Hierarchical Discrete Event Systems with Inputs and Outputs

Sebastian Perk, Thomas Moor, Klaus Schmidt

***Abstract*— We propose a framework for the hierarchical design of discrete event systems that addresses both safety and liveness properties. Technically, we build on a notion of inputs and outputs that is closely related to J.C. Willems' behavioural systems theory. We develop a structural admissibility condition that allows for abstraction-based controller synthesis similar to previous work on hybrid control systems. A key feature of our framework is an alternation of subsystem composition and controller synthesis that is expected to be computationally efficient whenever the complexity of the safety specifications is independent of the respective layer in the hierarchy.**

I. INTRODUCTION

It is common engineering practice to organise complex control systems in a hierarchical manner. For discrete event systems, complexity typically stems from an overall plant that is composed of a large number of subsystems with some nontrivial interaction between subsystems and/or specified cooperative behaviour. In such a situation monolithic controller synthesis procedures (e.g. [13]) that require an explicit representation of an overall plant model face a prohibitive computational cost since the overall number of states depends exponentially in the number of subsystems. Hierarchical approaches provide means to decompose the overall control problem into a number of less complex problems, typically employing a hierarchy of plant abstractions. One aim is to avoid explicit reference to the exact overall plant model in the controller synthesis procedure.

In this contribution we propose to alternate subsystem composition and controller synthesis in the design of hierarchical control systems; see Fig. 1. Our approach addresses applications in which the synthesis of each low-level controller requires a detailed plant model, but does so only for a small number of subsystems. Thus, on the lowest level of the hierarchy, we synthesise a large number of controllers for a large number of groups of subsystems with a small number of subsystems each. When proceeding to the next level of the hierarchy, we use the specifications of the preceeding level as an abstraction of the controlled groups of subsystems. We then synthesise controllers for groups of abstracted low-level control systems. The latter have been designed independently, so constraints in interconnection of groups of subsystems (e.g. shared resources) have not yet been considered. Our framework accounts for such constraints by a hierarchy of environment models that complements the hierarchy of controllers. The alternation of system composition, controller synthesis and environment interconnection is continued in a bottom-up fashion until a single top-level controller is synthesised to control an abstract overall model.

Lehrstuhl für Regelungstechnik, Universität Erlangen-Nürnberg, D-91058 Erlangen, Germany sebastian.perk@rt.eei.uni-erlangen.de

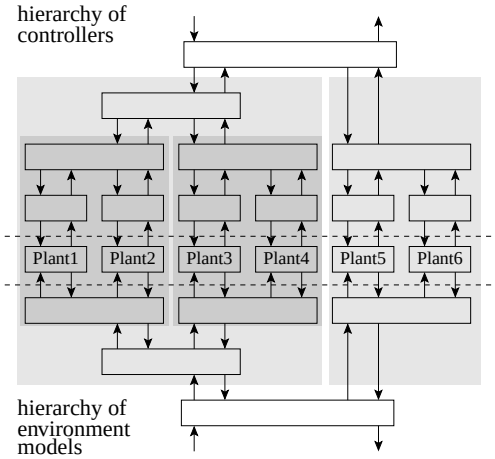

Fig. 1. Hierachical control system

Technically, the paper presents a system theoretic framework that allows for abstraction-based controller synthesis in the presence of hierarchical system interconnection. Abstraction-based synthesis has been studied extensively in the context hybrid control systems; e.g. [1], [5], [10]. In this paper, we build on the core ideas of [10] and a hierarchical extension [12], both stated within J.C. Willem's behavioural systems theory [16]. The results can, in principle, directly be applied to discrete event systems, with further extensions required for subsystem composition and a two-sided controller- and environment hierarchy. However, addressing discrete event systems, languages over unions of alphabets (e.g. control events U and measurement events Y amount to $\Sigma = U \dot{\cup} Y$) offer a more concise representation than behaviours (ω-languages) over Cartesian products of signal spaces (e.g. $\Sigma = U \times Y$). This shows in particular when one system (e.g. plant) is connected to two others (e.g. controller, environment) and signals run on different timescales. Therefore this paper presents results from [10], [12] in a format adapted for discrete event systems and extends those results to provide a framework for hierarchical control system design according to the above approach.

A number of hierarchical concepts have been discussed in the discrete event systems literature. In [2], [4], [17], [18], authors develop hierarchical system architectures that relate to our work in that each layer implements supervision and measurement aggregation and thus provides an abstract view on the layer below. There is also a strong conceptual link to [3], [7], [9], [14] where the vertical (de)-composition

introduced by a hierarchical architecture is complemented by a horizontal (de)-composition of modular or decentralised supervision. In all references given, the preservation of fundamental properties across levels of abstraction is a prime concern. In this contribution and in contrast to the references, relevant fundamental properties are derived from Willems' notion of free inputs and non-anticipating outputs.

The paper is organised as follows. Section II recalls basic notation and common operators on languages. Section III provides a language version of free inputs and non-anticipating outputs to define a class of I/O plants. We derive a control problem that allows for abstraction-based controller synthesis while preserving certain liveness and safety properties in Section IV. This setting already accounts for hierarchical controller design. Section V introduces an additional horizontal composition of subsystems to build the system architecture as proposed above.

II. PRELIMINARIES

Let Σ be a finite alphabet. The Kleene-closure Σ^* is the set of finite strings over Σ; i.e. $\Sigma^* = \{s | \exists n \in \mathbb{N}_0, \forall\, i < n : \sigma_i \in \Sigma, s = \sigma_0\sigma_1\cdots\sigma_n\}$ with the empty string $\epsilon \in \Sigma^*$. If for two strings $s, r \in \Sigma^*$ there exists $t \in \Sigma^*$, $t \neq \epsilon$, such that $s = rt$, we say r is a *strict prefix* of s and write $r < s$; r is a *prefix* of s if r is strict prefix of or equal to s and we write $r \leq s$. A prefix of s of length $n \in \mathbb{N}_0$ is denoted s^n. The *natural projection* $p_o\colon \Sigma^* \to \Sigma_o^*$, $\Sigma_o \subseteq \Sigma$, is defined iteratively: (1) let $p_o(\epsilon) := \epsilon$; (2) for $s \in \Sigma^*$, $\sigma \in \Sigma$, let $p_o(s\sigma) := p_o(s)\sigma$ if $\sigma \in \Sigma_o$, or $p_o(s\sigma) := p_o(s)$ otherwise. The set valued inverse of p_o is denoted $p_o^{-1}\colon \Sigma_o^* \to 2^{\Sigma^*}$.

A *language* over Σ is a subset $\mathcal{L} \subseteq \Sigma^*$. The *prefix-closure* of $\mathcal{L} \subseteq \Sigma^*$ is defined by $\overline{\mathcal{L}} = \{r | \exists s \in \mathcal{L} : r \leq s\} \subseteq \Sigma^*$. A language $\mathcal{L}$ is *prefix-closed* if $\mathcal{L} = \overline{\mathcal{L}}$. A language $\mathcal{L}$ is *complete* if for all $s \in \mathcal{L}$ there exists $\sigma \in \Sigma$ such that $s\sigma \in \overline{\mathcal{L}}$. Technically, $\mathcal{L} = \emptyset$ is complete. When extended to languages, the projection distributes over unions, and the inverse projection distributes over unions and intersection. Prefix-closure commutes with projection and inverse projection and distributes over unions. The *synchronous composition* of two languages $\mathcal{L}_i \subseteq \Sigma_i^*$, $i \in \{1, 2\}$, is defined $\mathcal{L}_1 \parallel \mathcal{L}_2 := p_1^{-1}(\mathcal{L}_1) \cap p_2^{-1}(\mathcal{L}_2)$, where the projections p_i are defined with domain $(\Sigma_1 \cup \Sigma_2)^*$ and range Σ_i^*.

The set of *ω-strings* over $A \subseteq \Sigma$ is denoted $A^\omega = \{s | \forall\, i \in \mathbb{N}_0\colon \sigma_i \in A, s = \sigma_0\sigma_1\sigma_2\cdots\}$. If for two strings $w \in \Sigma^\omega$, $r \in \Sigma^*$, there exists $v \in \Sigma^\omega$ such that $w = rv$, we say r is a *strict prefix* of w and write $r < w$. The strict prefix of w with length $n \in \mathbb{N}_0$ is denoted w^n. The *prefix* of an ω-language $\mathcal{L} \subseteq \Sigma^\omega$ is defined $\mathrm{pr}(\mathcal{L}) = \{r | \exists s \in \mathcal{L} : r < s\} \subseteq \Sigma^*$. For a language $\mathcal{L} \subseteq \Sigma^*$ the *limit* is defined $\mathcal{L}^\infty = \{w \in \Sigma^\omega | \exists (n_i)_{i\in\mathbb{N}_0}, n_{i+1} > n_i : w^{n_i} \in \mathcal{L}\}$. If and only if a language $\mathcal{L} \subseteq \Sigma^*$ is complete and prefix-closed, we have $\mathrm{pr}(\mathcal{L}^\infty) = \mathcal{L}$; see [6]. The natural projection for ω-strings carries over from finite strings. The range, however, is the union of finite and ω-strings. In contrast, the set valued inverse projection maps ω-strings to ω-languages. For prefix-closed languages $\mathcal{L}_1$ and $\mathcal{L}_2$ we have $\mathcal{L}_1{}^\infty \parallel \mathcal{L}_2{}^\infty \subseteq (\mathcal{L}_1 \parallel \mathcal{L}_2)^\infty$.

III. I/O PLANTS

We develop a class of discrete event systems that interact with an operator and an environment via input (control) and output (measurement) events; see Fig. 4. To begin with, a system consists of an alphabet (or signal space) and a language over that alphabet.

Definition III.1. A *system* is a tuple $\mathcal{S} = (\Sigma, \mathcal{L})$ with the alphabet Σ and the language $\mathcal{L} \subseteq \Sigma^*$. □

We say the system complete if $\mathcal{L}$ is complete, the system is regular if $\mathcal{L}$ is regular, the system is prefix-closed if $\mathcal{L}$ is prefix-closed etc. **Throughout this paper we consider prefix-closed systems only**.

The following notion of a plant-I/O port relates to Willems' I/O behaviours in that the input is free and the output does not anticipate the input. In contrast to e.g. [16], we do not require the output to process the input and thereby account for non-deterministic external behaviour. In contrast to the usual notion of controllable and uncontrollable events, we require alternation of measurement events $\nu \in Y$ and control events $\mu \in U$.

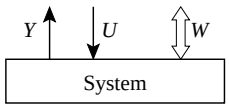

Fig. 2. Plant-I/O port

Definition III.2. A pair (U, Y) is a *plant-I/O port* of the system $(\Sigma, \mathcal{L})$ if
(i) $\Sigma = W \dot\cup U \dot\cup Y$, $U \neq \emptyset \neq Y$;
(ii) $\mathcal{L} \subseteq \overline{(W^*(YU)^*)^*}$; and
(iii) $(\forall s \in \Sigma^* Y,\ \mu \in U)\ [s \in \mathcal{L} \Rightarrow s\mu \in \mathcal{L}]$. □

When the system issues some measurement event $\nu \in Y$ on a plant-I/O port it will accept any control event $\mu \in U$ as an immediate successor. The following definition of a controller-I/O port is complementary in the sense that it requires the system to accept any measurement event $\nu \in Y$ and to reply by some control event $\mu \in U$, after an optional negotiation with some other system via the alphabet W.

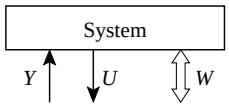

Fig. 3. Controller-I/O port

Definition III.3. A pair (U, Y) is a *controller-I/O port* of the system $(\Sigma, \mathcal{L})$ if
(i) $\Sigma = W \dot\cup U \dot\cup Y$, $U \neq \emptyset \neq Y$;
(ii) $\mathcal{L} \subseteq \overline{(YW^*U)^*}$; and
(iii) $(\forall s \in \Sigma^* U \cup \{\epsilon\},\ \nu \in Y)\ [s \in \mathcal{L} \Rightarrow s\nu \in \mathcal{L}]$. □

An I/O plant is defined as a system equipped with two distinguished plant-I/O ports. One port models the interaction of the plant with an operator (or controller), the other port

models the interaction of the plant with the environment. ¿From the perspective of the operator, the plant models the mechanism by which the environment can be manipulated.

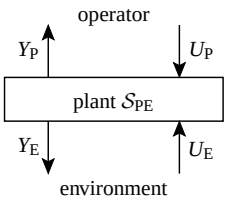

Fig. 4. I/O plant

Definition III.4. An *I/O plant* is a tuple $\mathcal{S}_{PE} = (U_P, Y_P, U_E, Y_E, \mathcal{L}_{PE})$, where

(i) $(\Sigma_{PE}, \mathcal{L}_{PE})$ is a system with $\Sigma_{PE} := \Sigma_P \dot\cup \Sigma_E$, $\Sigma_P := U_P \dot\cup Y_P$, $\Sigma_E := U_E \dot\cup Y_E$; and

(ii) (U_P, Y_P) and (U_E, Y_E) are plant-I/O ports of $(\Sigma_{PE}, \mathcal{L}_{PE})$. □

Remark. In this paper the relationship between systems, alphabets and languages is consequently indicated by matching subscripts; e.g. the system $\mathcal{S}_{ABC}$ always refers to the language $\mathcal{L}_{ABC}$ over the alphabet Σ_{ABC}. Furthermore, Σ_{ABC} denotes the disjoint union of Σ_A, Σ_B and Σ_C, and when inputs and outputs are relevant we use e.g. $\Sigma_A = U_A \dot\cup Y_A$. Similarly, the natural projection to Σ_{AB}^* is denoted p_{AB}; the natural projection to Y_A^* is denoted p_{YA}.

An I/O plant may be subject to constraints on the operator and/or the environment; e.g. the operator must comply to the operator's guidelines and the environment only provides a finite amount of resources. The separation of constraints from the plant model facilitates a discussion of the plant in a variety of different external configurations; e.g. we may change the actual environment by connecting another plant, or we may want to substitute the operator by a controller.

Definition III.5. A *constraint* is a tuple $(U, Y, \mathcal{L})$ if

(i) $(\Sigma, \mathcal{L})$ is a system with $\Sigma = U \dot\cup Y$;

(ii) (U, Y) is a controller-I/O port of $(\Sigma, \mathcal{L})$;

(iii) $\mathcal{L}$ is complete. □

We refer to the two extremal constraints $(U, Y, \mathcal{L})$ with $\mathcal{L} = \overline{(YU)^*}$ and $\mathcal{L} = \emptyset$ as the *minimal* and the *maximal constraint*, respectively. For faithful operation the plant must satisfy certain safety and liveness properties. Working with prefix-closed languages, only safety properties can be expressed as language inclusion. Regarding liveness we identify two properties adequate for our setting. The first one requires the plant to persistently issue events, and the second one requires that any infinite sequence of events must include an infinite number of measurement events reported to the operator.

Definition III.6. Let $\mathcal{S}_{PE} = (U_P, Y_P, U_E, Y_E, \mathcal{L}_{PE})$ be an I/O plant and let $\mathcal{S}_P = (U_P, Y_P, \mathcal{L}_P)$ and $\mathcal{S}_E = (U_E, Y_E, \mathcal{L}_E)$ be constraints. If $\mathcal{L}_P \parallel \mathcal{L}_{PE} \parallel \mathcal{L}_E$ is complete, then $\mathcal{S}_{PE}$ said to be *complete w.r.t. the constraints* $\mathcal{S}_P$ *and* $\mathcal{S}_E$. If

$$(\forall w \in (\mathcal{L}_P \parallel \mathcal{L}_{PE} \parallel \mathcal{L}_E)^\infty)[\, p_{YP}(w) \in Y_P^\omega \,], \qquad (1)$$

then the plant said to be Y_P*-live w.r.t. the constraints* $\mathcal{S}_P$ *and* $\mathcal{S}_E$. □

If $\mathcal{L}_P$, $\mathcal{L}_{PE}$, and $\mathcal{L}_E$ are all nonempty, then $\epsilon \in \mathcal{L}_P \parallel \mathcal{L}_{PE} \parallel \mathcal{L}_E$, and completeness implies that $(\mathcal{L}_P \parallel \mathcal{L}_{PE} \parallel \mathcal{L}_E)^\infty$ is nonempty. In this case Y_P-liveness indeed requires an infinite sequence of measurement events $\nu \in Y_P$ to be generated.

Suppose the plant model and the environment constraint are given. To establish an operator constraint that enforces the above liveness conditions amounts to solving a controller synthesis problem under partial observation [8]: control events U_P are regarded as controllable, all plant events Σ_P are observable, all other events are uncontrollable and unobservable; related synthesis problems that also address completeness are discussed in [10], [15], [6]. A least restrictive solution exists uniquely:

Proposition III.7. Let $\mathcal{S}_{PE} = (U_P, Y_P, U_E, Y_E, \mathcal{L}_{PE})$ be an I/O plant and $\mathcal{S}_E = (U_E, Y_E, \mathcal{L}_E)$ a constraint. Furthermore, let $\mathcal{S}_{P\alpha} = (U_P, Y_P, \mathcal{L}_{P\alpha})$ be a family of constraints and $\mathcal{L}_P := \cup_{\alpha \in A} \mathcal{L}_{P\alpha}$. Then $\mathcal{S}_P = (U_P, Y_P, \mathcal{L}_P)$ is a constraint. Moreover, if for all $\alpha \in A$ the plant $\mathcal{S}_{PE}$ is complete and Y_P-live w.r.t. $\mathcal{S}_E$ and $\mathcal{S}_{P\alpha}$, then $\mathcal{S}_{PE}$ is also complete and Y_P-live w.r.t. the constraints $\mathcal{S}_E$ and $\mathcal{S}_P$.

IV. I/O CONTROLLER SYNTHESIS

The task of the I/O controller is to assist the operator in manipulating the environment; see Fig. 5. Control events $\mu \in U_C$ issued by the operator trigger certain tasks to be performed by the controller and the plant. This eventually results in an abstract measurement event $\nu \in Y_C$ issued by the controller to indicate the status of the current task; e.g. successful completion or failure. The controller performs both control and measurement aggregation and thereby provides the operator with an abstract view on the plant.

Formally, we define the I/O controller as a system with a controller-I/O port to interact with the plant and a plant-I/O port to interact with the operator.

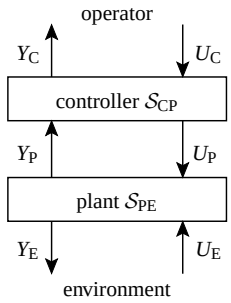

Fig. 5. Closed loop of plant and controller

Definition IV.1. An *I/O controller* is a tuple $\mathcal{S}_{CP} = (U_C, Y_C, U_P, Y_P, \mathcal{L}_{CP})$, where

(i) $(\Sigma_{CP}, \mathcal{L}_{CP})$ is a system with $\Sigma_{CP} = \Sigma_C \dot\cup \Sigma_P$, $\Sigma_C := U_C \dot\cup Y_C$, $\Sigma_P := U_P \dot\cup Y_P$;

(ii) (U_C, Y_C) and (U_P, Y_P) are a plant- and a controller-I/O port for $(\Sigma_{CP}, \mathcal{L}_{CP})$, respectively;

(iii) $\mathcal{L}_{\mathrm{CP}} \subseteq \overline{((Y_{\mathrm{P}}U_{\mathrm{P}})^*(Y_{\mathrm{P}}Y_{\mathrm{C}}U_{\mathrm{C}}U_{\mathrm{P}})^*)^*}$;
(iv) $\mathcal{L}_{\mathrm{CP}}$ is complete. □

When connecting a controller $\mathcal{S}_{\mathrm{CP}}$ and a plant $\mathcal{S}_{\mathrm{PE}}$ we obtain the system $(\Sigma_{\mathrm{CPE}}, \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}})$ to model the *full closed-loop behaviour*. Throughout this paper, we assume that the alphabets Σ_{C}, Σ_{P} and Σ_{E} are disjoint, hence, synchronisation of events happens only via the common alphabet Σ_{P}. Likewise, we obtain $(\Sigma_{\mathrm{CE}}, p_{\mathrm{CE}}(\mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}}))$ to model the *external closed-loop behaviour* which can be seen to be an I/O plant itself:

Proposition IV.2. Let $\mathcal{S}_{\mathrm{PE}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{PE}})$ be an I/O plant and let $\mathcal{S}_{\mathrm{CP}} = (U_{\mathrm{C}}, Y_{\mathrm{C}}, U_{\mathrm{P}}, Y_{\mathrm{P}}, \mathcal{L}_{\mathrm{CP}})$ be an I/O controller. Then the *external closed-loop system* $\mathcal{S}_{\mathrm{CE}} := \mathcal{S}_{\mathrm{CP}} \parallel_{\mathrm{ex}} \mathcal{S}_{\mathrm{PE}} := (U_{\mathrm{C}}, Y_{\mathrm{C}}, U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{CE}})$ with $\mathcal{L}_{\mathrm{CE}} = p_{\mathrm{CE}}(\mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}})$ is an I/O plant.

Note that the I/O structure itself is not sufficiently strong to imply completeness for the full or external closed loop. Hence the closed-loop system may run into a *deadlock* situation, which is considered undesirable. More subtle is the fact that arbitrary length strings $s \in (\Sigma_{\mathrm{P}} \cup \Sigma_{\mathrm{E}})^*$ may occur between each pair of control and measurement events $\mu \in U_{\mathrm{C}}$ and $\nu \in Y_{\mathrm{C}}$, which amounts to measurement aggregation. For the considered prefix-closed languages this implies that the closed-loop could also evolve on an infinite length string $s \in (\Sigma_{\mathrm{P}} \cup \Sigma_{\mathrm{E}})^{\omega}$. In the latter case the operator will not receive any further measurement events $\nu \in Y_{\mathrm{C}}$ and, hence, can not issue further control events. This *livelock* situation is also considered undesirable. The following admissibility condition addresses both issues in that it implies completeness and Y_{C}-liveness for the closed-loop system; see Proposition IV.4 and Theorem IV.5.

Definition IV.3. Let $\mathcal{S}_{\mathrm{PE}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{PE}})$ be an I/O plant and let $\mathcal{S}_{\mathrm{C}} = (U_{\mathrm{C}}, Y_{\mathrm{C}}, \mathcal{L}_{\mathrm{C}})$, $\mathcal{S}_{\mathrm{P}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, \mathcal{L}_{\mathrm{P}})$ and $\mathcal{S}_{\mathrm{E}} = (U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{E}})$ be constraints. Then, an I/O controller $\mathcal{S}_{\mathrm{CP}} = (U_{\mathrm{C}}, Y_{\mathrm{C}}, U_{\mathrm{P}}, Y_{\mathrm{P}}, \mathcal{L}_{\mathrm{CP}})$ is *admissible* to the plant $\mathcal{S}_{\mathrm{PE}}$ w.r.t. the constraints $\mathcal{S}_{\mathrm{C}}$, $\mathcal{S}_{\mathrm{P}}$, and $\mathcal{S}_{\mathrm{E}}$ if
(i) $p_{\mathrm{P}}(\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}) \subseteq \mathcal{L}_{\mathrm{P}}$;
(ii) $\mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}}$ is Y_{C}-live w.r.t. $\mathcal{S}_{\mathrm{C}}$ and $\mathcal{S}_{\mathrm{E}}$. □

The following proposition derives completeness of the full and the external closed-loop behaviour based on the above condition (i). As a technical consequence the set $(\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}})^{\infty}$ relevant to condition (ii) is non-empty.

Proposition IV.4. Let $\mathcal{S}_{\mathrm{CP}} = (U_{\mathrm{C}}, Y_{\mathrm{C}}, U_{\mathrm{P}}, Y_{\mathrm{P}}, \mathcal{L}_{\mathrm{CP}})$ be an I/O controller, let $\mathcal{S}_{\mathrm{PE}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{PE}})$ be an I/O plant, and let $\mathcal{S}_{\mathrm{C}} = (U_{\mathrm{C}}, Y_{\mathrm{C}}, \mathcal{L}_{\mathrm{C}})$, $\mathcal{S}_{\mathrm{P}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, \mathcal{L}_{\mathrm{P}})$ and $\mathcal{S}_{\mathrm{E}} = (U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{E}})$ be constraints. If $\mathcal{S}_{\mathrm{PE}}$ is complete w.r.t. $\mathcal{S}_{\mathrm{P}}$ and $\mathcal{S}_{\mathrm{E}}$, and $\mathcal{S}_{\mathrm{CP}}$ meets the admissibility condition (i), then $\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}$ is complete. If in addition $\mathcal{S}_{\mathrm{CP}}$ meets the admissibility condition (ii), then $\mathcal{L}_{\mathrm{C}} \parallel p_{\mathrm{CE}}(\mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}}) \parallel \mathcal{L}_{\mathrm{E}}$ is complete.

As indicated above, the admissibility condition implies that the external closed loop $\mathcal{S}_{\mathrm{CE}}$ is an I/O plant. Thus, in a hierarchical control architecture, the closed-loop can serve as a plant model for the design of the next layer of control and measurement aggregation.

Theorem IV.5. Let the I/O plant $\mathcal{S}_{\mathrm{PE}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{PE}})$ be complete and Y_{P}-live w.r.t. the constraints $\mathcal{S}_{\mathrm{P}}$ and $\mathcal{S}_{\mathrm{E}}$, and let $\mathcal{S}_{\mathrm{CP}} = (U_{\mathrm{C}}, Y_{\mathrm{C}}, U_{\mathrm{P}}, Y_{\mathrm{P}}, \mathcal{L}_{\mathrm{CP}})$ be admissible to $\mathcal{S}_{\mathrm{PE}}$ w.r.t. the constraints $\mathcal{S}_{\mathrm{C}}$, $\mathcal{S}_{\mathrm{P}}$, and $\mathcal{S}_{\mathrm{E}}$. Then the external closed-loop system $\mathcal{S}_{\mathrm{CE}} = (U_{\mathrm{C}}, Y_{\mathrm{C}}, U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{CE}})$, $\mathcal{L}_{\mathrm{CE}} = p_{\mathrm{CE}}(\mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}})$, is
(i) an I/O plant;
(ii) complete w.r.t. $\mathcal{S}_{\mathrm{C}}$ and $\mathcal{S}_{\mathrm{E}}$;
(iii) Y_{C}-live w.r.t. $\mathcal{S}_{\mathrm{C}}$ and $\mathcal{S}_{\mathrm{E}}$. □

We now are in the position to formally state the problem of I/O controller synthesis.

Definition IV.6. An *I/O controller synthesis problem* is a tuple $(\mathcal{S}_{\mathrm{PE}}, \mathcal{S}_{\mathrm{C}}, \mathcal{S}_{\mathrm{P}}, \mathcal{S}_{\mathrm{E}}, \mathcal{S}_{\mathrm{specCE}})$ where an $\mathcal{S}_{\mathrm{PE}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{PE}})$ is an I/O plant, $\mathcal{S}_{\mathrm{C}} = (U_{\mathrm{C}}, Y_{\mathrm{C}}, \mathcal{L}_{\mathrm{C}})$, $\mathcal{S}_{\mathrm{P}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, \mathcal{L}_{\mathrm{P}})$ and $\mathcal{S}_{\mathrm{E}} = (U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{E}})$ are constraints, and $\mathcal{S}_{\mathrm{specCE}} = (\Sigma_{\mathrm{CE}}, \mathcal{L}_{\mathrm{specCE}})$ is a system referred to as *safety specification*. A *solution to the I/O controller synthesis problem* is an I/O controller $\mathcal{S}_{\mathrm{CP}} = (U_{\mathrm{C}}, Y_{\mathrm{C}}, U_{\mathrm{P}}, Y_{\mathrm{P}}, \mathcal{L}_{\mathrm{CP}})$ that is admissible to $\mathcal{S}_{\mathrm{PE}}$ w.r.t. $\mathcal{S}_{\mathrm{C}}$, $\mathcal{S}_{\mathrm{P}}$, and $\mathcal{S}_{\mathrm{E}}$ and that enforces the safety specification $\mathcal{S}_{\mathrm{specCE}}$ on $\mathcal{S}_{\mathrm{PE}}$, i.e. $p_{\mathrm{CE}}(\mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}}) \subseteq \mathcal{L}_{\mathrm{specCE}}$. □

The above problem amounts to a controller synthesis problem under partial observation; we again refer to [10], [15], [6] where related problems are addressed. Note that the trivial controller (with empty language) solves the synthesis problem. Hence, the following theorem establishes unique existence of a least restrictive solution.

Theorem IV.7. Given an I/O controller synthesis problem $(\mathcal{S}_{\mathrm{PE}}, \mathcal{S}_{\mathrm{C}}, \mathcal{S}_{\mathrm{P}}, \mathcal{S}_{\mathrm{E}}, \mathcal{S}_{\mathrm{specCE}})$, let $\mathcal{S}_{\mathrm{CP}\alpha} = (U_{\mathrm{C}}, Y_{\mathrm{C}}, U_{\mathrm{P}}, Y_{\mathrm{P}}, \mathcal{L}_{\mathrm{CP}\alpha})$, $\alpha \in A$, denote a family of solutions. Then $\mathcal{S}_{\mathrm{CP}} = (U_{\mathrm{C}}, Y_{\mathrm{C}}, U_{\mathrm{P}}, Y_{\mathrm{P}}, \mathcal{L}_{\mathrm{CP}})$, $\mathcal{L}_{\mathrm{CP}} := \cup_{\alpha \in A} \mathcal{L}_{\mathrm{CP}\alpha}$, also solves the problem. □

Whilst considerably more general in scope, our framework makes similar use of the I/O structure as [10] and thereby allows for abstraction based controller synthesis; i.e. solutions obtained for a plant abstraction are guaranteed to solve the original problem. If the abstraction is of less complexity (number of states) the computational effort for controller synthesis is reduced accordingly.

Theorem IV.8. Given an I/O plant $\mathcal{S}_{\mathrm{PE}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{PE}})$, let $\tilde{\mathcal{S}}_{\mathrm{PE}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, U_{\mathrm{E}}, Y_{\mathrm{E}}, \tilde{\mathcal{L}}_{\mathrm{PE}})$ be a plant abstraction, i.e. $\mathcal{L}_{\mathrm{PE}} \subseteq \tilde{\mathcal{L}}_{\mathrm{PE}}$. If the plant $\mathcal{S}_{\mathrm{PE}}$ is complete and Y_{P}-live w.r.t. the constraints $\mathcal{S}_{\mathrm{P}}$ and $\mathcal{S}_{\mathrm{E}}$ and if $\mathcal{S}_{\mathrm{CP}}$ solves the I/O controller synthesis problem $(\tilde{\mathcal{S}}_{\mathrm{PE}}, \mathcal{S}_{\mathrm{C}}, \mathcal{S}_{\mathrm{P}}, \mathcal{S}_{\mathrm{E}}, \mathcal{S}_{\mathrm{specCE}})$, then $\mathcal{S}_{\mathrm{CP}}$ also solves $(\mathcal{S}_{\mathrm{PE}}, \mathcal{S}_{\mathrm{C}}, \mathcal{S}_{\mathrm{P}}, \mathcal{S}_{\mathrm{E}}, \mathcal{S}_{\mathrm{specCE}})$. □

On the downside of abstraction-based control, there is no guarantee that there exists a non trivial solution for the plant abstraction even if there does exist one for the original plant. Hence the question, how to obtain a "good" abstraction. In a hierarchical control architecture where the plant itself is a closed-loop system we propose the safety specification of the preceeding design step as a plant abstraction: we argue, that for many engineering applications the specification represents those aspects of the preceeding design step that are relevant for subsequent design. Consequently, we expect to obtain a non-trivial solution based on that abstraction. This line of thought has been further elaborated in the context of hybrid systems [11], [12].

V. COMPOUND I/O PLANTS

Suppose we are provided two plant components in a particular configuration that interact via shared resources. We suggest to: 1) model the individual plants independently (no shared events) with an environment constraint that always provides resources as requested; 2) formally obtain an overall model by a *shuffle product* composition; 3) model the interaction of the plant components by an *environment model* that shares environment events with both plant components and represents the limited amount of resources available. 4) Synthesise a controller that enforces a subset of the original environment constraint by only requesting resources when available. See Fig. 6 for the proposed system architecture.

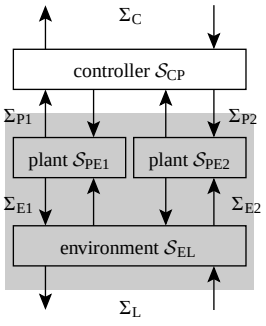

Fig. 6. Compound I/O plant with I/O controller

In our framework, step 1) leads to one I/O plant per component and corresponding constraints; i.e. for $i \in \{1,2\}$, $\mathcal{S}_{\mathrm{PE}i} = (U_{\mathrm{P}i}, Y_{\mathrm{P}i}, U_{\mathrm{E}i}, Y_{\mathrm{E}i}, \mathcal{L}_{\mathrm{PE}i})$, $\mathcal{S}_{\mathrm{P}i} = (U_{\mathrm{P}i}, Y_{\mathrm{P}i}, \mathcal{L}_{\mathrm{P}i})$ and $\mathcal{S}_{\mathrm{E}i} = (U_{\mathrm{E}i}, Y_{\mathrm{E}i}, \mathcal{L}_{\mathrm{E}i})$ where each the I/O plant $\mathcal{S}_{\mathrm{PE}i}$ is complete and $Y_{\mathrm{P}i}$-live w.r.t. the constraints $\mathcal{S}_{\mathrm{P}i}$ and $\mathcal{S}_{\mathrm{E}i}$. Recall that by Theorem IV.5 the external closed-loop system obtained by I/O controller synthesis exhibits the same properties. Thus, the following procedure applies uniformly to elementary plant models and closed-loop systems. Recall that at this stage both components are regarded as independent entities with no synchronisation built in; technically, all alphabets $\Sigma_{\mathrm{P}i} := U_{\mathrm{P}i} \dot\cup Y_{\mathrm{P}i}$, $\Sigma_{\mathrm{E}i} := U_{\mathrm{E}i} \dot\cup Y_{\mathrm{E}i}$, $i \in \{1,2\}$, are disjunct.

For step 2) we introduce the *I/O shuffle* operation. It is based on the usual shuffle product, but restricted by the additional condition $\mathcal{L}_{\mathrm{io}}$ on the ordering of input and output events and extended by a well-defined error behaviour $\mathcal{L}_{\mathrm{err}}$. The latter accounts for situations where a measurement event from the one plant component is replied to by a control event to the other plant component. A controller can be forced to avoid the error behaviour via a safety specification.

Definition V.1. Given two I/O plants $\mathcal{S}_{\mathrm{PE}i} = (U_{\mathrm{P}i}, Y_{\mathrm{P}i}, U_{\mathrm{E}i}, Y_{\mathrm{E}i}, \mathcal{L}_{\mathrm{PE}i})$, $i \in \{1,2\}$, the *I/O shuffle* $\mathcal{S}_{\mathrm{PE}} = \mathcal{S}_{\mathrm{PE1}} \parallel_{\mathrm{io}} \mathcal{S}_{\mathrm{PE2}}$ is defined as a tuple $\mathcal{S}_{\mathrm{PE}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{PE}})$, where:

(i) $U_{\mathrm{P}} := U_{\mathrm{P1}} \dot\cup U_{\mathrm{P2}}$, $Y_{\mathrm{P}} := Y_{\mathrm{P1}} \dot\cup Y_{\mathrm{P2}}$, $U_{\mathrm{E}} := U_{\mathrm{E1}} \dot\cup U_{\mathrm{E2}}$, $Y_{\mathrm{E}} := Y_{\mathrm{E1}} \dot\cup Y_{\mathrm{E2}}$;

(ii) $\mathcal{L}_{\parallel} := (\mathcal{L}_{\mathrm{PE1}} \parallel \mathcal{L}_{\mathrm{PE2}}) \cap \mathcal{L}_{\mathrm{io}}$ with $\mathcal{L}_{\mathrm{io}} := \overline{[(Y_{\mathrm{P1}}U_{\mathrm{P1}})^*(Y_{\mathrm{E1}}U_{\mathrm{E1}})^*(Y_{\mathrm{P2}}U_{\mathrm{P2}})^*(Y_{\mathrm{E2}}U_{\mathrm{E2}})^*]^*}$;

(iii) $\mathcal{L}_{\mathrm{err}} := \cup_{i=1}^{4} \overline{\mathcal{L}_{\mathrm{i}}}$ with
$\overline{\mathcal{L}_1} := \overline{(\mathcal{L}_{\parallel} Y_{\mathrm{P1}} \cap \mathcal{L}_{\parallel}) U_{\mathrm{P2}} ((Y_{\mathrm{P}}U_{\mathrm{P}})^*(Y_{\mathrm{E}}U_{\mathrm{E}})^*)^*}$,
$\overline{\mathcal{L}_2} := \overline{(\mathcal{L}_{\parallel} Y_{\mathrm{P2}} \cap \mathcal{L}_{\parallel}) U_{\mathrm{P1}} ((Y_{\mathrm{P}}U_{\mathrm{P}})^*(Y_{\mathrm{E}}U_{\mathrm{E}})^*)^*}$,
$\overline{\mathcal{L}_3} := \overline{(\mathcal{L}_{\parallel} Y_{\mathrm{E1}} \cap \mathcal{L}_{\parallel}) U_{\mathrm{E2}} ((Y_{\mathrm{P}}U_{\mathrm{P}})^*(Y_{\mathrm{E}}U_{\mathrm{E}})^*)^*}$,
$\overline{\mathcal{L}_4} := \overline{(\mathcal{L}_{\parallel} Y_{\mathrm{E2}} \cap \mathcal{L}_{\parallel}) U_{\mathrm{E1}} ((Y_{\mathrm{P}}U_{\mathrm{P}})^*(Y_{\mathrm{E}}U_{\mathrm{E}})^*)^*}$;

(iv) $\mathcal{L}_{\mathrm{PE}} := \mathcal{L}_{\mathrm{PE1}} \parallel_{\mathrm{io}} \mathcal{L}_{\mathrm{PE2}} := \mathcal{L}_{\parallel} \cup \mathcal{L}_{\mathrm{err}}$. □

It is readily shown that the I/O shuffle indeed is a shuffle composition in the sense that the behaviour of neither plant is restricted: $\mathcal{L}_{\mathrm{PE}i} \subseteq \mathcal{L}_{\mathrm{PE}}$. Moreover, the I/O shuffle retains the I/O structure of its arguments:

Proposition V.2. If $\mathcal{S}_{\mathrm{PE}i}$, $i \in \{1,2\}$ are I/O plants, so is $\mathcal{S}_{\mathrm{PE}} = \mathcal{S}_{\mathrm{PE1}} \parallel_{\mathrm{io}} \mathcal{S}_{\mathrm{PE2}}$. □

By the following proposition, constraints of the individual plant can be lifted to the compound plant by the (standard) shuffle product.

Proposition V.3. For $i \in \{1,2\}$, let $\mathcal{S}_{\mathrm{PE}i} = (U_{\mathrm{P}i}, Y_{\mathrm{P}i}, U_{\mathrm{E}i}, Y_{\mathrm{E}i}, \mathcal{L}_{\mathrm{PE}i})$ be an I/O plant, that is complete and $Y_{\mathrm{P}i}$-live w.r.t. the constraints $\mathcal{S}_{\mathrm{E}i} = (U_{\mathrm{E}i}, Y_{\mathrm{E}i}, \mathcal{L}_{\mathrm{E}i})$ and $\mathcal{S}_{\mathrm{P}i} = (U_{\mathrm{P}i}, Y_{\mathrm{P}i}, \mathcal{L}_{\mathrm{P}i})$. Denote the compound constraints $\mathcal{S}_{\mathrm{P}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, \mathcal{L}_{\mathrm{P}})$, $\mathcal{S}_{\mathrm{E}} = (U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{E}})$ with $\mathcal{L}_{\mathrm{P}} := (\mathcal{L}_{\mathrm{P1}} \parallel \mathcal{L}_{\mathrm{P2}}) \cap \mathcal{L}_{\mathrm{io}}$ and $\mathcal{L}_{\mathrm{E}} := (\mathcal{L}_{\mathrm{E1}} \parallel \mathcal{L}_{\mathrm{E2}}) \cap \mathcal{L}_{\mathrm{io}}$. Then $\mathcal{S}_{\mathrm{PE}} = \mathcal{S}_{\mathrm{PE1}} \parallel_{\mathrm{io}} \mathcal{S}_{\mathrm{PE2}}$ is complete and Y_{P}-live w.r.t. $\mathcal{S}_{\mathrm{P}}$ and $\mathcal{S}_{\mathrm{E}}$. □

We proceed with step 3) in modelling the interaction via sharing a common environment. Technically, we define the *environment model* to be of the same I/O structure as a controller. The environment model must not be confused with an environment constraint.

Definition V.4. An *I/O environment* is a tuple $\mathcal{S}_{\mathrm{EL}} = (U_{\mathrm{E}}, Y_{\mathrm{E}}, U_{\mathrm{L}}, Y_{\mathrm{L}}, \mathcal{L}_{\mathrm{EL}})$, where:

(i) $(\Sigma_{\mathrm{EL}}, \mathcal{L}_{\mathrm{EL}})$ is a system with $\Sigma_{\mathrm{EL}} := U_{\mathrm{E}} \dot\cup Y_{\mathrm{E}} \dot\cup U_{\mathrm{L}} \dot\cup Y_{\mathrm{L}}$;

(ii) $(U_{\mathrm{E}}, Y_{\mathrm{E}})$ and $(U_{\mathrm{L}}, Y_{\mathrm{L}})$ are a controller- and a plant-I/O port, respectively;

(iii) $\mathcal{L}_{\mathrm{EL}} \subseteq \overline{((Y_{\mathrm{E}}U_{\mathrm{E}})^*(Y_{\mathrm{E}}Y_{\mathrm{L}}U_{\mathrm{L}}U_{\mathrm{E}})^*)^*}$;

(iv) $\mathcal{L}_{\mathrm{EL}}$ is complete. □

Comparing the I/O structure of controller and environment, Proposition IV.2 carries over to the compound of plant and environment by uniform substitution.

Proposition V.5. Let $\mathcal{S}_{\mathrm{PE}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{PE}})$ be an I/O

plant and let $\mathcal{S}_{\mathrm{EL}} = (U_{\mathrm{E}}, Y_{\mathrm{E}}, U_{\mathrm{L}}, Y_{\mathrm{L}}, \mathcal{L}_{\mathrm{EL}})$ be an I/O environment. Then the external behaviour $\mathcal{S}_{\mathrm{PL}} = \mathcal{S}_{\mathrm{PE}} \parallel_{\mathrm{ex}} \mathcal{S}_{\mathrm{EL}}$ is an I/O plant. □

Also by uniform substitution we derive the following version of the first part of Proposition IV.4.

Proposition V.6. Let $\mathcal{S}_{\mathrm{PE}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{PE}})$ be an I/O plant, let $\mathcal{S}_{\mathrm{EL}} = (U_{\mathrm{E}}, Y_{\mathrm{E}}, U_{\mathrm{L}}, Y_{\mathrm{L}}, \mathcal{L}_{\mathrm{EL}})$ be an I/O environment, and let $\mathcal{S}_{\mathrm{P}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, \mathcal{L}_{\mathrm{P}})$, $\mathcal{S}_{\mathrm{E}} = (U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{E}})$ be constraints. If $\mathcal{S}_{\mathrm{PE}}$ is complete w.r.t. $\mathcal{S}_{\mathrm{P}}$ and $\mathcal{S}_{\mathrm{E}}$ and if $p_{\mathrm{E}}(\mathcal{L}_{\mathrm{EL}} \parallel \mathcal{L}_{\mathrm{L}}) \subseteq \mathcal{L}_{\mathrm{E}}$, then $\mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{EL}} \parallel \mathcal{L}_{\mathrm{L}}$ is complete. □

Step 4) requires the original environment constraint $\mathcal{S}_{\mathrm{E}}$ to be expressed by constraints $\mathcal{S}_{\mathrm{P}}$ and $\mathcal{S}_{\mathrm{L}}$ in order to guarantee liveness. The following theorem characterises suitable constraints. Typically, $\mathcal{S}_{\mathrm{L}}$ is given from an application context, and the below condition is solved for $\mathcal{S}_{\mathrm{P}}$.

Theorem V.7. For $i \in \{1,2\}$, let $\mathcal{S}_{\mathrm{PE}i} = (U_{\mathrm{P}i}, Y_{\mathrm{P}i}, U_{\mathrm{E}i}, Y_{\mathrm{E}i}, \mathcal{L}_{\mathrm{PE}i})$ be an I/O plant, that is complete and $Y_{\mathrm{P}i}$-live w.r.t. the constraints $\mathcal{S}_{\mathrm{E}i} = (U_{\mathrm{E}i}, Y_{\mathrm{E}i}, \mathcal{L}_{\mathrm{E}i})$ and $\mathcal{S}_{\mathrm{P}i} = (U_{\mathrm{P}i}, Y_{\mathrm{P}i}, \mathcal{L}_{\mathrm{P}i})$. Let $\mathcal{S}_{\mathrm{EL}} = (U_{\mathrm{E}}, Y_{\mathrm{E}}, U_{\mathrm{L}}, Y_{\mathrm{L}}, \mathcal{L}_{\mathrm{EL}})$ be an I/O environment and consider the compound system $\mathcal{S}_{\mathrm{PL}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, U_{\mathrm{L}}, Y_{\mathrm{L}}, \mathcal{L}_{\mathrm{PL}})$, $\mathcal{L}_{\mathrm{PL}} = p_{\mathrm{PL}}((\mathcal{L}_{\mathrm{PE1}} \parallel_{\mathrm{io}} \mathcal{L}_{\mathrm{PE2}}) \parallel \mathcal{L}_{\mathrm{EL}})$. Let $\mathcal{S}_{\mathrm{P}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, \mathcal{L}_{\mathrm{P}})$ and $\mathcal{S}_{\mathrm{L}} = (U_{\mathrm{L}}, Y_{\mathrm{L}}, \mathcal{L}_{\mathrm{L}})$ be constraints with

$$p_{\mathrm{E}}(\mathcal{L}_{\mathrm{P}} \parallel (\mathcal{L}_{\mathrm{PE1}} \parallel_{\mathrm{io}} \mathcal{L}_{\mathrm{PE2}}) \parallel \mathcal{L}_{\mathrm{EL}} \parallel \mathcal{L}_{\mathrm{L}}) \subseteq (\mathcal{L}_{\mathrm{E1}} \parallel \mathcal{L}_{\mathrm{E2}}) \cap \mathcal{L}_{\mathrm{io}},$$
$$p_{\mathrm{P}}(\mathcal{L}_{\mathrm{P}} \parallel (\mathcal{L}_{\mathrm{PE1}} \parallel_{\mathrm{io}} \mathcal{L}_{\mathrm{PE2}}) \parallel \mathcal{L}_{\mathrm{EL}} \parallel \mathcal{L}_{\mathrm{L}}) \subseteq (\mathcal{L}_{\mathrm{P1}} \parallel \mathcal{L}_{\mathrm{P2}}) \cap \mathcal{L}_{\mathrm{io}}.$$

Then $\mathcal{S}_{\mathrm{PL}}$ is

(i) an I/O plant;
(ii) complete w.r.t. $\mathcal{S}_{\mathrm{P}}$ and $\mathcal{S}_{\mathrm{L}}$,
(iii) Y_{P}-live w.r.t. $\mathcal{S}_{\mathrm{P}}$ and $\mathcal{S}_{\mathrm{L}}$. □

Thus, we end up with an I/O plant as discussed in Section IV and, hence, can approach the control problem accordingly. In particular, we can substitute the actual plant models $\mathcal{S}_{\mathrm{PE}i}$ by an abstraction: due to monotonicity of the applied language operations, this leads to an abstraction of the compound plant and to a conservative constraint $\mathcal{S}_{\mathrm{P}}$.

VI. CONCLUSIONS

In this contribution, we provide a system theoretic framework to discuss an approach to hierarchical control system design in which subsystem composition, controller synthesis and plant abstraction alternate. Our framework is equipped with a formal language version of Willems' free inputs and non-anticipating outputs. This is the key ingredient that allows for abstraction-based controller synthesis under preservation of safety- and liveness-properties.

To illustrate our expectations regarding computational complexity, consider the system design depicted in Fig. 1. Assume that we start with $n = k^m$ I/O plant components and that we can form groups of k plant components or closed-loop subsystems on each layer. Thus, we end up with m layers. The complexity of the design of an I/O controller for one group is expected to be polynomial in the number of states and, hence, exponential in k. When proceeding through the layers of the hierarchy, we use the safety specifications of the preceeding layer as an abstraction of each controlled group of subsystems. Thus, the exponential growth of the state space observed during the preceeding level is of no more relevance. On our way to the top level we need to solve $(1-n)/(1-k)$ I/O controller synthesis problems. Let M denote an upper bound on the size of the state space of the safety specifications, the lowest-level I/O plant components and environment models. Then the computational complexity of the synthesis of one controller is of order M^{ak} for some constant a. The overall complexity of the hierarchical design is $M^{ak}(1-n)/(1-k)$ and hence *exponential* is the number k of components that form a group — but only *linear* in the overall number of I/O plant components.

REFERENCES

[1] J.E.R. Cury, B.A. Krogh, and T. Niinomi. Synthesis of supervisory controllers for hybrid systems based on approximating automata. *IEEE Transactions on Automatic Control, Special issue on hybrid systems*, 43:564–568, 1998.

[2] A.E.C. da Cunha, J.E.R. Cury, and B.H. Krogh. An assume guarantee reasoning for hierarchical coordination of discrete event systems. *Workshop on Discrete Event Systems*, 2002.

[3] B. Gaudin and H. Marchand. Efficient computation of supervisors for loosely synchronous discrete event systems: A state-based approach. *IFAC World Congress*, 2005.

[4] P. Hubbard and P.E. Caines. Dynamical consistency in hierarchical supervisory control. *IEEE Transactions on Automatic Control*, 47(1):37–52, 2002.

[5] X. Koutsoukos, P.J. Antsaklis, J.A. Stiver, and M.D. Lemmon. Supervisory control of hybrid systems. *Proceedings of the IEEE*, 88:1026–1049, July 2000.

[6] R. Kumar, V. Garg, and S.I. Marcus. On supervisory control of sequential behaviors. *IEEE Transactions on Automatic Control*, 37:1978–1985, 1992.

[7] R.J. Leduc. Hierarchical interface based supervisory control. *PhD thesis, Department of Electrical and Computer Engineering, University of Toronto*, 2002.

[8] F. Lin and W.M. Wonham. On observability of discrete-event systems. *IEEE Transactions on Automatic Control*, 44:173–198, 1988.

[9] C. Ma. Nonblocking supervisory control of state tree structures. *Ph.D. Dissertation, Department of Electrical and Computer Engineering, University of Toronto*, 2004.

[10] T. Moor and J. Raisch. Supervisory control of hybrid systems within a behavioural framework. *Systems and Control Letters*, 38:157–166, 1999.

[11] T. Moor and J. Raisch. Hierarchical hybrid control of a multiproduct batch blant. In *Proc. 16th IFAC World Congress*, Prague, 2005.

[12] T. Moor, J. Raisch, and J.M. Davoren. Admissibility criteria for a hierarchical design of hybrid control systems. In *Proc. IFAC Conference on the Analysis and Design of Hybrid Systems (ADHS'03)*, pages 389–394, 2003.

[13] P.J. Ramadge and W.M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77:81–98, 1989.

[14] K. Schmidt. Hierarchical control of decentralized discrete event systems: Theory and application. *PhD-thesis, Lehrstuhl für Regelungstechnik, Universität Erlangen-Nürnberg*, 2005.

[15] J.G. Thistle and W.M. Wonham. Supevision of infinite behavior of finite automata. *SIAM J. Control and Optimization*, 32:1098–113, 1994.

[16] J.C. Willems. Paradigms and puzzles in the theory of dynamic systems. *IEEE Transactions on Automatic Control*, 36:258–294, 1991.

[17] K.C. Wong and W.M. Wonham. Hierarchical control of discrete-event systems. *Discrete Event Dynamic Systems: Theory and Applications*, 1996.

[18] H. Zhong and W.M. Wonham. On the consistency of hierarchical supervision in discrete-event systems. *IEEE Transactions on Automatic Control*, 35:1125–1134, October 1990.