or can remove certain state combinations completely. Thus, all timed safety properties are inherently preserved by the synthesis procedure.

For *preserving untimed liveness properties*, we have to ensure that each path of each single role automaton still exists in the refined (parallel composed) component automaton. This problem can be split up into analyzing the offered behavior of each refined component automaton location (cf. *protocol conformance* in [4]). This means that we verify that each refined location offers the same sending and receiving events as each of the corresponding role automaton locations. In the refined automaton of the RailCab component (Figure 9), for example, we have to verify for the (noConvoy,unregistered) location that it offers a startConvoy! event for the rear role automaton and a register! event for the registree role automaton.

The *offered behavior*, however, is defined such that it does not require the concerned location to have a direct outgoing transition with the corresponding event. Instead we also allow for transitions in-between, which are triggered by events of other role automata. This is possible because, for one particular role, the behavior of other roles is invisible. In the refined RailCab component automaton this means that the (noConvoy,unregistered) location also offers a startConvoy! event through the register! transition which originates from the registree automaton.

In addition to that we analyze timed systems. Therefore, we have to take the timing information in terms of clock values of the automaton into account. We cope with this by constructing the *zone automaton* [18,23] of the refined component automaton and verifying the offered behavior of each *zone location* instead of the automaton location. For this we also include the timing information of each original role automaton location in terms of location invariants and time guards of outgoing transitions in the analysis. The analysis is finally performed by applying operations on zones (cf. [18,23]) and comparing the offered events of each zone location with the offered events of each corresponding role automaton location in the time interval of the zone location.

If each zone location offers the required behavior, we also preserve all untimed liveness properties of the role behaviors and, thus, the refined component behavior is a correct refinement of the parallel composition of the particular role behaviors[5]. If this is not the case, one or more of the specified composition rules violate the externally visible behavior of at least one of the role behaviors. In this case the developer must either adapt the composition rules or go back to the specification of the corresponding real-time coordination patterns.

7 Related Work

Work which is related to our approach exists in the field of controller synthesis as well as in the field of component-based software development.

[5] The correct refinement is defined by a weaker form of a (timed) bisimulation equivalence [24,25] which we call observational timed bisimulation.

The field of controller synthesis [26,27,28] deals with the problem of synthesizing a behavioral model for a *controller* which interacts with some *environment*. In a controller, interaction is specified through alternating actions between the controller and the environment. Consequently, for the behavioral model a special type of timed automaton, a *timed game automaton* [26], is applied. In a timed game automaton, transitions are partitioned into those controllable by the controller and those controllable by the environment.

There exist a number of approaches for the controller synthesis of system and component-level behavior models from system specifications which considers no time (e.g. [29,30]). Current work in this domain focuses on synthesis approaches based on modal transition systems (e.g.[31]). The motivation of these approaches is to capture the possible system or component implementations. In general, these approaches are also able to restrict the forbidden behavior by properties.

As we presented in [32], we divide the specification in two phases. First, we specify and analyze the protocol behavior independently from the concrete application of a component which results in independent pattern role automata. These behavior, which we can synthesize by our parameterized synthesis approach [33,34,32], is multiple applicable by different component implementations. In a second step, we specify the restrictions for the different component implementations and synthesize the component behavior by considering these restrictions and a refinement relation which preserves the formal verification results of the protocol behavior.

Therefore, the main difference to our synthesis approach is that the given behavioral model of controller synthesis does not take a compositional character of this model into account as this is not necessarily given in the underlying controller behavior. As the compositional character is mandatory for safety critical systems to be able to handle the complexity especially for the analysis, these approaches are rather not appropriate. In our approach this is given by the independent pattern role automata. In the controller synthesis approach, the compositionality can consequently also not be considered for the specification of the properties which have to be synthesized. Altogether, this results in a different equivalence relation between the original and the synthesized model which in turn results in different synthesis algorithms. Furthermore, none of these approaches takes all the relevant characteristics of safety critical systems into account that are time, safety and bounded liveness properties.

In [4], Giese and Vilbig present a synthesis procedure for the behavior of interacting components. While the basic idea of their approach and our approach is the same, the main goal of the synthesis differs. Giese and Vilbig propose to synthesize a maximal consistent component behavior which allows for representational non-determinism by the explicit use of τ-transitions representing internal component behavior. Our goal, on the other hand, is to synthesize a correct refined component behavior with respect to safety and liveness properties where the behavior of other ports is treated as internal component behavior. Furthermore, we employ real-time behavioral models as input specifications in order to suit the requirements of safety critical systems.

8 Conclusion and Future Work

In this paper we proposed an approach to automatically synthesize the behavior of components applied in critical systems. Therefore, we propose to specify dependencies between several role behaviors separately by means of composition rules. Additionally, we defined a procedure to automatically integrate the composition rules for a given set of role behaviors. Afterwards it is checked that the resulting component behavior refines each of the role behaviors properly. We exemplify the approach by extending the MECHATRONIC UML. A first prototype and an evaluation of our approach is presented in [32,35]. For future work we plan to perform an exhaustive evaluation of the approach in the RailCab project and industrial applications. This way, it could also be evaluated if the proposed composition rule formalism is sufficient to specify dependencies between several coordination roles in a multi-cast setting.

References

1. Bosch, J., Szyperski, C.A., Weck, W.: Component-Oriented Programming. In: Malenfant, J., Moisan, S., Moreira, A.M.D. (eds.) ECOOP 2000 Workshops. LNCS, vol. 1964, pp. 55–64. Springer, Heidelberg (2000)
2. Dijkstra, E.: A Discipline of Programming. Prentice-Hall Series in Automatic Computation (1976)
3. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J.: Aspect-Oriented Programming. In: Aksit, M., Matsuoka, S. (eds.) ECOOP 1997. LNCS, vol. 1241, pp. 220–242. Springer, Heidelberg (1997)
4. Giese, H., Vilbig, A.: Separation of Non-Orthogonal Concerns in Software Architecture and Design. Software and System Modeling (SoSyM) 5(2), 136–169 (2006)
5. Tarr, P., Ossher, H., Harrison, W., Sutton Jr., S.M.: N Degrees of Separation: Multi-Dimensional Separation of Concerns. In: ICSE 1999: Proceedings of the 21st International Conference on Software Engineering, pp. 107–119. ACM, New York (1999)
6. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture, vol. 1. John Wiley & Sons, Chichester (1996)
7. Garlan, D., Perry, D.: (introduction to the) Special Issue on Software Architecture. IEEE Transactions on Software Engineering 21(4) (April 1995)
8. Gruber, T.R.: A Translation Approach to Portable Ontology Specifications. Knowl. Acquis. 5(2), 199–220 (1993)
9. Milner, R.: Communication and Concurrency. Prentice-Hall, Inc., Upper Saddle River (1989)
10. Selic, B.: Real-Time Object-Oriented Modeling (room). In: 2nd IEEE Real-Time Technology and Applications Symposium (RTAS 1996), Boston, MA, USA, June 10-12, p. 214. IEEE Computer Society, Los Alamitos (1996)
11. Jackson, E.K., Sztipanovits, J.: Using Separation of Concerns for Embedded Systems Design. In: EMSOFT 2005: Proceedings of the 5th ACM International Conference on Embedded Software, pp. 25–34. ACM, New York (2005)
12. Giese, H., Tichy, M., Burmester, S., Schäfer, W., Flake, S.: Towards the Compositional Verification of Real-Time UML Designs. In: Proc. of the 9th European Software Engineering Conference Held Jointly with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE-11), September 2003, pp. 38–47 (2003)

13. Giese, H., Burmester, S., Schäfer, W., Oberschelp, O.: Modular Design and Verification of Component-Based Mechatronic Systems with Online-Reconfiguration. In: Proc. of 12th ACM SIGSOFT Foundations of Software Engineering 2004 (FSE 2004), Newport Beach, USA, pp. 179–188. ACM Press, New York (2004)
14. Giese, H., Burmester, S.: Real-Time Statechart Semantics. Technical Report tr-ri-03-239, Lehrstuhl für Softwaretechnik, Universität Paderborn, Paderborn, Germany (June 2003)
15. Alur, R., Dill, D.L.: Automata for Modeling Real-time Systems. In: Paterson, M. (ed.) ICALP 1990. LNCS, vol. 443, pp. 322–335. Springer, Heidelberg (1990)
16. Henzinger, T.A., Nicollin, X., Sifakis, J., Yovine, S.: Symbolic Model Checking for Real-Time Systems. In: Proceedings of the Seventh Annual Symposium on Logic in Computer Science (LICS), pp. 394–406. IEEE Computer Society Press, Los Alamitos (1992)
17. Pettersson, P.: Modelling and Verification of Real-Time Systems Using Timed Automata: Theory and Practice. PhD thesis, Department of Computer Systems, Uppsala University (February 1999)
18. Bengtsson, J.E., Yi, W.: Timed Automata: Semantics, Algorithms and Tools. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) Lectures on Concurrency and Petri Nets. LNCS, vol. 3098, pp. 87–124. Springer, Heidelberg (2004)
19. Lamport, L.: Proving the Correctness of Multiprocess Programs. IEEE Transactions on Software Engineering SE-3(2), 125–143 (1977)
20. Henzinger, T.A.: Sooner is Safer than Later. Information Processing Letters 43(3), 135–141 (1992)
21. Yi, W., Pettersson, P., Daniels, M.: Automatic Verification of Real-time Communicating Systems by Constraint-solving. In: Hogrefe, D., Leue, S. (eds.) Proceedings of the 7th IFIP WG6.1 International Conference on Formal Description Formal Techniques, Berne, Switzerland. IFIP Conference Proceedings, vol. 6, pp. 243–258. Chapman & Hall, Boca Raton (1994)
22. Eckardt, T., Henkler, S.: Synthesis of Reconfiguration Charts. Technical Report tr-ri-10-314, University of Paderborn, Paderborn, Germany (January 2010)
23. Alur, R.: Timed Automata. In: NATO-ASI 1998 Summer School on Verification of Digital and Hybrid Systems (1998)
24. Clarke, E.M., Grumberg, O., Peled, D.: Model Checking (January 2000)
25. Tripakis, S., Yovine, S.: Analysis of Timed Systems Using Time-Abstracting Bisimulations. Formal Methods in System Design 18(1), 25–68 (2001)
26. Asarin, E., Maler, O., Pnueli, A.: Symbolic Controller Synthesis for Discrete and Timed Systems. In: Antsaklis, P.J., Kohn, W., Nerode, A., Sastry, S.S. (eds.) HS 1994. LNCS, vol. 999, pp. 1–20. Springer, Heidelberg (1995)
27. Altisen, K., Tripakis, S.: Tools for Controller Synthesis of Timed Systems. In: Pettersson, P., Yi, W. (eds.) Proceedings of the 2nd Workshop on Real-Time Tools (RT-TOOLS 2002) (August 2002)
28. Geist, S., Gromov, D., Raisch, J.: Timed Discrete Event Control of Parallel Production Lines with Continuous Outputs. Discrete Event Dynamic Systems 18(2), 241–262 (2008)
29. Harel, D., Kugler, H., Pnueli, A.: Synthesis Revisited: Generating Statechart Models from Scenario-Based Requirements. In: Kreowski, H.-J., Montanari, U., Orejas, F., Rozenberg, G., Taentzer, G. (eds.) Formal Methods in Software and Systems Modeling. LNCS, vol. 3393, pp. 309–324. Springer, Heidelberg (2005)
30. Whittle, J., Schumann, J.: Generating Statechart Designs from Scenarios. In: ICSE 2000: Proceedings of the 22nd International Conference on Software Engineering, pp. 314–323. ACM, New York (2000)

31. Uchitel, S., Brunet, G., Chechik, M.: Synthesis of Partial Behavior Models from Properties and Scenarios. IEEE Transactions on Software Engineering 35, 384–406 (2009)
32. Henkler, S., Greenyer, J., Hirsch, M., Schäfer, W., Alhawash, K., Eckardt, T., Heinzemann, C., Löffler, R., Seibel, A., Giese, H.: Synthesis of Timed Behavior from Scenarios in the Fujaba Real-Time Tool Suite. In: Proceedings of the 31st International Conference on Software Engineering (ICSE 2009), Vancouver, Canada, Washington, DC, USA, May 16-24, pp. 615–618. IEEE Computer Society, Los Alamitos (2009)
33. Giese, H., Klein, F., Burmester, S.: Pattern Synthesis from Multiple Scenarios for Parameterized Real-Timed UML Models. In: Leue, S., Systä, T.J. (eds.) Scenarios: Models, Transformations and Tools. LNCS, vol. 3466, pp. 193–211. Springer, Heidelberg (2005)
34. Giese, H., Henkler, S., Hirsch, M., Klein, F.: Nobody's Perfect: Interactive Synthesis from Parametrized Real-Time Scenarios. In: Proc. of the 5^{th} ICSE 2006 Workshop on Scenarios and State Machines: Models, Algorithms and Tools (SCESM 2006), Shanghai, China, May 2006, pp. 67–74. ACM Press, New York (2006)
35. Eckardt, T., Henkler, S.: Synthesis of Component Behavior. In: Gorp, P.V. (ed.) Proceedings of the 7th International Fujaba Days, November 2009, pp. 1–5. Eindhoven University of Technology, The Netherlands (2009)