Controller Synthesis for an I/O-Based Hierarchical System Architecture

Sebastian Perk, Thomas Moor, Klaus Schmidt

Abstract— **In our previous work, a framework for the hierarchical design of discrete event systems has been introduced that is based on a notion of inputs and outputs. I/O-plant models describe the interaction of each subsystem with the operator (or controller) and the environment. By alternation of subsystem composition and controller synthesis, a hierarchy of controllers is obtained that complements a hierarchy of environment models. An admissibility condition was presented that implies liveness while allowing for abstraction-based control. In this paper, we address the according controller synthesis problem and present an algorithmic synthesis procedure that respects admissibility and yields a solution to this problem. We illustrate our statements by the conceptional application example of a transport unit.**

I. Introduction

The supervisory control theory (SCT, e.g. [11]) provides a well-established method for model based design of controllers for discrete event systems (DES). However, direct usage of the SCT for large scale systems fails due to the computational cost, which stems from the need for an explicit representation of an overall plant model whose complexity grows exponentially with the number of subsystems.

Our concept avoids explicit reference to the overall plant model in the controller synthesis procedure by employing a hierarchy of plant abstractions; see [10]. Using an input/output-based description of DES, all subsystems are modelled independently from their environment aiming at reusability within various configurations. For each subsystem model, local controllers are designed to enforce local specifications that model the desired external behaviour of the closed loop. In the design step, additional assumptions on the external configuration can be taken into account by well-defined constraints. Their enforcement is passed on to the next level of superposed control.

For the next layer of the hierarchy, the interaction within the groups of locally controlled subsystems is described by dynamic environment models. Favourably, the I/O-based system structure allows for abstraction-based control, and the specifications of each subsystem can serve as an abstraction of the latter: rather than the controlled subsystems, groups of abstractions are composed with the environment model, which efficiently reduces the complexity of the result. As a benefit of our framework, the controllability and liveness of each hierarchical layer directly result from the I/O-based system structure. Superposed controllers designed for each group based on the abstractions solve the control problem provably also for the original groups of subsystems. The alternation of subsystem composition, controller synthesis and environment interconnection leads to a hierarchy of controllers that complements a hierarchy of environment models.

During the past decades a number of hierarchical concepts has been discussed in the discrete event systems literature. We refer to concepts like [2], [3], [14], where authors design hierarchical system architectures in which 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 [7], [12] where the vertical (de)-composition introduced by a hierarchical architecture is complemented by a horizontal (de)-composition of modular or decentralised supervision. In references such as [4], [6], [8], discrete event models are provided with different notions of inputs and outputs, each adequate to the considered problem. In all references concerning supervisory control, the preservation of fundamental properties in the closed loop is a primary concern.

In our approach and in contrast to the references, relevant fundamental properties like the novel notion of Y_{P}-liveness are derived from Willems' concept of free inputs and non-anticipating outputs according to the behavioural systems theory [13]. In our previous work [10], an admissibility condition has been identified that yields controllability and liveness for each hierarchical layer if met by all controllers in the hierarchy. However, the algorithmic construction of a controller featuring this admissibility condition remained as an open issue.

This article addresses this issue and the according controller synthesis problem as stated in Definition IV.6 of [10]. We present a controller synthesis algorithm that provides a solution to this synthesis problem. Our approach is illustrated by the conceptional example of a chain of transport units, see Fig. 1. The paper is organised as follows. After recalling basic notation in Section II, Section III gives a review of our framework. In Section IV we present a controller synthesis procedure that solves the synthesis problem stated in [10]. Section V illustrates the design of a hierarchical control system for the transport unit example and the resulting complexity. The results are summarised in Section VI.

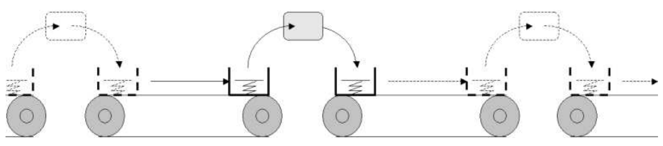


Fig. 1. Chain of transport units

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

II. Preliminaries

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

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 $s\sigma \in \overline{\mathcal{L}}$. The *synchronous composition* of two languages $\mathcal{L}_i \subseteq \Sigma_i^*$, $i \in \{1, 2\}$, is defined as $\mathcal{L}_1 \parallel \mathcal{L}_2 := p_1^{-1}(\mathcal{L}_1) \cap p_2^{-1}(\mathcal{L}_2)$, with the projections $p_i : (\Sigma_1 \cup \Sigma_2)^* \to \Sigma_i^*$.

The set of infinite length *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. For a language $\mathcal{L} \subseteq \Sigma^*$ the *limit* is defined as $\mathcal{L}^\infty = \{w \in \Sigma^\omega | \exists\, (n_i)_{i \in \mathbb{N}_0}, n_{i+1} > n_i : w^{n_i} \in \mathcal{L}\}$. The natural projection for ω-strings carries over from finite strings. The range, however, is the union of finite and ω-strings.

III. DES with Inputs and Outputs

The first step in model based control system design is to derive a model of the uncontrolled behaviour of the plant. In our framework, a *system* consists of an alphabet that carries the totality of all possible events and a language over that alphabet describing all possible event sequences.

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

We say the system is complete if $\mathcal{L}$ is complete, the system is regular if $\mathcal{L}$ is regular etc. As our notion of liveness is not expressed by marked strings, we consider prefix-closed languages only. In our formalism, the plant is represented by a particular system called I/O plant.

A. I/O Plant

I/O plants are a class of discrete event systems that interact with an operator and an environment via input and output events; see Fig. 2. 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.

Definition III.2. The 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) $(\forall s \in \Sigma^* Y,\ \mu \in U)\ [\, s \in \mathcal{L} \Rightarrow s\mu \in \mathcal{L}\,]$. $\square$

By item (i), we separate input events $\mu \in U$ from output events $\nu \in Y$. Remaining events, that e.g. are assigned to another I/O port of the system, are collected in W. By item (ii), we require alternation of output and input events, such that a dependence between cause and effect is established. 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 (item (iii)) respecting that the input can be imposed freely by the systems surroundings.

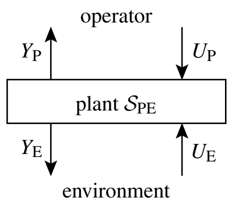


Fig. 2. I/O plant

The following definition of a controller-I/O port is a complement of the plant-I/O port 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. A controller-I/O port can be connected with a plant-I/O port.

Definition III.3. The 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) $(\forall s \in \Sigma^* U \cup \{\epsilon\},\ \nu \in Y)\ [\, s \in \mathcal{L} \Rightarrow s\nu \in \mathcal{L}\,]$. $\square$

From the perspective of the operator, the plant models the mechanism by which the environment can be manipulated. Hence, the I/O plant is defined as a system equipped with two distinguished plant-I/O ports; see Fig. 2.[1] One port models the interaction of the plant with an operator (or controller) via events Σ_{P}, the other port models the interaction of the plant with the environment via the events Σ_{E} that are not directly observable to the operator (or controller).

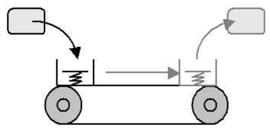
Definition III.4. An *I/O plant* is a tuple $\mathcal{S}_{\mathrm{PE}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{PE}})$, where

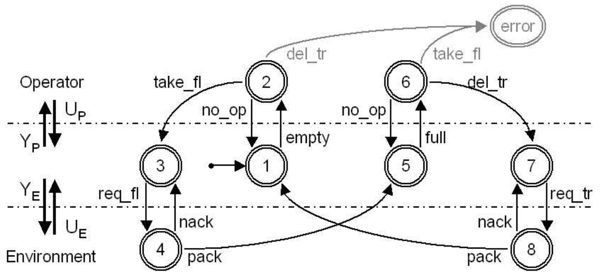
(i) $(\Sigma_{\mathrm{PE}}, \mathcal{L}_{\mathrm{PE}})$ is a system with $\Sigma_{\mathrm{PE}} := \Sigma_{\mathrm{P}} \dot{\cup} \Sigma_{\mathrm{E}}$, $\Sigma_{\mathrm{P}} := U_{\mathrm{P}} \dot{\cup} Y_{\mathrm{P}}$, $\Sigma_{\mathrm{E}} := U_{\mathrm{E}} \dot{\cup} Y_{\mathrm{E}}$; and
(ii) $(U_{\mathrm{P}}, Y_{\mathrm{P}})$ and $(U_{\mathrm{E}}, Y_{\mathrm{E}})$ are plant-I/O ports of $(\Sigma_{\mathrm{PE}}, \mathcal{L}_{\mathrm{PE}})$. $\square$

Note that an I/O plant always possesses the language format $\mathcal{L}_{\mathrm{PE}} \subseteq \overline{(Y_{\mathrm{P}}U_{\mathrm{P}} + Y_{\mathrm{E}}U_{\mathrm{E}})^*}$. To illustrate the above definition we introduce the following conceptional example.

Example. Consider a single transport unit (TU) as depicted in Fig. 3 a). Its behaviour can be modelled as an I/O plant $\mathcal{S}_{\mathrm{PE}} := (U_{\mathrm{P}}, Y_{\mathrm{P}}, U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{PE}})$ with $\mathcal{L}_{\mathrm{PE}}$ generated by the corresponding automaton model depicted in Fig. 3 b). The TU consists of a conveyor belt carrying a box that can hold the workpiece to be transported. A spring sensor inside the

[1]The relationship between systems, alphabets and languages is indicated by matching subscripts; e.g., the system $\mathcal{S}_{\mathrm{AB}}$ refers to the language $\mathcal{L}_{\mathrm{AB}}$ over the alphabet Σ_{AB}. Σ_{AB} denotes the disjoint union of Σ_{A} and Σ_{B}, and for inputs and outputs we use e.g. $\Sigma_{\mathrm{A}} = U_{\mathrm{A}} \dot{\cup} Y_{\mathrm{A}}$. The natural projections to Σ_{AB}^* and to Y_{A}^* are denoted p_{AB} and p_{YA}, respectively.


(a) physical layout



(b) I/O plant model

Fig. 3. Conceptional example: Transport Unit

box detects the absence or presence of a workpiece (*empty*, $full$). The initial state (state 1 in Fig. 3 b)) is defined such that the sensor reports $empty$. The operator can choose between three different commands (state 2). After the *no_op* (no operation) command, the TU does not move, and the system remains in the initial state. The command *del_tr* (deliver to right) leads to an error state as there is currently no workpiece present to deliver. Choosing the command $take\_fl$ (take from left) prompts the TU to move the box to its left border (state 3). Now it depends on the environment if a workpiece is provided from the left, which is modelled by the event req_fl (request from left) unobservable to the operator. For a plant description that is independent from the environment, we introduce the environment-events *pack* and *nack* (positive/negative acknowledge) as the environment may or may not comply with the requests of the plant. If the environment does not provide a workpiece (*nack*), the request is repeated. When a workpiece is provided from the environment, the sensor reports $full$. Now (state 6), the command $take\_fl$ leads to an error behaviour (the box can carry only one workpiece), and after *no_op* the plant still reports $full$. By the command *del_tr*, the belt moves the box to the right border. The event *req_tr* models the need for the workpiece to be withdrawn to the right by the environment. In case of *pack*, the system returns to its initial state. By $(U_\mathrm{P}, Y_\mathrm{P}) := (\{no\_op, take\_fl, del\_tr\}, \{empty, full\})$, we identify the interaction with the operator, $(U_\mathrm{E}, Y_\mathrm{E}) := (\{pack, nack\}, \{req\_fl, req\_tr\})$ describes the interaction with the environment. Note that $(U_\mathrm{P}, Y_\mathrm{P}, U_\mathrm{E}, Y_\mathrm{E}, \mathcal{L}_\mathrm{PE})$ features all I/O-plant properties of Definition III.4. □

For faithful operation, the plant must satisfy certain safety and liveness properties. Safety properties can be expressed as a language inclusion. The liveness properties of the plant depend on its external configuration; e.g., we may change the actual environment by reconfiguration, or substitute the operator by a controller. The variety of controller-I/O ports that can be connected to the I/O plant to obtain the desired liveness properties is described by *constraints*.

Definition III.5. A tuple $(U, Y, \mathcal{L})$ is a *constraint* if
(i) $(\Sigma, \mathcal{L})$ is a complete system with $\Sigma = U \dot{\cup} Y$;
(ii) (U, Y) is a controller-I/O port of $(\Sigma, \mathcal{L})$. □

We refer to the *minimal constraint* $(U, Y, \mathcal{L})$ with $\mathcal{L} = \overline{(YU)^*}$, if actually *no* constraint is considered. The operator and the environment constraint are denoted $\mathcal{S}_\mathrm{P} = (U_\mathrm{P}, Y_\mathrm{P}, \mathcal{L}_\mathrm{P})$ and $\mathcal{S}_\mathrm{E} = (U_\mathrm{E}, Y_\mathrm{E}, \mathcal{L}_\mathrm{E})$, respectively. Regarding liveness under constraints, we identify two properties adequate for our setting.

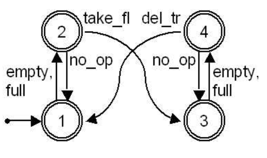
Definition III.6. [Liveness] Let $\mathcal{S}_\mathrm{PE} = (U_\mathrm{P}, Y_\mathrm{P}, U_\mathrm{E}, Y_\mathrm{E}, \mathcal{L}_\mathrm{PE})$ be an I/O plant, and let $\mathcal{S}_\mathrm{P} = (U_\mathrm{P}, Y_\mathrm{P}, \mathcal{L}_\mathrm{P})$ and $\mathcal{S}_\mathrm{E} = (U_\mathrm{E}, Y_\mathrm{E}, \mathcal{L}_\mathrm{E})$ be constraints.

If $\mathcal{L}_\mathrm{P} \parallel \mathcal{L}_\mathrm{PE} \parallel \mathcal{L}_\mathrm{E}$ is complete, then $\mathcal{S}_\mathrm{PE}$ is said to be *complete w.r.t. the constraints* $\mathcal{S}_\mathrm{P}$ *and* $\mathcal{S}_\mathrm{E}$.

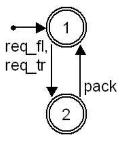
If $(\forall w \in (\mathcal{L}_\mathrm{P} \parallel \mathcal{L}_\mathrm{PE} \parallel \mathcal{L}_\mathrm{E})^\infty)[p_\mathrm{YP}(w) \in Y_\mathrm{P}^\omega]$, then $\mathcal{S}_\mathrm{PE}$ is said to be Y_P*-live w.r.t. the constraints* $\mathcal{S}_\mathrm{P}$ *and* $\mathcal{S}_\mathrm{E}$. □

Completeness requires the plant to persistently issue events, i.e., prohibits deadlocks. Y_P-liveness requires that any infinite sequence of events must include an infinite number of measurement events reported to the operator, and hence, in return, the operators influence is persistently possible.

Example. Note that $\mathcal{S}_\mathrm{PE}$ is neither complete nor Y_P-live with respect to minimal constraints. As seen in Fig. 3 b), completeness is violated in the *error* state because no further event is possible. The avoidance of this *deadlock* is described by an operator constraint $\mathcal{S}_\mathrm{P}$ on the correct alternation of the commands $take\_fl$ and *del_tr*, see Fig. 4 a). Moreover, as the I/O plant is designed independently of the environment, the case that the environment *never* complies with requests of the plant is included in the model. This *livelock* violates the Y_P-liveness and is represented by a $(req\_fl\ nack)$-loop between states 3 and 4 and a $(req\_tr\ nack)$-loop between states 7 and 8 in Fig. 3 b). The environment constraint $\mathcal{S}_\mathrm{E}$ as seen in Fig. 4 b) models the prohibition of the event *nack*, i.e., the assumption that requests of the plant are *always* accepted by the environment. The liveness properties of the plant are preserved if a controller connected to the plant complies with the operator constraint and if the external configuration meets the environment constraint. □



(a) Operator constraint



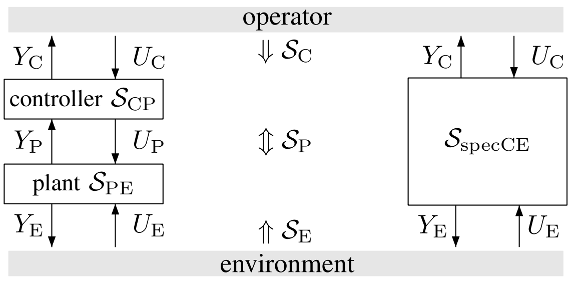
(b) Environment constraint

Fig. 4. Constraints for the TU

In the following section, we recall the term of an I/O controller (enforcing a safety specification) and admissibility conditions for a complete and Y_P-live closed loop.

B. I/O CONTROLLER

The task of the I/O controller is to assist the operator in manipulating the environment according to a given specification; see Fig. 5 a). Control events $\mu \in U_\mathrm{C}$ issued by the


operator
Y_{C}
U_{C}
controller $\mathcal{S}_{\mathrm{CP}}$
Y_{P}
U_{P}
plant $\mathcal{S}_{\mathrm{PE}}$
Y_{E}
U_{E}
$\Downarrow \mathcal{S}_{\mathrm{C}}$
$\Updownarrow \mathcal{S}_{\mathrm{P}}$
$\Uparrow \mathcal{S}_{\mathrm{E}}$
$\mathcal{S}_{\mathrm{specCE}}$
environment

(a) Closed loop (b) Constraints (c) Specification

Fig. 5. I/O Controller Synthesis Problem

operator trigger more or less complex tasks to be performed by the controller and the plant. This eventually results in an abstract measurement event $\nu \in Y_{\mathrm{C}}$ issued by the controller to indicate the status of the current task; e.g. successful completion or failure. Hence, the controller performs both control and measurement aggregation and thereby provides an abstract view $\mathcal{S}_{\mathrm{CE}} = (\Sigma_{\mathrm{CE}}, \mathcal{L}_{\mathrm{CE}}) := (\Sigma_{\mathrm{CE}}, p_{\mathrm{CE}}(\mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}}))$ of the closed loop between operator and environment. Accordingly, we propose to draft the specification as an I/O plant model $\mathcal{S}_{\mathrm{specCE}} = (\Sigma_{\mathrm{CE}}, \mathcal{L}_{\mathrm{specCE}})$ of the *desired* external closed loop with the environment constraint $\mathcal{S}_{\mathrm{E}}$ and an optional operator constraint $\mathcal{S}_{\mathrm{C}}$; see Fig. 5 c).

Example. For the TU, a specification can be designed by the system $\mathcal{S}_{\mathrm{specCE}} = (\Sigma_{\mathrm{C}} \dot{\cup} \Sigma_{\mathrm{E}}, \mathcal{L}_{\mathrm{specCE}})$ with $\Sigma_{\mathrm{C}} := U_{\mathrm{C}} \dot{\cup} Y_{\mathrm{C}} = \{\mathit{stby}, \mathit{l2r}\} \dot{\cup} \{\mathit{idle}\}$ and $\mathcal{L}_{\mathrm{specCE}}$ as seen in Fig. 6. By the measurement event idle we introduce a feedback to the operator notifying that the TU is ready for transport of the next workpiece. We specify that the operator can choose between two operational modes. After the command stby (standby), no interaction with the environment is desired. With the command $\mathit{l2r}$ (left to right) we specify that a workpiece from left is requested from the environment ($\mathit{req\_fl}$) and, in case of positive acknowledge, the workpiece shall be provided to the right ($\mathit{req\_tr}$). It is the controller's task to enforce appropriate Σ_{P}-sequences on the plant to achieve the specified behaviour with respect to the environment. $\square$

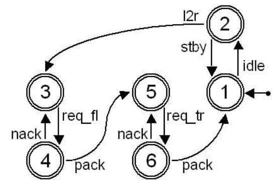


Fig. 6. Specification for the TU

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

Definition III.7. An *I/O controller* is a tuple $\mathcal{S}_{\mathrm{CP}} = (U_{\mathrm{C}}, Y_{\mathrm{C}}, U_{\mathrm{P}}, Y_{\mathrm{P}}, \mathcal{L}_{\mathrm{CP}})$, where

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

(ii) $(U_{\mathrm{C}}, Y_{\mathrm{C}})$ and $(U_{\mathrm{P}}, Y_{\mathrm{P}})$ are a plant- and a controller-I/O port for $(\Sigma_{\mathrm{CP}}, \mathcal{L}_{\mathrm{CP}})$, respectively;

(iii) $\mathcal{L}_{\mathrm{CP}} \subseteq \overline{(Y_{\mathrm{P}}(U_{\mathrm{P}} + Y_{\mathrm{C}} U_{\mathrm{C}} U_{\mathrm{P}}))^*}$. $\square$

The language format defined in item (iii) ensures that an operator command $\mu_{\mathrm{C}} \in U_{\mathrm{C}}$ is actually applied to the plant beginning with a control event $\mu_{\mathrm{P}} \in U_{\mathrm{P}}$. Note that controller and plant synchronise only via the alphabet Σ_{P}; from the perspective of the plant, the controller conforms with the alternation $\overline{(Y_{\mathrm{P}} U_{\mathrm{P}})^*}$ and, in particular, the controller cannot observe environment events. When connecting controller and plant we obtain the *full closed-loop behaviour* $\mathcal{S}_{\mathrm{CPE}} = (\Sigma_{\mathrm{CPE}}, \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}})$ and the *external closed-loop behaviour* $\mathcal{S}_{\mathrm{CE}} = (\Sigma_{\mathrm{CE}}, p_{\mathrm{CE}}(\mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}}))$.

The I/O structure of plant and controller itself is not sufficiently strong to imply liveness of the closed loop. As the controller may not comply with the operator constraint $\mathcal{S}_{\mathrm{P}}$ identified for liveness of the plant, 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. 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 this *livelock* situation the operator no longer receives measurement events $\nu \in Y_{\mathrm{C}}$ and, hence, can not issue further control events. Our admissibility condition addresses both issues and implies completeness and Y_{C}-liveness for the closed-loop system; see Theorem III.9.

Definition III.8. 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* for 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 $\mathcal{S}_{\mathrm{E}}$. $\square$

The above definition provides each constraint depicted in Fig. 5 b) with a certain role; $\mathcal{S}_{\mathrm{P}}$ has to be met by the I/O controller in item (i), while $\mathcal{S}_{\mathrm{C}}$ and $\mathcal{S}_{\mathrm{E}}$ must be fulfilled by the external configuration in both items (i) and (ii). For the external closed loop we obtain the following result.

Theorem III.9. [10] 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 for $\mathcal{S}_{\mathrm{PE}}$ w.r.t. the constraints $\mathcal{S}_{\mathrm{C}}$, $\mathcal{S}_{\mathrm{P}}$, and $\mathcal{S}_{\mathrm{E}}$. Then the external closed-loop system $\mathcal{S}_{\mathrm{CE}} = (\Sigma_{\mathrm{CE}}, p_{\mathrm{CE}}(\mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}}))$ 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 $\mathcal{S}_{\mathrm{E}}$.

This means the admissibility condition implies that the external closed loop $\mathcal{S}_{\mathrm{CE}}$ is an I/O plant that is complete and Y_{C}-live with respect to the given constraints. Thus, in a hierarchical control architecture, the closed loop provides a plant model for the next layer of control and measurement aggregation. Hence, the problem to be solved is the synthesis of an admissible I/O controller. In the next section we present an algorithmic design procedure that provably yields a solution to the synthesis problem.

IV. I/O Controller Synthesis

The setting depicted in Fig. 5 forms a controller synthesis problem, with the I/O controller as the desired solution.

Definition IV.1. 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 $\mathcal{S}_{\mathrm{PE}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{PE}})$ 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 $\mathcal{S}_{\mathrm{specCE}} = (U_{\mathrm{C}}, Y_{\mathrm{C}}, U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{specCE}})$ is a *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 for $\mathcal{S}_{\mathrm{PE}}$ 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 $\mathcal{S}_{\mathrm{PE}}$ w.r.t. $\mathcal{S}_{\mathrm{C}}$ and $\mathcal{S}_{\mathrm{E}}$, i.e., $p_{\mathrm{CE}}(\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}) \subseteq \mathcal{L}_{\mathrm{specCE}}$. □

Example. The I/O-plant model $\mathcal{S}_{\mathrm{PE}}$ of the TU as in Fig. 3 b), the constraints $\mathcal{S}_{\mathrm{P}}$ and $\mathcal{S}_{\mathrm{E}}$ as in Fig. 4, a minimal constraint $\mathcal{S}_{\mathrm{C}}$ and the specification $\mathcal{S}_{\mathrm{specCE}}$ as in Fig. 6 pose an I/O controller synthesis problem. □

A major aspect in finding a solution to the I/O controller synthesis problem is to restrict a given language to a Y_{C}-live sublanguage. This involves the detection of strings that compromise Y_{C}-liveness. In the automata representation of the considered language, such a string is indicated by a so-called Y_{C}-less cycle of states, within which each state can be visited arbitrarily often without the occurrence of any Y_{C}-event. This implies that such string features two properties:
a) Nerode-equivalence to at least one of its own strict prefixes (i.e., a cycle is closed)
b) The extension from each Nerode-equivalent prefix to the considered string does not contain any Y_{C}-event.
A Y_{C}-live sublanguage of an arbitrary language is achieved by allowing only finite sequences of transitions between states within the Y_{C}-less cycle. Unfortunately, it can be shown that, in general, the *supremal* Y_{C}-live sublanguage of a given language does not exist.[2] However, also the *finite* iteration of Y_{C}-less cycles can be seen as a needless back step on the path to the next Y_{C}-event. Hence, we propose to derive a so-called Y_{C}-Acyclic sublanguage that guarantees that a Y_{C}-less cycle of the original language is *never* closed.

Definition IV.2. Let $\mathcal{L}$ be a regular language over $\Sigma \supseteq Y_{\mathrm{C}}$. A string $t \in \Sigma^*$ is Y_{C}-Acyclic w.r.t. $\mathcal{L}$, if

$$\forall r, s \in \Sigma^*,\ r < t:\ (rs = t \wedge rs \equiv_{\mathcal{L}} r) \Rightarrow p_{Y\mathrm{C}}(s) \neq \epsilon$$

where $\equiv_{\mathcal{L}}$ denotes the Nerode equivalence over Σ^* w.r.t. $\mathcal{L}$.
The language $\mathcal{K} \subseteq \mathcal{L}$ is a *Y_{C}-Acyclic sublanguage of $\mathcal{L}$* if

$$\forall s \in \mathcal{K}:\ s \text{ is } Y_{\mathrm{C}}\text{-Acyclic w.r.t } \mathcal{L}.$$ □

Thus, the *supremal* Y_{C}-Acyclic sublanguage of a language $\mathcal{L} \subseteq \Sigma^*$ is the set of *all* strings that are Y_{C}-Acyclic w.r.t. $\mathcal{L}$.

Proposition IV.3. Let $\mathcal{L}$ be a regular language over the alphabet $\Sigma \supseteq Y_{\mathrm{C}}$. The language

$$Y_{\mathrm{C}}\text{Acyclic}(\mathcal{L}) := \{s \in \mathcal{L} \mid s \text{ is } Y_{\mathrm{C}}\text{-Acyclic w.r.t.}\mathcal{L}\}$$

(i) is the *supremal* Y_{C}-Acyclic sublanguage of $\mathcal{L}$, and
(ii) is Y_{C}-live.

[2]As a consequence to this new result, Proposition III.7 and Theorem IV.7 in [10] - in general - only hold for finite unions!

Our I/O controller synthesis algorithm computes the supremal Y_{C}-Acyclic sublanguage to yield an admissible I/O controller; details of each step are explained below.

I/O Controller Synthesis Algorithm (I/O CSA)
Let $\Pi := (\mathcal{S}_{\mathrm{PE}}, \mathcal{S}_{\mathrm{C}}, \mathcal{S}_{\mathrm{P}}, \mathcal{S}_{\mathrm{E}}, \mathcal{S}_{\mathrm{specCE}})$ be an I/O controller synthesis problem, where $\mathcal{S}_{\mathrm{specCE}}$ is an I/O plant model of the desired external closed-loop behaviour. The system $\mathcal{S}_{\mathrm{CP}} = (\Sigma_{\mathrm{CP}}, \mathcal{L}_{\mathrm{CP}})$ is computed as follows.

(I) Restrict the behaviour of the full closed loop:

$$\mathcal{K}_0 := \mathcal{L}_{\mathrm{PE}_c} \parallel \mathcal{L}_{\mathrm{P}} \parallel \overline{(Y_{\mathrm{P}}(\epsilon + Y_{\mathrm{C}}U_{\mathrm{C}})U_{\mathrm{P}})^*} \parallel \mathcal{L}_{\mathrm{specCE}}$$

where $\mathcal{L}_{\mathrm{PE}_c}$ is the plant under constraints $\mathcal{L}_{\mathrm{PE}_c} := \mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}$.

(II) Compute the supremal Y_{C}-Acyclic sublanguage:

$$\mathcal{K}_1 := Y_{\mathrm{C}}\text{Acyclic}(\mathcal{K}_0)$$

(III) Define the events $\Sigma_{\mathrm{uc}} := U_{\mathrm{C}} \cup Y_{\mathrm{P}} \cup \Sigma_{\mathrm{E}}$ uncontrollable and the events $\Sigma_{\mathrm{o}} := \Sigma_{\mathrm{CP}}$ observable. Compute a complete, controllable and normal sublanguage of $\mathcal{K}_1$ w.r.t. $\mathcal{L}_{\mathrm{PE}_c}$, Σ_{uc} and Σ_{o}:

$$\mathcal{K}_2 := (\mathcal{K}_1)^{(cCN)}$$

(IV) Compute the projection to the controller alphabet:

$$\mathcal{K}_{\mathrm{CP}} := p_{\mathrm{CP}}(\mathcal{K}_2)$$

(V) Add error behaviour to make Y_{P} and U_{C} free in $\mathcal{L}_{\mathrm{CP}}$:

$$\mathcal{L}_{\mathrm{CP}} := \mathcal{K}_{\mathrm{CP}} \cup \mathcal{K}_{\mathrm{CP}}^{err}$$

Step (I): By parallel composition, we restrict the possible plant behaviour $\mathcal{L}_{\mathrm{PE}_c}$ to the language format $\overline{(Y_{\mathrm{P}}(\epsilon + Y_{\mathrm{C}}U_{\mathrm{C}})U_{\mathrm{P}})}$ (required by Definition III.7 of the I/O controller), to the constraint $\mathcal{L}_{\mathrm{P}}$ (required by the admissibility condition (i) in Definition III.8) and to the safety specification $\mathcal{L}_{\mathrm{specCE}}$.
Step (II): The supremal Y_{C}-Acyclic sublanguage of $\mathcal{K}_0$ is computed according to Definition IV.3. Note that any sublanguage of $Y_{\mathrm{C}}\text{Acyclic}(\mathcal{L})$ is also a Y_{C}-Acyclic (and thus Y_{C}-live) sublanguage of $\mathcal{L}$. Hence, the restriction in the following step does not compromise this property.
Step (III): For brevity, we omit a formal definition of the operator $(\cdot)^{(cCN)}$. Our current implementation iteratively computes the supremal complete and controllable sublanguage according to [5] and the supremal normal sublanguage according to [1] until a fixpoint is reached. The *supremal* complete, controllable and normal sublanguage $(\cdot)^{\uparrow(cCN)}$ is subject of current research.
Step (IV): The unobservability of the environment alphabet is taken into account in this step. As $\mathcal{K}_2$ is normal w.r.t. $\mathcal{L}_{\mathrm{PE}_c}$ and Σ_{CP} (see Step (III)), we can conclude: $\mathcal{K}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}_c} = p_{\mathrm{CP}}^{-1}(p_{\mathrm{CP}}(\mathcal{K}_2)) \cap \mathcal{L}_{\mathrm{PE}_c} = \mathcal{K}_2 = \mathcal{K}_2 \parallel \mathcal{L}_{\mathrm{PE}_c}$
Hence, the partial observation does not affect the closed-loop behaviour.
Step (V): To formally account for Y_{P}- and U_{C}-events that do

not occur in the closed-loop behavior, we insert a strategic error behavior $\mathcal{K}_{\mathrm{CP}}^{err}$ with $\mathcal{K}_{\mathrm{CP}}^{err} \parallel (\mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}c}) = \emptyset$.

As a main result, I/O CSA leads to a solution to Π.

Theorem IV.4. Let $\Pi := (\mathcal{S}_{\mathrm{PE}}, \mathcal{S}_{\mathrm{C}}, \mathcal{S}_{\mathrm{P}}, \mathcal{S}_{\mathrm{E}}, \mathcal{S}_{\mathrm{specCE}})$ be an I/O controller synthesis problem according to Definition IV.1, where $\mathcal{S}_{\mathrm{specCE}}$ is an I/O plant model of the desired external closed-loop behaviour. If the language $\mathcal{L}_{\mathrm{CP}}$ is constructed by applying I/O CSA to Π, then:

$$\mathcal{S}_{\mathrm{CP}} := (\Sigma_{\mathrm{CP}}, \mathcal{L}_{\mathrm{CP}}) \text{ is a solution to } \Pi.$$

Example. For the I/O controller synthesis problem of the TU, our synthesis algorithm returns the controller $\mathcal{S}_{\mathrm{CP}}$ with $\mathcal{L}_{\mathrm{CP}}$ as depicted in Fig. 7. Formally, the I/O controller accepts all measurement events of the plant, even those that can actually not occur; the respective transitions are denoted by grey arrows leading to error states that represent $\mathcal{K}_{\mathrm{CP}}^{err}$ and are never reached. It is verified that if the environment constraint $\mathcal{S}_{\mathrm{E}}$ is fulfilled, the closed loop is complete and Y_{C}-live and features the external behaviour specified by $\mathcal{S}_{\mathrm{specCE}}$. $\square$

Similar to [9], our framework allows for *abstraction based controller synthesis*; i.e., solutions obtained for a plant abstraction provably also solve the original problem, see [10]. If the abstraction is of less complexity, the computational effort for controller synthesis is reduced accordingly. We propose the safety specification $\mathcal{S}_{\mathrm{specCE}}$ of the preceding design step as a plant abstraction of the external closed-loop behaviour $\mathcal{S}_{\mathrm{CE}}$, as it meets the abstraction condition $\mathcal{L}_{\mathrm{CE}} \subseteq \mathcal{L}_{\mathrm{specCE}}$ required in [10] and represents those aspects of the preceding design step that are relevant for subsequent controller design.

V. Chain of Transport Units: Complexity

The design of a hierarchical control architecture is explained in detail in [10]. We sketch the application to the chain of TUs. *Controller design:* A local controller for each TU is designed for the specification $\mathcal{S}_{\mathrm{specCE}}$ according to the previous sections. *Abstraction step*: As proposed above, the external closed loop $(\Sigma_{\mathrm{CE}}, \mathcal{L}_{\mathrm{CE}})$ of each TU is replaced by $\mathcal{S}_{\mathrm{specCE}}$. *Subsystem composition:* The abstractions of each two neighbored TUs are composed using the *I/O shuffle* composition. For each pair, the interaction of the two TUs among themselves and with the remaining environment is captured by a subordinate *I/O environment* model, which counts 14 states. *Design of superposed controllers:* For simplicity, we keep up the specification in Fig. 6 also for the compounds of two TUs. The controller for two TUs and that specification counts 28 states. *Overall hierarchy:* Subsystem composition and controller synthesis are alternated, until a top-level controller for the whole chain of TUs is synthesised.

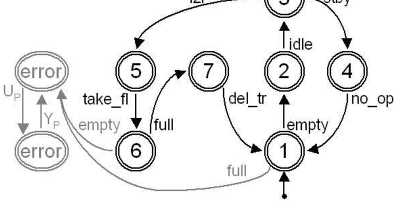


Fig. 7. Controller for the TU

TABLE I
Transport Unit: Sum of States

No. of TUs	plant hierarchy	controller hierarchy	monolithic plant model
1	9	9	6
2	$2 \cdot 9 + 14 = 32$	$2 \cdot 9 + 28 = 46$	36
4	78	120	1296
8	170	278	7776
16	354	594	approx. $2,8 \cdot 10^{12}$

Table I shows the sum of states for a chain of up to 16 TUs: both, the plant model hierarchy (comprising all I/O plants and the environment hierarchy) and the controller hierarchy feature linear complexity compared to the exponential growth of a monolithic plant model.

VI. Conclusion

In this contribution, we address the controller synthesis problem that arises in the I/O-based framework [10] for the hierarchical design of discrete event systems. We present an algorithmic controller design procedure that yields a solution to the synthesis problem - the supremal solution is subject of future work. The transport unit example presented for the illustration of our approach verifies that the overall complexity is limited efficiently, while safety- and liveness-properties are preserved.

References

[1] R. D. Brandt, V. Garg, R. Kumar, F. Lin, S. I. Marcus, and W. M. Wonham. Formulas for calculating supremal controllable and normal sublanguages. *System and Control Letters*, 15(2):111–117, 1990.

[2] P.E. Caines and Y.J. Wei. The hierarchical lattices of a finite machine. *Systems and Control Letters*, 25:257–263, 1995.

[3] A.E.C. da Cunha, J.E.R. Cury, and B.H. Krogh. An assume guarantee reasoning for hierarchical coordination of discrete event systems. *WODES*, 2002.

[4] T. Jeron, H. Marchand, V. Rusu, and V. Tschaen. Ensuring the conformance of reactive discrete-event systems using supervisory control. *Conference on Decision and Control*, 2003.

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

[6] R. Kumar, V.K. Garg, and S.I. Marcus. Finite buffer realization of input-output discrete-event systems. *IEEE Transactions on Automatic Control*, 40(6):1042–1053, 1995.

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

[8] N. Lynch and M. Tuttle. An introduction to input/output automata. *CWI-Quarterly*, 2(3):219–246, 1989.

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

[10] S. Perk, T. Moor, and K. Schmidt. Hierarchical discrete event systems with inputs and outputs. *WODES*, 2006.

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

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

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

[14] K.C. Wong and W.M. Wonham. Hierarchical control of discrete-event systems. *Discrete Event Dynamic Systems: Theory and Applications*, 6(3):241–273, 1996.