A Design Framework for Concurrent Hybrid Controllers with Safety and Stability Annotations

by

Werner Damm, Eike Möhlmann, Astrid Rakow

AVACS Technical Report No. 105

April, 2016

ISSN: 1860-9821

Publisher: Sonderforschungsbereich/Transregio 14 AVACS
(Automatic Verification and Analysis of Complex Systems)
Editors: Bernd Becker, Werner Damm, Bernd Finkbeiner, Martin Fränzle,
Ernst-Rüdiger Olderog, Andreas Podelski
ATRs (AVACS Technical Reports) are freely downloadable from `www.avacs.org`

A Design Framework for Concurrent Hybrid Controllers with Safety and Stability Annotations

Werner Damm, Eike Möhlmann, Astrid Rakow
Carl von Ossietzky University of Oldenburg
Department of Computer Science
D-26111 Oldenburg, Germany
{werner.damm,eike.moehlmann,astrid.rakow}@informatik.uni-oldenburg.de

April 29, 2016

At loosely coupled systems, the composed system has to accomplish a task that may depend on several of its subcomponents while little coordination between them is necessary. In the following we present an assume guarantee framework for hybrid systems which implements design principles for compositional reasoning for loosely coupled controllers on a common plant. The framework introduces time bounded promises, so called *events*, for coordination between concurrent controllers. To bridge the gap between design and implementation level, signal latencies and potential loss of events are modelled. Safety as well as stability properties are deferrable from subcomponents to the composed system.

Keywords:
control component, interface specification, assume guarantee reasoning, stability, Lyapunov function

1 Introduction

A component can be thought of as an encapsulation of an implementation that interacts with its environment via its interface and that can be (re-)used in certain deployment contexts, where it guarantees to provide its services. In many control applications, the controlled system continuously evolves whereas its controllers are triggered by discrete conditions and perform nearly instantaneous actions. Which leads to so called hybrid system models, where continuous as well as discrete dynamics can be reflected.

In [3] Damm et al. proposed a library based design methodology for constructing hybrid controllers. Interfaces and interface satisfaction were defined to support cataloguing components for later reuse. Their framework defines *transition composition*,

a way of sequential orchestration of controllers, that preserves safety and stability properties. Verification conditions were identified to alleviate the application of automatic verification tools. Our research aims to extend this approach by concurrent orchestration of control components.
The importance of contract-based compositional design – as e.g. imposed by interface satisfaction – has long been recognized as an instrument to reduce the number of late integration errors [14] and to boost re-use of components [7]. Contract-based specifications have been integrated into the Core meta Model underlying the CESAR reference technology platform for critical system design, which is recognized by the Artemis Industrial Association as the unifying platform for building a suite of interoperable methods and tools covering the complete system life cycle for critical systems [1]. Projects like Speeds[1] have provided formal contract based component interface specifications for real-time and safety requirements (as in [5, 6]).

There certainly is a trade-off between the benefits of re-using components and the cost payed for decoupling a component from its environment in order to make it reusable. We focus on loosely coupled systems, where one component needs to know little about its concurrent environment. In particular, our framework provides interface specifications to describe lightweight base contracts and dynamic contracts. An interface specifies allowed component deployment contexts, i.e. , the environments where it can be used. The interface also specifies which services the component provides, given it is used in a valid environment. We suggest to use *base* assumptions/guarantees and *event* assumptions/guarantees. Base assumptions and base guarantees describe the normal modus operandi (e.g. they may reflect physical laws), while event assumptions and guarantees may be used to describe situational achievements. They can be used to inform other components about situations where a component relaxes its assumptions and strengthens its guarantees. Event assumptions and guarantees imply a kind of dynamic contract. In order to gain tractable verification conditions, we specify assumptions and guarantees simply as intervals of reachable variable ranges. Additionally, interfaces declare guaranteed safety and stability properties, which are propagated to the composed system, but are not used within the assume-guarantee reasoning.

As an example of a loosely coupled control application we studied an advanced driver assistance system (ADAS) [4, 8, 10]. In the following we will use this case study as a running example. We briefly introduce the main characteristics of the system: The ADAS had to
(o1) maintain a centrifugal force comfortable for a driver,
(o2) bring and then keep the car on the center of its lane,
(o3) control the speed also considering driver requests for a certain speed value.
The ADAS was composed of two concurrent controllers, a velocity controller, VC, and a steering controller, SC. The velocity controller is responsible for its *local objective* of controlling the speed, the steering controller is responsible for lane keeping and together they have to accomplish the *global objective* of guaranteeing a comfortable centrifugal force. The global objective implies a loose coupling between VC and SC, because maintaining a comfortable centrifugal force means that a car cannot be arbitrarily fast in a narrow turn and a fast car may not decide to change its steering

[1]IST Project 03347, see www.speeds.eu.com

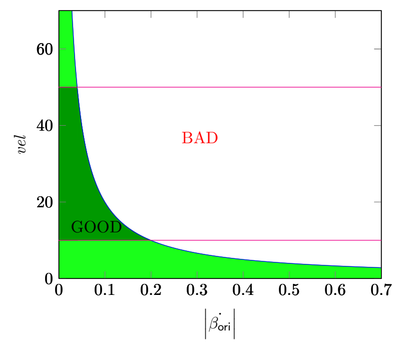


Figure 1: Velocity and steering control is coupled via the centrifugal force, $|F_{\mathsf{cntrif}}| = |mass \cdot vel_{\mathsf{cur}} \cdot \dot{\beta_{\mathsf{ori}}}|$. A driver feels comfortable within at centrifugal forces within the GOOD range.

angle too abruptly. But as steering angles usually are small, the coupling is quite loose, which is illustrated in Fig. 1.
In this document we first give the formal basis of our framework in terms of hybrid input output automata (HIOA) (cf. Sect. 2). Then, we give an overview of the basic conceptual ingredients, *controller, plant and event communication,* in Sect. 3. We get in more detail on events in Sect. 4. We proof an assume-guarantee reasoning result for safety properties and stability propertie in Sect. 5. Then in Sect. 6 we formally introduce interfaces and define an interface satisfaction relation. We show how an interface of a composed component can be derived from the interfaces of its constituent components.

Before drawing the conclusions, we give an outlook on support via tool and techniques for establishing interface annotations: We highlight the relation of stability annotations of our interfaces and stability theory. Therefore we present an excert of the disseration [9] that sketches how Lyapunov functions can be used as certificates for stability properties within our framework.

2 Basic Definitions

In this paper we use *linear temporal logic (LTL)* and its extension *metric temporal logic* MTL to specify properties. The reader is referred to [2] and [13] for an introduction of these formalisms. Let φ be an LTL formula. We denote the *globally* operator as $\Box$ and *eventually* as $\Diamond$. The order of precedence is $\{\neg\}, \{\Box, \Diamond\}, \{\wedge\}, \{\vee, \rightarrow, \leftrightarrow\}, \{\mathsf{U}, \mathsf{UNLESS}\}$, where φ UNLESS ψ abbreviates $(\varphi \mathsf{U} \neg\psi) \vee \Box\varphi$. MTL augment the modalities of LTL with time constraints. That is we can use U_I, for an interval $I \subseteq (0, \infty)$ of reals with endpoints in $\mathbb{N} \cup \infty$, and the derived operators $\Diamond_I$ and $\Box_I$. As a short hand, we annotate U_I for an interval $I = [0, b]$ also as $\mathsf{U}_{\leq b}$

The following definition introduces hybrid automata (HIOA), a variant of hybrid automata apt to provide a formal basis for our component framework. An HIOA distinguishes between input, output and local variables and explicitly refers to an assumption.

Definition 1 ***Hybrid I/O Automaton*** *A* hybrid I/O automaton *(HIOA) H given by* $(\mathbb{M}, Var^{loc}, Var^{in}, Var^{out}, R^{\mathsf{dscr}}, R^{\mathsf{cnt}}, \Phi^{\mathit{init}}, \Theta^{\mathit{inv}})$

1. $\mathbb{M}$ *is a finite set of* modes.
2. Var^{loc}, Var^{in} *and* Var^{out} *are disjoint sets of* local, input and output variables *over* $\mathbb{R}$. *We denote* $Var := Var^{loc} \cup Var^{in} \cup Var^{out}$ *and the controlled variables as* $Var^C := Var^{loc} \cup Var^{out}$.

3. Φ^{init} *is a first-order predicate over Var and a variable md_var which takes values in* $\mathbb{M}$, *describing all combinations of* initial states and modes.
4. Θ^{inv} *is a mapping that associates with each mode* $m \in \mathbb{M}$ *a* local invariant $\Theta^{\mathsf{inv}}(m)$, *which is a quantifier-free formula over Var.*
5. R^{dscr} *is the* discrete transition relation *with elements* $(m, G, \mathcal{A}, m')$ *where* $m, m' \in \mathbb{M}$, *G is a first-order predicate over Var.* $\mathcal{A}$ *is a first-order predicate over* $Var \cup Var^{C+}$, *where* Var^{C+} *holds decorated variants of variables in* Var^C *and represents the value after the discrete update.* R^{dscr} *consists of disjoint sets* R^{dscr}_U, *the urgent transitions, and* R^{dscr}_L, *the lazy transitions.*
6. R^{cnt} *is a function defining for every* $m \in \mathbb{M}$ *a first-order predicate of the form* $\bigwedge \overset{\bullet}{v} \bowtie e$, *where* $v \in Var^C, \bowtie \in \{\leq, =, \geq\}$, *and e is a real linear arithmetic expression over Var.* $R^{\mathsf{cnt}}(m)$ *defines* differential inclusions *modeling the continuous evolution of mode m.*

For graphical representation, lazy transitions will be labeled with $\phi/\mathcal{A}$, and urgent transitions $u{:}\,\phi/\mathcal{A}$, with $\mathcal{A}$ either in predicate or assignment notation. If a set of assignments is empty, it will be left out, meaning that all variables in *Var* remain unchanged upon switching. For a predicate ϕ, $\phi[\mathcal{A}]$ is the result of the substitution induced by $\mathcal{A}$ on ϕ.

We now give the formal definition of runs of a HIOA H capturing the evolution of modes and the real-valued variables over time. To this end, we consider the *continuous time domain* $Time := \mathbb{R}_{\geq 0}$ of non-negative reals and time sequences, $(\tau_i) \in Time^\omega \cup Time^*$. $(\tau_i)_{i\in\mathbb{N}} \in Time^\omega$ is an *infinite time sequence* where $\tau_0 = 0$ and (τ_i) monotonically increases. A *finite time sequence* is a finite subsequence $(\tau_i)_{\{i\in 1,\ldots,n | n\in\mathbb{N}\}}$ of an infinite time sequence $(\tau_i)_{i\in\mathbb{N}}$. The *mode function* is given as $M_i : Time \to \mathbb{M}$, and function $\mathbf{X}_i : Time \to \mathbb{R}^{|Var|}$, with $\mathbf{X}_i(t) = [\mathbf{X}^C_i(t)^T, \mathbf{X}^I_i(t)^T]^T$, describes for each time point $t \in [\tau_i, \tau_{i+1}]$ the current value of all variables in *Var*. The order of variables in each of the two sub-vectors is fixed but arbitrary. Here, $\mathbf{X}^C_i$ covers all controlled variables and $\mathbf{X}^I_i$ all input variables. Furthermore, the vector $\pi_i(t) = [M_i(t), \mathbf{X}^C_i(t)^T, \mathbf{X}^I_i(t)^T]^T$ describes the overall (hybrid) state of a HIOA. For a vector $\mathbf{X} \in \mathbb{R}^{|Var|}$ we use $\mathbf{X}|_{Var'}$ to denotes its projection onto $Var' \subseteq Var$. We also denote the elementwise projection of state sequences, in this way.

For simplicity, we identify a state vector $\mathbf{X}$ with the corresponding valuation of the variables in $\{md\_var\} \cup Var$ and for a state predicate Ψ, we use the notation $\mathbf{X} \models \Psi$, if the valuation associated with $\mathbf{X}$ fulfills Ψ. We also define a substitution based on vectors, such that the predicate $\Psi[Var/\mathbf{X}]$ is the predicate Ψ with the variable values given by the valuation $\mathbf{X}$ substituting the corresponding variables in *Var* and $\Psi[Var/Var^+]$ is the predicate Ψ with decorated variables substituting the undecorated variables in *Var*. The vectors $\mathbf{X}^C(t)$ and $\mathbf{X}^I(t)$ are handled in the same manner. The *time derivative* of $\mathbf{X}^I(t)$ is denoted by $d\mathbf{X}^I/dt(t)$ or $\overset{\bullet}{\mathbf{X}}_i(t)$.

During a run of a HIOA H, values of controlled variables are constrained by H and φ^{hyb}, while input variables changes are only under the the hybrid predicate's control. We will usually capture the effect of the environment in form of hybrid predicates. A hybrid predicate φ^{hyb} specifies discrete, continuous and invariant aspects of the environment: $\varphi^{hyb} = (\neg\mathsf{flow} \Rightarrow \varphi^{hyb}_{dscr}) \wedge (\mathsf{flow} \Rightarrow \varphi^{hyb}_{cnt}) \wedge \varphi^{hyb}_{inv}$. It will be used to specify that inputs are allowed to change discretely satisfying φ^{hyb}_{dscr} and continuously satisfying φ^{hyb}_{cnt}, and at all times φ^{hyb}_{inv} has to hold.

Definition 2 ***Interval Conjunction, Hybrid Predicate*** *Given a set of variables Var.*

An interval conjunction *is a predicate φ of the form $\bigwedge_{v \in Var} v \in I_v$ with $I_v \subseteq \mathbb{R}$ an interval. φ's* restriction *to Var', $\varphi|_{Var'}$, denotes the predicate true $\wedge \bigwedge_{v \in Var' \cap Var} v \in I_v$.*

An hybrid predicate *has the form $\varphi^{hyb} = (\neg \mathsf{flow} \Rightarrow \varphi^{hyb}_{dscr}) \wedge (\mathsf{flow} \Rightarrow \varphi^{hyb}_{cnt}) \wedge \varphi^{hyb}_{inv}$, where*

- *flow is an atomic proposition which is true during a flow and false otherwise,*
- *φ^{hyb}_{dscr} is of the form $\bigwedge_{v \in Var}(\bigvee_{i \in \{1,\ldots,n\}} v \in I^i_v)$,*
- *φ^{hyb}_{inv} is of the form $\bigwedge_{v \in Var}(v \in I_v)$,*
- *φ^{hyb}_{cnt} is of the form $\bigwedge_{v \in Var}(\dot{v} \in I_v)$, where I^i_v and I_v are intervals of reals.*

We extend the restriction of interval predicates *to hybrid predicates by restricting the interval predicates of each conjunct.*

Note, that with $I_v = \mathbb{R}$ we can express that certain variables are unconstraint by φ^{hyb}.

Definition 3 ***Trajectory/Run of an HIOA*** *Let (τ_i) be a time sequence and φ^{hyb} be an hybrid predicate.*

Let Φ^{start} be a first-order predicate over $Var \cup \mathbb{M}$ of a hybrid I/O automaton

$$H = (\mathbb{M}, Var^{loc}, Var^{in}, Var^{out}, R^{\mathsf{dscr}}, R^{\mathsf{cnt}}, \Phi^{\mathsf{init}}, \Theta^{\mathsf{inv}})$$

A sequence (π_i) is a trajectory of H from Φ^{start} obeying φ^{hyb} with

$$\mathbf{X}_i = \begin{bmatrix} \mathbf{X}^C_i \\ \mathbf{X}^I_i \end{bmatrix},$$

where $M_i : [\tau_i, \tau_{i+1}] \to \mathbb{M}$, $\mathbf{X}^C_i : [\tau_i, \tau_{i+1}] \to \mathbb{R}^{|Var^C|}$, and $\mathbf{X}^I_i : [\tau_i, \tau_{i+1}] \to \mathbb{R}^{|Var^{in}|}$ are continuously differentiable functions
when it satisfies

(1) start states: $\pi_0(0) \models \Phi^{\mathsf{start}} \wedge \varphi^{hyb}$

(2) mode switching times: $\forall i \in \mathbb{N}\ \forall t \in [\tau_i, \tau_{i+1}) : M_i(t) = M(\tau_i)$

*(3) continuous evolution: $\forall i \in \mathbb{N}\ \forall t \in (\tau_i, \tau_{i+1}) :$
$(d\mathbf{X}^C_i/dt(t), \mathbf{X}_i(t)) \models R^{\mathsf{cnt}}(M_i(\tau_i)) \wedge d\mathbf{X}_i/dt(t) \models \varphi^{hyb}_{cnt}$*

(4) invariants: $\forall i \in \mathbb{N}\ \forall t \in [\tau_i, \tau_{i+1}) : \mathbf{X}_i(t) \models \Theta^{\mathsf{inv}}(M_i(t)) \wedge \mathbf{X}_i(t) \models \varphi^{hyb}_{inv}$

(5) urgency: $\forall i \in \mathbb{N}\ \forall t \in [\tau_i, \tau_{i+1})\ \forall (M_i(t), \phi, \mathcal{A}, m') \in R^{\mathsf{dscr}}_U$ we have that $\mathbf{X}_i(t) \not\models \phi$

(6) discrete evolution: $\forall i \in \mathbb{N}$:

$$\begin{aligned}
&(M_i(\tau_{i+1}) = M_{i+1}(\tau_{i+1}) \wedge \mathbf{X}_i(\tau_{i+1}) = \mathbf{X}_{i+1}(\tau_{i+1})) && (\textbf{stuttering})\\
&\vee(M_i(\tau_{i+1}) = M_{i+1}(\tau_{i+1}) \wedge \mathbf{X}_i^C(\tau_{i+1}) = \mathbf{X}_{i+1}^C(\tau_{i+1}) && (\textbf{discr. input change})\\
&\qquad \wedge \mathbf{X}_{i+1}^I(\tau_{i+1}) \models \varphi_{dscr}^{hyb})\\
&\vee(\exists(m, \phi, \mathcal{A}, m') \in R^{\mathsf{dscr}} : M_i(\tau_{i+1}) = m \wedge M_{i+1}(\tau_{i+1}) = m'\\
& && (\textbf{transition execution})\\
&\qquad \wedge \mathbf{X}_{i+1}(\tau_{i+1}) \models \varphi_{dscr}^{hyb}\\
&\qquad \wedge \mathbf{X}_{i+1}(\tau_{i+1}) \models \mathcal{A}[\mathit{Var}^C/\mathbf{X}_i^C(\tau_{i+1})]\\
&\qquad \wedge \mathbf{X}_i^I(\tau_{i+1}) = \mathbf{X}_{i+1}^I(\tau_{i+1})).
\end{aligned}$$

A run *of H is a trajectory from Φ^{init}.*

A time sequence (τ_i) of trajectory (π_i) is called a sequence of switching times. *Define $\pi(t)$ as the $\pi_i(t)$, such that $\forall j > i : \tau_j > t$, i.e., $\pi(t)$ is the system state after all (possibly super-dense) switches that occur at time t. Define $\mathbf{X}(t), M(t), \mathbf{X}^C(t)$ and $\mathbf{X}^I(t)$ in the same manner.*

Note, that in (**discr. input change**) values of input variables may change, while in (**transition execution**) the values of input variables are fixed. Hence concurrent discrete changes – within the component and its environment – are sequentialized, while the continuous variables evolve concurrently.

Clause (1) anchors a trajectory in a set of overall (hybrid) states. The time sequence $(\tau_i)_{i\in\mathbb{N}}$ identifies the points in time, at which mode-switches may occur, which is expressed in Clause (2). Only at those points discrete transitions (having a noticeable effect on the state) may be taken. On the other hand, it is not required that any transition fires at some point τ_i, which permits to cover behaviors with a finite number of discrete switches within the framework above. Clause (3) forces all variables to actually obey their respective differential inclusions. Clause (4) requires, for each mode, the valuation of continuous variables to meet both local and global invariants while staying in this mode. Clause (5) forces an urgent discrete transition to fire when its trigger condition becomes true. The effect of a discrete transition is described by Clause (6). Whenever a discrete transition is taken, local and output variables may be assigned new values. If there is no such assignment, the variable maintains its previous value, which is determined by taking the limit of the trajectory of the variable as t converges to the switching time τ_{i+1}. Define $\pi(t)$ as the $\pi_i(t)$, such that $\forall j > i : \tau_j > t$, i.e., $\pi(t)$ is the system state after all (possibly super-dense) switches that occur at time t. Define $\mathbf{X}(t), M(t), \mathbf{X}^C(t)$ and $\mathbf{X}^I(t)$ in the same manner.

Definition 4 *$H \models_{(\phi^{assm})} \varphi$ Given an MTL formula φ and a HIOA H and an hybrid predicate φ^{hyb}.*

H satisfies φ within φ^{hyb}, $H \models_{(\varphi^{hyb})} \varphi$, iff every run (π_i) of H within φ^{hyb} satisfies φ.

Note 1 *We will also interpret hybrid predicates as MTL properties. Therefore we require that φ^{hyb}_{dscr} holds for Var^C only at the start of each continous flow. In contrast, all the time φ^{hyb}_{dscr} on Var^{in}, φ^{hyb}_{inv}, φ^{hyb}_{cnt} have to hold.*

Definition 5 ***Prefix, Deadlock, Livelock, Non-Zeno*** *Let H be an HIOA. A trajectory is* finite *if it has finitely many switching times, and* infinite*, if it has infinitely many switching times. It is called* maximal*, if it is not a proper prefix of any other trajectory of H.*

A deadlock state $\mathbf{X}_{dlk}$ is a reachable state of H where neither continuous evolution nor discrete evolution is possible, i.e., given $\mathbf{X}_{dlk}$ is reached at time τ and at mode m, it holds for all functions $\mathbf{X}_i$

- $\exists \tau_2 : \tau < \tau_2 : \forall t \in [\tau, \tau_2) :$ (no continuous evolution)
  $(d\mathbf{X}^I/dt(t), X^I(t)) \models \varphi^{hyb} \wedge\ \exists \tau_3, \tau < \tau_3 \leq \tau_2 :$
  $\forall t' \in (\tau, \tau_3) : (d\mathbf{X}^C/dt(t'), \mathbf{X}(t')) \not\models (R^{cnt}(m) \wedge \Theta^{inv}(m)),$
- $\forall (m, \phi, \mathcal{A}, m') \in R^{dscr}, \forall \mathbf{X} :$ (no discrete evolution)
  $\big(\mathbf{X}_{dlk} \models \phi \wedge \mathbf{X}^+ = \mathbf{X}^C[Var^C/Var^{C^+}] \wedge (X^+, \mathbf{X}_{dlk}) \models \mathcal{A}\big) \Rightarrow \mathbf{X} \not\models \Theta^{inv}(m')).$

A livelock *or* zeno *trajectory of H is an infinite trajectory of H of finite time duration and infinite transition executions.*

H is lockfree *or guarantees* time progress*, iff none of its runs reach a deadlock state or are livelock trajectory. We say H is* lockfree *under φ^{hyb} for the future of Δ^{time}_{lat}, iff time progresses for Δ^{time}_{lat} at any state $\mathbf{X}$ reached on a run under ϕ.*

As we study physical systems, we study systems where time progresses. Within our compositional framework it suffices to require that any (open) component itself is lockfree, i.e., does not stall time. We do not care whether its assumed most general environment stalls time, therefore we define a livelock trajectory as a trajectory of infinite transition executions and do not restrict the input behavior. It will follow compositionally that the trajectories of the closed composed system are non-Zeno.

Lemma 1 Projection and Conjunction Let φ be a temporal logic formula referring only to variables in $Var' \subseteq Var$ of the overall state space $\mathbf{X} = \mathbb{R}^{|Var|}$.
A sequence $(\mathbf{X})_{i \in \mathbb{N}} \in \mathbf{X}^* \cup \mathbf{X}^\omega$ satisfies φ iff $(\mathbf{X}')_{i\in\mathbb{N}} = (\mathbf{X}|_{Var'})_{i\in\mathbb{N}}$ satisfies φ.

Definition 6 ***Reach Set*** *For some $t \geq 0$, define a* time bounded reach set $\mathit{reach}(H, \Phi^{start}, \varphi^{hyb}, t)$ *of HIOA H from predicate Φ^{start} and within environment φ^{hyb} as the closure of*

$$\{\mathbf{X} | \exists \textit{ trajectory } (\pi) \textit{ of } H \textit{ within } \varphi^{hyb}, t \geq t' \geq 0 : \pi(0) \models \Phi^{start} \wedge \pi(t') = \mathbf{X}\}.$$

Analogously, define the unbounded reach set $\mathit{reach}(H, \Phi^{start}, \varphi^{hyb})$ *of hybrid automaton H from predicate Φ^{start} within environment φ^{hyb} as the closure of*

$$\{\mathbf{X} | \exists \textit{ trajectory } (\pi) \textit{ of } H \textit{ within } \varphi^{hyb}, t' \geq 0 : \pi(0) \models \Phi^{start} \wedge \pi(t') = \mathbf{X}\}.$$

For convenience we denote $\mathit{reach}(H, \Phi^{start}, \varphi^{hyb})$ and $\mathit{reach}(H, \Phi^{start}, \varphi^{hyb}, t)$ as $\mathit{reach}(H, \varphi^{hyb})$ and $\mathit{reach}(H, \varphi^{hyb}, t)$, if $\Phi^{start} = \varphi^{init}_H$.

The parallel composition of two HIOA H_1 and H_2 results in a new automata $H_1 \parallel H_2$ in which H_1 and H_2 synchronously evolve during a flow while their discrete updates are interleaved. For composition, we only require that local variables are disjoint from variables of the other automata. We do not demand that output and input variables of the two automata are disjoint. Hence, one automata may read the same input as the other or read the other's output or write to a shared output. The latter may cause write conflicts, but will not occur within our component framework. In our framework, usually outputs of the plant will be read by several components, while the outputs of a control component will be read only by the plant. For the later case, hiding of outputs could be applied.

Note that inputs that get connected to outputs remain as outputs but are no longer inputs.

Definition 7 ***Parallel Composition***
Let two HIOA

$$H_i = (\mathbb{M}_i, Var_i^{loc}, Var_i^{in}, Var_i^{out}, R_i^{\mathsf{dscr}}, R_i^{\mathsf{in}}, R_i^{\mathsf{cnt}}, \Phi_i, \Theta_i),$$

$i = 1,2$ *be given with* $(Var_1^{loc} \cap Var_2) \cup (Var_2^{loc} \cap Var_1) = \varnothing$. *Then the parallel composition*

$$H_1 \parallel H_2 = (\mathbb{M}, Var^{loc}, Var^{in}, Var^{out}, R^{\mathsf{dscr}}, R^{\mathsf{in}}, R^{\mathsf{cnt}}, \Phi, \Theta)$$

is given by:

- $\mathbb{M} = \mathbb{M}_1 \times \mathbb{M}_2$,
- $Var^{loc} = Var_1^{loc} \dot\cup Var_2^{loc}$,
- $Var^{out} = Var_1^{out} \cup Var_2^{out}$,
- $Var^{in} = (Var_1^{in} \cup Var_2^{in}) - Var^{out}$,
- $R^{\mathsf{cnt}}((m_1, m_2)) = R_1^{\mathsf{cnt}}(m_1) \wedge R_2^{\mathsf{cnt}}(m_2)$
- R_U^{dscr} *consists of the following transitions:*
  (1) $((m_1, m_2), \Phi_1, \mathcal{A}_1, (m_1', m_2))$ *for each* $(m_1, \Phi_1, \mathcal{A}_1, m_1') \in R_{U\ ,1}^{\mathsf{dscr}}$, *and*
  (2) transitions of the form (1) with the role of H_1 *and* H_2 *interchanged,*
- R_L^{dscr} *consists of the following transitions:*
  (1) $((m_1, m_2), \Phi_1, \mathcal{A}_1, (m_1', m_2))$ *for each* $(m_1, \Phi_1, \mathcal{A}_1, m_1') \in R_{L\ ,1}^{\mathsf{dscr}}$, *and*
  (2) transitions of the form (1) with the role of H_1 *and* H_2 *interchanged,*
- $\Phi = \Phi_1 \wedge \Phi_2$,
- $\Theta((m_1, m_2)) = \Theta_1(m_1) \wedge \Theta_2(m_2)$.

Note 2 (Renaming locals) *We require in Def. 7 that the set of local variables of one automata is disjoint from the set of variables of the other and vice versa. This condition can in general be satisfied after appropriate renaming of local variables. In the following we hence assume that local variables are renamed appropriately at composition.*

At this point we have introduced the basic notions that we need to give a semantics to our framework. In the next section we given an overview of our framework.

3 Overview

Our approach targets a basic set up (cf. Fig. 2), where a plant is given modelling a physical system and a set of controllers has to govern the plant to be safe. The controllers are loosely coupled – to achieve their local control task they do not need to exchange much information. All control components may read via their sensors the continuous plant state.

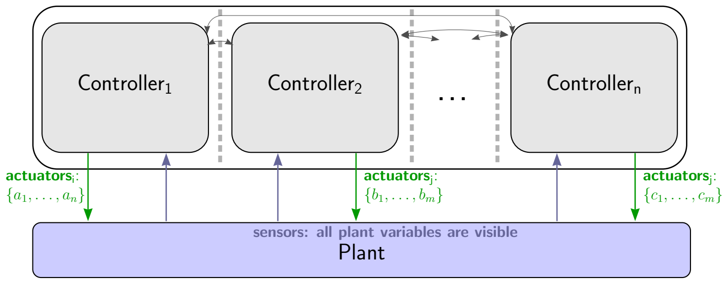

Controller$_1$
Controller$_2$
...
Controller$_n$
actuators$_i$: $\{a_1, \ldots, a_n\}$
actuators$_j$: $\{b_1, \ldots, b_m\}$
actuators$_j$: $\{c_1, \ldots, c_m\}$
sensors: all plant variables are visible
Plant

Figure 2: Application class. A common plant is governed by the parallel composite of controllers. The controllers are loosely coupled, communicate via events and have separate sets of actuators.

Control components are catalogued in a library and a composite controller can be build from existing catalogued controllers. The library catalogue lists components by their *interface specification*. A(n interface) *specification* gives an external and abstract view on a controller and hides implementation details. It specifies when a component can be used and what its services are in an abstract way. So, a controller implementation can satisfy several interface(specification)s and an interface can be implemented by several controllers.

To describe *when a control component can be used* (its deployment context), the essence of the controller's interaction with its environment has to be captured. For safety critical systems, where no failure is tolerated and achievement of the control task needs to be guaranteed no matter what, an interface must specify in which environment the component can guarantee to establish its control task. Hence the interface annotates constraints on the environment that are sufficient to achieve its *local control tasks*. The interaction with its environment needs to be captured in more detail, the closer the control component is coupled with its environment, the more intertwined the interaction is.

We target systems, where little information about the other components is necessary to achieve their control task. For these we suggest a quite sleek model of interaction: A component makes assumptions and guarantees for the default case. It can additionally dynamically send events to announce loosening of the current assume-guarantee contract and is able to benefit from the loosening of contracts when it receives an event. The base assumption specifies the tolerated input behavior in terms of value ranges for open actuators and affected sensors. Likewise, a component specifies its base guarantee in terms of controlled actuator values and sensors. In the default case, when no event is active, the component relies on the base assumption and is obliged to establish its base guarantee. We use *hybrid predicates* on visiable variables, to specify base assumptions and guarantees and thereby restrict variable values to ranges during a flow, at discrete updates and give an in-

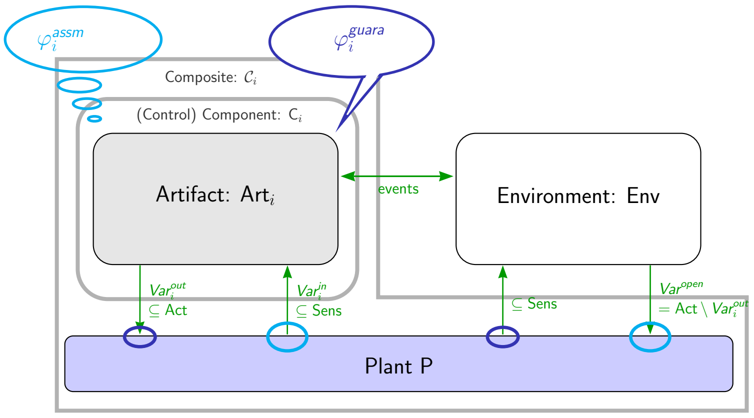

φ_i^{assm}
φ_i^{guara}
Composite: $\mathcal{C}_i$
(Control) Component: C_i
Artifact: Art_i
events
Environment: Env
Var_i^{out} $\subseteq$ Act
Var_i^{in} $\subseteq$ Sens
$\subseteq$ Sens
Var^{open} $=$ Act $\setminus$ Var_i^{out}
Plant P

Figure 3: Analyzed in isolation: assumptions and guarantees define deployment context and the component's service.

variant bound. A component can only submit itself dynamically to event contracts by sending an event to another component. Events represent promises of the sender to the receiving component. The event encodes a time-limited assumption-guarantee pair which is also specified via hybrid predicates.

In our framework, a controller's interface lists base assumptions, base guarantees, events as well as safety and stability guarantees on the plant to describe what the component does and how it may be deployed. The safety guarantee is a first-order predicate that specifies a safety property that is globally true on the plant. The stability guarantee, also specified via a first-order predicate, specifies a property that is established within a certain time and then contiguously holds.

In this paper we will use the terms plant, controller, implementation and interface. For an overview, we summarize their meaning in the following list neglecting events. We will refine these notions later on.

- The common *plant* P (cf. Def 8) is a restricted HIOA.
  The plant will be regulated by controllers via actuators. Hence actuators are input variables of P and outputs of the controllers. Sensors are output variables of P and inputs of a controller. Disturbances model uncontrolled influences in the system's environment.
- We call the realization of a controller also *implementation*.
  Implementations are usually denoted as C and are defined by a tuple $\mathsf{C} = (\mathsf{Art}_\mathsf{C}, \mathsf{P}, \varphi_\mathsf{C}^{assm})$ of artifact, reference to its plant, and a hybrid predicate on open actuators of its plant and sensors it reads (see Fig. 3 for a visualization).
- An *artifact* Art (see Def. 9) is a HIOA. Art is *applicable to P* (see. Def. 12) if P and Art are disjoint except for Act and Sens.
- A (control) *component*, denoted as CC, is a realization of a controller together with an interface it implements.
- We use the term *controller* to refer to a control component or to an implementation or an artifact.
- $\mathcal{C}_i = (\mathsf{Art}_{\mathsf{C}_i} \parallel \mathsf{P})_{(\varphi_{\mathsf{C}_i}^{assm})}$ is the *composite* of C_i and plant P.[2]

[2]The framework will ensure that assumptions restrict only open actuators and sensors, so that the composite is well-defined.

- An interface describes a controller abstractly. It lists its plant, input and output variables, assumptions and guarantees for the base case and events. Safety and stability properties that the controller guarantees, if its assumption is met.

To build a composite controller from subcontrollers, we establish an assume-guarantee result (AGR) in the following of this paper. We will define *interface satisfaction* such that given the component's assumptions are satisfied, it will realize its guarantees and we make it an obligation of the component to maintain its guarantees on its plant for a certain time onwards after its assumptions cease. Our AGR is sketched as follows:

> Given CC_1 assumes $\phi_a \wedge \phi_b$ and guarantees $\psi_x \wedge \psi_y$, and
>
> given CC_2 assumes $\psi_y \wedge \psi_z$ and guarantees $\phi_b \wedge \phi_c$.
>
> The composite controller $\mathsf{CC}_1 \parallel \mathsf{CC}_2$ assumes $\phi_a \wedge \psi_z$ and guarantees $\psi_x \wedge \psi_y \wedge \phi_b \wedge \phi_c$.

The controller components interact with each other by reading sensor values of the same plant. They may also send events to each other. We specify the event communication formally via the notion of *event connector* and consider an exemplary operational implementation via a communication link automaton Com in this paper.

We will use the remainder of this section to introduce formally the basic building blocks of our framework: P, Impl and event connectors.

3.1 Plant

A plant is the model of the physical system the control components supervise and regulate. A controller can only perceive the plant via designated *sensors* and influence the plant via designated *actuators*.

Example 1 *ADAS Plant. For example, in our ADAS case study [4, 8, 10] we modeled aspects of the car as well as the street as part of the plant. In general the physical system is more than the mere physical object the controller is for, but comprises certain aspects of its environment as well.*

Definition 8 ***Plant*** *A* plant *is a lockfree HIOA of the form*

$$P = (\mathbb{M}_P, Var_P^{loc}, Var_P^{in}, Var_P^{out}, R_P^{dscr}, R_P^{cnt}, \Phi_P^{init}, \Theta_P^{inv})$$

where discrete updates $(m, G, \mathcal{A}, m') \in R_P^{dscr}$ only refer to $Var^{loc} \cup \mathbb{M}_P$ in guard G and assignment $\mathcal{A}$.

We call Var_P^{out} also Sens, the sensors *of the plant. The set Var_P^{in} consists of disjoint sets, $\mathsf{Act} \dot\cup \mathsf{Disturb}$,* actuators *and* disturbances.

Definition 8 defines the plant as a restricted hybrid automaton. This hybrid automaton will be regulated by control components via actuators. Hence actuators are input variables of the plant and output variables of the control components. Similarly, sensors are output variables of the plant and input variables of a controller. Disturbances model uncontrolled influences in the system's environment.

A controller's sensor may hence not be discretely set by the plant. This implements a natural point of view: the plant models a physical system where changes take time whereas the control actions may be considered as instantaneous. Internally the plant may switch instantaneously which influences the continuous evolutions of its outputs. [3]

Note, that the plant does not make assumptions on its actuators. Restrictions on actuator values may be encoded via plant invariants in our framework.

3.2 Controller

A controller may read the continuously evolving plant outputs and controls the plant via its actuators, which it may drive discretely and continuously. It makes assumptions about its environment. It makes base assumptions, which have to hold if no event is sent or received. Additionally, a controller can adopt its assumption about the environment, if it has received additional information via an incoming event, an *inevent.* Inevents encode strengthened assumptions on the environment for a certain future time frame and may allow weakened guarantees for that time frame. A controller itself can send an event to another controller to announce guaranteed future achievements. Outevents strengthen a controller's guarantees and weaken its assumptions.

For now we can think of events as identifiers to which a time frame, an assumption and guarantee is associated. We do not specify a mechanism for sending and receiving events within our framework, but we assume that there are variables indicating the sending and receiving of events. A controller declares what events it may send, Ev^{out}, and what events it may receive, Ev^{in}. An implementation C will provide variables –a dedicated variable for each event– that will be used to signal sending and receiving of the respective event. We denote with $CodeVarEv^{out} = \{v_{\mathcal{E}_{out}} \mid \mathcal{E}_{out} \in Ev^{out}_{\mathsf{C}}\} \subseteq Var^{out}$ the set of variables that encode sending of events and the set of variables $CodeVarEv^{in} = \{v_{\mathcal{E}_{in}} \mid \mathcal{E}_{in} \in Ev^{in}_{\mathsf{C}}\} \subseteq Var^{in}$ to encode event reception. We therefore extend the definition of artifact, as follows:

Definition 9 ***Artifact*** *An* artifact Art *is a triple* (H, Ev^{in}, Ev^{out})*, where* H *is an HIOA* $H = (\mathbb{M}, Var^{loc}, Var^{in}, Var^{out}, R^{\mathsf{dscr}}, R^{\mathsf{cnt}}, \Phi^{\mathsf{init}}, \Theta^{\mathsf{inv}})$ *and* Ev^{in}*,* Ev^{out} *are two sets of* Ev^{in} *and* Ev^{out}*, respectively, disjoint from* Var*.*

Further there are $CodeVarEv^{out} = \{v_{\mathcal{E}_{out}} \mid \mathcal{E}_{out} \in Ev^{out}_{\mathsf{C}}\} \subseteq Var^{out}$*, the set of variables that encode sending of events, and the set of variables* $CodeVarEv^{in} = \{v_{\mathcal{E}_{in}} \mid \mathcal{E}_{in} \in Ev^{in}_{\mathsf{C}}\} \subseteq Var^{in}$*, to encode receiving of events.*

We introduce the inevent hiding operator $\ominus$.

Definition 10 ***Inevent Hiding*** *Given an artifact* Art *=(*H*,*Ev^{in}*,*Ev^{out}*) and a set of events* Ev'*.*

$\mathsf{Art} \ominus Ev'$ *denotes* $(H', Ev^{in} \setminus Ev', Ev^{out})$ *with*

$H' = (\mathbb{M}, Var^{loc} \dot{\cup} \{v_{\mathcal{E}} \mid \mathcal{E} \in Ev' \cap Ev^{in}\}, Var^{in} \setminus \{v_{\mathcal{E}} \mid \mathcal{E} \in Ev'\}, Var^{out}, R^{\mathsf{dscr}}, R^{\mathsf{cnt}}, \Phi^{\mathsf{init}} \wedge \bigwedge_{\mathcal{E} \in Ev^{in} \cap Ev'} v_{\mathcal{E}} = 0, \Theta^{\mathsf{inv}}, \mathbf{A}, \phi^{assm})$.

[3]Lockfreedom may seem unachievable without controllers. We imagine the plant to transit into an error mode.

Note that all runs of $\mathsf{CC} \ominus Ev'$ satisfy $\Box \bigwedge_{\mathcal{E} \in Ev^{in} \cap Ev'} v_{\mathcal{E}} = 0$ holds. As CC cannot change input variables, $\mathsf{CC} \ominus Ev'$ will not change the hidden event variables.

Definition 11 ***Implementation*** *The tuple $\mathsf{C} = (\mathsf{Art}, \mathsf{P}, \varphi^{assm})$ of artifact, reference to its plant, and hybrid predicate on $(Var^{in}_{,} \cap \mathsf{Sens}) \cup (\mathsf{Act} \setminus Var^{out}_{,})$ the open actuators of its plant and sensors it reads, is called* implementation *of a controller. We identify the variable sets of the implementation with respective variable sets of its artifact, e.g. $Var_{\mathsf{C}} = Var_{\mathsf{Art}}$.*

As necessary requirement for applicable of a controller to a plant we use:

Definition 12 ***Applicability to a Plant*** *An artifact Art =(H,Ev^{in},Ev^{out}) is applicable as controller to plant P, if*

1. *$Var^{out}_{\mathsf{Art}} \setminus CodeVarEv^{out} \subseteq \mathsf{Act}$ (artifact outputs are actuators),*
2. *$Var^{in}_{\mathsf{Art}} \setminus CodeVarEv^{in} \subseteq \mathsf{Sens}$ (artifact inputs are sensors),*
3. *$Var_{\mathsf{P}} \cap (Var^{loc}_{\mathsf{Art}} \cup CodeVarEv^{in} \cup CodeVarEv^{out}) = \varnothing$ (artifact and plant are disjoint except for actuators and sensors),*
4. *$\varphi^{\mathsf{init}}_{\mathsf{Art}} \Rightarrow \forall v \in CodeVarEv^{in} : v = 0$ (artifact event variables are initialized to 0),*

An implementation $\mathsf{C} = (\mathsf{Art}_{\mathsf{C}}, \mathsf{P}, \varphi^{assm}_{\mathsf{C}})$ is applicable as controller to plant P, if

1. *its artifact is applicable to P, and*
2. *$\varphi^{\mathsf{init}}_{\mathsf{P}} \Rightarrow \forall v \in Var^{in}_{\mathsf{C}} \setminus CodeVarEv^{in} : \varphi^{assm}(v)$ (controller assumptions on initial sensor values match plant initialization),*
3. *$\varphi^{assm}_{\mathsf{Art}}|_{EvVar^{in}} = \bigwedge_{v \in CodeVarEv^{in}} \big((\mathsf{flow} \Rightarrow \dot{v} = 0) \wedge (\neg\mathsf{flow} \Rightarrow (v^+ = 0 \vee v^+ = 1))\big)$ (controller event variables are boolean variables),*
4. *$\varphi^{assm}_{\mathsf{Art}} \Rightarrow \forall v \in Var^{in} \setminus CodeVarEv^{in} : (\neg\mathsf{flow} \Rightarrow v^+ = v)$ (sensor variables are assumed not to change discretely).*

Example 2 *The ADAS Controllers. In our running example, we consider two loosely coupled controllers, a velocity controller VC and a steering controller SC. The VC reads the current velocity vel_{cur}, the steering controller reads the car's orientation β_{ori} and its position relative to the mid of the lane $dist_{\mathsf{cur}}$. Both have disjoint sets of actuators. SC sets $\dot{\beta}_{\mathsf{ori}}$ while VC sets acc.*

In our approach components belong to a plant library. That is for each abstract model of a physical system, we image there is a separate library where applicable components are organized. For the remainder of this paper we fix the library to be P.

3.2.1 Loosely Coupled Controllers: Local and Global Objectives

We provide in the paper a framework that will be well suited to a restricted class of system. For these systems it is a light weight framework that allows tractable verification. We will not provide an exact characterization of the class of systems, that can be conveniently modelled within our framework, but we will discuss the notion of coupling based on a few examples in the following, in order to transport an intuition of which systems can be modelled.

We consider systems as loosely coupled, if the controllers need to exchange little information to achieve their control task and a coarse over approximation of other controllers is sufficient. Given a library, an engineer may compose several such loosely coupled controllers to build a controller that achieves the global control task.

Given a global control task, first a suitable decomposition into local control tasks needs to be determined. The control objectives certainly influence the degree of coupling and for a given a global control task, it is not always possible to find a decomposition into loosely coupled control tasks. But sometimes at least control phases can be identified within which the coupling is loose. In the following we give a list examples from different domains and examine their degree of coupling.

Example 3 *(A) In our ADAS the local objective of the steering controller is to steer the car to the center of the targeted lane controlling $\dot{\beta}_{\mathsf{ori}}$. Local objective of the velocity controller is to make the car drive at a (user chosen) velocity controlling $\dot{vel}$. The global objectives to be achieved cooperatively are (i) the car has to stabilize on the center of its lane, and (ii) the centrifugal force on a passenger ($|F| = |v \cdot \dot{\beta}_{\mathsf{ori}} \cdot m|$) does not exceed a certain threshold.*

(B) At a production line, controller for conveyor and controller for a press have to cooperate. The conveyor controller is in charge of transporting and placing the product. The controller for the press has regulate how the press presses the product into its form. As global objective it has to achieved that the press may only be closed when the product is properly positioned.

(C) At an air conditioner, the air humidity and temperature may be controlled by a hygro controller which regulates a humidifier-dehumidifier and a temperature controller regulating a heater-cooler. Their global objective is to maintain a pleasant humidity at a chosen temperature.

(D) Finally, let us consider a variant of the ADAS scenario. Let the velocity and steering controller have the same local objectives as in (A). As before, the global objective is to make the car stay on its road and maintain an acceptable centrifugal force, but now the car is supposed to drive as fast as possible.

As the above examples illustrate, coupling between subcontrollers results from

(i) the effect of external control actions
In example C, the air humidity needs to be adjusted, if the temperature decreases.

(ii) the control objectives
In our ADAS example, coupling and hence the need for cooperation is induced by the global objective.
Whereas in scenario (A) the coupling is more loose: the velocity control needs only to know about steering when the car is (supposed to drive) fast. VC and

> *SC are closer coupled in scenario (D). To achieve the global objective of driving as fast as possible without leaving the road, the velocity control needs to know exactly what steering is currently applied and what steering will be necessary to stay on the road.*

The control objectives hence need to be carefully chosen:

Example 4 *In our ADAS example, the local objective of VC to reach a user chosen velocity, is intuitive but may get into conflict with the global objective of a comfortable centrifugal force: If the car drives in a narrow road, the user may not force the car to drive arbitrarily fast. As we consider deceleration a safe control action, we reformulate the local objective: VC has to make the car reach the minimum of user chosen velocity and the comfort velocity, that is the velocity that is comfortable at the current steering.*

So decomposition into local control tasks is an intricate process, where one has to be attentive to assign not conflicting control tasks and tasks that allow a great looseness.

3.3 Communication Links: Sending Events between Composed Components

In our framework one controller may send events to another. Inevents are declared at the receiver whereas outevents are declared at the sender. When we compose two controllers, we define how to connect (some) outevents to the other's inevent.

The partial mapping of outevents of a sender to triggered inevents of a receiver is called *event connector.* An event connector $\mathcal{R}$ for (Art_1,Art_2) defines which inevent of Art_2 is triggered by which outevent of Art_1. Additionally $\mathcal{R}$ may declare which outevents of Art_1 are disabled, i.e. the engineer decides to hide.

Definition 13 ***event connector*** *Given Art_1 and Art_2 are two artifacts. An* event connector *$\mathcal{R}$ for (Art_1, Art_2) is a partial function $\mathcal{R} : Ev^{out}_{\mathsf{Art}_1} \nrightarrow Ev^{in}_{\mathsf{Art}_2} \dot\cup \{\mathsf{disable}\}$.*

We image a kind of wire over which events are transmitted between two distributed control components. The transmission of events takes time. We model the connection between two controllers for an event via an HIOA. We do not dictate the exact implementation for this wire -or rather transmission- but we constrain the implementation of the link automaton Com axiomatically (Def. 18). We will give a simple exemplary implementation in Sect. 4.3.

4 Events

Events, just like assumptions and guarantees, are conceptually elements of the interface of a controller. In our framework a controller makes a base assumption about its environment. The base assumption is a coarse overapproximation of every possible behavior that the controller can tolerate at its environment, while it is still in a position to accomplish its services. The base assumption will be specified at the interface as an hybrid predicate on $(Var^{in}_{\mathsf{Art}} \cap \mathsf{Sens}) \cup (\mathsf{Act} \setminus Var^{out}_{\mathsf{Art}})$, its sensors and open plant actuators (cf. Def. 33). A controller also declares a base guarantee, a

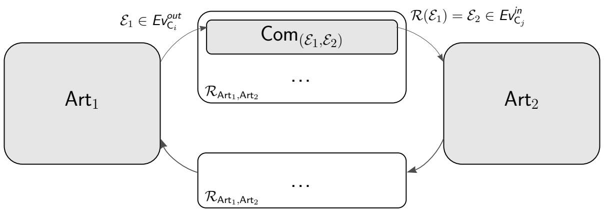

$\mathcal{E}_1 \in Ev^{out}_{C_i}$
$\mathsf{Com}_{(\mathcal{E}_1,\mathcal{E}_2)}$
…
$\mathcal{R}_{\mathsf{Art}_1,\mathsf{Art}_2}$
$\mathcal{R}(\mathcal{E}_1) = \mathcal{E}_2 \in Ev^{in}_{C_j}$
Art_1
Art_2
…
$\mathcal{R}_{\mathsf{Art}_1,\mathsf{Art}_2}$

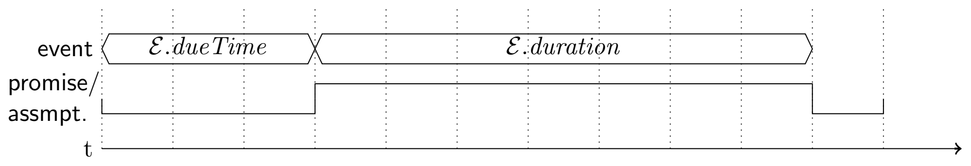
Figure 4: Communication link. Art_1 sends outevent $\mathcal{E}_1$ to Art_2 via $\mathsf{Com}_{(\mathcal{E}_1,\mathcal{E}_2)}$. $\mathsf{Com}_{(\mathcal{E}_1,\mathcal{E}_2)}$ delays event delivery and triggers then the according inevent of Art_2. Art_2 receives $\mathcal{E}_2 \in \mathcal{R}(\mathcal{E}_1)$. One event connector defines the transmission from Art_1 to Art_2, a second one defines the transmission from Art_2 to Art_1.

coarse overapproximation of the possible behavior that the controller will show at its outputs. The base guarantee will be specified at an interface via an hybrid predicate on $(Var^{out}_{\mathsf{Art}} \cap \mathsf{Act}) \cup \mathsf{Sens}$. In the default mode, the controller relies on the base assumption and establishes the base guarantee. To allow to communicate more fine grained information we introduce events. Our events can be used to communicate an optimization potential: When a controller is able to foresee that it will comfortably fulfill its purpose, it can use an event to allow another controller more freedom to achieve its control task for a foreseeable future.

Example 5 *For instance, if the steering controller SC identifies a straight lane ahead and needs to make only minor steering actions, it can notify the velocity controller by sending an event. The VC can use this information, to make the car accelerate, if the user wishes to go faster.*

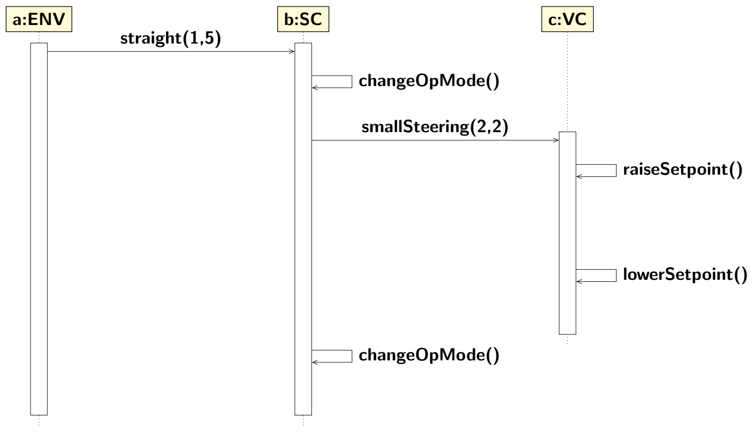
The sending controller cannot force the receiver to certain actions and vice versa a controller cannot require from other controllers to send certain events. Events hence represents an offer of an refined contract, the guarantee is strengthened and the assumption weakened. There is hence no need for the receiving controller to signal consent to the offered event contract.

An event $\mathcal{E} = (\varphi^{prom}, \varphi^{assm}, duration, dueTime)$ encodes an event promise, an event assumption, a due time and a duration.



Events represent dynamic contract(offer)s between the sender and designated receiver. In particular, event promises are time-stamped and are only valid for a certain time. The sending controller has to guarantee the event promise $\mathcal{E}.\varphi^{prom}$ and makes a relaxed assumption $\mathcal{E}.\varphi^{assm}$. That is, sent outevents strengthen guarantees, $\mathcal{E}.\varphi^{prom} \Rightarrow \varphi^{guara}$, and relax the base assumption, $\varphi^{assm} \Rightarrow \mathcal{E}.\varphi^{assm}$. An event promise, if not *released*, is guaranteed from the due time on and at least for the promised validity duration. The sending controller tolerates every behavior specified at the event assumption for this time.

Example 6 *In the scenario sketched below, the steering controller is informed by the environment that a straight lane is guaranteed in one time step and for five time steps.* SC *then sends an event where it guarantees to apply smaller steering actions than promised in its base guarantee and assumes that the velocity may get greater than assumed in the base assumption in two time steps and for two time steps.* VC *uses this event promise to actually increase the speed within the active time of the received event.*



In our framework, VC *is not obliged to increase the speed.* VC *may ignore* SC*'s event. But if* VC *decides to increase its speed then* VC *is obliged to lower its speed before the event promise ends, so that it afterwards fulfills* SC*'s base assumption.*

When we compose two controllers we define via which events the two controllers can communicate. At the controller's interface hence the set of outevents Ev^{out} and the set of inevents Ev^{in} are declared. From the interface declaration we can infer whether events send by the one controller are compatible to inevents of the other controller.

Although, events are abstract concepts that mainly exist at the interface, we require that an implementation provides certain variables to model events: We require that an artifact provides output variables to signal event sending and input variables to encode event receiving. The following definition introduces events referring to the implementation's artifact.

Definition 14 ***Event*** *An event $\mathcal{E}$ is given by a tuple*

$$\mathcal{E} = (\varphi^{prom}, \varphi^{assm}, dueTime, duration)$$

with

- φ^{prom} *is an hybrid predicate on* $(Var^{out}_{\mathsf{Art}} \cap \mathsf{Act}) \cup \mathsf{Sens}$
- φ^{assm} *is an hybrid predicate on* $(Var^{in}_{\mathsf{Art}} \cap \mathsf{Sens}) \cup (\mathsf{Act} \setminus Var^{out}_{\mathsf{Art}})$
- $dueTime > 0$ *is the time duration before the event becomes active*
- $duration \geq 0$ *is the time duration for which the event is active*

An event promise specifies which behavior is guaranteed at the artifact's outputs and the event assumption specifies which behavior is tolerated at the artifact's environment.

A controller can declare two kinds of events, *inevents*, which are events it can receive, and *outevents*, which are events it can send. We denote $Ev := Ev^{in} \cup Ev^{out}$.

Definition 15 ***Inevent, Outevent, Active Time*** *An* outevent $\mathcal{E}_{out}$ *is an event the controller may send, where*

- $\mathcal{E}_{out}.\varphi^{prom}$ *strengthens the base guarantee* φ^{guara}*, i.e.* $\mathcal{E}_{out}.\varphi^{prom} \Rightarrow \varphi^{guara}$
- $\mathcal{E}_{out}.\varphi^{assm}$ *is weaker than the base assumption* φ^{assm}*, i.e.* $\varphi^{assm} \Rightarrow \mathcal{E}_{out}.\varphi^{assm}$.

An inevent $\mathcal{E}_{in}$ *is an event the controller may receive.*

- $\mathcal{E}_{in}.\varphi^{prom}$ *is weaker than the base guarantee, i.e.* $\varphi^{guara} \Rightarrow \mathcal{E}_{in}.\varphi^{prom}$.
- $\mathcal{E}_{in}.\varphi^{assm}$ *is stronger than the base assumption, i.e.* $\mathcal{E}_{in}.\varphi^{assm} \Rightarrow \varphi^{assm}$.

Let an inevent $\mathcal{E}_{in}$ *be received at time* t_{rec}*. It is* active *at the time interval* $[t_{\mathsf{rec}} + \mathcal{E}_{in}.dueTime, t_{\mathsf{rec}} + \mathcal{E}_{in}.dueTime + \mathcal{E}_{in}.duration]$*. The* active times *of an outevent* $\mathcal{E}_{out}$ *send at time* t_{send}*, is the interval* $[t_{\mathsf{send}} + \mathcal{E}_{out}.dueTime, t_{\mathsf{send}} + \mathcal{E}_{out}.dueTime + \mathcal{E}_{out}.duration]$.

$Var(\mathcal{E}_{in})$ *denotes the set of actually constrained variables by an inevent* $\mathcal{E}_{in}$*,*

$$\{v \in Var \mid (\phi^{assm}(v) \not\Rightarrow \mathcal{E}_{in}.\varphi^{assm}(v)) \vee (\mathcal{E}_{in}.\varphi^{prom} \not\Rightarrow \phi^{guara}(v))\}.$$

Likewise $Var(\mathcal{E}_{out})$ *denotes the set of actually constrained variables by an outevent* $\mathcal{E}_{out}$*,*

$$\{v \in Var \mid (\mathcal{E}_{out}.\varphi^{assm}(v) \not\Rightarrow \phi^{assm}) \vee (\phi^{guara}(v) \not\Rightarrow \mathcal{E}_{out}.\varphi^{prom}(v))\}.$$

Var^{inEv} *denotes* $\bigcup_{\mathcal{E} \in Ev^{in}} Var(\mathcal{E})$*. Analogously,* Var^{outEv} *is defined.*

Receiving and sending of events is encoded at the implementation via values on distinguished variables. The following definition introduces this concept.

Definition 16 **rec, send** *Let an implementation* $\mathsf{C} = (\mathsf{Art}_\mathsf{C}, \mathsf{P}, \varphi^{assm}_\mathsf{C})$ *of a controller for plant* P *be given. We require that sending and receiving of events is implemented so that propositions* $\mathsf{rec}(\mathsf{C}, \mathcal{E})$ *and* $\mathsf{send}(\mathsf{C}, \mathcal{E})$ *can be assigned to any reachable state* $\mathbf{X}(t)$ *of* $\mathsf{C} \parallel \mathsf{P}$ *such that*

- $\mathsf{rec}(\mathsf{C}, \mathcal{E})$ *is true at* $\mathbf{X}(t)$ *if and only if* C *receives an event* $\mathcal{E}$ *at* $\mathbf{X}(t)$*.*
- $\mathsf{send}(\mathsf{C}, \mathcal{E})$ *is true at* $\mathbf{X}(t)$ *if and only if* C *sends an event* $\mathcal{E}$ *at* $\mathbf{X}(t)$*.*

Let C *declare sets of event encoding variables* $CodeVarEv^{out} = \{v_{\mathcal{E}_{out}} \mid \mathcal{E}_{out} \in Ev^{out}_\mathsf{C}\}$ *and* $CodeVarEv^{in} = \{v_{\mathcal{E}_{in}} \mid \mathcal{E}_{in} \in Ev^{in}_\mathsf{C}\}$*. For the following we assume that at any reachable state* $\mathbf{X}(t)$ *it holds*

- $\forall v_{\mathcal{E}_{out}} \in CodeVarEv^{out}$*:*
  - *if and only if* $\mathbf{X}(v_{\mathcal{E}_{out}}) = 1$*, then* $\mathsf{send}(\mathsf{C}, \mathcal{E}_{out})$ *is true and*
  - *if and only if* $\mathbf{X}(v_{\mathcal{E}_{out}}) = 0$*, then* $\mathsf{send}(\mathsf{C}, \mathcal{E}_{out})$ *is false and*

  - $\mathbf{X}(v_{\mathcal{E}_{out}}) = 0 \ \vee \ \mathbf{X}(v_{\mathcal{E}_{out}}) = 1$.

- $\forall v_{\mathcal{E}_{in}} \in \mathit{CodeVarEv}^{in}$:
  - *if and only if* $\mathbf{X}(v_{\mathcal{E}_{in}}) = 1$, *then* $\mathsf{rec}(C, \mathcal{E}_{in})$ *is true and*
  - *if and only if* $\mathbf{X}(v_{\mathcal{E}_{in}}) = 0$, *then* $\mathsf{rec}(C, \mathcal{E}_{in})$ *is false, and*
  - $\mathbf{X}(v_{\mathcal{E}_{in}}) = 0 \ \vee \ \mathbf{X}(v_{\mathcal{E}_{in}}) = 1$.

We assume that the transmission of events takes time. We fix bounds on the message travelling time. We also assume that events may get lost. So either an event gets lost, or it is delivered within minimal and maximal latency. We also require that the signal for receiving an event and the signal for sending an event is persistent at least for Δ^{pers}.

Definition 17 ***MinSendLat, MaxSendLat,*** Δ^{pers} *We define a global constants* $0 < \mathit{MinSendLat} < \mathit{MaxSendLat} \in \mathit{Time}$, *representing the* minimal *and* maximum latency *between sending and receiving an event, if the event is successfully sent.*
$\Delta^{\mathsf{pers}} > 0$ *is the required minimal signal persistency time for a send and receive signal.*

Events can be thought of as being sent via private channels between two controllers (cf. Sect. 3.3). We model the transmission of an event $\mathcal{E}$ from C_i to C_j as
1. sending $\mathcal{E}_{out}$ at C_i, that is $\mathsf{send}(\mathsf{C}_i, \mathcal{E})$ true, followed by a
2. a delay, followed by
3. receiving of $\mathcal{R}(\mathcal{E}_{out})$ at C_j, that is $\mathsf{rec}(\mathsf{C}_j, \mathcal{R}(\mathcal{E}_{out}))$ is true.

We will give a simple exemplary implementation in 4.3.
We require that
- any implementation of the link satisfies that only sent events are received, but we allow that events may get lost.
- the transmission times are between minimal and maximal latency and
- no locks are introduced by the automaton implementing the link.

Definition 18 ***Event Transmission*** *Given controller implementations,* C_1 *with outevents* $Ev^{out}_{C_1}$ *and* C_2 *with inevents* $Ev^{in}_{C_2}$. *Let the two be applicable to a plant* P *and have disjoint sets of actuators. Let* $\mathcal{R}$ *be an event connector,* $\mathcal{R} : Ev^{out}_{C_1} \nrightarrow Ev^{in}_{C_2} \dot\cup \{\mathit{disable}\}$.
The event transmission is implemented via a HIOA $\mathit{Com}_{\mathcal{R}}$ *that has to satisfy*

1. $C_1 \parallel C_2 \parallel \mathit{Com}_{\mathcal{R}} \parallel P \models$
$\forall \mathcal{E}_i \in Ev^{in}(C_2) \cap \mathcal{R}(Ev^{out}_{C_1}) : \Box(\mathit{rec}(C_2, \mathcal{E}_i) \Rightarrow \exists \mathcal{E}_o \in Ev^{out}_{C_1} : \mathit{send}(C_1, \mathcal{E}_o) \ \wedge \ \mathcal{E}_i = \mathcal{R}(\mathcal{E}_o)$
$\wedge\, \mathcal{E}_o.t\_\mathit{send} + \mathit{MinSendLat} \leq \mathcal{E}_i.t\_\mathit{rec} \leq \mathcal{E}_o.t\_\mathit{send} + \mathit{MaxSendLat})$,
where $\mathcal{E}_i.t\_\mathit{rec}$ *is the time at which* $\mathcal{E}_i$ *is received and* $\mathcal{E}_o.t\_\mathit{send}$ *is the time at which* $\mathcal{E}_o$ *has been sent.*
(controller C_2 *may only receive an event, if* C_1 *did send the matching event. The delay between sending and receiving varies between MinSendLat and MaxSendLat, if the event is received at all.)*

2. $\mathit{Com}_{\mathcal{R}}$ *is lockfree.*

3. *The variables of $\mathsf{Com}_{\mathcal{R}}$ satisfy*
   - *The automaton $\mathsf{Com}_{\mathcal{R}}$ is variable disjoint from plant P.*
   - $Var_{\mathsf{Com}_{\mathcal{R}}} \cap Var_{\mathsf{C}_1} \subseteq CodeVarEv^{out}_{\mathsf{C}_1} \cap Var^{in}_{\mathsf{Com}_{\mathcal{R}}} = \{v_{\mathcal{E}} \mid \mathcal{E} \in Ev^{out}_{\mathsf{C}_1} \cap \mathcal{R}^{-1}(Ev^{in}_{\mathsf{C}_2})\}$
     (If $\mathsf{Com}_{\mathcal{R}}$ and C_1 share variables, then the shared variables are outevent variables of C_1 and input variables of $\mathsf{Com}_{\mathcal{R}}$.) and
   - $Var_{\mathsf{Com}_{\mathcal{R}}} \cap Var_{\mathsf{C}_2} \subseteq CodeVarEv^{in}_{\mathsf{C}_2} \cap Var^{out}_{\mathsf{Com}_{\mathcal{R}}} = \{v_{\mathcal{E}} \mid \mathcal{E} \in Ev^{in}_{\mathsf{C}_2} \cap \mathcal{R}(Ev^{out}_{\mathsf{C}_1})\}$.
     (If $\mathsf{Com}_{\mathcal{R}}$ and C_2 share variables, then these variables are inevent variables of C_2 and output variables of $\mathsf{Com}_{\mathcal{R}}$.)

   ($\mathsf{Com}_{\mathcal{R}}$ reads event-outputs of C_1 and $\mathsf{Com}_{\mathcal{R}}$ writes to C_2's event-inputs. Otherwise $\mathsf{Com}_{\mathcal{R}}$ is disjoint from the remainder.)

4. $\forall v \in Var^{out}_{\mathsf{Com}_{\mathcal{R}}} : \mathsf{Com}_{\mathcal{R}} \models \Box((\mathit{flow} \Rightarrow \dot{v} = 0) \wedge (v = 0 \vee v = 1))$
   (Outevent variables are updated as discrete variables and take on boolean values only.)

5. *The initial state of $\mathsf{Com}_{\mathcal{R}}$ encodes*
   - *that initially no event is received,*
   - *no event is in the process of being delivered and*
   - *an event is sent at the initial state, iff the corresponding sender defines this.*

   (Φ^{init} of $\mathsf{Com}_{\mathcal{R}}$'s input and output variables are determined by connected sender and receiver controllers.)

6. $\forall v \in Var^{out}_{\mathsf{Com}_{\mathcal{R}}} : \mathsf{Com}_{\mathcal{R}} \models \Box(\uparrow (v = 1) \Rightarrow \Box_{\Delta^{pers}}(v = 1))$
   (The signal that an event has been transmitted to the receiver is persistent for Δ^{pers}.)

After composition (cf. Sect. 5) inevents and outevents that get connected are no longer available to the outside. Hence we will hide these events in the composite automaton.

4.1 Releasing Events

Example 7 *Suppose SC does not need to apply much steering, because the lane ahead is straight and the car on course. SC notifies VC via $\mathcal{E}_1$ that it will not need to apply much steering and accepts that the car drives fast. Suppose our car is currently in a play street and VC is bound to drive very slow. After VC did receive SC's offer, $\mathcal{R}(\mathcal{E}_1)$, to bring the car to a faster speed while SC simultaneously applies minimal steering, VC can itself send an event, $\mathcal{E}_2$, that guarantees that VC will not drive such high speeds as guaranteed by its base assumption and that it accepts stronger steering actions. When SC receives this event, it is released from its event promise given by $\mathcal{E}_1$ and it may now apply stronger steering to achieve lower priority goals as fun cruising.*

A controller $\mathsf{C_{send}}$ that has sent an event $\mathcal{E}$ is obliged to satisfy the given event promise, unless it is *released* from its promise by the receiver $\mathsf{C_{rec}}$. The controller is released when it receives an (incoming) releasing event. This inevent is releasing if it expresses that $\mathsf{C_{rec}}$ does not expect fulfillment of the given promise.

$\mathsf{C_{rec}}$ can release $\mathsf{C_{send}}$ from satisfying the promise given by $\mathcal{E}$ by sending an event $\mathcal{E}_{release}$ with assumptions that are weaker than the promise given by $\mathcal{E}$.

In our framework it suffices to check which variables are referenced, as outevents always strengthen the sender's base guarantee and weaken its base assumption, while inevents declare promises weaker than the base guarantee and assumptions stronger than the base assumption. This also implies that loosing a releasing event does not endanger contract satisfaction.

Def. 19 captures necessary conditions on an event $\mathcal{E}_{in}$ to be able to release an event $\mathcal{E}_{out}$. Note that potentially releasing is defined in terms of in- and outevents of the same controller.

Definition 19 ***potentially releasing event, potentially released event*** *We call $\mathcal{E}_{out} \in Ev^{out}_{\mathsf{C}_1}$* potentially releasing *$\mathcal{E}_{in} \in Ev^{in}_{\mathsf{C}_1}$ under $\mathcal{R}_1 : Ev^{out}_{\mathsf{C}_1} \nrightarrow Ev^{in}_{\mathsf{C}_2} \dot\cup \{\mathsf{disable}\}$ and $\mathcal{R}_2 : Ev^{out}_{\mathsf{C}_2} \nrightarrow Ev^{in}_{\mathsf{C}_1} \dot\cup \{\mathsf{disable}\}$ if*

- *$\mathcal{E}_{out}.\varphi^{assm}$ covers all variables referred to by $\mathcal{E}_{in}.\varphi^{prom}$ and*
- *$\mathcal{E}_{out}$, $\mathcal{E}_{in}$ are events of C_1 and $\mathcal{R}_1(\mathcal{E}_{out})$, $\mathcal{R}_2^{-1}(\mathcal{E}_{in})$ are events of C_2, i.e., $\mathcal{R}_1(\mathcal{E}_{out}) \in Ev^{in}(\mathsf{C}_2) \wedge \mathcal{R}_2^{-1}(\mathcal{E}_{in}) \subseteq Ev^{out}(\mathsf{C}_2)$.*

Given appropriate timing, sending $\mathcal{E}_{out}$ releases the sender of $\mathcal{R}^{-1}(\mathcal{E}_{in})$ from its promise.

We call $\mathcal{E}_{out} \in Ev^{out}_{\mathsf{C}_1}$ potentially released by *$\mathcal{E}_{in} \in Ev^{in}_{\mathsf{C}_1}$, if*

- *$\mathcal{E}_{in}.\varphi^{assm}$ covers all variables referred to by $\mathcal{E}_{out}.\varphi^{prom}$.*
- *$\mathcal{E}_{out}$, $\mathcal{E}_{in}$ are events of C_1 and $\mathcal{R}_1(\mathcal{E}_{out})$, $\mathcal{R}_2^{-1}(\mathcal{E}_{in})$ are events of C_2.*

Given appropriate timing, receiving $\mathcal{E}_{in}$ releases the controller from the promise given by sending $\mathcal{E}_{out}$.

The notion of *potentially releasing* neglects activation times of releasing and released events. The decision of when a received event is considered as releasing, has to be taken with care, as releasing a contract bears the danger of misconception:

Example 8 *Suppose SC sends event $\mathcal{E}_1$ to VC promising minimal steering and allowing great velocities. Sometime before twice the maximal latency elapses, SC receives a potentially releasing event $\mathcal{E}_2$ from VC. $\mathcal{E}_2$ allows more steering and guarantees slower velocities than $\mathcal{E}_1$. How can SC tell, whether $\mathcal{E}_2$ is a response to $\mathcal{E}_1$? Suppose SC in turn applies stronger steering than promised by $\mathcal{E}_1$. If VC now believes that $\mathcal{E}_1$ is still guaranteed and decides to accelerate as strong as $\mathcal{E}_1$ does allow, bad things may happen.*

We call $\mathcal{E}_i \in Ev^{out}_{\mathsf{C}_1}$ *released by* $\mathcal{E}_j \in Ev^{in}_{\mathsf{C}_1}$ at time t, if $\mathcal{E}_j$ is potentially releasing and $\mathcal{E}_i$'s (remaining) active time is covered by the active time of $\mathcal{E}_j$ and $\mathcal{E}_j$ is received after the maximal latency elapsed at least twice after sending $\mathcal{E}_i$. This termination of the dynamic event contracts is captured in the Def. 21.

If a controller sends an event to a receiver, it thereby offers a dynamic contract "I weaken my assumptions about you and strengthen my guarantees for you". Unless the receiver releases the sender from its offer, the sender is obliged to satisfy the offered contract.

But what if the sender sends events repeatedly and what if the events overlap? To allow event concatenation, i.e. concatenation of consecutive events to derive a contiguous event promise, it is necessary that a sender sends overlapping events. It does not suffice to send events with contiguous active times, as the receiver has to conservatively derive active times considering that one event is delayed by maximal latency while the other was delayed only by minimal latency. In the following we will examine what sending sequences of events means for our framework.

Let us consider a situation where a controller sends two outevents, first $\mathcal{E}_1$ then $\mathcal{E}_2$ and $\mathcal{E}_1$ and $\mathcal{E}_2$ are simultaneously active at a time t (and no inevents are active). To derive a combined dynamic contract at overlapping active times, we formalize a partial order on events based on their implied dynamic contracts.

Definition 20 $\mathcal{E}_1 \preceq \mathcal{E}_2$ *Let $\mathcal{E}_1$ and $\mathcal{E}_2$ be two events. The partial* event contract order $\preceq$ *is defined by*

- *event contract $\mathcal{E}_2$ is stronger than the event contract $\mathcal{E}_1$, $\mathcal{E}_1 \preceq \mathcal{E}_2$, if and only if $\mathcal{E}_1.\varphi^{assm} \Rightarrow \mathcal{E}_2.\varphi^{assm} \wedge \mathcal{E}_2.\varphi^{prom} \Rightarrow \mathcal{E}_1.\varphi^{prom}$.*
- *event contracts of $\mathcal{E}_1$ and $\mathcal{E}_2$ are incomparable, $\mathcal{E}_1 \prec\!\succ \mathcal{E}_2$, if and only if neither $\mathcal{E}_1 \preceq \mathcal{E}_2$ nor $\mathcal{E}_2 \preceq \mathcal{E}_1$.*

The minimal element in the $\preceq$-order is the event $\mathcal{E}_{\mathsf{min}}$ that assumes *false* and guarantees *true*, the maximal element $\mathcal{E}_{\mathsf{max}}$ is an event that assumes *true* and guarantees *false*.

To rule out that a sender weakens a once offered event contract, one approach could be to require that if a sequel of events $(\mathcal{E}_i)$ is sent, then $\mathcal{E}_i \preceq \mathcal{E}_{i+1}$. A sender may still iteratively strengthen the dynamic contract. But we want more; we want to allow chaining of events even if they are not increasing: A sender should also be able to express that it first guarantees an excellent contract but right after it guarantees at slightly degraded contract which is still better than the base assumption. Therefore we assume that an outevent gets active only if there is no stronger outevent. A sender is always obliged to fulfill the strongest event contract and a receiver may always rely on the strongest received contract offer. But since $\preceq$ is only a partial order, we require that a sender guarantees that simultaneously activated outevents are comparable wrt. $\preceq$. This is formalized in the notion of consistent sender, cf. Def. 22. Before we introduce the notion of consistent sender, we introduce propositions $\mathsf{activated\_rec}$, $\mathsf{active\_rec}$, $\mathsf{activated\_send}$ and $\mathsf{active\_send}$ that encode the status of an send or received event for a controller C and an event $\mathcal{E}$. Their valuation at state $\mathbf{X}(t)$ depends on the valuation along the trajectory $\mathbf{X}(\cdot)$ upto state $\mathbf{X}(t)$. An active event determines the current contract, whereas an activated event is *dominated* by another concurrently activated event. Whether an event $\mathcal{E}$ dominates an event $\mathcal{E}'$ depends on whether the event is received or send: For inevents, $\mathcal{E}$ *dominates* $\mathcal{E}'$, if $\mathcal{E} \leq \mathcal{E}'$. For outevents $\mathcal{E}$ *dominates* $\mathcal{E}'$, if $\mathcal{E}' \leq \mathcal{E}$. That is for inevents we choose the weakest contract, while for outevents we choose the strongest contract, as contract offers cannot be retracted.

Definition 21 activated_rec, activated_send, active_rec, active_send

- *$\mathsf{activated\_rec}(C, \mathcal{E})$ is true at state $\mathbf{X}(t)$ if and only if C has received an event $\mathcal{E}$ and $\mathcal{E}$ is active at state $\mathbf{X}(t)$.*

  $\mathcal{E}$ is active at state $\mathbf{X}(t)$, if $\mathbf{X}(t)$ is reached at time t and there is an earlier state $\mathbf{X}_{\mathsf{rec}}(t_{\mathsf{rec}})$ reached at time t_{rec} at which $\mathsf{rec}(C, \mathcal{E})$ is true and $t \in [t_{\mathsf{rec}} + \mathcal{E}.dueTime, t_{end})$, where t_{end} is $t_{end} = \min(t_{release}, t_{max})$ with

  - *$t_{max} = t_{\mathsf{rec}} + \mathcal{E}.dueTime + \mathcal{E}.duration$,*
    if C does not send a potentially releasing event $\mathcal{E}_{release}$ with covering active times during the interval $(t_{\mathsf{rec}}, t_{max}]$; otherwise $t_{max} = \infty$.
  - *if C sends a potentially releasing event $\mathcal{E}_{release}$ with covering active times within $(t_{\mathsf{rec}}, t_{max}]$,*
    let t_{send} be the earliest time at which such an event $\mathcal{E}_{release}$ is sent,
    then $t_{release} = t_{\mathsf{send}} + \mathcal{E}_{release}.dueTime$ is the point in time at which $\mathcal{E}_{release}$ may become active; otherwise $t_{release} = \infty$.

  $\mathcal{E}_{release}$ has active times covering $\mathcal{E}$, if $[t_{\mathsf{rec}} + \mathcal{E}.dueTime, t_{\mathsf{rec}} + \mathcal{E}.dueTime + \mathcal{E}.duration] \subseteq [t_{release}, t_{release} + \mathcal{E}_{release}.duration]$.

- *$\mathsf{active\_rec}(C, \mathcal{E})$ is true at state $\mathbf{X}(t)$ if and only if $\mathsf{activated\_rec}(C, \mathcal{E})$ is true at state $\mathbf{X}(t)$ and there is no other event $\mathcal{E}'$ with $\mathsf{activated\_rec}(C, \mathcal{E}')$ and $\mathcal{E}' \preceq \mathcal{E}$.*

- *$\mathsf{activated\_send}(C, \mathcal{E})$ is true at state $\mathbf{X}(t)$ if and only if C has sent event $\mathcal{E}$ and $\mathcal{E}$ is active at state $\mathbf{X}(t)$.*

  $\mathcal{E}$ is active at state $\mathbf{X}(t)$, if $\mathbf{X}(t)$ is reached at time t and there is an earlier state $\mathbf{X}_{\mathsf{send}}(t_{\mathsf{send}})$ reached at time t_{send} at which $\mathsf{send}(C, \mathcal{E})$ is true and $t \in [t_{\mathsf{send}} + \mathcal{E}.dueTime, t_{end})$, where $t_{end} = \min(t_{release}, t_{max})$

  - *$t_{max} = t_{\mathsf{send}} + \mathcal{E}.dueTime + \mathcal{E}.duration$,*
    if C does not receive a potentially releasing event $\mathcal{E}_{release}$ with covering active times during the interval $(t_{send} + 2 \cdot MaxSendLat, t_{max}]$; $t_{max} = \infty$ otherwise.
  - *if C receives a potentially releasing event $\mathcal{E}_{release}$ with covering active times within $(t_{send} + 2 \cdot MaxSendLat, t_{max}]$,*
    let t_{rec} be the earliest time at which such an event is received,
    then $t_{release} = t_{\mathsf{rec}} + \mathcal{E}_{release}.dueTime$ is the point in time at which $\mathcal{E}_{release}$ may become active; otherwise $t_{release} = \infty$.

  $\mathcal{E}_{release}$ has active times covering $\mathcal{E}$, if it holds that $[t_{\mathsf{send}} + \mathcal{E}.dueTime, t_{\mathsf{send}} + \mathcal{E}.dueTime + \mathcal{E}.duration] \subseteq [t_{release}, t_{release} + \mathcal{E}_{release}.duration]$.

- *$\mathsf{active\_send}(C, \mathcal{E})$ is true at state $\mathbf{X}(t)$ if and only if $\mathsf{activated\_send}(C, \mathcal{E})$ is true at state $\mathbf{X}(t)$ and there is no other event $\mathcal{E}'$ with $\mathsf{activated\_send}(C, \mathcal{E}')$ and $\mathcal{E} \preceq \mathcal{E}'$*

Note that it is possible that $\mathsf{active\_send} \wedge \mathsf{active\_rec}$ are simultaneously true, i.e., in- and outevents become simultaneously active referring to the same variable. We further introduce $\mathcal{E}.t_{\mathsf{act}}$ to denote the most recent time at which $\mathcal{E} \in Ev^{in} \cup Ev^{out}$ became active along a trajectory. We define that at any state along a trajectory,

$\mathcal{E}.t_{\mathsf{act}}$ equals 0 when $\mathcal{E}$ was not previously active. By definition, events are not active at the initial state of the controller.

Note 3 *According to the above definition an event that releases some event does not necessarily become active. It can be released itself.*

Note 4 *In our framework events are a private communication mean between two controllers. We decided on this directed link instead of allowing for instance event broadcasts. Event broadcasts can be implemented for framework easily if broadcasted events are not released. But even otherwise, only a suitable arbitrating protocol is necessary that resolves releasing an event as earliest when all controllers agree. This can be done by an external arbitrating controller or by the sending controller itself.*

To summarize, in our framework a sender is always obliged to fulfill the strongest event contract and a receiver may always rely on the strongest received contract offer. But since $\preceq$ is only a partial order, we require that a sender guarantees that simultaneously activated outevents are comparable wrt. $\preceq$. This is formalized in the notion of consistent sender, cf. Def. 22. Additionally, a consistent event sender guarantees to keep up the sending signal for Δ^{pers} time.

Definition 22 ***Consistent Sender*** *Let $\mathsf{C} = (\mathsf{Art_C}, \mathsf{P}, \varphi_{\mathsf{C}}^{assm})$ be a controller on plant P.*

1. $\mathsf{Art_C} \models \bigwedge_{\mathcal{E}_1,\mathcal{E}_2 \in Ev^{out}} \Box \Big(\mathsf{activated\_send}(\mathcal{E}_1) \wedge \mathsf{activated\_send}(\mathcal{E}_2) \Rightarrow \mathcal{E}_1 \preceq \mathcal{E}_2 \vee \mathcal{E}_2 \preceq \mathcal{E}_1 \Big)$,
2. $\mathsf{Art_C} \models \Box \bigwedge_{\mathcal{E} \in Ev^{out}} \big(\uparrow \mathsf{send}(\mathcal{E}) \Rightarrow \Box_{\Delta^{\mathsf{pers}}} \mathsf{send}(\mathcal{E}) \big)$

4.2 Dynamic Assumptions and Dynamic Guarantees

In this section we consider the effect of events on the contracts between controllers. By using events the assume-guarantee-contracts become dynamic. Within our framework, assumptions are statically declared at the respective controller interfaces for the case they do not send and receives events. Events provide the opportunity to dynamically modify assumptions and promises. In any case, a controller may only be used, if its assumptions are satisfied at all times, at any reachable state.

Sending and receiving events makes the assumptions of a component dynamic: Receiving events, means strengthening assumptions to the event assumptions associated with the received event. Sending an event means weakening assumptions to the assumptions of the sent event. To specify the resulting *dynamic assumptions* we define a function $\varphi_{dyn}^{assm}(.)$ that assigns a propositional predicate $\varphi_{dyn}^{assm}(t, \mathbf{X}(.))$ to each state $X(t)$ along a trajectory $\mathbf{X}(.)$. The dynamic assumption describes the currently active assumption, which is influenced by sending and receiving events. $\varphi_{dyn}^{assm}(t, \mathbf{X}(.))$ holds at a trajectory $X(.)$ if the propositional formula assigned at $X(t)$ holds at every reached state $X(t)$.

For the formal definition of $\varphi_{dyn}^{assm}(.)$ we use $\mathsf{active\_rec}(\mathcal{E})(x)$ as short hand for $\mathsf{active\_rec}(\mathcal{E}) \wedge x \in \mathsf{Var}(\mathcal{E})$ and respectively $\mathsf{active\_send}(\mathcal{E})(x)$ to abbreviate $\mathsf{active\_send}(\mathcal{E}) \wedge x \in \mathsf{Var}(\mathcal{E})$.

Definition 23 ***Dynamic Assumption*** *Let $\mathsf{C} = (\mathsf{Art}_\mathsf{C}, P, \varphi_\mathsf{C}^{assm})$ be a controller where $\mathsf{Art}_\mathsf{C} = (H_{\mathsf{Art}}, Ev^{in}, Ev^{out})$, base assumption $\varphi_\mathsf{C}^{assm}$, a set Ev^{in} of receivable events and a set Ev^{out} of events it may send.*

The dynamic assumption $dynAssm(\varphi^{assm}, Ev^{in}, Ev^{out})$ of C is defined by the following dynamic safety predicate $\varphi_{dyn}^{assm}(.)$:

1. $\bigwedge_{v \in (Var_{\mathsf{Art}}^{in} \cap \mathsf{Sens}) \cup (\mathsf{Act} \setminus Var_{\mathsf{Art}}^{out})} \Big(\neg \bigvee_{\mathcal{E}_{out} \in Ev^{out}(\mathsf{C})} \mathsf{active\_send}(\mathsf{C}, \mathcal{E}_{out})(v) \Rightarrow \varphi^{assm}(v) \;\wedge$
2. $\bigwedge_{\mathcal{E}_{in} \in Ev^{in}(\mathsf{C})} \big(\mathsf{active\_rec}(\mathsf{C}, \mathcal{E}_{in})(v) \Rightarrow \;\mathcal{E}_{in}.\varphi^{assm}(v)\big)\Big) \;\wedge$
3. $\bigwedge_{\mathcal{E}_{out} \in Ev^{out}(\mathsf{C})} \Big(\mathsf{active\_send}(\mathsf{C}, \mathcal{E}_{out})(v) \Rightarrow \;\mathcal{E}_{out}.\varphi^{assm}(v)\Big)$

According to 1 of the above safety predicate the base assumption holds for a variable v at any state up to a state where a sent event on v becomes active. Additionally after a received event $\mathcal{E}$ becomes active, the controller may assume that the behavior promised by the event on its inputs satisfies $\mathcal{E}.\varphi^{assm}$ until the received promise is not active anymore (cf. 2). By 3 the above assumption may not need to hold, in case the controller itself sends an event. Then the event's assumption holds (cf. 3) until the event is not active anymore.

Analogously to Def. 23 we define the dynamic guarantee of a controller C.

Definition 24 ***Dynamic Guarantees*** *Let $\mathsf{C} = (\mathsf{Art}_\mathsf{C}, P, \varphi_\mathsf{C}^{assm})$ be a controller where $\mathsf{Art}_\mathsf{C} = (H_{\mathsf{Art}}, Ev^{in}, Ev^{out})$, a set Ev^{in} of receivable events and a set Ev^{out} of events it may send. Let $\varphi_\mathsf{C}^{guara}$ be an hybrid predicate on $(Var_{\mathsf{Art}}^{out} \cap \mathsf{Act}) \cup \mathsf{Sens}$, specifying C's base guarantee.*

The dynamic guarantee $dynProm(\varphi^{guara}, Ev^{in}, Ev^{out})$ of a controller C is defined by the following safety predicate $\varphi_{dyn}^{guara}(.)$:

1. $\bigwedge_{v \in (Var_{\mathsf{Art}}^{out} \cap \mathsf{Act}) \cup \mathsf{Sens}} \Big(\Big(\;\neg \bigvee_{\mathcal{E}_{in} \in Ev^{in}(\mathsf{C})} \mathsf{active\_rec}(\mathsf{C}, \mathcal{E}_{in}) \;\Rightarrow \varphi^{guara}(v)\Big) \;\wedge$
2. $\Big(\;\neg \bigvee_{\mathcal{E}_{out} \in Ev^{out}(\mathsf{C})} \mathsf{active\_send}(\mathsf{C}, \mathcal{E}_{out}) \;\Rightarrow$
3. $\bigwedge_{\mathcal{E}_{in} \in Ev^{in}(\mathsf{C})} \big(\mathsf{active\_rec}(\mathsf{C}, \mathcal{E}_{in})(v) \Rightarrow \;\mathcal{E}_{in}.\varphi^{prom}(v)\,)\,)\Big) \;\wedge$
4. $\Big(\bigwedge_{\mathcal{E}_{out} \in Ev^{out}(\mathsf{C})} \big(\mathsf{active\_send}(\mathsf{C}, \mathcal{E}_{out})(v) \Rightarrow \;\mathcal{E}_{out}.\varphi^{prom}(v)\,)\big)\Big)$

According to 1 the base guarantee holds for a variable v at any state where no inevent on v is active. If at C currently no send event is active and if an inevent $\mathcal{E}_{in}$ is active, then C's behavior complies with $\mathcal{E}_{in}.\varphi^{prom}$ (cf. 2), which does not necessarily imply the base guarantee. But if at C there is an active outevent, cf. 4, then the outevent's guarantee holds.

Note 5 *Simultaneous sent and received events may lead to a partial release, as illustrated in the following scenario: Suppose $\mathsf{active\_rec}(\mathcal{E}_1) \wedge \mathsf{active\_send}(\mathcal{E}_2)$ holds at C with $\{x_1, x_2\} \subseteq Var(\mathcal{E}_1)$, $\{x_2\} \subseteq Var(\mathcal{E}_2)$. C assumes that x_1 stays within the interval promised by $\mathcal{E}_1$ but does not rely on the promise concerning x_2, since it itself has weakened its assumptions on x_2.*

4.3 Sending Events - Exemplary Implementation

As already outlined in Sect. 3.3, we use event connectors (Def. 13) to define which outevents of one controller trigger which inevents of the other controller. We in-

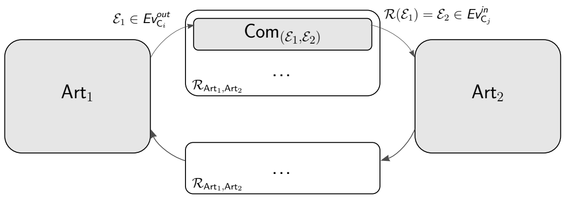

$\mathcal{E}_1 \in Ev^{out}_{\mathsf{C}_i}$
$\mathsf{Com}_{(\mathcal{E}_1,\mathcal{E}_2)}$
$\mathcal{R}(\mathcal{E}_1) = \mathcal{E}_2 \in Ev^{in}_{\mathsf{C}_j}$
Art_1
...
$\mathcal{R}_{\mathsf{Art}_1,\mathsf{Art}_2}$
Art_2
...
$\mathcal{R}_{\mathsf{Art}_1,\mathsf{Art}_2}$

Figure 5: Communication links between two controllers.

troduced event connectors as partial mapping from outevents of one component to inevents of another component. But so far, we have not settled on how to implement event based communication in terms of HIOA. In the following we will discuss a simple implementation for event based communication between two concurrent control components. We implement sending as writing an variable and receiving as reading variables. The sender sets a variable $v^{\mathcal{E}}_{\mathsf{send}}$ to indicate the sending of an event. A timed automaton Com delays the event on its way to the receiving component.

Let C_1 be the sender of an event $\mathcal{E}_{out}$ and let the event connector $\mathcal{R}_1 : Ev^{out}_1 \nrightarrow Ev^{in}_2 \dot\cup \{\mathsf{disable}\}$ define C_2 as receiver, that is $\mathcal{R}(\mathcal{E}_{out}) = \mathcal{E}_{in} \in Ev^{in}_2$.

To indicate that C_1 sends $\mathcal{E}_{out}$, it sets $v^{\mathcal{E}_{out}}_{\mathsf{send}}$ from 0 to 1. $v^{\mathcal{E}_{out}}_{\mathsf{send}}$ is only read by the timed automaton $\mathsf{Com}^{\mathcal{R}_1}_{\mathcal{E}_{out},\mathcal{E}_{in}}$. $\uparrow v^{\mathcal{E}_{out}}_{\mathsf{send}}$ triggers $\mathsf{Com}^{\mathcal{R}_1}_{\mathcal{E}_{out},\mathcal{E}_{in}}$ to a monitor the transmission delay.

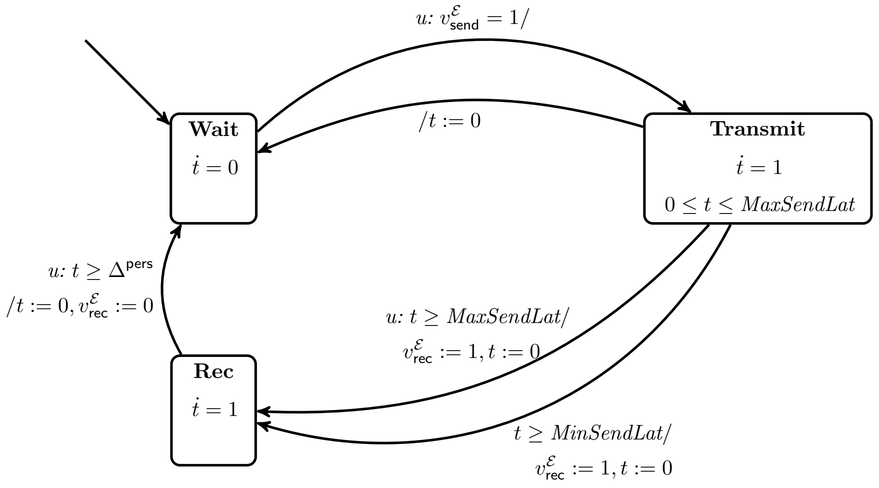
1. C_1, the sender of event $\mathcal{E}_{out}$ has to satisfy:
   - $v^{\mathcal{E}_{out}}_{\mathsf{send}} \in Var^{out}_{\mathsf{C}_1}$
   - $\forall \mathbf{X} \in \mathsf{reach}(\mathsf{C}_1) : \mathbf{X} \models \uparrow v^{\mathcal{E}_{out}}_{\mathsf{send}} \Leftrightarrow \mathbf{X} \models \mathsf{send}(\mathcal{E}_{out})$.
2. The receiver C_2 of event $\mathcal{E}_{in}$ has to satisfy
   - $v^{\mathcal{E}_{in}}_{\mathsf{rec}} \in Var^{in}_{\mathsf{C}_2}$,
   - $\forall \mathbf{X} \in \mathsf{reach}(\mathsf{C}_2) : \mathbf{X} \models \uparrow v^{\mathcal{E}_{in}}_{\mathsf{rec}} \Leftrightarrow \mathbf{X} \models \mathsf{rec}(\mathcal{E}_{in})$.

$\mathsf{Com}^{\mathcal{R}_1}_{\mathcal{E}_{out},\mathcal{E}_{in}}$ is illustrated in Fig. 6. Variable sets of the automaton are: $Var^{loc} = \{t\}$, $Var^{out} = \{v^{\mathcal{E}_{in}}_{\mathsf{rec}}\}$ and $Var^{in} = \{v^{\mathcal{E}_{out}}_{\mathsf{send}}\}$.

The automaton implements transmission of a single event. An automaton $\mathsf{Com}_{\mathcal{R}_1}$ realizing $\mathcal{R}_1$ can be derived via parallel composition of several such automata. $\mathsf{Com}_{\mathcal{R}_1}$ is built for every $\mathcal{E}_{out} \in \mathsf{dom}(\mathcal{R}_i)$ with $\mathcal{R}_i(\mathcal{E}_{out}) \neq \mathsf{disable}$. v_{rec} is set to 1 if any $\mathsf{Com}^{\mathcal{R}_i}_{\mathcal{E}_{out}}$ sets v_{rec} to 1 and v_{rec} is set to 1 if any $\mathsf{Com}^{\mathcal{R}_i}_{\mathcal{E}_{out}}$ sets $v_{\mathsf{rec}}$to 1.

Given event connectors $\mathcal{R}_1 : Ev^{out}_1 \nrightarrow Ev^{in}_2 \dot\cup \{\mathsf{disable}\}$, $\mathcal{R}_2 : Ev^{out}_2 \nrightarrow Ev^{in}_1 \dot\cup \{\mathsf{disable}\}$, we model the composition of C_1 and C_2 via $\mathcal{R}_1$ and $\mathcal{R}_2$ as parallel composition of the two controller automata and their communication automata. The above implementation mechanism satisfies Def. 18, as it is easy to see.

The above automaton is just one possible implementation of the communication link. If we would remove the transition from mode `Transmit` back to `Wait` and additionaly require that senders to seperate the sending of the same kind of event



u: $v^{\mathcal{E}}_{\mathsf{send}} = 1/$
$/t := 0$
Wait
$\dot{t} = 0$
Transmit
$\dot{t} = 1$
$0 \leq t \leq MaxSendLat$
u: $t \geq \Delta^{\mathsf{pers}}$
$/t := 0, v^{\mathcal{E}}_{\mathsf{rec}} := 0$
u: $t \geq MaxSendLat/$
$v^{\mathcal{E}}_{\mathsf{rec}} := 1, t := 0$
Rec
$\dot{t} = 1$
$t \geq MinSendLat/$
$v^{\mathcal{E}}_{\mathsf{rec}} := 1, t := 0$

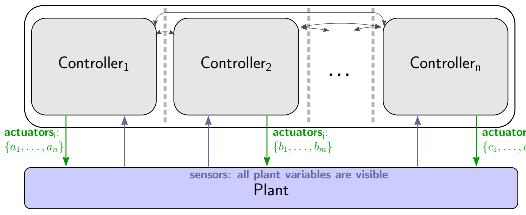
Figure 6: $\mathsf{Com}^{\mathcal{R}_i}_{\mathcal{E}_{out},\mathcal{E}_{in}}$ to model transmission of event $\mathcal{E}$
At an increasing the value of v_{send} to 1, the automation transits from mode `Wait` to mode `Transmit`. Mode `Transmit` is eventually left to enter either mode `Wait` or mode `Rec`. `Transmit` may be left without setting v_{rec}. Then the event is lost. `Transmit` is left urgently, if *MaxSendLat* time was spend, then variable v_{rec} is set to 1 and mode `Rec` is entered. `Transmit` may be left as soon as *MinSendLat* time was send and $v_{\mathsf{rec}}$is set to 1 and mode `Rec` is entered. Mode `Rec` has a fixed dwell time of $\Delta^{\mathsf{pers}}$to signal that an inevent has been received. An event is also lost, if it is sent while the automaton is not at mode `Wait`.

by more than *MaxSendLat* − *MinSendLat*, we would rule out loosing events during transmission.

5 Composing Controllers

In the following we will formally introduce the composition of controllers. In this paper parallel composition is the operation to build composed controllers from simpler controllers.

Within the composite controller the composed controllers together achieve a global control task, while each subcontroller is in charge of its respective actuators, A_i, and has to achieve its local control task. We target application classes where the control of actuators of different controllers will only be loosely coupled, whereas the control of actuators A_i within the same controller may be coupled tightly (cf. Fig. 7).



Controller$_1$
Controller$_2$
...
Controller$_n$
actuators$_i$: $\{a_1, \ldots, a_n\}$
actuators$_j$: $\{b_1, \ldots, b_m\}$
actuators$_j$: $\{c_1, \ldots, c_m\}$
sensors: all plant variables are visible
Plant

Figure 7: A composite controller of parallel subcontrollers. Each subcontroller controls its set of actuators. The controllers are loosely coupled. For optimization controllers communicate via events.

Example 9 *We already sketched local and global objectives in our ADAS running example (cf. Exp. 4, 3.A): We compose* SC *and* VC *to build a global controller* $\mathsf{SC} \parallel \mathsf{VC}$*.* SC *is in charge of* $\dot{\beta}_{\mathsf{ori}}$ *and* VC *determines* $acc = \dot{vel}$*. The local objective of* SC *is to steer the car to the center of the targeted lane. Local objective of* VC *is to make the car drive at a velocity that is safe and at best the user chosen velocity. The global objectives are: (i) the car has to stabilize on the center of its lane, and (ii) the centrifugal force on a passenger* ($|F| = |v \cdot \dot{\beta}_{\mathsf{ori}} \cdot m|$) *does not exceed a certain threshold.*

In the following we will define, when and how two controllers may be composed to build a composite controller.

Figure 8 illustrates the composition step. From separate concurrent controllers (left), each with its assumption and guarantee, a composite controller (right) with a pair of assumption and guarantee is generated. We allow that one controller contributes to the other's assumptions. Remaining assumptions φ^{assm}_{rem} are propagated to the composite controller. The remaining assumption φ^{assm}_{rem} refers to the environment of the composite, that is φ^{assm}_{rem} refers to the composite's sensors and the plant actuators that remain open after composition.[4] Outevents of the one controller must be connected to compatible inevents of the other controller.

[4]We assume that composed controllers C_1 and C_2 are variable disjoint except on their sensors, so that φ^{assm}_{rem} refers to $((Var^{in}_{\mathsf{C}_1} \cup Var^{in}_{\mathsf{C}_2}) \cap \mathsf{Sens}) \cup (\mathsf{Act} \setminus (Var^{out}_{\mathsf{C}_1} \cup Var^{out}_{\mathsf{C}_2}))$.

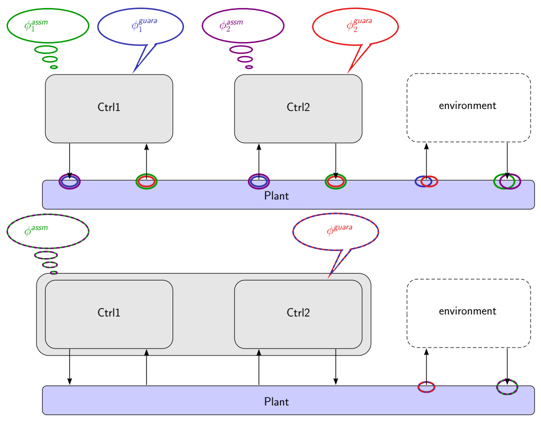


Figure 8: Composing controllers: Subcontrollers contribute to the assumptions of other subcontrollers. Remaining assumptions are propagated to the composite controller. On the left, subcontrollers still rely on their local assumptions and promise local guarantees, on the right the composed component needs only the remaining assumptions and establishes combined guarantees.

A simple precondition for composability is captured by the notion *structurally* $\|$-*composable*. We call two controllers for plant P *structurally* $\|$-*composable* if their output variables are disjoint, so that we get no write-conflict on plant actuators.

Definition 25 *structurally* $\|$-*composable Given two controller* C_1 *and* C_2. C_1 *and* C_2 *are structurally* $\|$-*composable on plant* P, *iff they are both applicable to* P *and* $Var_{\mathsf{C}_1} \cap Var_{\mathsf{C}_2} \subseteq \mathsf{Sens}$.

5.1 Composing Controllers and Linking Events

We introduced event connectors as partial mapping from outevents of one component to inevents of another component (cf. Def. 13). To formalize that only *compatible* events get connected we introduce the notion of *admissible event connector*. An admissible event connector only connects an outevent $\mathcal{E}_{out}$ to an inevent $\mathcal{E}_{in}$ if the promise given by $\mathcal{E}_{out}$ (together with the remaining assumption φ^{assm}_{rem}) implies the assumptions of the triggered inevent $\mathcal{E}_{in}$ and if the assumption associated to $\mathcal{E}_{out}$ is satisfied by the guarantee of $\mathcal{E}_{in}$ (together with φ^{assm}_{rem}).

Definition 26 ***admissible event connector*** *Given* $\mathsf{C}_1 = (\mathsf{Art}_1, P, \phi^{assm}_{\mathsf{C}_1})$ *and* $\mathsf{C}_2 = (\mathsf{Art}_2, P, \phi^{assm}_{\mathsf{C}_2})$ *are composable controllers and given an* event connector $\mathcal{R}$ *for* $(\mathsf{Art}_1, \mathsf{Art}_2)$, $\mathcal{R} : Ev^{out}_{\mathsf{Art}_1} \nrightarrow Ev^{in}_{\mathsf{Art}_2} \,\dot{\cup}\, \{\mathsf{disable}\}$. *Let* φ^{assm}_{rem} *be a hybrid predicate over* $Var^{assm}_{rem} := ((Var^{in}_{\mathsf{C}_1} \cup Var^{in}_{\mathsf{C}_2}) \cap \mathsf{Sens}) \cup (\mathsf{Act} \setminus (Var^{out}_{\mathsf{C}_1} \cup Var^{out}_{\mathsf{C}_2}))$.

An event connector $\mathcal{R}$ *is* admissible for composition of C_1, C_2 under φ^{assm}_{rem} *iff it holds* $\forall \mathcal{E}_{out} \in \mathcal{R}^{-1}(Ev^{in}_{\mathsf{C}_2}) \cap Ev^{out}_{\mathsf{C}_1}$:

1. $\mathcal{E}_{out}.\varphi^{prom}_{\mathsf{C}_1} \wedge \varphi^{assm}_{rem} \Rightarrow \ \mathcal{R}(\mathcal{E}_{out}).\varphi^{assm}_{\mathsf{C}_2} \wedge$
2. $\mathcal{R}(\mathcal{E}_{out}).\varphi^{prom}_{\mathsf{C}_2} \wedge \varphi^{assm}_{rem} \Rightarrow \ \mathcal{E}_{out}.\varphi^{assm}_{\mathsf{C}_1}$

Note, that φ^{assm}_{rem} refers to Var^{assm}_{rem} and $\varnothing = Var^{assm}_{rem} \cap (Var^{out}_{\mathsf{C}_1} \cup Var^{out}_{\mathsf{C}_2})$. Hence an event that is sent between C_1 and C_2 defines constraints on actuators that are otherwise unconstrained via the remaining assumption φ^{assm}_{rem}. The additional assumptions of C_2 have to be satisfied by the remaining assumption (cf. Def. 26.1). An event that is received from C_2 implies constraints on actuators of C_2; which are also otherwise unconstrained via the remaining assumption φ^{assm}_{rem}. The additional assumptions of C_1 have to be satisfied by the remaining assumption (cf. Def 26.2).

Sending an event means offering a contract to the receiver. It is the obligation of the sender to establish the event promise as long as the event is active – given its dynamic assumptions are satisfied. When a controller receives an event and it becomes active, it is also obliged to satisfy the contract. In the latter case it has to establish the promises associated to an active inevent. Note, that at the interface a base guarantee will be specified, that implies any such event promise (cf. 15) – so ignoring an event is fine. The obligations arising from event-based communication are formally introduced in Def. 27.

Definition 27 ***Event-contractual Obligations*** *Given event connectors $\mathcal{R}_1$ for $(\mathsf{Art}, \mathsf{Art}_2)$ and $\mathcal{R}_2$ for $(\mathsf{Art}_2, \mathsf{Art})$.*
A component C establishes its event-contractual obligations *iff*

$$\mathsf{C} \models_{(true)} \Box_{\leq \Delta^{time}_{\mathsf{lat}}} \Big(\bigwedge_{\mathcal{E}_{out} \in \mathsf{dom}(\mathcal{R}_1)} (\mathsf{active\_send}(\mathcal{E}_{out}) \Rightarrow \mathcal{E}_{out}.\varphi^{prom}) \wedge$$
$$\bigwedge_{\mathcal{E}_{in} \in \mathsf{image}(\mathcal{R}_2) \cap Ev^{in}_1} (\mathsf{active\_rec}(\mathcal{E}_{in}) \Rightarrow \mathcal{E}_{in}.\varphi^{prom}) \Big) \ \mathsf{UNLESS} \ \neg \varphi^{assm}_{dyn\ 1}.$$

For brevity we often refer to the global event connector of a composed system instead of enumerating all communication links.

Definition 28 ***global event connector*** *Let S be a composed system built of controllers CC_i. The* global event connector *$\mathcal{R}_S$ is defined as $\mathcal{R}_S ::= \bigcup_{\mathcal{R}_{C_i \in S}} \mathcal{R}_{C_i} : Ev^{out}(C_i) \to Ev^{in}(C_j) \dot\cup \{\mathsf{disable}\}$.*

The composed control component will hide events send between its subcomponents.

5.2 Composability

We now define when two controller implementations are composable via event connectors. While structurally $\|$-composability is a simple constraint that implies that both controllers are applicable to the plant and the absence of write-conflicts, ${}_{\mathcal{R}_1,\mathcal{R}_2}\|$-composability is far more. The subcontrollers themselfes need to satisfy certain constraints and the subcomponents need to be compatible to each other (e.g. in terms of assumptions and guarantees). The following definition captures both kinds of constraints.

Definition 29 ${}_{\mathcal{R}_1,\mathcal{R}_2}\|$***-Composability of Controller Implementations*** *Let C_1 and C_2 be two implementations. Let Var^{assm}_{rem} be $((Var^{in}_{\mathsf{C}_1} \cup Var^{in}_{\mathsf{C}_2}) \cap \mathsf{Sens}) \cup (\mathsf{Act} \setminus (Var^{out}_{\mathsf{C}_1} \cup Var^{out}_{\mathsf{C}_2}))$.*
C_1 and C_2 are ${}_{\mathcal{R}_1,\mathcal{R}_2}\|$-composable on P iff

1. *C_1 and C_2 are structurally $\|$-composable on P,*

2. *C_1 and C_2 are consistent event senders (cf. Def. 22),*

3. *there is an assumption predicate φ_{rem}^{assm} on Var_{rem}^{assm} such that*
   (i) *event connectors $\mathcal{R}_1 : Ev_1^{out} \nrightarrow Ev_2^{in} \dot\cup \{\mathsf{disable}\}, \mathcal{R}_2 : Ev_2^{out} \nrightarrow Ev_1^{in} \dot\cup \{\mathsf{disable}\}$ are admissible for composition under φ_{rem}^{assm} (cf. Def 26),*
   (ii) *C_1 guarantees the $(\mathcal{R}_1, \mathcal{R}_2, \varphi_{rem}^{assm})$-contractual obligation and*
   C_2 guarantees the $(\mathcal{R}_2, \mathcal{R}_1, \varphi_{rem}^{assm})$-contractual obligation according to Def. 27,
   (iii) *Let $Ev_i^{intl} = Ev_{\mathsf{C}_i}^{in} \cap \mathcal{R}_j(Ev_{\mathsf{C}_j}^{out})$.*
   Let $Ev_i^{extl\text{-}in}$ be the set of events $Ev_i^{in} \setminus \mathsf{image}(\mathcal{R}_j)$ of C_i that are not send(able) from C_j.
   Let $Ev_i^{extl\text{-}out}$ be the set of events $Ev_i^{out} \setminus \mathsf{dom}(\mathcal{R}_i)$ of C_i that are not send(able) to C_j.
   Let $\varphi_{dyn_i}^{rem}$ be $dynAssm(\varphi_{rem}^{assm}, Ev_i^{extl\text{-}in}, Ev_i^{extl\text{-}out})$.

   $\mathsf{C}_i \parallel P \models_{(true)} \Box_{\leq \Delta_{lat}^{time}} \big(\neg\varphi_{dyn_i}^{rem} \vee \varphi_j^{assm} \vee \bigvee_{\mathcal{E} \in Ev_i^{intl}} \mathsf{active\_rec}(\mathcal{E})\big) \mathsf{UNLESS} \neg(\varphi_{dyn\ i}^{assm} \wedge \varphi_{dyn_i}^{rem})$
   (Base assumptions of the other component are implied by this component and the global assumptions.)

4. *$\mathsf{C}_i \parallel P$ is lockfree under $\phi_{\mathsf{C}_i}^{assm}$ for Δ_{lat}^{time}, $i \in \{1, 2\}$.*

   (Forward progress: As long as the controller's assumption is satisfied and is running on its plant, the controller-plant composite will be lockfree for the next Δ_{lat}^{time} time instances (cf. Def. 5))

5. *Let Ev_{in}^{intl} be the set of internally receivable events $Ev_{in}^{intl} = \bigcup_{i\in\{1,2\}}\{\mathcal{E}_{in} \mid \exists \mathcal{E}_{out} \in Ev_{\mathsf{C}_i}^{out} : \mathcal{R}_i(\mathcal{E}_{out}) = \mathcal{E}_{in}\}$.*
   Let $\mathcal{E}.t_{\mathsf{rec}}$ denote the time at which an event $\mathcal{E}$ is received and $\mathcal{R}^{-1}(\mathcal{E}).t_{\mathsf{send}}$ the time at which the corresponding outevent has been sent.

   $$\begin{aligned}&\mathsf{C}_1 \parallel \mathsf{C}_2 \parallel P \models_{(\varphi_{rem}^{assm})} \Box(\exists\, \mathcal{E}_{\mathsf{rec}} \in Ev_{in}^{\mathsf{intl}} : \mathsf{rec}(\mathcal{E}_{\mathsf{rec}}) \ \wedge\ \mathcal{E}_{\mathsf{send}} = \mathcal{R}^{-1}(\mathcal{E}_{\mathsf{rec}}) \Rightarrow \\ &\quad [\mathcal{E}_{\mathsf{rec}}.t_{\mathsf{rec}} + \mathcal{E}_{\mathsf{rec}}.dueTime, \mathcal{E}_{\mathsf{rec}}.t_{\mathsf{rec}} + \mathcal{E}_{\mathsf{rec}}.dueTime + \mathcal{E}_{\mathsf{rec}}.duration] \subseteq \\ &\qquad [\mathcal{E}_{\mathsf{send}}.t_{\mathsf{send}} + \mathcal{E}_{\mathsf{send}}.dueTime, \mathcal{E}_{\mathsf{send}}.t_{\mathsf{send}} + \mathcal{E}_{\mathsf{send}}.dueTime + \mathcal{E}_{\mathsf{send}}.duration])\end{aligned}$$
   (Sent and received events match in time: the time interval of a sent event covers the time interval of a received event.)

We also say, that $\mathcal{C}_1$ and $\mathcal{C}_2$ are ${}_{\mathcal{R}_1,\mathcal{R}_2}^{\ \ \parallel}$-composable on P under assumption φ_{rem}^{assm}.

Definition 30 formally introduces the *composite of two controllers*, i.e., the result of composing two controllers. Basically, Tte composite is the parallel composition of controllers and their communication link, but the composite gets an updated assumption, φ_{rem}^{assm}, and updated event sets. Inevents of the composite are the union of inevents of the subcontrollers minus inevents that got connected. Similarly, outevents of the composite are the union of outevents of the subcontrollers minus those that got connected (or hidden). Aso, output variables of the communication link and event variables that got connected are removed.

Definition 30 $({}_{\mathcal{R}_1,\mathcal{R}_2}^{\ \ \parallel}, \varphi_{rem}^{assm})$***-composite controller*** *Let C_1 and C_2 be implementations.*
Let $\mathcal{R}_i$ be an event connector for (Art_i, Art_j), $\mathcal{R}_i : Ev_{\mathsf{Art}_i}^{out} \nrightarrow Ev_{\mathsf{Art}_j}^{in} \dot\cup \{\mathsf{disable}\}$,

$i \neq j \in \{1,2\}$.
Let C_1 and C_2 be ${}_{\mathcal{R}_1}\|_{\mathcal{R}_2}$ composable under φ^{assm}_{rem}.
Let $\mathsf{Com}_{\mathcal{R}_i}$ satisfy Def. 18, $i \in \{1,2\}$.
Let Var_{lnk} be the set $\bigcup_{\mathcal{R}\in\{\mathcal{R}_1,\mathcal{R}_2\}} Var^{in}_{\mathsf{Com}_{\mathcal{R}}} \cup Var^{out}_{\mathsf{Com}_{\mathcal{R}}}$ of variables that link C_1 and C_2 for event communication.

The $({}_{\mathcal{R}_1}\|_{\mathcal{R}_2}, \varphi^{assm}_{rem})$-composite controller *of C_1 and C_2 is defined as $(\mathsf{Art}_{\|}, \mathsf{P}, \varphi^{assm}_{rem})$,*

where $\mathsf{Art}_{\|} = \Big(\big((\mathsf{Com}_{\mathcal{R}_1} \parallel \mathsf{Com}_{\mathcal{R}_2} \parallel H_{\mathsf{C}_1} \parallel H_{\mathsf{C}_2}) \setminus Var_{\mathsf{lnk}}\big) \ominus \big(\mathsf{image}(\mathcal{R}_1) \cup \mathsf{image}(\mathcal{R}_2)\big),$

$(Ev^{in}_1 \setminus \mathcal{R}_2(Ev^{out}_2)) \cup (Ev^{in}_2 \setminus \mathcal{R}_1(Ev^{out}_1)),$

$(Ev^{out}_1 \setminus \mathcal{R}_1^{-1}(Ev^{in}_2)) \cup (Ev^{out}_2 \setminus \mathcal{R}_2^{-1}(Ev^{in}_1))\Big)$

Note that Var_{lnk} consists of event variables only. These event variables are removed as outputs, hidden as input and become local variables of the composite system. This implies that events have one sender and one receiver component only.
In the following $\mathcal{C}_{1\|2}$ denotes the $({}_{\mathcal{R}_1}\|_{\mathcal{R}_2}, \varphi^{assm}_{rem})$-*composite controller* of C_1 and C_2, $(\pi_i)|_{\mathcal{C}_{1\|2}}$ is the projection of the trajectory (π_i) to variables of the plant composites, $\mathcal{C}_i = \mathsf{C}_i \parallel \mathsf{P}$, where in particular a mode (m_1, m_2) of $\mathcal{C}_{1\|2}$ is mapped to m_i of $\mathcal{C}_i$.

As first result on the composite, we present Lemma 2 on dynamic contracts. The lemma expresses that within the composite event assumptions and event promises between the two components are satisfied. Whenever an inevent $\mathcal{E}_{in}$ is active in the composite, the inevent's assumptions are satisfied and its guarantee hold; the sender did send the triggering outevent and this outevent is activated. The respective outevent may not be active though, since it may be dominated by a stronger event.

Lemma 2 Event Contract Let C_1 and C_2 be two ${}_{\mathcal{R}_1}\|_{\mathcal{R}_2}$-composable controllers under φ^{assm}_{rem}. The $({}_{\mathcal{R}_1}\|_{\mathcal{R}_2}, \varphi^{assm}_{rem})$-composite of C_1 and C_2 satisfies
$\forall \mathcal{E}_{in} \in Ev^{in}(\mathsf{C}_1) \cap \mathsf{image}(\mathcal{R}_2) : \exists \mathcal{E}_{out} \in \mathsf{dom}(\mathcal{R}_2) : \exists \mathcal{E}'_{out} \in Ev^{out}(\mathsf{C}_2) :$
$\Box(\mathsf{active\_rec}(\mathsf{C}_1, \mathcal{E}_{in}) \Rightarrow$
$\mathsf{activated\_send}(\mathsf{C}_2, \mathcal{E}_{out}) \wedge \mathcal{R}_2(\mathcal{E}_{out}) = \mathcal{E}_{in} \wedge \mathsf{active\_send}(\mathsf{C}_2, \mathcal{E}'_{out}) \wedge$
$(\mathcal{E}_{in}.\varphi^{prom} \wedge \varphi^{assm}_{rem} \Rightarrow \mathcal{E}'_{out}.\varphi^{assm}) \wedge (\mathcal{E}'_{out}.\varphi^{prom} \wedge \varphi^{assm}_{rem} \Rightarrow \mathcal{E}_{in}.\varphi^{assm})$

Proof of Lemma 2 Let $\mathcal{C}_{1\|2}$ be the $({}_{\mathcal{R}_1}\|_{\mathcal{R}_2}, \varphi^{assm}_{rem})$-*composite controller* of C_1 and C_2. Let us consider a reachable state $\mathbf{X}$ of $\mathcal{C}_{1\|2}$ that satisfies $\mathsf{active\_rec}(\mathsf{C}_1, \mathcal{E}_{in})$. Let $\mathcal{E}_{in}.t_{\mathsf{rec}}$ be the time at which $\mathcal{E}_{in}$ has been received leading to activation at $\mathbf{X}$, i.e., let $\mathcal{E}_{in}.t_{\mathsf{rec}}$ be the earliest time in $(\mathbf{X}.t - \mathcal{E}_{in}.dueTime - \mathcal{E}_{in}.duration, \mathbf{X}.t - \mathcal{E}_{in}.dueTime]$ at which $\mathsf{rec}(\mathsf{C}_1, \mathcal{E}_{in})$ holds. From Def. 21 follows that such an $\mathcal{E}_{in}.t_{\mathsf{rec}}$ exists. From Def. 18.1, it follows that there is a time $\mathcal{E}_{out}.t_{\mathsf{send}}$ with $\mathcal{E}_{out}.t_{\mathsf{send}} + MinSendLat \leq \mathcal{E}_{in}.t_{\mathsf{rec}} \leq \mathcal{E}_{out}.t_{\mathsf{send}} + MaxSendLat$ at which C_2 did send an event $\mathcal{E}_{out}$ with $\mathcal{R}_2(\mathcal{E}_{out}) = \mathcal{E}_{in}$. Hence $\mathsf{activated\_send}(\mathsf{C}_2, \mathcal{E}_{out})$ holds during $[\mathcal{E}_{out}.t_{send} + \mathcal{E}_{out}.dueTime, \mathcal{E}_{out}.t_{end})$, where t_{end} is defined as $\min(\mathcal{E}_{out}.t_{send} + \mathcal{E}_{out}.dueTime + \mathcal{E}_{out}.duration, \mathcal{E}_{release}.t_{release})$, where $\mathcal{E}_{release}$ is a releasing event. As we require that $[\mathcal{E}_{in}.t_{\mathsf{rec}} + \mathcal{E}_{in}.dueTime, \mathcal{E}_{in}.t_{\mathsf{rec}} + \mathcal{E}_{in}.dueTime + \mathcal{E}_{in}.duration] \subseteq [\mathcal{E}_{out}.t_{\mathsf{send}} + \mathcal{E}_{out}.dueTime, \mathcal{E}_{out}.t_{\mathsf{send}} + \mathcal{E}_{out}.dueTime +$

$\mathcal{E}_{out}.duration]$, it follows $\mathsf{activated\_send}(\mathsf{C}_2, \mathcal{E}_{out})$ has to hold at $\mathbf{X}$ given no releasing events.

Suppose C_2 received an event $\mathcal{E}_{release}$ at $\mathcal{E}_{release}.t_{\mathsf{rec}}$ that releases $\mathcal{E}_{out}$. By Def. 21 $\mathcal{E}_{release}$ was received at earliest some time later than $\mathcal{E}_{out}.t_{\mathsf{send}} + 2 \cdot MaxSendLat$ and by Def. 19 C_1 is the sender of $\mathcal{R}^{-1}(\mathsf{C}_2, \mathcal{E}_{release})$. So C_1 did send $\mathcal{E}_{release}$ at earliest at a time later than $\mathcal{E}_{out}.t_{\mathsf{send}} + MaxSendLat$. As C_1 received $\mathcal{E}_{in}$ at latetest after maximal signal latency, $\mathcal{E}_{in}.t_{\mathsf{rec}} \leq \mathcal{E}_{out}.t_{\mathsf{send}} + MaxSendLat$, $\mathcal{E}_{release}$ was hence send as response of C_1 to receiving $\mathcal{E}_{in}$. But then by Def. 21, $\mathsf{active\_rec}(\mathsf{C}_1, \mathcal{E})$ does not hold at C_1.

So far we have shown that at a state $\mathbf{X}$ $\mathsf{activated\_send}(\mathsf{C}_2, \mathcal{E}_{out})$ holds for an $\mathcal{E}_{out}$ with $\mathcal{R}_2(\mathcal{E}_{out}) = \mathcal{E}_{in}$. We now show that at C_2 a send event is active that implies the contract of $\mathcal{E}_{in}$. From Def. 21 follows that $\mathcal{E}_{out}$ is active or an event $\mathcal{E}'_{out}$ is active with $\mathcal{E}_{out} \preceq \mathcal{E}'_{out}$. By Def. 26 $(\mathcal{E}_{out}.\varphi^{prom} \wedge \varphi^{assm}_{rem} \Rightarrow \mathcal{E}_{in}.\varphi^{assm}) \wedge (\mathcal{E}_{in}.\varphi^{prom} \wedge \varphi^{assm}_{rem} \Rightarrow \mathcal{E}_{out}.\varphi^{assm})$ holds. As $\mathcal{E}'_{out}.\varphi^{prom} \Rightarrow \mathcal{E}_{out}.\varphi^{prom} \wedge \mathcal{E}_{out}.\varphi^{assm} \Rightarrow \mathcal{E}'_{out}.\varphi^{assm}$, an active event at C_2 satisfies the contract offered to C_1 via $\mathcal{E}_{in}$.

■

So, whether an incoming event is releasing depends on its timing wrt a preceding send outevent, its activation times, assumptions and guarantees, but also on the source of the event. Whereas all but the last information is encoded in the triggered inevent at the receiver's interface, the source is determined by the event connector. So that this information is derived from the connection structure of automata.

5.3 Assume Guarantee Reasoning

According to the following Lemma 3 a component establishes the other component's assumptions, if its own assumptions and the remaining assumptions are given. So C_1 (on the plant and connected via $\mathcal{R}$) establishes ${\varphi^{assm}_{dyn}}_2$ at each state for the future of $\Delta^{time}_{\mathsf{lat}}$, if the state has been reached while ${\varphi^{assm}_{dyn}}_1 \wedge \varphi^{assm}_{rem}$ holds and as long as the remaining assumption continues to hold.

Lemma 3 Assumptions of composable components Given two components C_1 and C_2 that are $\underset{\mathcal{R}_1,\mathcal{R}_2}{\|}$-composable under assumptions φ^{assm}_{rem}.
Let Ev_2^{extl} be the set of events $Ev_2^{in} \setminus \mathsf{image}(\mathcal{R}_1) \cup Ev_2^{out} \setminus \mathsf{dom}(\mathcal{R}_2)$ of C_2 that are not used to communicate with C_1.
Let $\mathcal{C}_1^{\mathcal{R}}$ be $(\mathsf{C}_1 \parallel \mathsf{P} \parallel \mathsf{Com}_{\mathcal{R}_1} \parallel \mathsf{Com}_{\mathcal{R}_2})$, the composition of controller C_1, communication links to C_2 and plant.
Let φ^{rem}_{dyn} be $dynAssm(\varphi^{assm}_{rem}, Ev_2^{extl\text{-}in}, Ev_2^{extl\text{-}out})$.

$$\mathcal{C}_1^{\mathcal{R}} \models_{(true)} (\Box_{\leq \Delta^{time}_{\mathsf{lat}}} ({\varphi^{assm}_{dyn}}_2 \vee \neg \varphi^{rem}_{dyn})\ \mathsf{UNLESS}\ \neg {\varphi^{assm}_{dyn}}_1)\ \mathsf{UNLESS}\ \neg \varphi^{rem}_{dyn}.$$

Proof of Lemma 3 We show that $(\Box_{\leq \Delta^{time}_{\mathsf{lat}}} {\varphi^{assm}_{dyn}}_2 \mathsf{UNLESS} \neg {\varphi^{assm}_{dyn}}_1) \mathsf{UNLESS} \neg \varphi^{rem}_{dyn}$ holds. Therefore let us consider a time t up to which ${\varphi^{assm}_{dyn}}_1$ and φ^{rem}_{dyn} are satisfied and assume that ${\varphi^{assm}_{dyn}}_2$ has been established at all times $t', t' < t$.

We show that $\Box_{\leq \Delta^{time}_{\mathsf{lat}}} {\varphi^{assm}_{dyn}}_2$ holds at time t. Therefore let us consider a time $t', t \leq t' \leq t + \Delta^{time}_{\mathsf{lat}}$, and show that ${\varphi^{assm}_{dyn}}_2$ holds at t'.

We distinguish the following cases for $\mathcal{C}_2$ (cf. Def. 23):

1. $\mathcal{C}_2$ itself did not send any event and

a) $\mathsf{active\_rec}(\mathsf{C}_2, \mathcal{E}_{in})$ does not hold. For this case we have to show that φ_2^{assm} has to hold.

b) $\mathsf{active\_rec}(\mathsf{C}_2, \mathcal{E}_{in})$ is *true* at time t for a $\mathcal{E}_{in} \in Ev^{extl}$. Then $\mathcal{E}_{in}.\varphi^{assm}$ has to hold, which directly follows from φ_{dyn}^{rem}.

c) $\mathsf{active\_rec}(\mathsf{C}_2, \mathcal{E}_{in})$, $\mathcal{E} \in Ev^{intl}$, holds, then we have to show that $\mathcal{E}_{in}.\varphi^{assm}$ holds.

2. $\mathcal{C}_2$ currently has an $\mathsf{active\_send}(\mathsf{C}_2, \mathcal{E}_{out})$ then we have to show $\mathcal{E}_{out}.\varphi^{assm}$ has to hold.

   a) In case $\mathcal{E}_{out}$ in Ev^{extl}, this is implied by φ_{dyn}^{rem}.

Note, that $\mathcal{E}.\varphi^{assm}(x)$ for a $\mathcal{E} \in Ev_2^{in}$ implies $\varphi_2^{assm}(x)$.

Since we assume that up to t ${\varphi_{dyn}^{assm}}_1$ holds, by Def. 29.3iii C_1 hence guarantees $\Box_{\leq \Delta_{\mathsf{lat}}^{time}} ({\varphi^{assm}}_2 \vee \bigvee_{\mathcal{E} \in Ev_1^{intl}} \mathsf{active\_rec}(\mathcal{E}))$. By Def. 29.3ii C_1 satisfies its event-contractual obligations as defined in Def. 27. Hence C_1 guarantees $\big(\bigwedge_{\mathcal{E}_{out} \in \mathsf{dom}(\mathcal{R}_1)} (\mathsf{active\_send}(\mathcal{E}_{out}) \Rightarrow \mathcal{E}_{out}.\varphi^{prom}) \wedge \bigwedge_{\mathcal{E}_{in} \in \mathsf{image}(\mathcal{R}_2) \cap Ev_1^{in}} (\mathsf{active\_rec}(\mathcal{E}_{in}) \Rightarrow \mathcal{E}_{in}.\varphi^{prom})\big)$. Since $\mathcal{R}_1$ and $\mathcal{R}_2$ are admissible, this implies $\big(\bigwedge_{\mathcal{E}_{out} \in \mathsf{dom}(\mathcal{R}_1)} (\mathsf{active\_send}(\mathcal{E}_{out}) \Rightarrow \mathcal{R}_1(\mathcal{E}_{out}).\varphi^{assm}) \wedge \bigwedge_{\mathcal{E}_{in} \in \mathsf{image}(\mathcal{R}_2) \cap Ev_1^{in}} (\mathsf{active\_rec}(\mathcal{E}_{in}) \Rightarrow \forall \mathcal{E}_{out} \in \mathcal{R}_2^{-1}(\mathcal{E}_{in}) : \mathcal{E}_{out}.\varphi^{assm})\big)$.
We distinguish the following two cases for $\mathcal{C}_1$ at time t':

- $\neg \bigvee_{\mathcal{E}_{in} \in Ev_1^{intl}} \mathsf{active\_rec}(\mathsf{C}_1, \mathcal{E}_{in})$ holds.
  C_1 guarantees φ_2^{assm} by Def. 29.3iii. So case 1a is satisfied. For case 1c and an inevent $\mathcal{E}_{in} \in Ev_1^{intl}$ follows by Lemma 2, that the inevent's assumption is satisfied. The assumption for case 2 holds, since the assumption of any send event is weaker than the base assumption according to Def.14.

- $\bigvee_{\mathcal{E}_{in} \in Ev_1^{intl}} \mathsf{active\_rec}(\mathsf{C}_1, \mathcal{E}_{in})$ holds.
  By definition of composite controllers (Def. 30) and Lemma 2, C_2 did send the according events; i.e. case 2 follows for C_2. Let *activeEvs* be the set of internal inevents currently active at C_1. C_1 guarantees for all $\mathcal{E}_{in} \in \mathit{activeEvs}$ $\mathcal{E}_{in}.\varphi^{prom}$ by Def. 29.3ii and Def. 27. This implies by Def. 26 that for any $\mathcal{E}_{out} \in \mathcal{R}^{-1}(\mathit{activeEvs})$ $\mathcal{E}_{out}.\varphi^{assm}$ holds. So C_2's assumptions are satisfied.

■

In section 3 we briefly introduced the basic set up of plant, controllers and communication links. In a nutshell, the controller-plant interaction is restricted. Controllers are allowed to read plant variables via sensors and write on plant actuators. The plant makes sensor values change continuously, not discrete. Actuators may be set discretely and also evolve continuously. Also the plant has the simple assumption *true*. In the following we refer to the controller implementation as C_i and to the controller-plant composite $\mathsf{C}_i \parallel \mathsf{P}$ as $\mathcal{C}_i$.
For the following central lemma saying *"A global run imply local runs"* (Lemma 4), we examine runs of automata stepwise. We consider as a *step* a discrete transition execution as well as a continuous flow of a maximal length of $\Delta_{\mathsf{lat}}^{time}$. Longer continuous flows are considered as concatenation of steps. This is captured in Def. 31.

Definition 31 $\Delta_{\mathsf{lat}}^{time}$***-Length of trajectory*** *The* Δ_{lat}^{time}*-length of a trajectory* (π_i) *is the number of discrete transition firings plus the number of* Δ_{lat}^{time}*-flow steps.*

A Δ_{lat}^{time}*-flow step ends after a duration of* Δ_{lat}^{time} *unless the flow is terminated by a switching time* $t \in \{t_i \mid i \in Time\}$ *earlier than* Δ_{lat}^{time}*.*

For non-maximal trajectories we assume a switching time to indicate the end of the trajectory.

A run of the composite of two composable controllers on the plant satisfies a strong relation with runs of its subcontrollers: If started on according states, the subcontrollers have according runs (including the switching times) and reached states satisfy a projection relation.

Lemma 4 A global run implies local runs

Given C_1 and C_2, two ${}_{\mathcal{R}_1,\mathcal{R}_2}\|$-composable controllers on P under assumption φ_{rem}^{assm}.
Let $\mathsf{C}_{1\|2}$ be the $({}_{\mathcal{R}_1,\mathcal{R}_2}\|, \varphi_{rem}^{assm})-$composite of C_1 and C_2 with inevents Ev^{in} and outevents Ev^{out}.
Let φ_{dyn}^{rem} be $dynAssm(\varphi_{rem}^{assm}, Ev^{in}, Ev^{out})$.
Let (π_i) be a finite trajectory of $\mathsf{C}_{1\|2}$ from X_0 to $X_{(\pi_i)}$ satisfying φ_{dyn}^{rem}, where X_0 is a reachable state of $\mathcal{C}_{1\|2}$ such that $X_0|_{\mathcal{C}_j}$ is reachable on $\mathcal{C}_j$ under assumption ${\varphi_{dyn}^{assm}}_j$.

For each subcomponent $\mathcal{C}_j$, $j \in \{1, 2\}$, there is a trajectory (π_i^j) on $\mathcal{C}_j$ from $X_0|_{\mathcal{C}_j}$ to $X^j_{(\pi_i^j)}$ such that

- $X^j_{(\pi_i^j)} = X_{(\pi_i)}|_{\mathcal{C}_j}$
- assumption ${\varphi_{dyn}^{assm}}_j$ holds during (π_i) on $\mathcal{C}_j$, and
- $(\pi_i^j) = (\pi_i)|_{\mathcal{C}_j}$

Proof of Lemma 4 The proof is by induction on the $\Delta_{\mathsf{lat}}^{time}$-length l of the trajectory (π_i). We inductively construct trajectories (π_i^1) on $\mathcal{C}_1$, (π_i^2) on $\mathcal{C}_2$ for the given trajectory (π_i) and show that any state $\mathbf{X}$ reached via a trajectory (π_i) on $\mathcal{C}_{1\|2}$ can be projected to a state $\mathbf{X}^j$ of subsystem $\mathcal{C}_j$ that is reachable via a trajectory (π_i^j) with $(\pi_i^j) = (\pi_i)|_{\mathcal{C}_j}$ and $\mathbf{X}|_{\mathcal{C}_j} = \mathbf{X}^j$.

This holds for case $l = 0$ trivially.

For the induction step, let us consider a run (π_i) of length l on $\mathcal{C}_{1\|2}$ from $\mathbf{X}_0$ to $\mathbf{X}$ and a step π' from $\mathbf{X}$ to $\mathbf{X}'$ both under assumption φ_{dyn}^{rem}. Let (π_i^j) be the run on $\mathcal{C}_j$ under assumption ${\varphi_{dyn}^{assm}}_j$ corresponding to (π_i), which exists by the induction hypothesis. Let $\mathbf{X}^1$ be the state reached via (π_i^1) and $\mathbf{X}^2$ be the state reached via (π_i^2) on $\mathcal{C}_2$.

Note, that • since all visited states on $\mathcal{C}_{1\|2}$ along (π_i) satisfy the projection relation with according states on $\mathcal{C}_1$ and $\mathcal{C}_2$, it follows by Lemma 1 that ${\varphi_{dyn}^{assm}}_j$ also holds along (π_i), • as $\Theta(m_1, m_2)$ is defined as $\Theta_1(m_1) \wedge \Theta_2(m_2)$, it follows by Lemma 1 that any state reached on $\mathcal{C}_{1\|2}$, projected to a subsystem $\mathcal{C}_i$ satisfies the invariant of $\mathcal{C}_i$.

Let us first summarize important aspects of φ_{dyn}^{rem}. φ_{rem}^{assm} constrains only variables v where (i) $v \in \mathsf{Sens} \cap (Var_{\mathsf{C}_1}^{in} \cup Var_{\mathsf{C}_2}^{in})$ and sensor variables cannot be changed discretely, or (ii) $v \in \mathsf{Act} \setminus (\mathit{Var}^{out}_{\mathsf{C}_1} \cup \mathit{Var}^{out}_{\mathsf{C}_2})$, which can be changed discretely or continuously change. φ^{rem}_{dyn} additionally expresses that external inevents are truthful and the assumptions of external outevents are satisfied, i.e. expresses the assumption that in case an outevent is send, its assumptions hold.

Let us first assume that π' is a discrete step, i.e., a discrete transition tr is executed. By construction of $\mathcal{C}_{1\|2}$, $R^{dscr} = R^{dscr}_{\mathsf{C}_1} \cup R^{dscr}_{\mathsf{C}_2} \cup R^{dscr}_{\mathsf{P}} \cup R^{dscr}_{\mathsf{Com}_{\mathcal{R}_1}} \cup R^{dscr}_{\mathsf{Com}_{\mathcal{R}_2}}$.

(1) Let us say $tr \in R^{dscr}_{\mathsf{C}_1}$. As tr is enabled at state $\mathbf{X}$ of $\mathcal{C}_{1\|2}$ and by the induction hypothesis $\mathbf{X}^1 = \mathbf{X}|_{\mathcal{C}_1}$, tr is also enabled at $\mathbf{X}^1$ of $\mathcal{C}_1$. Let $\mathbf{X}'^1$ be the state reached on $\mathcal{C}_1$ via tr. The projection relation $\mathbf{X}'^1 = \mathbf{X}'|_{\mathcal{C}_1}$ holds, as tr has the same effect on $\mathcal{C}_1$ and $\mathcal{C}_{1\|2}$. By Def. 1, tr may affect $\mathit{Var}^{loc}_{\mathsf{C}_1} \cup \mathit{Var}^{out}_{\mathsf{C}_1}$. In case tr does not affect $\mathit{Var}_{\mathcal{C}_2}$ we introduce a stutter step at (π^2_i). Otherwise tr affects $\mathit{Var}^{out}_{\mathsf{C}_1} \subseteq \mathit{Var}^{in}_{\mathsf{P}}$, i.e. it affects subcomponent $\mathcal{C}_2$ via plant actuators. We show, that a state $\mathbf{X}'^2$ on $\mathcal{C}_2$ is reachable with $\mathbf{X}'^2 = \mathbf{X}'|_{\mathcal{C}_2}$, i.e. on $\mathcal{C}_2$ it is possible to set the plant actuator matching the effect of tr. By the induction hypothesis, ${\varphi^{assm}_{dyn}}_1 \wedge {\varphi^{assm}_{dyn}}_2$ hold during (π_i) on $\mathcal{C}_{1\|2}$ and ${\varphi^{assm}_{dyn}}_1$ during (π^1_i) on $\mathcal{C}_1$. As φ^{rem}_{dyn} is assumed to hold during $(\pi_i)\pi'$, $\mathcal{C}_1$ guarantees that ${\varphi^{assm}_{dyn}}_2$ holds at $\mathbf{X}'$ (cf. Lemma 3). Hence ${\varphi^{assm}_{dyn}}_2$ continuously holds during $(\pi_i)\pi'$. So the change on $\mathcal{C}_2$'s input corresponds to a discrete input change according to Def. 3.6. The case $tr \in R^{dscr}_{\mathsf{C}_2}$ follows analogously.

(2) In case the discrete transition tr is in R^{dscr}_{P}, it may only affect $\mathit{Var}^{loc}_{\mathsf{P}}$ according to Def. 8 and hence $\mathbf{X}|_{\mathit{Var}_{\mathcal{C}_i}}$ satisfies ${\varphi^{assm}_{dyn}}_i$. Obviously tr is enabled in both $\mathcal{C}_1$ and $\mathcal{C}_2$. Hence $(\pi_i)^i tr$ is a run on $\mathcal{C}_i$ satisfying continuously ${\varphi^{assm}_{dyn}}_i$.

(3) In case the discrete transition tr is in $R^{dscr}_{\mathsf{Com}_{\mathcal{R}_1}}$, it may affect $V^{loc}_{\mathsf{Com}_{\mathcal{R}_1}}$ and $\mathit{Var}^{out}_{\mathsf{Com}_1}$. If tr affects $\mathit{Var}^{out}_{\mathsf{Com}_{\mathcal{R}_1}}$, it sets $v^{\mathcal{E}}_{\mathsf{rec}} \in \mathit{Var}^{in}_{\mathsf{C}_2}$ to either 1 or 0 by Def. 18 Both satisfy $\phi^{assm}_{\mathsf{C}_2}$ (cf. Def. 12). Lemma 2 implies that the event has also been send and that C_1 has an active outevent strong enough to satisfy the dynamic contract. If tr does not affect $\mathit{Var}^{out}_{\mathsf{Com}_1}$ we introduce a stutter step at both local runs.

(4) If input variables change discretely according to φ^{rem}_{dyn}, then the change is on inputs of the composite system, i.e. on actuators $\mathsf{Act} \setminus (\mathit{Var}^{out}_{\mathsf{C}_1} \cup \mathit{Var}^{out}_{\mathsf{C}_2})$ or on external event variables. A change on open actuators satisfies by Def. 29 the assumptions of C_1 and C_2. For the case that external inevent variables of $\mathsf{C}_{1\|2}$ change, let us consider an inevent variable $v_{\mathcal{E}}$ of C_1. By Def. 29 φ^{rem}_{dyn} is such that ${\varphi^{assm}_{dyn}}_1$ is implied. As C_1 and C_2 are structurally composable any event has a well-defined recipient. As $v_{\mathcal{E}} \notin \mathit{Var}_{\mathsf{C}_2}$, we extend $(\pi_i)_2$ by a stutter step. Hence the projection relation is satisfied.

Let us now assume that π' is a flow step. By assumption, π' starts at state $\mathbf{X}$ satisfying $\mathbf{X}|_{\mathcal{C}_i} = \mathbf{X}^i$. Since $R^{\mathsf{cnt}} = R^{\mathsf{cnt}}_1 \wedge R^{\mathsf{cnt}}_2 \wedge R^{\mathsf{cnt}}_{\mathsf{P}} \wedge R^{\mathsf{cnt}}_{\mathsf{Com}}$, $\Theta^{\mathsf{inv}} = \Theta^{\mathsf{inv}}_1 \wedge \Theta^{\mathsf{inv}}_2 \wedge \Theta^{\mathsf{inv}}_{\mathsf{P}} \wedge \Theta^{\mathsf{inv}}_{\mathsf{Com}}$ and $R^{\mathsf{dscr}}_U = {R^{\mathsf{dscr}}_U}_1 \cup {R^{\mathsf{dscr}}_U}_2 \cup {R^{\mathsf{dscr}}_U}_{\mathsf{P}} \cup {R^{\mathsf{dscr}}_U}_{\mathsf{Com}}$ (cf. Def. 7), satisfaction of the projection relation of states implies satisfaction of an according flow relation and invariants during (π_i) and $(\pi_i)^i$.

Let t_{start} be the time (π_i) ends and π' starts. Without loss of generality, we assume that π' is not realizable on $\mathcal{C}_1$. By Def. 3 there is an earliest time instant t', $t_{start} \leq t'$, such $\forall t \leq t', i \in \{1, 2\} : \mathbf{X}(t)|_{\mathcal{C}_i} = \mathbf{X}^i(t)$ while

1. $(d\mathbf{X}^{\mathcal{C}_1}/dt(t'), \mathbf{X}^{\mathcal{C}_1}(t')) \not\models R^{\mathsf{cnt}}_1(M_i(t')) \wedge R^{\mathsf{cnt}}_{\mathsf{P}}(M_i(t'))$ or

2. $d\mathbf{X}^{\mathcal{C}_1 in}/dt(t') \not\models {\varphi^{env}_{cnt}}_1$ or

3. $d\mathbf{X}^{\mathcal{C}_1 in}/dt(t') \not\models \varphi^{env}_{inv\,1}$ or

As R^{cnt} has $R^{\mathsf{cnt}}_{\mathcal{C}_1}$ as conjunct, case 1 is ruled out. Since by the induction hypothesis ${\varphi^{assm}_{dyn}}_2$ holds during (π_i) and since we assume φ^{rem}_{dyn} for π', $\mathcal{C}_2$ guarantees ${\varphi^{assm}_{dyn}}_1$ to hold during π'. This rules out case 2 and 3. The case that π' is not realizable on $\mathcal{C}_2$ follows analogously. ■

Remark 1 *The proof argument ad infinitum allows to extend Lemma 4 to infinite trajectories.*

Note 6 *In our framework components are initialized in a state that satisfies all components' assumptions. Additionally they have to satisfy their guarantees for an arbitrary small time horizon without their assumptions being satisfied, i.e., relying solely on the plant. Thus, it can be concluded inductively that all components' guarantees are always fulfilled and thereby all component's assumptions are satisfied.*
The underlying rational of this restriction is motivated by signal latencies of controllers and inertia of physical models that usually prevent instantaneous responses as well as dramatic environmental changes.
To establish the assume-guarantee reasoning result it would suffice to require that one component establishes its guarantees for the small time horizon without relying on the other. But since we have a component library in mind, this distinction would have to annoted at the component interface. Which seems to make think more complicated.

The previous proof argues that one component C_1 establishes the assumption of the other component C_2 for the future of $\Delta^{time}_{\mathsf{lat}}$ and hence C_2 is obliged to behave well and establish, vice versa, C_1's assumption for the future of $\Delta^{time}_{\mathsf{lat}}$. This mutual providing of assumptions relies on the components being lockfree for the future of $\Delta^{time}_{\mathsf{lat}}$. Hence a nice feature is to derive that the composed system guarantees to be lockfree for the future of $\Delta^{time}_{\mathsf{lat}}$ as well.
To prove that also lock freedom for the composite controller follows, we need the following result on global and local runs: A global run implies that there is a local run under appropriate assumptions

Lemma 5 Given a run (π_i) on $\mathcal{C}_{1\|2}$ from $\mathbf{X}_0$ to $\mathbf{X}$ under assumption $\varphi^{assm}_{1\|2}$.
$(\pi^j_i) = (\pi_i)|_{\mathcal{C}_j}$ is a run from $\mathbf{X}_0|_{\mathcal{C}_j}$ to $\mathbf{X}|_{\mathcal{C}_j}$ for the input trajectory $(\pi_i)|_{Var^{in}_{\mathcal{C}_j}}$.

Proof The proof is a simple instantiation of the poof arguments for Lemma 4 but can be summarized briefly: A discrete transition of $\mathcal{C}_{1\|2}$ affects local variables of the constituting automata or -via outputs- inputs of partner components. The inputs of components in isolation can be chosen as necessary to match the a transition execution in the composed system, as there are no restricting assumptions. The effect of a transition in the local automata is the same as the effect in the composed system. So the projection relation is satisfied. A continuous step on the composed system is also possible on the components in isolation as the constraints (flow relation, invariants, assumptions on inputs) are weakened. ■

Lemma 6 Composite Controllers are lockfree Given C_1 and C_2, two $\|_{\mathcal{R}_1,\mathcal{R}_2}$-composable controllers on P.

If C_1 under ${\varphi_{dyn}^{assm}}_1$ and C_2 under ${\varphi_{dyn}^{assm}}_2$ are lock-free for the future of $\Delta_{\mathsf{lat}}^{time}$,

then $\mathcal{C}_{1\|2}$, the composite $\mathsf{C}_{1\|2}$ on plant P under φ_{dyn}^{rem}, is lock-free for the future of $\Delta_{\mathsf{lat}}^{time}$.

Proof Assume that a run (π_i) of $\mathcal{C}_{1\|2}$ has a lock within $\Delta_{\mathsf{lat}}^{time}$ time after a state $\mathbf{X}$ has been reached up to which φ_{dyn}^{rem} did continuously hold. Let $t_{\mathbf{X}}$ be that time at which $\mathbf{X}$ is reached. Let (π_i') be the prefix of (π_i) up to time $t_{\mathbf{X}}$, i.e. (π_i') leads to $\mathbf{X}$. Let (π_i'') be (π_i)'s the suffix such that $(\pi_i)_{i\in\{1,\ldots|(\pi_i')|+|(\pi_i'')|\}} = \pi_1' \ldots \pi_{|(\pi_i')|}' \pi_1'' \ldots \pi_{|(\pi_i'')|}''$. By Lemma 4, $(\pi_i'^j)$, the projection of $\mathcal{C}_{1\|2}$'s run (π_i') to subcomponent $\mathcal{C}_j$, is also a run of $\mathcal{C}_j$, $j \in \{1,2\}$ under assumption ${\varphi_{dyn}^{assm}}_j$ and leads to $\mathbf{X^j} = \mathbf{X}|_{\mathcal{C}_j}$. By Lemma 5, $(\pi_i''^j)$, the projection of (π_i'') to subcomponent $\mathcal{C}_j$, is also a run of $\mathcal{C}_j$ under assumption *true*, $j \in \{1,2\}$, maintaining the projection relation of reached states. It follows that $(\pi_i^j) = (\pi_i)|_{\mathcal{C}_j}$ is a run on $\mathcal{C}_j$.

Firstly, let us assume that (π_i) has infinitely many discrete transitions in finite time. Since $\mathsf{Com}_{\mathcal{R}}$ is lockfree for all input behaviors, at least for one $j \in \{1,2\}$ (π_i^j) is a run on $\mathcal{C}_j$ with infinitely many discrete transitions. This contradicts the assumption that $\mathcal{C}_j$ is lock-free for $\Delta_{\mathsf{lat}}^{time}$ after a state has been reached under ${\varphi_{dyn}^{assm}}_j$.

Now, let us assume that (π_i) is a finite maximal trajectory terminating in deadlock state $\mathbf{X}_{lck}$ from where for an input trajectory (π_i^{in}) neither further continuous nor discrete evolution is possible and $t_{lck} - t_{\mathbf{X}} < \Delta_{\mathsf{lat}}^{time}$. We show that $\mathcal{C}_j$ being lockfree implies that (π_i) can be extended, which contradicts the assumption of $\mathbf{X}_{lck}$ being a deadlock state.

As the $\mathcal{C}_j$s guarantee not to lock within $\Delta_{\mathsf{lat}}^{time}$ time after a state has been reached under ${\varphi_{dyn}^{assm}}_j$ and also plant P is lockfree, there are trajectories extending (π_i^j) by at least $\delta_{lck} = \Delta_{\mathsf{lat}}^{time} - (t_{lck} - t_{\mathbf{X}})$ for every input behavior of $\mathcal{C}_j$.

Given such a trajectory starts with a discrete transition $tr \in R_{\mathcal{C}_j}^{\mathsf{dscr}}$, tr is also executable on $\mathcal{C}_{1\|2}$, since $R_{\mathcal{C}_1}^{\mathsf{dscr}} \cup R_{\mathcal{C}_2}^{\mathsf{dscr}} \subseteq R_{\mathcal{C}_{1\|2}}^{\mathsf{dscr}}$.

So let $\mathbf{X}_{lck}$ and (π_i^{in}) be such that $\mathcal{C}_1$ and $\mathcal{C}_2$ both can only answer with continuous steps, that is no discrete transition is enabled at $\mathbf{X}_{lck}$. As both local steps start at $\mathbf{X}_{lck}$, they agree on the plant mode m_p, which specifies the possible continuous evolutions of the plant via differential inclusions $R_{\mathsf{P}}^{\mathsf{cnt}}(m_p)$. Likewise the projection relation of states implies the mode m_c of Com, m_1 of C_1 and m_2 of C_2 at state $\mathbf{X}_{lck}$ and hence the linear differential inclusions are fixed to $R_{\mathsf{Com}}^{\mathsf{cnt}}(m_c)$, $R_1^{\mathsf{cnt}}(m_1)$ and $R_2^{\mathsf{cnt}}(m_2)$.

So, a combined continuous step $\pi^{1\|2}$ executable on $\mathcal{C}_{1\|2}$ is a solution of the initial value problem with input for the system of linear differential inclusions given by

$$\begin{array}{c} R_{\mathsf{Com}}^{\mathsf{cnt}}(m_c) \\ R_{\mathsf{C}_1}^{\mathsf{cnt}}(m_1) \\ R_{\mathsf{C}_2}^{\mathsf{cnt}}(m_2) \\ R_{\mathsf{P}}^{\mathsf{cnt}}(m_p) \end{array} \tag{1}$$

with initial value implied by $\mathbf{X}_{lck}$ and input (π_i^{in}), while a step on $\mathcal{C}_j$ is a solution to the initial value problem with input for

$$\begin{array}{l} R^{\mathsf{cnt}}_{\mathsf{C}_j}(m_j) \\ R^{\mathsf{cnt}}_{\mathsf{P}}(m_p) \end{array} \tag{2}$$

and an input $(\pi_i^{in_j})$ satisfying $(\pi_i^{in_j})|_{Var^{in}_{\mathcal{C}_{1\|2}}} = (\pi_i^{in})$.

As both controller $\mathcal{C}_j$ can only start with a continuous step for such an input sequence $(\pi_i^{in_j})$, it follows that any trajectory on $\mathcal{C}_j$ from $\mathbf{X}_{lck}{}^j$ spends a time $\delta^j_{cnt} > 0$ within a continuous step. This step reaches a state where the guard of a discrete transition becomes enabled or lasts at least for δ_{lck}.

By Def. 18.2 Com is lockfree and by Def. 8 plant P is lockfree as well. As $\mathcal{C}_j$ being lockfree implies that there is a solution of Eq. (2) for any possible sequence of valuations of $Var^{out}_{\mathsf{C}_i} \cap Var^{in}_{\mathcal{C}_j}$ along (π_i^{in}) and as plant P, C_1, C_2, Com define differential inclusions for disjoint sets of variables, there is also a solution for $[t_{lck}, t_{lck} + \min(\delta^1_{cnt}, \delta^2_{cnt})]$ of

$$\begin{array}{l} R^{\mathsf{cnt}}_{\mathsf{C}_1}(m_1) \\ R^{\mathsf{cnt}}_{\mathsf{C}_2}(m_2) \\ R^{\mathsf{cnt}}_{\mathsf{P}}(m_p) \end{array} \tag{3}$$

In case $\min(\delta^1_{cnt}, \delta^2_{cnt}) < \delta_{lck}$, it follows that there is a discrete transition of $\mathcal{C}_1$ or $\mathcal{C}_2$ is enabled. So at $\mathbf{X}_{lck}$ at least a further continuous step of length $\min(\delta^1_{cnt}, \delta^2_{cnt})$ is possible at $\mathsf{C}_1 \parallel \mathsf{C}_2 \parallel \mathsf{P}$. This step is also executable on $\mathcal{C}_{1\|2}$, except when Com interrupts this continuous step on $\mathcal{C}_{1\|2}$ by executing a discrete transition.

In any case $\mathbf{X}_{lck}$ cannot be a deadlock state of $\mathcal{C}_{1\|2}$. ■

5.4 Summary

So far, we have introduced our application domain of loosely coupled controllers on a common plant (Sect. 3) where controllers may communicate with each other via events (Sect. 4). For composability of controller implementation we imposed constraints on the controllers itself and on the combination (cf. Def. 29). We established two central lemmas on which our an assume-guarantee result will be founded: a global run on the composite defines local run the subcontrollers (Lemma 4) and lockfreedom propagates from subcontrollers to the composite (Lemma 6). In the next section we lift the results on controller implementations to component interfaces. So instead of knowing the exact implementation of a controller, we know only that the underlying implementation satisfies the properties annotated at its interface.

6 Interface Specification and Implementation

To support reuse we introduce interface specifications. An interface Spec provides an external view on a control component. It specifies structural as well as behavioral aspects of the controller that enable its deployment and composition to achieve a certain control task. We annotate safety as well as stability properties at a component's interface.

To support assume-guarantee reasoning, the interface defines guarantees that the component provides and assumptions under which the component can be deployed. The interfaces are such that it can be decided based on the components' interfaces, whether two controllers may be composed. We also show that an interface for a composed component can be automatically derived from the interfaces of its constituent components.

Note 7 *Whereas before we usually refer to a controller implementation* C*, we now argue about control components* CC*, that is about controllers for which a certain interface is defined.*

6.1 Interface Specification

Interface specifications give a condensed view on a component where implementation details are hidden but sufficient information for deployment and composition have to be provided. The information provided at an interface has to describe a component's deployment context and its services sufficiently; it should enable to efficiently check the verification conditions for deployment and composition.

In Def. 33 we present our proposal for an interface specification for loosely coupled systems. The interface lists a reference to the plant the component is applicable to, sensors it depends on, and actuators it controls. The interface specifies assumptions, φ^{assm} and guarantees φ^{guara} for composition. Assumptions are constraints on the controller's sensors and the open actuator. Their values are constrained via an hybrid predicate. Guarantees are constraints on the controller's actuators and on plant sensors and is also expressed via hybrid predicates. Additionally the services of a component are described via φ^{safe}, a predicate on the plant that is guaranteed for all times, and by stability constraints on the plant, that describe that the component drives the plant into a certain stable region (cf. Def. 32). Finally, the interface describes its initial state in terms of input and output variables.

Definition 32 ***(event-conditioned) stability constraint*** *A* stability constraint *of a component* CC *for plant* P *is a pair* $(\varphi^{stab}, \Delta^{stab})$ *with*

- φ^{stab} *a convex, open first-order predicate over* $Var^{in}_{\mathsf{P}} \cup Var^{out}_{\mathsf{P}}$*,*
- $\Delta^{stab} \in Time$ *the time to reach* φ^{stab}*.*

An event-conditioned stability constraint *of a component* CC *for plant* P *is a tuple* $(\varphi^{stab}, \Delta^{stab}, Ev^{stab}, \Delta^{re})$ *with*

- $(\varphi^{stab}, \Delta^{stab})$ *a stability constraint*
- $Ev^{stab} \subseteq Ev^{in}_{\mathsf{CC}}$*, a set of events specifying the condition under which convergence is guaranteed,*
- $\Delta^{re} \in Time$*, the time to get ready for the base assumption at the end of the event-guaranteed time period.*

Definition 33 ***Interface Specification*** *A component interface specification* Spec *on plant* P *consists of*

1. *a reference to a plant* P
2. $Var^{in}_{\mathsf{Spec}} \subseteq \mathsf{Sens}$*,* $Var^{out}_{\mathsf{Spec}} \subseteq \mathsf{Act}$*, sets of real valued input and output variables, respectively.*
3. $\varphi^{assm}_{\mathsf{Spec}}$ *an hybrid predicate on* $Var^{in}_{\mathsf{Spec}} \cup (\mathsf{Act} \setminus Var^{out}_{\mathsf{Spec}})$

4. *$\varphi^{guara}_{\mathsf{Spec}}$ an hybrid predicate on $Var^{out}_{\mathsf{Spec}} \cup \mathsf{Sens}$*
5. *$\varphi^{safe}_{\mathsf{Spec}}$, an open, first-order predicate on $Var^{in}_{\mathsf{P}} \cup Var^{out}_{\mathsf{P}}$*
6. *$Stab_{\mathsf{Spec}} = \{(\varphi^{stab\ i}_{\mathsf{Spec}}, \Delta^{stab\ i}_{\mathsf{Spec}}) \mid 1 \leq i \leq N\}$, a finite set of stability constraints*
7. *$\varphi^{entry}_{\mathsf{Spec}}$, a first-order predicate on $Var^{in}_{\mathsf{Spec}} \cup Var^{out}_{\mathsf{Spec}}$ describing initial states of the component.*
8. *Ev^{in}_{Spec}, Ev^{out}_{Spec}, finite sets of events the component can receive or send, respectively. It holds $Ev^{in}_{\mathsf{Spec}} \cap Ev^{out}_{\mathsf{Spec}} = \varnothing$,*
9. *$EvStab_{\mathsf{Spec}} = \{(\varphi^{stab\ i}_{\mathsf{Spec}}, \Delta^{stab\ i}_{\mathsf{Spec}}, Ev^{stab^i}, \Delta^{re\ \ i}_{\mathsf{Spec}}) \mid 1 \leq i \leq M\}$, a finite set of event-conditioned stability constraints,*

In the sequel we also refer to the component interface specification briefly as interface or specification.

Note 8 *Sanity of assumptions and guarantees. For the sake of simplicity, we use a very restrictive form for assumptions and guarantees in our framework. In general it is possible that assumptions and guarantees can refer to both input and output variables. Although this way easily unrealisable specifications can be defined, we could allow this freedom, to support the specification of conditioned properties, e.g., a velocity controller may assume that the change in the orientation is limited at certain velocities. It certainly is possible to define syntactic restrictions as guidance for defining realizable component specifications, but its out of the scope of this paper.*

In the following we will define when a controller implementation satisfies an interface specification. In order to define when a controller implementation satisfies an event-conditioned stability constraint, we define a proposition that signals that the component does no longer guarantee convergence. This proposition is relative to the end of the event-guaranteed time period and provides the component a window in time to get ready for the base assumption. Note, that the component either determines the end of $\mathsf{active\_rec}(\mathcal{E})$ itself by sending a releasing event or knows about the end already before $\mathcal{E}$ becomes even active, as $\mathcal{E}.dueTime > 0$.

Definition 34 *Given a trajectory of component C defining a state sequence with $\mathbf{X}(t)$ for $t \geq 0$ and a set of inevents $Ev' \subset Ev^{in}_{\mathsf{C}}$.*

$\mathsf{active\_recov}(Ev')$ is true at state $\mathbf{X}(t)$ iff
$\bigvee_{\mathcal{E}Ev'} \mathsf{active\_rec}(\mathcal{E})$ is true at $\mathbf{X}(t)$ and $\bigvee_{\mathcal{E}\in Ev'} \mathsf{active\_rec}(\mathcal{E})$ is false at state $\mathbf{X}(t + \Delta^{re}))$.

Def. 35 defines when a controller implementation satisfies an interface specification. So when a new component with interface Spec and implementation C is added to the component library, these constraints have to be checked.

Definition 35 ***Interface Satisfaction*** *An implementation C satisfies the interface specification Spec (denoted as $\mathsf{C} \models \mathsf{Spec}$) iff*

1. *$\mathsf{P_C} = \mathsf{P_{Spec}}$*
   Interface and implementation consent on the plant.

2. $Var^{in}_{\mathsf{Spec}} = Var^{in}_{\mathsf{C}} \setminus CodeVarEv^{in}_{\mathsf{CC}}$, $Var^{out}_{\mathsf{Spec}} = Var^{out}_{\mathsf{C}} \setminus CodeVarEv^{out}_{\mathsf{CC}}$
   Interface and implementation have matching input and output variables.

3. $\varphi^{assm}_{\mathsf{Spec}} = \varphi^{assm}_{\mathsf{C}}$
   The interface declares the implementation's assumption.

4. $\mathcal{C} \models \varphi^{\mathsf{init}}_{\mathsf{C}} \Rightarrow \varphi^{entry}_{\mathsf{Spec}}$.
   The entry declared at the interface is guaranteed by the implementations initial state.

5. $Ev^{in}_{\mathsf{Spec}} = Ev^{in}_{\mathsf{CC}}$, $Ev^{out}_{\mathsf{Spec}} = Ev^{out}_{\mathsf{CC}}$
   The interface declares the implementation's events.

6. $\mathcal{C} \models \Box_{\leq\Delta^{time}_{\mathsf{lat}}} \varphi^{guara}_{dyn}$ UNLESS $\neg\varphi^{assm}_{dyn}$.
   Unless $\mathcal{C}$*'s dynamic assumption is violated,* $\mathcal{C}$ *fulfils its base guarantee –now and for the future of* $\Delta^{time}_{\mathsf{lat}}$.

7. $\mathcal{C}$ *is lockfree under assumption* φ^{assm}_{dyn}*for the future of*$\Delta^{time}_{\mathsf{lat}}$.

8. $\mathcal{C}$ *is a consistent event sender.*

9. $\mathcal{C} \models \neg \bigvee_{\mathcal{E}_{in} Ev^{in}} \mathsf{active\_rec}(\mathcal{E}_{in})$.
   Initially no inevent is active.

10. $\mathcal{C} \models \varphi^{safe}_{\mathsf{Spec}}$ UNLESS $\neg\varphi^{assm}_{dyn}$.
    Components guarantee $\varphi^{safe}_{\mathsf{Spec}}$.

11. *For all* $(\varphi^{stab}, \Delta^{stab}) \in Stab_{\mathsf{Spec}}$ *holds*
    $\mathcal{C} \models \Diamond_{\leq\Delta^{stab}}(\varphi^{stab}\ \mathsf{UNLESS}\ \neg\varphi^{assm}_{dyn}) \vee \neg\Box_{\leq\Delta^{stab}}\varphi^{assm}_{dyn}$.
    Components guarantee to converge to $\varphi^{stab}_{\mathsf{Spec}}$ *within* $\Delta^{stab}_{\mathsf{Spec}}$ *time units given* φ^{assm}_{dyn}*holds.*

12. *For all* $(\varphi^{stab}, \Delta^{stab}, Ev^{stab}, \Delta^{re}) \in EvStab_{\mathsf{Spec}}$ *holds*
    $$\mathcal{C} \models (\ \Box_{\leq\Delta^{stab}} \mathsf{active\_rec}(Ev^{stab}) \Rightarrow \Big(\Diamond_{\leq\Delta^{stab}}\big(\varphi^{stab}\ \mathsf{UNLESS}\ \ \mathsf{active\_recov}(Ev^{stab})\big)\Big)\ \mathsf{UNLESS}\ \neg\varphi^{assm}_{dyn},$$
    Active components guarantee convergence to $\varphi^{stab}_{\mathsf{Spec}}(Ev)$ *within* $\Delta^{stab}_{\mathsf{Spec}}$ *time units, given the event is sufficiently long active.*

Interface and implementation agree on the reference plant (1), on input and output variables (2) –variables encoding events are not visible as such at the interface–, on their events (5) and on their assumption (3) . The interface specification declares an entry φ^{entry}, that is implied by the initial state of the implementation (4) and initially no inevent is assumed to be active (9).

The implementation has to guarantee to be lockfree (7). A component is required to establish the (dynamic) guarantees for $\Delta^{time}_{\mathsf{lat}}$ as long as its dynamic assumptions hold (6) and has to maintain the plant in a safe state (10). The component also guarantees to drive the plant into the stable region φ^{stab} within Δ^{stab} time and to remain there, given its assumptions hold (11). Given stability-condition events are sufficiently long active, the component guarantees to drive the plant into the respective stable region within Δ^{stab} time and to remain there as long as the events will be valid for more than Δ^{re} (12).

6.2 Composition

In this section we will lift the composition results established for controller implementations to control components, i.e., we lift the results to controllers that we only know by their interfaces. We define criteria on interfaces when two components are composable and introduce a composite interface that is derived from the subcontrollers' interfaces.

Lemma 7 links guarantees annotated at the subcomponent interfaces to the composite implementation.

Lemma 7 Composed guarantees Given C_1 and C_2, two ${}^{\|}_{\mathcal{R}_1,\mathcal{R}_2}$-composable controllers on P under assumption φ^{assm}_{rem}, where C_i satisfies Spec_i.

The ${}^{\|}_{\mathcal{R}_1,\mathcal{R}_2}$-composite of C_1 and C_2 under assumption φ^{rem}_{dyn} satisfies $\Box_{\leq \Delta^{time}_{\mathsf{lat}}}(\varphi^{guara}_{dyn\ \mathcal{C}_1} \wedge \varphi^{guara}_{dyn\ \mathcal{C}_2})$.

Proof of Lemma 7 By Lemma 4, the dynamic local assumptions hold for all runs from of the composing component. Hence both components fulfill their guarantees for a future of $\Delta^{time}_{\mathsf{lat}}$ according to Def. 35. 6. ■

By Lemma 7 a composite implementation $\mathsf{CC}_{1\|2} := \mathsf{CC}_1\ {}^{\|}_{\mathcal{R}_1,\mathcal{R}_2}\ \mathsf{CC}_2$ establishes the conjunction of *dynamic* guarantees, or more precisely $\Box_{\leq \Delta^{time}_{\mathsf{lat}}}(\varphi^{guara}_{dyn\ \mathcal{C}_1} \wedge \varphi^{guara}_{dyn\ \mathcal{C}_2})$. The dynamic guarantee takes active in- and outevents into account. During a valid inevent a controller may provide a weakened base guarantee. Whether and when such a weakened guarantee is provided depends on whether and when events are received and made use of. As it depends on runtime information, we cannot know in advance, when a component provides its base guarantee ϕ^{guara} or when it may weaken its base guarantee in response to an inevent. However, we know that if events are received, a new contract is negotiated and the controllers adhere to the contract.

If we reason about when two controllers are composable, we need to establish that one controller alleviates the assumptions of the other. Therefore we overapproximate the guaranteed behavior of a controller component CC_i by the so-called *encapsulated guarantee* of CC_i, $\varphi^{guara}_{encaps,i} := \varphi^{guara}_i \vee \bigvee_{\mathcal{E}\in Ev^{in}_i \setminus Ev^{intl}_i} \mathcal{E}.\varphi^{prom}$ where $Ev^{intl}_i := Ev^{in}_i \cap \mathcal{R}_j(Ev^{out}_j)$. The encapsulated guarantee is the disjunction of a component's base guarantee and all (weakening) inevent promises that may be triggered by the environment apart from inevents that are sendable by CC_j. [5]

In Def. 36 we introduce with the notion of *composability of components* a sufficient condition for composability that can be decided based on the interface specification of a component.

Definition 36 $({}^{\|}_{\mathcal{R}_1,\mathcal{R}_2}, \boldsymbol{\varphi^{assm}_{rem}})$***-Composability of Component(Interface)s*** *Given two components CC_1 and CC_2 satisfying interface specification Spec_1 and Spec_2, respectively. Given $\mathcal{R}_1 : Ev^{out}_1 \to Ev^{in}_2, \mathcal{R}_2 : Ev^{out}_2 \to Ev^{in}_1$ event connectors and a hybrid predicate φ^{assm}_{rem} on $(\mathsf{Sens} \cap (Var^{in}_{\mathsf{Spec}_1} \cup Var^{in}_{\mathsf{Spec}_2})) \cup (\mathsf{Act} \setminus (Var^{out}_{\mathsf{Spec}_1} \cup Var^{out}_{\mathsf{Spec}_2}))$.*

CC_1 and CC_2 are $({}^{\|}_{\mathcal{R}_1,\mathcal{R}_2}, \varphi^{assm}_{rem})$-composable on P iff

[5]Inevent hiding results in stronger encapsulated guarantees.

1. *Spec_1 and Spec_2 are for plant P.*
2. *$Var^{out}_{\mathsf{Spec}_1} \cup CodeVarEv_{\mathsf{Spec}_1}$, $Var^{out}_{\mathsf{Spec}_2} \cup CodeVarEv_{\mathsf{Spec}_2}$ are disjoint,* (no write conflict) *only if*
3. *Let the set of externally receivable events of CC_i be $Ev^{extl}_i := Ev^{in}_i \setminus \mathcal{R}_j(Ev^{out}_j)$ and $\varphi^{guara}_{encaps,i} := \varphi^{guara}_i \vee \bigvee_{\mathcal{E}_i \in Ev^{extl}_i} \mathcal{E}_i.\varphi^{prom}$, $i \neq j \in \{1,2\}$.*

   φ^{assm}_{rem} satisfies

   a) $\varphi^{assm}_{rem} \wedge \varphi^{guara}_{encaps,i} \Rightarrow \varphi^{assm}_j, i \neq j \in \{1,2\}$

   b) $\mathcal{R}_1, \mathcal{R}_2$ are admissible for composition under φ^{assm}_{rem}

   (base assumptions of the other component are implied by this component's guarantees and the remaining assumptions)
4. $\forall \mathcal{E}_{in} \in Ev^{in}_{\mathsf{CC}_i} : (\exists \mathcal{E}_{out} \in Ev_j : \mathcal{R}_j(\mathcal{E}_{out}) = \mathcal{E}_{in} \Rightarrow$

   $\mathcal{E}_{out}.dueTime - MinSendLat \leq \mathcal{E}_{in}.dueTime \leq \mathcal{E}_{out}.dueTime + \Delta \wedge$

   $\mathcal{E}_{out}.duration - MaxSendLat - \Delta \geq \mathcal{E}_{in}.duration)$

   (send events are connected to appropriate receive events)

We also say, that $\mathcal{C}_1$ and $\mathcal{C}_2$ are ${}_{\mathcal{R}_1}\|_{\mathcal{R}_2}$-composable on P under assumption φ^{assm}_{rem}.

To compose two components with interface Spec_1 and Spec_2, respectively, we require that they are for the same plant (1) and that there is no write-conflict (also not on the internally introduced event-variables, implicitly declared via events at the interface) (2). The encapsulated guarantee $\varphi^{guara}_{encaps,i}$ and φ^{assm}_{rem}, together, imply the base guarantee of the other component, and event connectors have to be admissible under φ^{assm}_{rem} (3). This requirement makes sure, that the guarantees are strong enough for the contract between the two components.

Requirement 4 ensures that the timing for connected events is appropriate. An inevent may become active only after the triggering outevent at the sender became active. This is conservatively ensured by requiring that the inevent's due time is at least the send event's due time plus the maximal send latency. The inevent may be valid at most until the corresponding sent outevent is valid. Since the exact delay of sending an event is unknown, we again conservatively adapt the duration of the triggered inevent.

We have chosen a notion of composability for interfaces that implies that components with composable interfaces have also composable implementations (cf. Lemma 8). So, an engineer who takes components from a library does not need to examine – or even know– the internal implementation of the components in order to check whether they are composable.

Lemma 8 Composable components have composable implementations. Let CC_1 and CC_2 be two components with implementations C_1 and C_2 and interface specifications Spec_1 and Spec_2, respectively.

If component interfaces CC_1 and CC_2 are ${}_{\mathcal{R}_1}\|_{\mathcal{R}_2}$-composable under assumption φ^{assm}_{rem}, then implementations C_1 and C_2 are ${}_{\mathcal{R}_1}\|_{\mathcal{R}_2}$-composable under φ^{assm}_{rem}.

Proof of Lemma 8 The two implementations are structurally composable (29.1), since due to interface composability they both refer to the same plant (36.1) and since they satisfy their interfaces by Def. 35.1 both implementations are applicable to P. Further they may only intersect on sensors, as we assume that local variables of controllers are made disjoint prior to composition and (36.2) rules out that the two controllers have common variables in their sets $Var^{out} \cup CodeVarEv^{out}$.

29.2 follows from Def. 35. 8.

That event connectors are admissible under φ^{assm}_{rem} (29.3i) is directly implied by Def. 36.3b. 29.3ii follows by 356. To show 29.3iii, let us consider a trajectory that satisfies $\varphi^{assm}_{dyn\ i} \wedge \varphi^{rem}_{dyn}$ upto state $\mathbf{X}$. Since CC_i satisfies its interface, $\mathcal{C}_i \models \Box_{\leq \Delta^{time}_{\mathsf{lat}}} \varphi^{guara}_{dyn\ i}$ UNLESS $\neg\varphi^{assm}_{dyn\ i}$ holds (Def. 35.6). Hence $\Box_{\leq \Delta^{time}_{\mathsf{lat}}} \varphi^{guara}_{dyn\ i}$ is guaranteed after reaching $\mathbf{X}$. By Def. 36.3a holds that $\varphi^{guara}_{encaps,i} \wedge \varphi^{assm}_{rem} \Rightarrow \varphi^{assm}_{j}$ with $\varphi^{guara}_{encaps,i} = \varphi^{guara}_{i} \vee \bigvee_{\mathcal{E}_i \in Ev^{extl}_i} \mathcal{E}_i.\varphi^{prom}$. The dynamic guarantee implies the encapsulated guarantee in all cases but when CC_i receives an internal event. Hence the dynamic guarantee in conjunction with the remaining assumptions implies φ^{assm}_{j} in case CC_i does not receive an event.

Def. 29.4 is directly given by Def. 35.7.

Def. 29.5 is implied by Def. 36.4. Let $\mathcal{E}_{out}.t_{send}$ be the time an event is sent. Such an event is received at time $\mathcal{E}_{in}.t_{rec} \in [\mathcal{E}_{out}.t_{send} + MinSendLat, \mathcal{E}_{out}.t_{send} + MaxSendLat]$. The sender guarantees that the promise is truthful for all times $t \in [\mathcal{E}_{out}.t_{send} + \mathcal{E}_{out}.dueTime, \mathcal{E}_{out}.t_{send} + \mathcal{E}_{out}.dueTime + \mathcal{E}_{out}.duration)$. The event receiver makes at most use of the corresponding inevent at time $t' \in$

$$\begin{aligned}
&[\mathcal{E}_{in}.t_{rec} + \mathcal{E}_{in}.dueTime, \mathcal{E}_{in}.t_{rec} + \mathcal{E}_{in}.dueTime + \mathcal{E}_{in}.duration) \\
&\subseteq [(\mathcal{E}_{out}.t_{send} + MinSendLat) + (\mathcal{E}_{out}.dueTime - MinSendLat), \\
&\quad (\mathcal{E}_{out}.t_{send} + MaxSendLat) + (\mathcal{E}_{out}.dueTime + \Delta) + (\mathcal{E}_{out}.duration - \Delta - MaxSendLat)).
\end{aligned}$$

■

Next we give an interface for a composite component that is derived from the subcontrollers interfaces. It represents just one possible interface for a composite component, but can be derived directly from the interfaces of the subcontrollers.

Definition 37 **($\mathsf{Spec}_1 \ {}_{\mathcal{R}_1}\!\overset{\|}{}_{\mathcal{R}_2}\ \mathsf{Spec}_2, \varphi^{assm}_{rem}$)-*interface*** *Given two composable interface specifications Spec_1 and Spec_2. Let φ^{assm}_{rem} be a hybrid predicate on $Var^{in}_{\mathsf{Spec}_{1\|2}} \cup (\mathit{Act} \setminus Var^{out}_{\mathsf{Spec}_{1\|2}})$.*

The interface specification $\mathsf{Spec}_{1\|2} = (\mathsf{Spec}_1 \ {}_{\mathcal{R}_1}\!\overset{\|}{}_{\mathcal{R}_2}\ \mathsf{Spec}_2, \varphi^{assm}_{rem})$ is defined as

1. *a reference to plant P,*
2. $Var^{in}_{\mathsf{Spec}_{1\|2}} := Var^{in}_{\mathsf{Spec}_1} \cup Var^{in}_{\mathsf{Spec}_2}$, $Var^{out}_{\mathsf{Spec}_{1\|2}} := Var^{out}_{\mathsf{Spec}_1} \dot\cup Var^{out}_{\mathsf{Spec}_2}$,
3. $\varphi^{assm}_{\mathsf{Spec}_{1\|2}} := \varphi^{assm}_{rem}$,
4. *Let the sets of internally receivable events be $Ev^{intl}_{\mathsf{Spec}_i} := Ev^{in}_{\mathsf{Spec}_i} \cap \mathcal{R}_j(Ev^{out}_{\mathsf{Spec}_j})$.* $\varphi^{guara}_{\mathsf{Spec}_{1\|2}} := (\varphi^{guara}_{\mathsf{Spec}_1} \vee \bigvee_{\mathcal{E}_1 \in Ev^{intl}_{\mathsf{Spec}_1}} \mathcal{E}_1.\varphi^{prom}) \wedge (\varphi^{guara}_{\mathsf{Spec}_2} \vee \bigvee_{\mathcal{E}_2 \in Ev^{intl}_{\mathsf{Spec}_2}} \mathcal{E}_2.\varphi^{prom})$
5. $\varphi^{safe}_{\mathsf{Spec}_{1\|2}} := \varphi^{safe}_{\mathsf{Spec}_1} \wedge \varphi^{safe}_{\mathsf{Spec}_2}$

6. $\varphi^{entry}_{\mathsf{Spec}_{1\|2}} := \varphi^{entry}_{\mathsf{Spec}_1} \wedge \varphi^{entry}_{\mathsf{Spec}_2}$.

7. $Ev^{out}_{\mathsf{Spec}} := (Ev^{out}_{\mathsf{Spec}_1} \setminus \mathcal{R}_1^{-1}(Ev^{in}_2)) \cup (Ev^{out}_{\mathsf{Spec}_2} \setminus (\mathcal{R}_2^{-1}(Ev^{in}_{\mathsf{Spec}_1})),$
   $Ev^{in}_{\mathsf{Spec}} \;\; := (Ev^{in}_{\mathsf{Spec}_1} \setminus \mathcal{R}_2(Ev^{out}_{\mathsf{Spec}_2})) \cup (Ev^{in}_{\mathsf{Spec}_2} \setminus (\mathcal{R}_1(Ev^{out}_{\mathsf{Spec}_1}))$

8. $Stab_{\mathsf{Spec}_{1\|2}} := Stab_{\mathsf{Spec}_1} \cup Stab_{\mathsf{Spec}_2}$
   $\cup\{(\varphi_1^{stab} \wedge \varphi_2^{stab}, \max(\Delta_1^{stab}, \Delta_2^{stab})) \mid (\varphi_i^{stab}, \Delta_i^{stab}) \in Stab_{\mathsf{Spec}_i}, i \in$
   $\{1,2\}\}$

9. *Let the set of internally receivable events* $Ev^{intl}_{\mathsf{Spec}_1} \cup Ev^{intl}_{\mathsf{Spec}_2}$ *be denoted as* Ev^{intl}.
   $EvStab_{\mathsf{Spec}_{1\|2}} := \{(\varphi^{stab\ i}_{\mathsf{Spec}}, \Delta^{stab\ i}_{\mathsf{Spec}}, Ev_i^{stab} \setminus Ev^{intl}, \Delta^{re\ \ i}_{\mathsf{Spec}}) \mid (\varphi^{stab\ i}_{\mathsf{Spec}}, \Delta^{stab\ i}_{\mathsf{Spec}}, Ev_i^{stab}, \Delta^{re\ \ i}_{\mathsf{Spec}}) \in$
   $EvStab_{\mathsf{Spec}_i}\}$

Note 9 *The composite interface derives its property annotation from the properties annotated at the subcontrollers' interface. As these properties are approximations, the composite interface also only approximates the composite component's behavior. This approximation usually gets coarser, as for instance the guarantee is approximated by assuming that any external event can be received at any time. An engineer hence carefully has to determine which events should remain external. Usually, a composite component will satisfy other interfaces with stronger guarantees and/or weaker assumptions as well.*

It is left to show that a composite component actually satisfies the composite of subcomponent interfaces. We establish this result as Theorem 1. By this result, an engineer can select two components from a library, compose them and annotate the composite interface for the resulting composite component.

Theorem 1 Given components CC_i with implementations C_i, event connectors $\mathcal{R}_i$, interfaces Spec_i, $i \in \{1,2\}$.
Given a hybrid predicate φ^{assm}_{rem} on $(\,Var^{in}{}_{\mathsf{Spec}_{1\|2}} \cup \mathsf{Act} \setminus Var^{out}_{\mathsf{Spec}_{1\|2}}$ and given CC_1 and CC_2 are composable under φ^{assm}_{rem}.

The interface specification $\mathsf{Spec}_{1\|2} = (\mathsf{Spec}_1 \;{}_{\mathcal{R}_1}\!\overset{\|}{}_{,\mathcal{R}_2}\; \mathsf{Spec}_2, \varphi^{assm}_{rem})$ is satisfied by $\mathsf{C}_{1\|2}$, the $({}_{\mathcal{R}_1}\!\overset{\|}{}_{,\mathcal{R}_2}, \varphi^{assm}_{rem})$-composite of C_1 and C_2.

Proof of Theorem 1 In the following $\mathcal{C}$ denotes $(\mathsf{Com} \parallel \mathsf{C}_1 \parallel \mathsf{C}_2 \parallel \mathsf{P}) \setminus (\,Var^{in}_{\mathsf{Com}} \cup Var^{out}_{\mathsf{Com}})$ where Com is the hybrid automaton implementing the communication link between C_1 and C_2.
The composite implementation refers to the same plant as the composite interface (35.1), since each control component satisfies 35.1.
Input (output) variables of Spec equals the union of input (output) variables of Spec_1 and Spec_2. As the input (output) variables of C_i, equals the input (output) variables of Spec_i and as Com variables are hidden, it follows that input (output) variables of Spec equals the union of input (output) variables of C_1 and C_2 (35.2).
35.3 holds by assumption.
The entry condition of the composite interface is implied by the initial state of the composite implementation (35.4) holds, since both controller implementations

satisfy 35.4 and $\varphi^{entry}_{\mathsf{Spec}} = \varphi^{entry}_1 \wedge \varphi^{entry}_2$ and $\varphi^{\mathsf{init}}_{\mathsf{C}_{1\|2}} = \varphi^{\mathsf{init}}_{\mathsf{C}_1} \wedge \varphi^{\mathsf{init}}_{\mathsf{C}_2}$ and $\varphi^{entry}_{\mathsf{Spec}}$ does not refer to the initial state of the communication link.

35.5 follows directly from Def. 30.

That *the component satisfies its guarantee given its assumption hold (35.6)* follows, since the components are composable. This implies (cf. Lemma 8) that the implementations are composable. Hence we can apply Lemma 7 to derives that $\Box_{\leq\Delta^{time}_{\mathsf{lat}}}(\varphi^{guara}_{dyn\ \ \mathcal{C}_1} \wedge \varphi^{guara}_{dyn\ \ \mathcal{C}_2})$ UNLESS $\neg\varphi^{rem}_{dyn}$ holds.

The composite $\mathsf{C}_1 \parallel \mathsf{C}_2 \parallel \mathsf{P}$ is lockfree (35.7) by Lemma 6, which is applicable as both implementations are composable. Composition with the communication link automaton does not introduce locks, as the communication link automaton is lockfree by (18.2) and no locks can be introduced at the composites.

That the composite implementation is a consistent event sender (35.8), follows as both subcontrollers are consistent senders and the communication link does not generate send events by itself (18.1).

35.9 holds, because interface satisfaction of the two subcontrollers implies that initially no events are received at the subcontrollers and also Com cannot induce that an event is initially received. Hence also the controller composite does not initially receive events.

That the two controllers both still establish their safety obligation on the plant (35.10) follows by Lemma 4 and Lemma 1.

Similarly follows that the original convergence properties are preserved 35.11 and 35.12. That composite convergence properties with $\max(\Delta^{stab}_1, \Delta^{stab}_2)$ can be derived, is proven in [9]. ■

6.3 Summary

We have introduced a framework for loosely coupled controllers on a common plant, where controllers need to know very little about the other to accomplish their control task. For optimization controller can communicate via events. We defined interfaces for controllers to support reuse and management within a component library. The interface annotates services, safety and stability properties, and a deployment context of the component. A composite interface can be derived for the composite component from the subcomponent interfaces.

When an engineer enters a new atomic implementation to the library, he will have to dismiss the constraints for interface satisfaction of Def. 35. When an engineer builds a composite component, she will have to dismiss the constraints for composability of Def. 36. Our goal for the framework was to create constraints for the verification objectives that are as simple as possible and are well tool supported. To demonstrate usability, we applied this framework to a case study in [10, 8].

7 Annotating Stability Properties

In this section we will give an excerpt of the disseration [9] to illustrate how the stability annotations for our components relate to classical stability notions and can be certified by Lyapunov functions. Hence, we can employ techniques to determine

Lyapunov functions and use tools like STABHYLI [11] to determine Lyapunov functions for a given implementation. Further details and the proofs omitted here can be found in [9].

At the interface (cf. Def. 33) we annotate two kinds of stability properties:
(`Stab`) stability constraints $(\varphi^{stab}, \Delta^{stab})$ and
(`CStab`) event-conditioned stability constraints $(\varphi^{stab}, \Delta^{stab}, Ev^{stab}, \Delta^{re})$
These properties express that the component guarantees to evolve into a certain region, φ^{stab}, within a certain time Δ^{stab} and remain there. Whereas the stability constraint `Stab` requires only the base assumption to be true, for event-conditioned stability `CStab` the convergence relies on an additional precondition on the environment. Only if certain events promise that a "better" environment can be assumed, convergence is guaranteed. Also with `CStab` being at the convergence region is only established as long as the better environment can be assumed – minus the time Δ^{re}, which allows the component to get ready again for the default assumption.

7.1 Global Asymptotic Stability

Stability basically describes that a system, under a slight disturbance, still behaves approximately the same. Serval notions of stability exist. One of them is Lyapunov stability, which expresses that if a system trajectory starts close enough to $\mathbf{0}$, it remains close (that is within a certain neighborhood).

Global asymptotic stability is stronger type of stability. If a trajectory is Lyapunov stable and all trajectories that start out near $\mathbf{0}$ converge to $\mathbf{0}$. Global asymptotic stability (GAS) is defined for closed loops, that is, systems without inputs.

Definition 38 ***Closed Loop*** *We refer to the parallel composition $\mathcal{C} = C \parallel P$ of a controller C and a plant P as a* closed loop *if $\mathcal{C}$ has no input variables, i. e., $Var^I = \varnothing$.*

To cope with inputs, traditional approaches from control theory community use vanishing terms and relax GAS to bounded-input-bounded-output stability, where the output response of a system is bounded by the input signal. Then, if a system's input vanishes, then the system's output converges to $\mathbf{0}$.

Definition 39 ***Global Asymptotic Stability [12].*** *Let H be a closed loop HIOA, and let $Var^S \subseteq Var$ be the set of variables that are required to converge to the equilibrium point $\mathbf{0}$. A continuous-time dynamic system H is called* Lyapunov stable (LS) with respect to *Var^S if for all trajectories $\mathbf{X}(\cdot)$ of H,*

$$\forall \epsilon > 0 : \exists \delta > 0 : \forall t \geq 0 : ||\mathbf{X}(0)|| < \delta \Rightarrow ||\mathbf{X}|_{Var^S}(t)|| < \epsilon.$$

H is called globally attractive (GA) with respect to *Var^S if for all trajectories $\mathbf{X}(\cdot)$ of $\mathcal{H}$,*

$$\lim_{t \to \infty} \mathbf{X}|_{Var^S}(t) = \mathbf{0},\ i.\,e., \forall \epsilon > 0 : \exists t_0 \geq 0 : \forall t > t_0 : ||\mathbf{X}|_{Var^S}(t)|| < \epsilon,$$

where $\mathbf{0}$ is the origin of $\mathbb{R}^{|Var^S|}$. If a system is both globally stable with respect to Var^S and globally attractive with respect to Var^S, then it is called globally asymptotically stable (GAS) with respect to *Var^S.*

In the following we will extend GAS to our setting so that we can profit from the additional information that inputs are constrained by hybrid predicates.

7.2 Extension to Parallel Composition

We are going to show that if H_i is provably GAS wrt. Var_i^S via Lyapunov functions then $H_1 \parallel H_2$ is provably GAS wrt. $Var_1^S \cup Var_2^S$ via Lyapunov functions.

First we define modified versions of global asymptotic stability, namely,
(`GAS-Asm`) global asymptotic stability under assumptions (Def. 40) and
(`conGAS`) conditional global asymptotic stability (Def. 41).
We also give the accordingly modified versions of the Lyapunov Theorem, saying that Lyapunov functions can be used to prove a system as `GAS-Asm` (Thm. 2)and `conGAS` (Thm. 3), respectively. Further, a structural parallel composition preserves the `GAS-Asm` and `conGAS`.

In [9] it is also shown how to construct Lyapunov functions for $H_1 \parallel H_2$ from the Lyapunov functions for H_i. As Lyapunov functions certify the stability properties of components, this result allows us to propagate the stabiliy certificates of subcomponents to composed components. Additionally it allows us to perform further analysis of a composed system using the Lyapunov functions as abstract knowledge without explicitly constructing the system's trajectories.

7.2.1 Global Asymptotic Stability in an Environment

Definition 40 ***Global Asymptotic Stability under Assumptions.*** *Let H be an HIOA. Let φ^{env} be an hybrid predicate on $Var_H^{in} \cup Var_H^{loc}$, and let $Var^S \subseteq Var^C$ be a subset of variables required to converge to the equilibrium point $\mathbf{0} \in \mathbb{R}^{|Var^S|}$. A continuous-time dynamic system H is called* Lyapunov stable (LS) wrt. Var^S under assumptions φ^{env}, *if for all trajectories of $\mathbf{X}(\cdot)$ of H with $\forall t : \mathbf{X}(t) \models \varphi^{env}$, holds*

$$\forall \epsilon > 0 : \exists \delta > 0 : \forall t \geq 0 : ||\mathbf{X}(0)|| < \delta \Rightarrow \left|\left|\mathbf{X}^S(t)\right|\right| < \epsilon.$$

H is called globally attractive (GA) wrt. Var^S under assumptions φ^{env}, *if for all trajectories $\mathbf{X}(\cdot)$ of H with $\forall t : \mathbf{X}(t) \models \varphi^{env}$,*

$$\lim_{t \to \infty} \mathbf{X}^S(t) = \mathbf{0}, \quad i.e., \forall \epsilon > 0 : \exists t_e \geq 0 : \forall t > t_e : \left|\left|\mathbf{X}^S(t)\right|\right| < \epsilon.$$

$\mathbf{X}^S$ refers to the projection of $\mathbf{X}|_{Var^S}$. A system is globally asymptotically stable wrt. Var^S under assumptions φ^{env} (`GAS-Asm`) *iff it is, both, Lyapunov stable wrt. Var^S under assumption φ^{env} and globally attractive wrt. Var^S under assumption φ^{env}.*

`GAS-Asm` is a generalized version of `GAS` that allows an hybrid automaton to have inputs and outputs and requires the trajectories to converge in case the automaton's environment (the valuation of the input variables) behaves according to a certain assumption φ^{env}. The special case of `GAS` for hybrid automata without inputs and outputs can be obtained by having no inputs and setting φ^{env} to `true`.

Next we show how to prove a system as `GAS-Asm`. We use a slightly modified version of the Lyapunov theorem that refers to the assumption predicate.

Theorem 2 Discontinuous Lyapunov Functions under Assumptions. Let H be a HIOA, let φ^{env} be an hybrid predicate on $Var_H^{in} \cup Var_H^{loc}$, and let $Var^S \subseteq Var^C$

be a subset of variables required to converge to the equilibrium point $\mathbf{0} \in \mathbb{R}^{|Var^S|}$. Let $\mathbf{X}^S$ denote the projection of $\mathbf{X}|_{Var^S}$ and $\mathbf{X}^{S_m}$ the projection of $\mathbf{X}|_{Var^{S_m}}$.

If for each mode $m \in \mathbb{M}$, exists a set of variables Var^{S_m} with $Var^S \subseteq Var^{S_m} \subseteq Var^C$ and there exists a continuously differentiable function $V_m : \mathcal{X} \to \mathbb{R}$ such that

1. for each $m \in \mathbb{M}$, there exist two class-$\mathcal{K}^\infty$ functions α and β such that
$$\forall \mathbf{X} \models \Theta^{\mathsf{inv}}(m) : \mathbf{X} \models \varphi^{env}_{inv} \Rightarrow \alpha\big(||\mathbf{X}^{S_m}||\big) \leq V_m(\mathbf{X}) \leq \beta\big(||\mathbf{X}^{S_m}||\big),$$
2. for each $m \in \mathbb{M}$, there exists a class $\mathcal{K}^\infty$ function γ such that
$$\forall \mathbf{X} \models \Theta^{\mathsf{inv}}(m) : \mathbf{X} \models \varphi^{env}_{inv} \Rightarrow \dot{V}_m(\mathbf{X}) \leq -\gamma\big(||\mathbf{X}^{S_m}||\big)$$
for each
$$\dot{V}_m(\mathbf{X}) \in \left\{ \left\langle \begin{bmatrix} \frac{dV_m(\mathbf{X})}{d\mathbf{X}^C} \\ \frac{dV_m(\mathbf{X})}{d\mathbf{Y}} \end{bmatrix} \middle| \begin{bmatrix} f(\mathbf{X}) \\ \dot{\mathbf{Y}} \end{bmatrix} \right\rangle \middle| \begin{array}{c} f(\mathbf{X}) \in R^{\mathsf{cnt}}(m) \\ \wedge \dot{\mathbf{Y}} \models \varphi^{env}_{flow} \end{array} \right\},$$
3. for each transition $(m, G, \mathcal{A}, m') \in R^{\mathsf{dscr}}$,
$$\begin{aligned} \forall \mathbf{X} \models G, \mathbf{X}^+ : \quad & \Big(\mathbf{X} \models \varphi^{env}_{inv} \wedge \mathbf{X}, \mathbf{X}^+ \models \mathcal{A} \wedge \mathbf{X}^{I^+} = \mathbf{X}^I\Big) \\ & \Rightarrow V_{m'}\big(\mathbf{X}^+\big) \leq V_m(\mathbf{X}), \end{aligned}$$
4. for each $m \in \mathbb{M}$
$$\begin{aligned} \forall \mathbf{X} \models \Theta^{\mathsf{inv}}(m), \mathbf{X}^+ : \quad & \Big(\mathbf{X} \models \varphi^{env}_{inv} \wedge \mathbf{X}^{C^+} = \mathbf{X}^C \wedge \mathbf{X}^+ \models \varphi^{env}_{dscr}\Big) \\ & \Rightarrow V_m\big(\mathbf{X}^+\big) \leq V_m(\mathbf{X}), \end{aligned}$$

then H is global asymptotically stable with respect to Var^S under assumptions φ^{env} (`GAS-Asm`) and V_m is called a *local Lyapunov function (LLF)* for location m, and the function V given by $V(m, \mathbf{X}) := V_m(\mathbf{X})$ is called a *global Lyapunov function* (GLF) for H.

In the following we introduce a modified version of `GAS`, namely, conditional global asymptotic stability (`conGAS`). A HIOA H is φ^{con}-`conGAS` if it is `GAS` as long as an additional condition φ^{con} over Var^I holds. This notion allows to prove convergence to distinct equilibria. To which equilibrium the system converges depends on the environment condition φ^{con}. We formally define `conGAS` as follows:

Definition 41 ***Conditional Global Asymptotic Stability.*** *Let H be a HIOA, let φ^{con} be a hybrid predicate on $Var^{in} \cup Var^{loc}$, and let $Var^S \subseteq Var^C$ be a subset of variables required to converge to the equilibrium point $\mathbf{0} \in \mathbb{R}^{|Var^S|}$. We denote the projection of $\mathbf{X}|_{Var^S}$ as $\mathbf{X}^S$.*

*A continuous-time dynamic system H is called φ^{con}-*conditionally Lyapunov stable (LS) wrt. *Var^S if for all trajectories $\mathbf{X}(t)$ holds*

$$\begin{aligned} & \forall t_0 \geq 0, \forall \epsilon > 0, \forall t_f \geq t_0 : \exists \delta > 0 : \\ & \qquad (\forall t \in [t_0, t_f] : \mathbf{X}(t) \models \varphi^{con}) \Rightarrow \big(\forall t \in [t_0, t_f] : ||\mathbf{X}(t_0)|| < \delta \Rightarrow \big|\big|\mathbf{X}^S(t)\big|\big| < \epsilon\big). \end{aligned}$$

H is called φ^{con}-conditionally globally attractive (GA) wrt. *Var^S if for all trajectories $\mathbf{X}(t)$ holds*

$$\forall t_0 \geq 0, \forall \epsilon > 0, \exists t_\epsilon \geq t_0 : \forall t_f \geq t_\epsilon : \\ \left(\forall t \in [t_0, t_f] : \mathbf{X}(t) \models \varphi^{con}\right) \Rightarrow \left(\forall t \in [t_\epsilon, t_f] : \left|\left|\mathbf{X}^S(t)\right|\right| < \epsilon\right),$$

A system is φ^{con}-conditionally globally asymptotically stable (φ^{con}-conGAS) wrt. *Var^S iff it is, both, φ^{con}-conditionally Lyapunov stable wrt. Var^S and φ^{con}-conditionally globally attractive wrt. Var^S.*

Like for `GAS-Asm` we give an adapted version of the Lyapunov theorem for `conGAS` which exploits discontinuous conditional Lyapunov functions.

Theorem 3 Discontinuous Conditional Lyapunov Functions. Let H be an HIOA, let φ^{con} be a hybrid predicate on $Var^{in} \cup Var^{loc}$, and let $Var^S \subseteq Var^C$ be a subset of variables required to converge to the equilibrium point $\mathbf{0} \in \mathbb{R}^{|Var^S|}$. $\mathbf{X}^S$ denotes the projection of $\mathbf{X}|_{Var^S}$ and $\mathbf{X}^{S_m}$ denotes the projection of $\mathbf{X}|_{Var^{S_m}}$.
If for each mode $m \in \mathbb{M}$ there exists a set of variables Var^{S_m} with $Var^S \subseteq Var^{S_m} \subseteq Var^C$ and there exists a continuously differentiable function $V_m : \mathcal{X} \to \mathbb{R}$ such that

1. for each $m \in \mathbb{M}$, there exist two class-$\mathcal{K}^\infty$ functions α and β such that
$$\forall \mathbf{X} \models \Theta^{\mathsf{inv}}(m) : \mathbf{X} \models \varphi^{con}_{inv} \Rightarrow \alpha\left(\left|\left|\mathbf{X}^{S_m}\right|\right|\right) \leq V_m(\mathbf{X}) \leq \beta\left(\left|\left|\mathbf{X}^{S_m}\right|\right|\right),$$
2. for each $m \in \mathbb{M}$, there exists a class $\mathcal{K}^\infty$ function γ such that
$$\forall \mathbf{X} \models \Theta^{\mathsf{inv}}(m) : \mathbf{X} \models \varphi^{con}_{inv} \Rightarrow \dot{V}_m\left(\mathbf{X}^C\right) \leq -\gamma\left(\left|\left|\mathbf{X}^{S_m}\right|\right|\right)$$
for each $\dot{V}_m(\mathbf{X}) \in \left\{ \left\langle \begin{bmatrix} \frac{dV_m(\mathbf{X})}{d\mathbf{X}^C} \\ \frac{dV_m(\mathbf{X})}{d\mathbf{Y}} \end{bmatrix} \middle| \begin{bmatrix} f(\mathbf{X}) \\ \dot{\mathbf{Y}} \end{bmatrix} \right\rangle \middle| \begin{array}{c} f(\mathbf{X}) \in R^{\mathsf{cnt}}(m) \\ \wedge \dot{\mathbf{Y}} \models \varphi^{env}_{flow} \end{array} \right\},$
3. for each transition $(m, G, \mathcal{A}, m') \in R^{\mathsf{dscr}}$,
$$\forall \mathbf{X} \models G, \mathbf{X}^+ : \left(\mathbf{X} \models \varphi^{env}_{inv} \wedge \mathbf{X}, \mathbf{X}^+ \models \mathcal{A} \wedge \mathbf{X}^{I+} = \mathbf{X}^I\right) \\ \Rightarrow V_{m'}\left(\mathbf{X}^+\right) \leq V_m(\mathbf{X}),$$
4. for each $m \in \mathbb{M}$
$$\forall \mathbf{X} \models \Theta^{\mathsf{inv}}(m), \mathbf{X}^+ : \left(\mathbf{X} \models \varphi^{env}_{inv} \wedge \mathbf{X}^{C+} = \mathbf{X}^C \wedge \mathbf{X}^+ \models \varphi^{env}_{dscr}\right) \\ \Rightarrow V_m\left(\mathbf{X}^+\right) \leq V_m(\mathbf{X}),$$

then H is φ^{con}-conditionally globally asymptotically stable (`conGAS`) with respect to Var^S and V_m is called a *Local Lyapunov Function (LLF)* of m, and the function $V(m, \mathbf{X}) = V_m(\mathbf{X})$ is called a *global Lyapunov function* (GLF) for H.

Note 10 *The difference between discontinuous Lyapunov function and conditional discontinuous Lyapunov functions is that in the former the system has to converge all the time and in the latter it has to converge only while the φ^{con} evaluates to* `true`*. Furthermore, by replacing φ^{con} with* `true` *one obtains the original Lyapunov theorem.*

7.2.2 Global Asymptotic Stability under Parallel Composition

In the following we show that `GAS-Asm` or `conGAS` properties are preserved under composition. This essentially allows to prove the properties in isolation.

Note 11 (Controller) *In the following C refers to a controller implemenation $(\mathit{Art}, \mathsf{P}, \varphi^{env})$. We refer a controller implementation as* controller*, for short. We denote $\mathsf{C} \parallel \mathsf{P}$ as $\mathcal{C}$. We denote by $Var_{\mathcal{C}} := Var_{\mathsf{C}} \cup Var_{\mathsf{P}}$ the set of variables of the controller.*

Definition 42 ***Runs of a Controller*** *A – possibly infinite – sequence (π_i) is a called* run of $\mathcal{C}$ *iff (π_i) is a run of $\mathsf{C} \parallel \mathsf{P}$ under assumption φ^{env}.*

Definition 43 ***Global Asymptotic Stability of a Controller*** *A controller C is* globally asymptotically stable wrt. Var^S *iff $\mathsf{C} \parallel \mathsf{P}$ is GAS-Asm wrt. Var^S under φ^{env}.*

Theorem 4 Given two φ^{env}, $\parallel$-composable controllers $\mathsf{C}_1, \mathsf{C}_2$.

If C_i is GAS wrt. Var_i^S, then $\mathsf{C}_1 \parallel \mathsf{C}_2$ is GAS wrt. $Var_1^S \uplus Var_2^S$.

Definition 44 ***Conditional Global Asymptotic Stability of a Control Component*** *A controller C is* φ^{con}-conditional globally asymptotically stable (GAS) wrt. Var^S (φ^{con}-conGAS)
iff
- *φ^{con} is a hybrid predicate on $\mathsf{Sens} \uplus (Var_{\mathsf{P}}^I \setminus Var_{\mathsf{C}}^O)$ and*
- *$\mathsf{C} \parallel \mathsf{P}$ is φ^{con}-conditional globally asymptotically stable wrt. Var^S.*

Theorem 5 Given two φ^{env}, $\parallel$-composable controllers $\mathsf{C}_1, \mathsf{C}_2$ and a hybrid predicate φ^{con} on $\mathsf{Sens} \uplus (Var_{\mathsf{P}}^I \setminus Var_{\mathsf{C}}^O)$.
If one subcomponent C_i is φ^{con}-conGAS wrt. Var_i^S, then $\mathsf{C}_1 \parallel \mathsf{C}_2$ is φ^{con}-conGAS wrt. Var_i^S.

8 Conclusion

We presented the formal basis for a design framework for control components. The hybrid controllers operate on a common hybrid system plant. Each controller is in charge of its own set of actuators and its local control task. A controller known its plant but makes assumptions about the unknown environment, thereby specifying its deployment context. Our framework additionally introduces events as a communication mean between controllers. Events allow to communicate dynamic contract offers to other controllers.

The verification conditions that an engineer has to establish – to enter a controller with interface into the library and to establish whether two controller components are composable – are intendedly simple and sleek. We focused to give verification conditions, for which tool support exists or which can be established by sufficient conditions in many cases, as we believe. We regard providing a framework with simple verification conditions which at the same time allows to express relevant properties as the main contribution of this line work.

We demonstrated the usability of the framework on a case study in [10]. There we also profited from the tool support for Lyapunov theory, that provides certificates for the stability annotations at component interfaces.

References

[1] L. J. Alun Foster, Iris Hamelink, editor. *ARTEMIS Book of Successes.* ARTEMIS, 2013.

[2] C. Baier and J.-P. Katoen. *Principles of Model Checking (Representation and Mind Series).* The MIT Press, 2008.

[3] W. Damm, H. Dierks, J. Oehlerking, and A. Pnueli. Towards component based design of hybrid systems: Safety and stability. In Z. Manna and D. Peled, editors, *Essays in Memory of Amir Pnueli*, volume 6200 of *LNCS*. Springer, 2010.

[4] W. Damm, W. Hagemann, E. Möhlmann, and A. Rakow. Component based design of hybrid systems: A case study on concurrency and coupling. Reports of SFB/TR 14 AVACS 95, 2014.

[5] W. Damm, A. Mikschl, J. Oehlerking, E.-R. Olderog, J. Pang, A. Platzer, M. Segelken, and B. Wirtz. Automating Verification of Cooperation, Control, and Design in Traffic Applications. In *Formal Methods and Hybrid Real-Time Systems, Essays in Honor of Dines Bjørner and Chaochen Zhou on the Occasion of Their 70th Birthdays*, 2007.

[6] W. Damm, T. Peikenkamp, and B. Josko. Contract Based ISO CD 26262 Safety Analysis. In *SAE World Congress – Session on Safety-Critical Systems*, 2009.

[7] W. Damm, A. Votintseva, A. Metzner, B. Josko, and E. Peikenkamp, Thomas; Böde. Boosting re–use of embedded automotive applications through rich components. In *Proceedings of FIT 2005*, 08 2005.

[8] W. Hagemann, E. Möhlmann, and A. Rakow. Verifying a pi controller using soapbox and stabhyli: Experiences on establishing properties for a steering controller. April 2014.

[9] E. Möhlmann. *Automatic Stability Verification via Lyapunov functions – Representations, Transformations, and Practical Issues.* PhD thesis, Carl von Ossietzky University of Oldenburg, Department of Computer Science, Oldenburg, Germany, to appear.

[10] E. Möhlmann, A. Rakow, and W. Damm. Component based design of hybrid systems: A case study on concurrency and coupling. In *HSCC*, 2014.

[11] E. Möhlmann and O. E. Theel. Stabhyli: a Tool for Automatic Stability Verification of Non-Linear Hybrid Systems. In C. Belta and F. Ivancic, editors, *HSCC*. ACM, 2013.

[12] J. Oehlerking. *Decomposition of Stability Proofs for Hybrid Systems.* PhD thesis, Carl von Ossietzky University of Oldenburg, Department of Computer Science, Oldenburg, Germany, 2011.

[13] J. Ouaknine and J. Worrell. *Formal Modeling and Analysis of Timed Systems: 6th International Conference, FORMATS 2008, Saint Malo, France, September 15-17, 2008. Proceedings*, chapter Some Recent Results in Metric Temporal Logic, pages 1–13. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[14] A. L. Sangiovanni-Vincentelli, W. Damm, and R. Passerone. Taming Dr. Frankenstein: Contract-based design for cyber-physical systems. *Eur. J. Control*, 18(3), 2012.