A guide to modelling and control with modules of signal nets

Jörg Desel, Hans-Michael Hanisch, Gabriel Juhás, Robert Lorenz, Christian Neumair

Angaben zur Veröffentlichung / Publication details:

Desel, Jörg, Hans-Michael Hanisch, Gabriel Juhás, Robert Lorenz, and Christian Neumair. 2004. "A guide to modelling and control with modules of signal nets." In *Integration of software specification techniques for applications in engineering: priority program SoftSpez of the German Research Foundation (DFG), Final Report*, edited by Hartmut Ehrig, Werner Damm, Jörg Desel, Martin Große-Rhode, Wolfgang Reif, Eckehard Schnieder, and Engelbert Westkämper, 3147:270–300. Berlin [u.a.]: Springer. https://doi.org/10.1007/978-3-540-27863-4_16.

A Guide to Modelling and Control with Modules of Signal Nets

Jörg Desel[1], Hans-Michael Hanisch[2], Gabriel Juhás[1], Robert Lorenz[1], and Christian Neumair[1]

[1] Lehrstuhl für Angewandte Informatik
Katholische Universität Eichstätt-Ingolstadt, Germany
name.surname@ku-eichstaett.de
[2] Lehrstuhl für Automatisierungstechnik
Martin Luther Universität Halle-Wittenberg, Germany
hans-michael.hanisch@iw.uni-halle.de

Abstract. In this paper we summarize syntax and semantics of modules of elementary signal nets and explain how to synthesize the control for discrete event systems modelled by such modules.
Signal nets, introduced in [8,9,10,12], are based on Petri net modules which communicate via signals. Two kinds of signals are employed, namely active signals which force occurrence of (enabled) events, and passive signals which enable/prohibit occurring of events. Modelling with such modules appears to be very natural from an engineering perspective. It enables hierarchical structuring and supports the locality principle.
Given an uncontrolled system (a plant), modelled by a module of an elementary signal net, and a control specification, given as a regular language representing the desired signal output behavior of this system, we show step-by-step how to automatically synthesize the maximally permissive and nonblocking behavior of the plant respecting the control specification. Finally, we show how to synthesize the controller (as a module of an elementary signal net) forcing the plant to realize the controlled behavior.

1 Introduction

In complex applications, models are usually constructed in several steps and are described on several levels of abstraction. Systems are parts of bigger systems, such as a robot is a part of a manufacturing cell. Conversely, many systems are composed from subsystems. This fact motivates the principle of modularity and compositionality. Considering a certain level of abstraction, one does not need to reason about all details of subsystems which were taken into consideration on a sublevel. It is usually sufficient to consider just those parts of subsystems which are in contact with the environment, i.e. the "input/output" parts and to consider the "inside" of the subsystems being a "black box". Such an approach supports local changes in the whole system, i.e. it enables the replacement of one subsystem by another with the same "input/output"-functionality.

Considering discrete event systems (DES), Petri nets are a very successfully successfully used modelling formalism [13,29]. The main reason is that they offer both, nice graphical representation and formal background. In addition, modelling with Petri nets is popular because Petri nets usually allow a more compact and structured representation of the system behavior than automata. There are many case studies using Petri nets in modelling and control and many tools supported by sophisticated analytical methods. However, Petri nets (at least in their basic version) do not support the above mentioned features which are very essential for engineering applications: The absence of input/output structure seems to be a strong limitation. Additionally, the important feature of hierarchical structuring is not directly supported by Petri nets.

There are many compositional frameworks for Petri nets, mostly based on gluing common places and/or transitions. However, it is desirable that the composition of modules preserves the structure of modules. Modules of signal nets constitute an extension of Petri nets which supports input/output structuring, modularity and compositionality in an intuitive graphical way. This formalism was developed in a series of papers under the name *net condition/event system* and is widely used for modelling of complex DES, see e.g. [9,10,12]. A signal net is a Petri net enriched by *event signals*, which force the occurrence of (enabled) events, and *condition signals* which enable/prohibit the occurrence of events. Adding input and output signals to a signal net, one gets a *module of a signal net*. Modules of signal nets can be composed by connecting their respective input and output signals.

In the first part of this paper we summarize syntax and semantics of modules of elementary signal nets, where the underlying Petri nets are elementary Petri nets. In the second part, we give a survey on control synthesis for DES modelled by modules of elementary signal nets. In this part technical details are replaced by illustrations (for a detailed presentation see [18,19]). Furthermore, a brief comparison to the supervisory control synthesis approach based on automata is provided.

In the problem of control synthesis for DES, a system is given which can interfere with its environment via inputs and outputs. This is the object to be controlled, and it is called "plant". The goal of control is to ensure a specified behavior of this plant which is given as a set of desired sequences of inputs and outputs. The plant is therefore equipped with sensors that provide information about some (usually not all) so called *observable* states and state transitions of the plant. It is also equipped with actuators that allow to control the behavior of the plant by enforcing or preventing some (usually not all) so called *controllable* state transitions in the plant. The central idea is that plant and control build a so called *closed loop* (or *feedback loop*) which means, roughly speaking, that the control gives inputs to the actuators of the plant based on the observed sensor outputs of the plant.

Modelling a plant by a module of a signal net, sensors in the plant may provide condition signals to give information about a reached observable state to the control. For example, a condition signal can indicate that a process variable of the plant is within a given range of its value. Sensors in the plant may also provide event signals to give information about the occurrence of an observable state transition to the control. For example, it can be indicated by an event signal that a process variable in the plant is just reaching a threshold.

A controller that controls the plant may use both types of signals as well. Via condition signals, the controller prohibits/enables controllable state transitions in the plant whereas via event signals, the controller tries to force controllable state transitions in the plant to occur.

In [18,19], we identify which event signal inputs have to be sent to the plant module in order to observe only such sequences of event signal outputs which are prefixes of and can be completed to sequences of event signal outputs belonging to the control specification. This control specification is given as a regular language. The resulting output behavior of the plant is maximal with this properties. In other words, we construct a language over event signal inputs and outputs of the module of the plant which represents the maximally permissive nonblocking controllable behavior satisfying the control specification. Finally, we show in [19] that for such a behavior there exists a control module (of a signal net) which, composed with the plant module, realizes this behavior. As the main result of [19], we construct such a control module.

The formal definitions in [18,19] are based on low-level Petri nets, where tokens carry no data structure. In particular, the interfaces and the communication between modules are low-level. These elementary signal net models are close to the physical level (similar to assembler code in the area of programming languages). Of course, one can achieve more compact representations (for example of protocols, services, data types et cetera) by using appropriate high level concepts (such as high-level Petri nets in this case). However, the fundamental problems arising in controller synthesis considered in [18,19] (such as observability and controllability of behavior) are of low-level nature. For real applications, a higher-level modelling language would be more suitable. In [5] we present such a language based on signal nets extended by high-level features (such as data types, annotated condition signals, timers etc.) and employed it in a case study from automotive industry (modelling of controllers for the new AUDI A8 model).

Several related work employs modules of signal nets in the control of discrete event systems. In [9,10,12] effective solutions for particular classes of specifications, such as forbidden states, or simple desired and undesired sequences of events are described. An approach for control specification given by cycles of observable events was presented in [21]. Up to recent time, the problem of control synthesis for the general class of specifications given by regular languages (as in supervisory control theory for systems modelled by automata) remained open for modules of signal nets. In [19], we filled this gap.

The paper splits into two parts: In the first part we present *modules of elementary signal nets* with definition of step semantics, composition rules and input/output behavior. In the second part we illustrate control synthesis of DES with modules of elementary signal nets: In Subsection 3.1 we show how to automatically synthesize the maximally permissive nonblocking controllable behavior of a module of a signal net (representing the plant) respecting a given regular specification language. In Subsection 3.2 we present how to construct the controller as a module of a signal net. Finally, we take a short view on methods that use the structure of some models rather than the complete enumeration of the state space in Subsection 3.3. A conclusion and an outlook on further work is given in Section 4.

Part I

2 Modules of Signal Nets

As mentioned in the introduction, we use an extension of elementary Petri nets (1-safe Petri nets) which allows to model condition and event signals, supports modularity, and preserves the essential benefits of Petri nets. We assume the underlying elementary Petri nets to be equipped with the so called *first consume, then produce* semantics (since we allow loops, see e.g. [16]). The first step of the extension is to add two kinds of signals, namely active signals which force the occurrence of (enabled) transitions, and passive signals which enable/prohibit the occurrence of transitions. These signals are represented respectively by two kinds of arcs. A Petri net extended with such signals is simply called a *signal net*.

Active signals, also called event arcs, are represented by arcs connecting transitions. They are interpreted in the following way: An *event arc* leading from transition t_1 to transition t_2 specifies that, if transition t_1 occurs and transition t_2 is enabled to occur then the occurrence of t_2 is forced (synchronized) by the occurrence of t_1, i.e. then transitions t_1 and t_2 occur in one (synchronized) step. If t_2 is not enabled, t_1 occurs without t_2, while an occurrence of t_2 without t_1 is not possible. As an example, an event turning on a switch would be modelled via transition t_1, while the event lighting the bulb would be modelled via transition t_2.

In general, (synchronized) steps of transitions are defined inductively in the above way. Every step starts at one so called *spontaneous* transition which is not synchronized by another transition. It is required that there are no cycles of event arcs.

Consider a transition t which is synchronized by transitions $t_1, \ldots, t_n, n \geqslant 2$. Then there are two dialects in the literature to interpret such a situation. For simplicity we consider the case $n = 2$. In the first approach [9,10,12] both transitions t_1 and t_2 have to agree to synchronize t. Thus the only possible step of transitions involving t has to include transitions t_1 and t_2, too. We call this dialect AND-semantics (see Figure 1, part (b)). In the second one [4] the occurrence of at least one of the transitions t_1 and t_2 synchronizes transition t, if t is enabled. We call this dialect OR-semantics (see Figure 1, parts (a) and (c)).

In general, the relation given by event arcs builds a forest of arbitrary depth. We introduce the most general interpretation, where we distinguish between OR- and AND-synchronized transitions. An OR-synchronized transition demands to be synchronized by at least one of its synchronizing transitions, whereas an AND-synchronized transition demands to be synchronized by all of its synchronizing transitions. Since we allow loops w.r.t. single transitions, i.e. transitions connected to a place with flow arcs in both directions, we also allow loops w.r.t. steps of transitions (see Figure 2, part (a)).

Passive signals are expressed by so called *condition arcs* (also called read arcs or test arcs in the literature) connecting places and transitions. A condition arc leading from a place to a transition models the situation that the transition can only occur if the place is in a certain state but this state remains unchanged by the transition's occurrence (read operation) (see Figure 2, part (b)). There are no condition arcs leading from a transition to a place. Several transitions belonging to a synchronized step can test a place to be in a

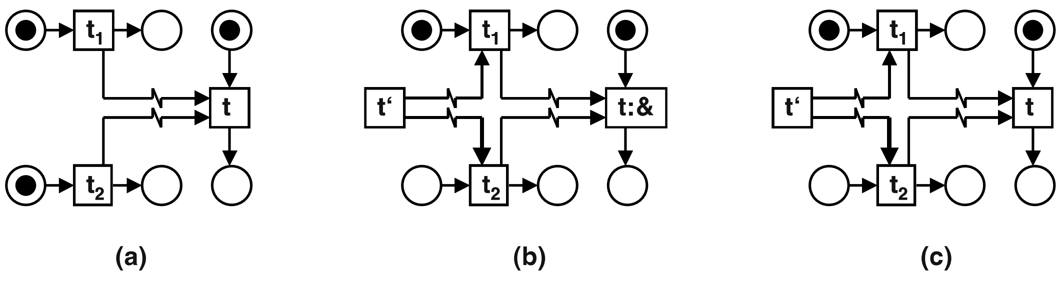

Fig. 1. In (a) the enabled steps are $\{t_1, t\}$ and $\{t_2, t\}$. (b) shows a signal net with AND-semantics: Here the only enabled step is $\{t', t_1\}$, i.e. t is not synchronized. In (c) the same net is shown in OR-semantics: Here we have the enabled step $\{t', t_1, t\}$, i.e. t is synchronized.

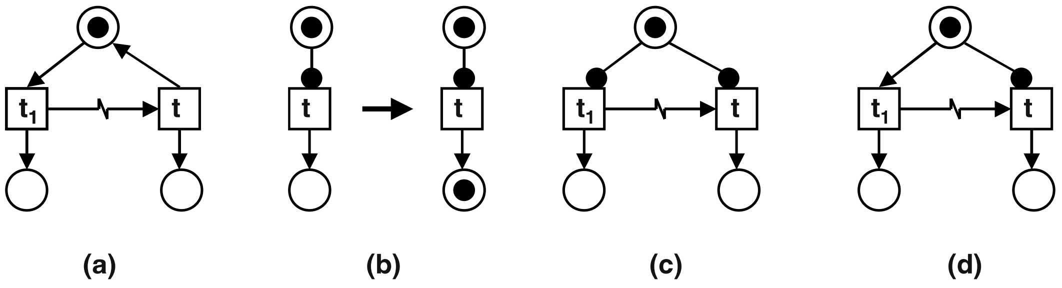

Fig. 2. (a) shows an enabled step $\{t_1, t\}$. The left part of (b) shows an enabled transition t which tests a place if it is marked. The occurrence of t leads to the marking shown in the right part of (b). Figures (c) and (d) again present situations of an enabled step $\{t_1, t\}$.

certain state via passive signals simultaneously since the state of this place is not changed by their occurrence (see Figure 2, part (c)). We also allow that a transition belongs to a synchronized step of transitions testing a place to be in a certain state via a passive signal, whereas the state of this place is changed by the occurrence of this or of another transition in this step. That means we use the so called *a priori* semantics [15] for the occurrence of steps of transitions, where testing of states precedes changing of states by occurrence of steps of transitions (see Figure 2, part (d)).

As usual, places, transitions and the flow relation are drawn using circles, boxes and arrows respectively. To distinguish between OR- and AND-synchronized transitions, AND-synchronized transitions are additionally labelled by the symbol "&". Event arcs and condition arcs are visualized using arcs of a special shape, as shown in Figure 1 and Figure 2.

Let x be a place or a transition: ${}^\bullet x$ is the set of transitions (places) connected with x by an arc ingoing to x, called *preset* of x. $x^\bullet$ is the set of transitions (places) connected with x by an arc outgoing from x, called *postset* of x. For a transition t, we denote in a similar fashion: ${}^+t$ is the set of places which are tested on presence of tokens by t (via a condition arc), called the *positive context* of t. Given a set $\xi \subseteq T$ of transitions, we extend the above notations to ${}^\bullet\xi$, $\xi^\bullet$ and ${}^+\xi$ via the union of sets.

A transition t is *enabled* at a marking m if all places in ${}^\bullet t$ and ${}^+t$ are marked and the places in $t^\bullet \setminus {}^\bullet t$ are unmarked at m.

A *(synchronized) step* of transitions is a set of transitions which can be constructed inductively in the following way: For each spontaneous transition t the set $\{t\}$ is a step. If ξ is a step, t is an OR-synchronized transition not in ξ and ξ contains at least one synchronizing transition of t, then $\xi \cup \{t\}$ is a step. If ξ is a step, t is an AND-synchronized transition not in ξ and ξ contains all synchronizing transition of t, then $\xi \cup \{t\}$ is a step.

A step ξ is *potentially enabled* at a marking m if

- all places in ${}^\bullet\xi$ and ${}^+\xi$ are marked at m,
- the places in $\xi^\bullet \setminus ({}^\bullet\xi \cup {}^+\xi)$ are unmarked at m, and
- all transitions $t_1, t_2 \in \xi$ are not in conflict w.r.t. to their pre- or postsets, i.e. ${}^\bullet t_1 \cap {}^\bullet t_2 = \emptyset$ and $t_1^\bullet \cap t_2^\bullet = \emptyset$.

From all steps potentially enabled at a marking m only those are *enabled* which are maximal with this property.

The *occurrence* of an enabled step ξ removes a token from each place of the preset of ξ and adds a token to each place of the postset of ξ. A sequence of steps which are enabled subsequently from the initial marking is called an *occurrence sequence*.

We add to a signal net an *input/output structure*. This structure consists of sets of event signal inputs and outputs, condition signal inputs and outputs, and arcs connecting these inputs and outputs with places and transitions of the signal net. The event signal inputs and outputs are connected via event arcs with transitions of the signal net. The condition signal inputs are connected with transitions of the signal net via condition arcs. The condition signal outputs are connected with places of the signal net via condition arcs. For the condition signal inputs, their initial states are fixed (either in *on*- or *off-state*). A signal net together with such an input/output structure defines a *module of a signal net* (see Figure 3).

We extend the notions of preset, postset and positive context to the added event and condition signal inputs and outputs in the obvious way.

Two modules A and B can be composed by identifying event resp. condition inputs of module A one by one with event resp. condition outputs of module B, and vice versa, employing a composition mapping Ω (see Figure 4). The identification of inputs and outputs via Ω is required to satisfy the following properties:

- A place of A connected to a transition of B via a condition signal output of A which is identified with a condition signal input of B is initially marked if and only if the condition signal input is in on-state, and vice versa.
- No cycles of event arcs are generated.

The connections of places and transitions of one module to places and transitions of the other module via identified inputs and outputs are replaced by direct signal arcs (see Figure 5). The *composition of A and B w.r.t. Ω* is denoted by $A *_\Omega B$.

We are interested in the behavior of a module of a signal net A w.r.t. a given environment: Transitions connected by an event signal input to the environment are not able to occur spontaneously but need to be synchronized by the event input in order to occur. Similar, a transition connected by a condition signal input to the environment is only able to occur if the condition signal input is in on-state. In the most general case, this

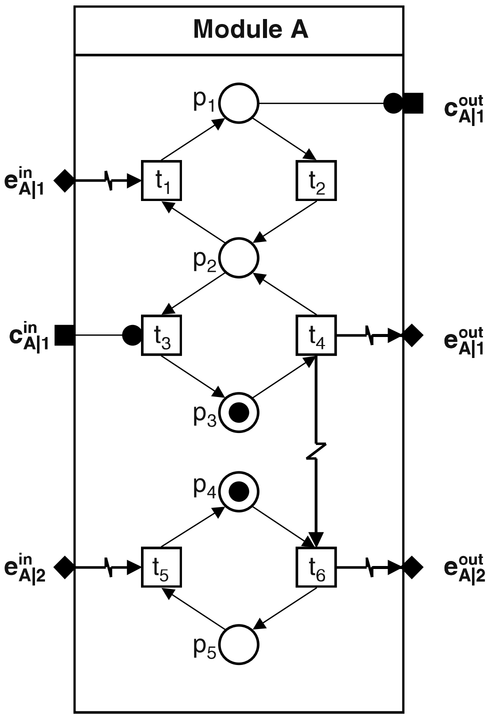

Fig. 3. A module of a signal net with condition inputs $C^{in} = \{c^{in}_{A|1}\}$, event inputs $E^{in} = \{e^{in}_{A|1}, e^{in}_{A|2}\}$, condition outputs $C^{out} = \{c^{out}_{A|1}\}$ and event outputs $E^{out} = \{e^{out}_{A|1}, e^{out}_{A|2}\}$.

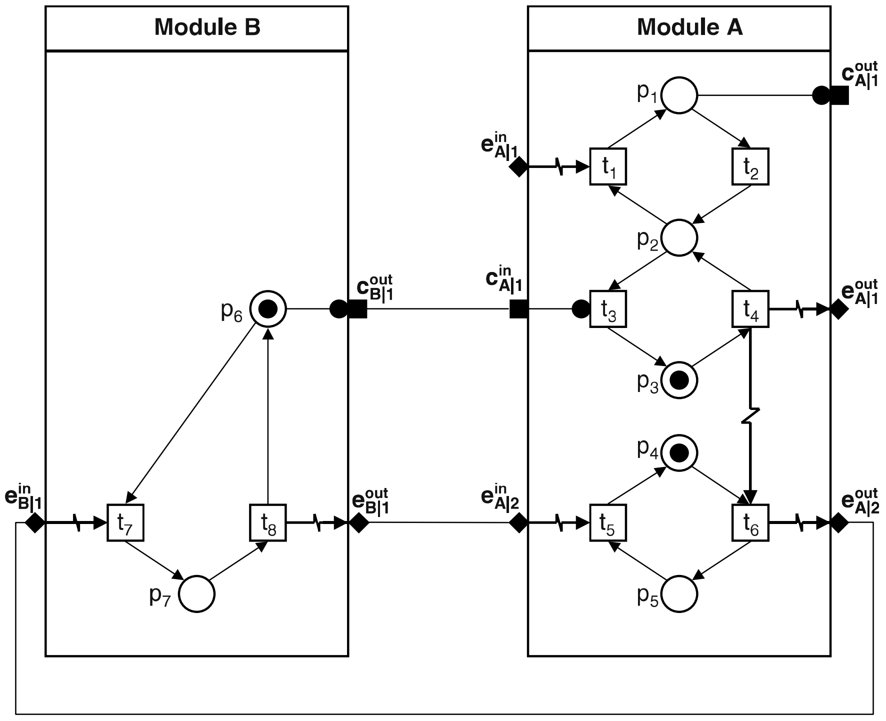

Fig. 4. The composition of two modules.

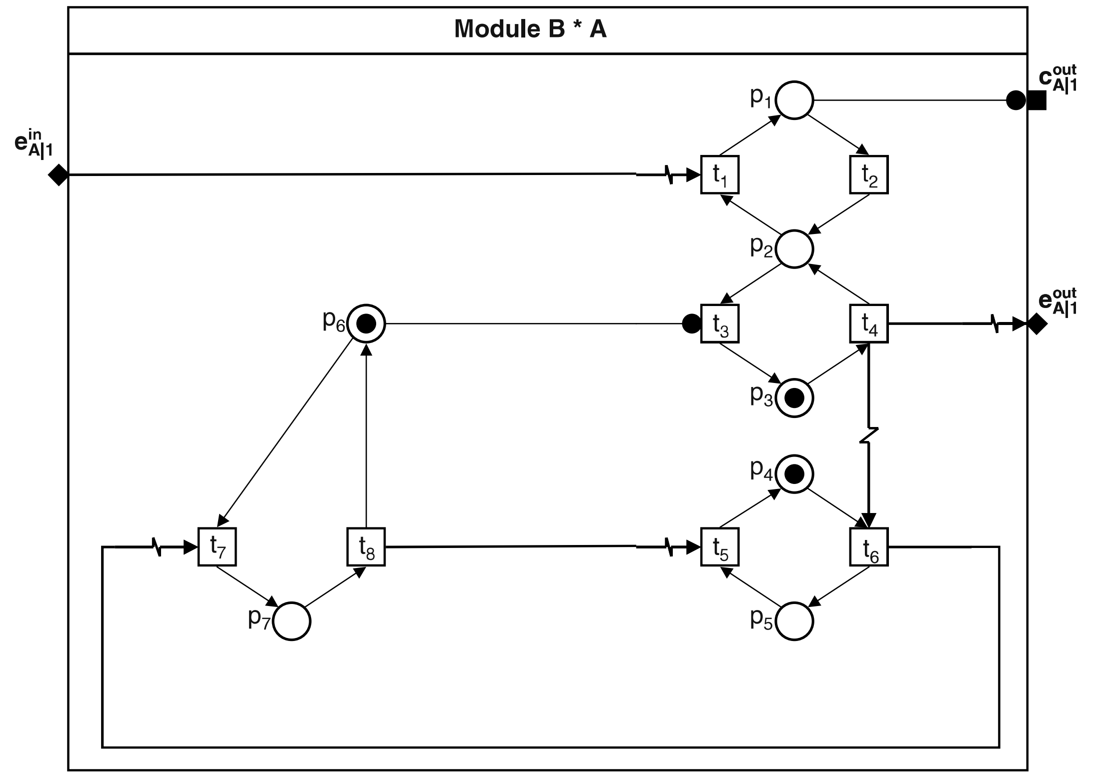

Fig. 5. The result of the composition of the modules from Figure 4.

environment is assumed to be maximally permissive in the sense that there are no causal dependencies between sending event signal inputs and switching on and off condition signal inputs. We model such an environment as a module $\mathcal{E}$ of a signal net and then compose the environment module appropriately with the module A such that $\mathcal{E}$ realizes a *maximally permissive environment* in the following sense (see Figure 6):

– At any moment, $\mathcal{E}$ can send event signal inputs to A: each event signal input of A is synchronized by a corresponding so called *input transition* in $\mathcal{E}$ that is always enabled;
– at any moment, $\mathcal{E}$ can send condition signal inputs to A: Each condition input of M is switched on resp. off by marking resp. unmarking a corresponding so called *input place* in $\mathcal{E}$;
– $\mathcal{E}$ can observe signal outputs of A: Every event signal output of A synchronizes a corresponding so called *output transition* in $\mathcal{E}$ that is always enabled. Every condition signal output enables resp. disables a corresponding *output transition* in $\mathcal{E}$ that is always enabled.

By this definition, in $\mathcal{E}$ no synchronization between its transitions is allowed. In particular, input signals can not be sent in steps from $\mathcal{E}$ to A, and output signals of A can only be observed by $\mathcal{E}$ and not synchronize input signals of A via $\mathcal{E}$.

The assumption that the environment (which later becomes the controller) changes at most one of its inputs to the plant at each moment does not always hold in practice. A controller might change more than one input to the plant within one cyclic run of its control program. In such a case, an environment enabling steps of input signals has to be

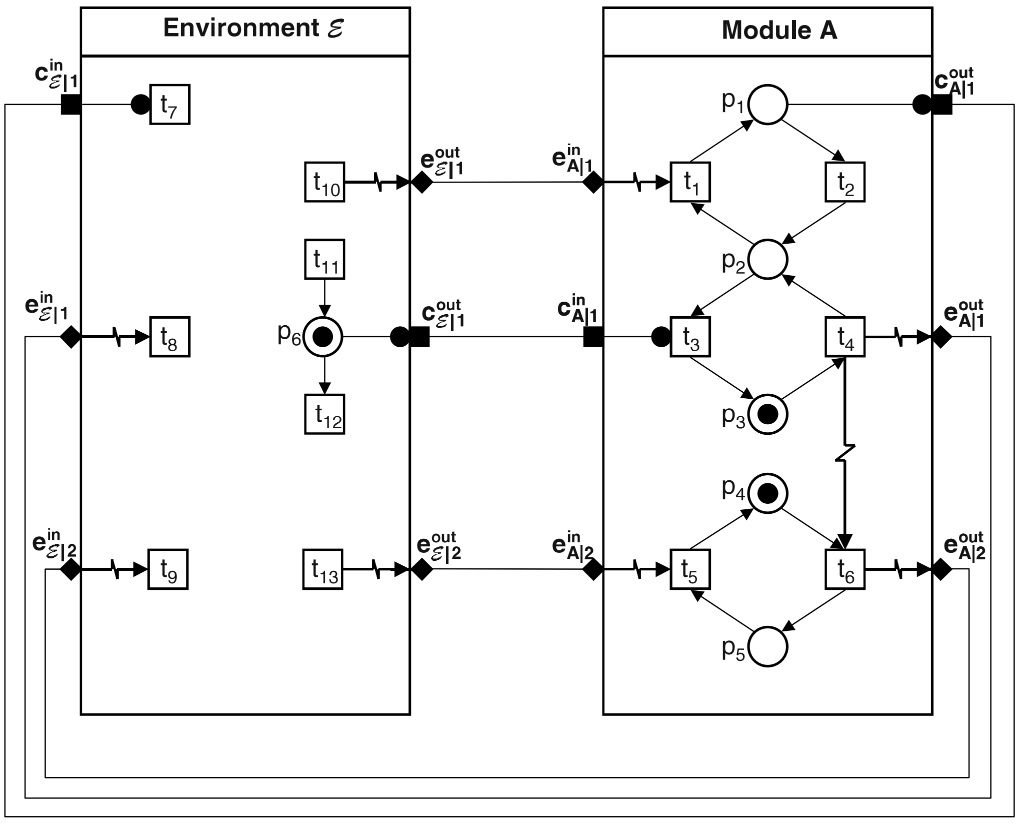
Environment $\mathcal{E}$
Module A
$c^{in}_{\mathcal{E}|1}$
$e^{in}_{\mathcal{E}|1}$
$e^{in}_{\mathcal{E}|2}$
$e^{out}_{\mathcal{E}|1}$
$c^{out}_{\mathcal{E}|1}$
$e^{out}_{\mathcal{E}|2}$
$e^{in}_{A|1}$
$c^{in}_{A|1}$
$e^{in}_{A|2}$
$c^{out}_{A|1}$
$e^{out}_{A|1}$
$e^{out}_{A|2}$
t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10} t_{11} t_{12} t_{13}
p_1 p_2 p_3 p_4 p_5 p_6

Fig. 6. The composition of the module in Figure 3 with its maximally permissive environment module.

considered. It is straightforward to modify all synthesis algorithms presented in the next part of this paper in order to deal with an environment allowing such steps of inputs.

The composition of A with its maximally permissive environment $\mathcal{E}$ is called the *standalone of* A. This composition has empty input/output structure. As an example, see Figure 7. The standalone is a model representing the uncontrolled behavior of the plant. The set L_A of all occurrence sequences of the standalone of A is called the *behavior of* A.

Since the underlying Petri net is assumed to be 1-safe, the behavior L_A of A is a regular language and can be represented as a finite automaton.

In [17] we introduced the *input/output behavior* of A as the set of all occurrence sequences of the standalone of A in which the transitions of A are hidden. Further, we defined two modules to be *input/output equivalent* if they have the same input/output structure and identical input/output behavior. By defining an appropriate composition operation for standalones, we showed that input/output equivalence is preserved by the composition of modules. This is a crucial concept for hierarchical modelling which allows to replace a module by a more abstract/concrete module with the same "input/output" functionality. Moreover, using the composition operation for standalones, it can easily be seen that the input/output behavior of the composition of two modules A and B can be represented by the composition of the standalones of A and B.

Summarizing, modules of signal nets are an extension of Petri nets supporting input/output structures, modularity and compositionality in an intuitive graphical way with precise syntax and semantics. This fact gives a motivation for a more detailed theoretical

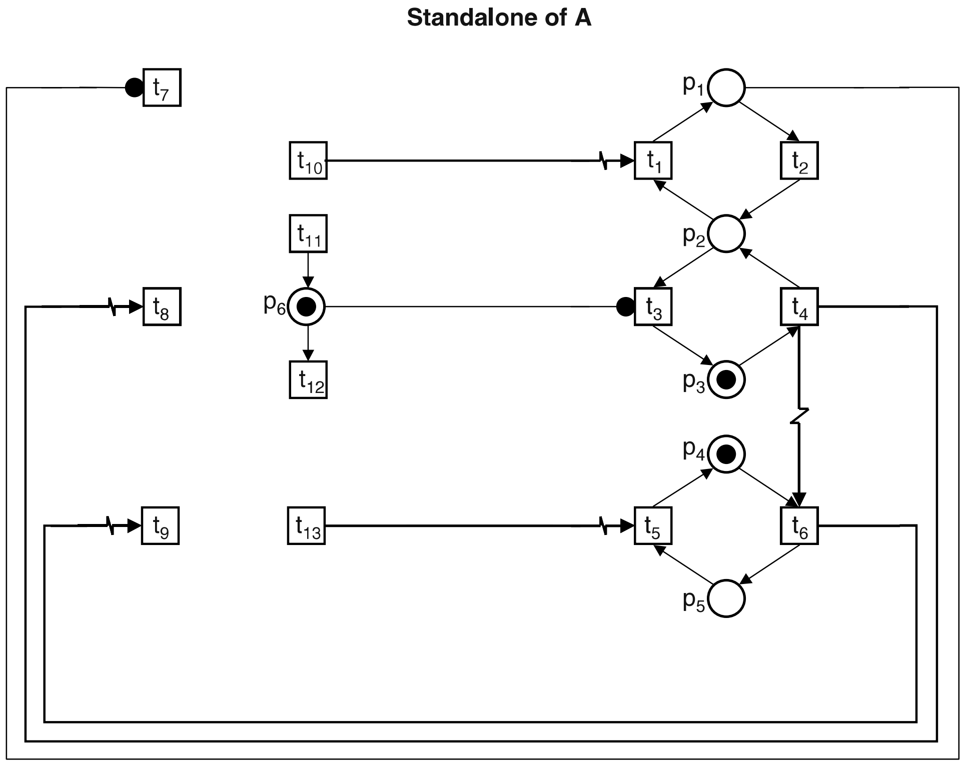

Fig. 7. The standalone of the module of a signal net in Figure 3.

investigation of this extension of Petri nets. In the following part of the paper we discuss the role of both kinds of signals in control tasks and we focus on control aspects in general.

Part II

3 Controller Synthesis

As mentioned in the introduction, the aim in control synthesis of DES is to influence the behavior of a system by a control via passive and active signals in order to get a specified desired behavior. In principle, there are two possibilities to specify a desired behavior (see [2] for an actual survey, and [1,28] for recent developments):

- The event-based approach used in the seminal work of Ramadge and Wonham on supervisory control of DES [23]. In this framework, automata are used to model the behavior of the plant and of the control. The desired behavior is given by legal sequences of events.
- The state-based approach [13], where Petri nets are used to model plant and control. The desired behavior is derived from a set of legal resp. forbidden states.

In both approaches, the main problem is that the considered modelling formalism (languages, automata, Petri nets) does not provide a straightforward mechanism for enforcing events in the plant.

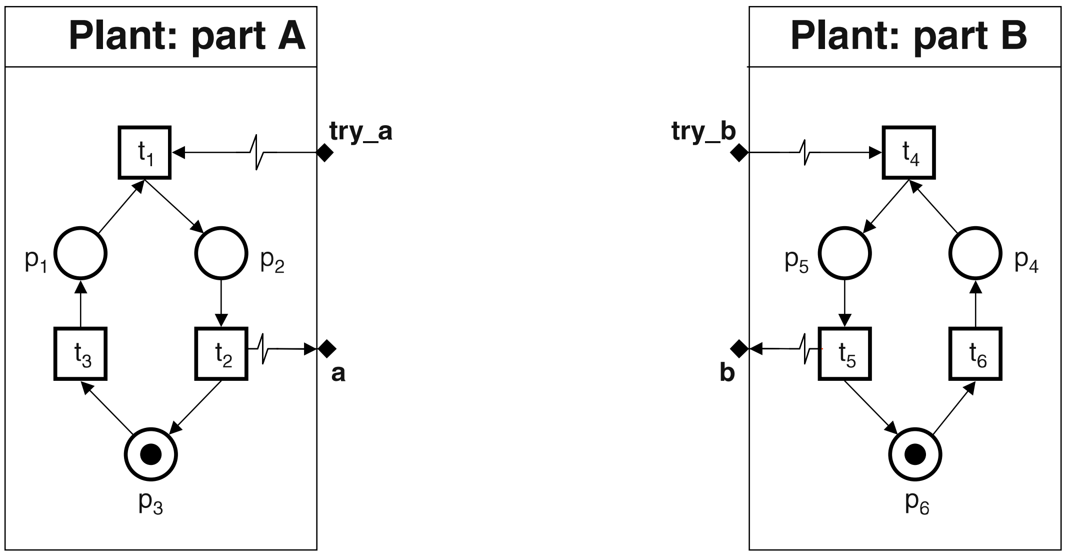

Fig. 8. Model of the plant.

In classical supervisory control, this problem is solved by modelling the enforcing of an event in the plant via prohibiting all other possible events [6]. As a consequence, the behavior of the plant cannot be forced by the control, now called supervisor, but only be restricted. Formally, a regular prefix closed language over a set of events representing the uncontrolled behavior of the plant and a regular subset of this language representing the restricted desired behavior is given. In the most general case, one distinguishes between controllable events (which can be prohibited by the supervisor) and uncontrollable events, and between observable events (which can be observed by the supervisor) and unobservable events. The question is, which controllable events should be prohibited by the supervisor after observing a certain sequence of observable events in order to disable all undesired behavior in a *minimal restrictive way*.

We present an alternative to the existing approaches to control of DES, employing *direct enforcing of events* in our models. The aim of control is to *maximally force* the behavior of the plant in order to ensure the specified desired behavior. Our formalism is suitable for both kinds of specifications of the desired behavior. In the literature, the event-based approach is further developed than the state-based approach in the sense that it allows more general specifications [30]. Therefore, we concentrate in this paper on an event-based specification of the desired behavior.

As an illustrative running example, consider the model of a plant consisting of two modules A and B given in Figure 8. The modules are independent. Each module models a task. The behavior of a module is cyclic and consists of three events. The event t_1 (resp. t_4) occurs if it is synchronized by the event signal try_a (resp. try_b) and if the module is ready to start (place p_1 (resp. p_4) is marked). The event t_2 (resp. t_5), which represents that the task is finished, is spontaneous but can be observed through occurrence of the event signal a (resp. b). Finally, the event t_3 (resp. t_6), which initializes the module to be ready to start, is spontaneous and unobservable.

In the figures we draw the two modules separately but we understand them as one module, obtained by a composition with an empty composition mapping. In other words,

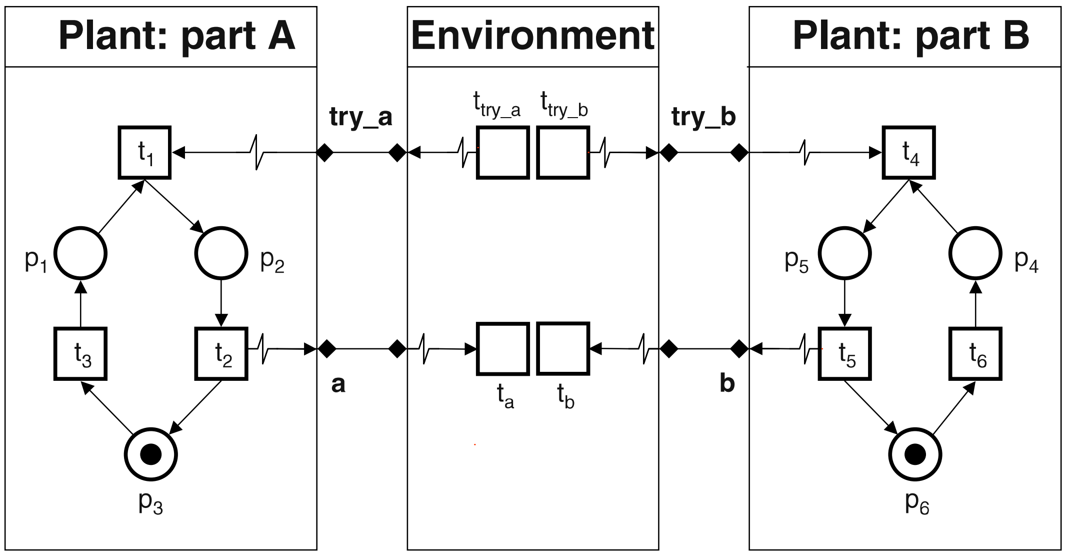

Fig. 9. Model of the plant together with its environment.

we consider the module obtained by putting the both modules side-by-side without connections between their input/output signals. We call this module $\mathcal{P}$.

The aim of the control specification is to coordinate the behavior of both modules in order to ensure the following behavior: Both tasks alternate strictly, starting with the task of module A and ending with the task of module B. More precisely, since only output behavior is relevant, the aim is to observe desired sequences of event output signals. The set of desired sequences is given by the regular language $L_c = (ab)^*$. In general, the desired behavior is given by a regular language which involves event signal inputs. Moreover, since the event arc relation produces a semantics of (synchronized) steps of transitions, the desired behavior must be specified as a regular language over steps of event signal inputs and outputs.

Let $T = \{t_1, \dots, t_6\}$ be the set of transitions and $P = \{p_1, \dots, p_6\}$ be the set of places of the plant. To denote a marking in figures, we use the vector-like notation, which is more usual in control literature.

Consider the maximally permissive environment $\mathcal{E}$ of $\mathcal{P}$, shown in Figure 9, and the standalone of $\mathcal{P}$, shown in Figure 10, which is a composition of $\mathcal{P}$ and $\mathcal{E}$ w.r.t. to a composition mapping Ω. We denote by $I = \{t_{try\_a}, t_{try\_b}\}$ the set of input transitions of $\mathcal{E}$ corresponding to the event signal inputs try_a, try_b and by $O = \{t_a, t_b\}$ the set of output transitions of $\mathcal{E}$ corresponding to the event signal outputs a, b. The behavior $L_{\mathcal{P}}$ of $\mathcal{P}$ is given as the set of all occurrence sequences of the standalone of $\mathcal{P}$. It can be represented by the finite automaton shown in Figure 11. In order to be able to compare $L_{\mathcal{P}}$ with the specification $(ab)^*$, we restate the specification in the form $L_c = (t_a t_b)^*$, replacing event signal outputs by the corresponding output transitions in $\mathcal{E}$.

A sublanguage K of $L_{\mathcal{P}}$ *satisfies the specification* if each occurrence sequence of K is a prefix of an occurrence sequence of K whose projection onto the set O of output transitions of $\mathcal{E}$ is a string in L_c. In particular, each projection of an occurrence sequence of K onto O is required to be a prefix of a string in L_c. The above condition implies that K represents a nonblocking behavior where the tasks specified by L_c always can

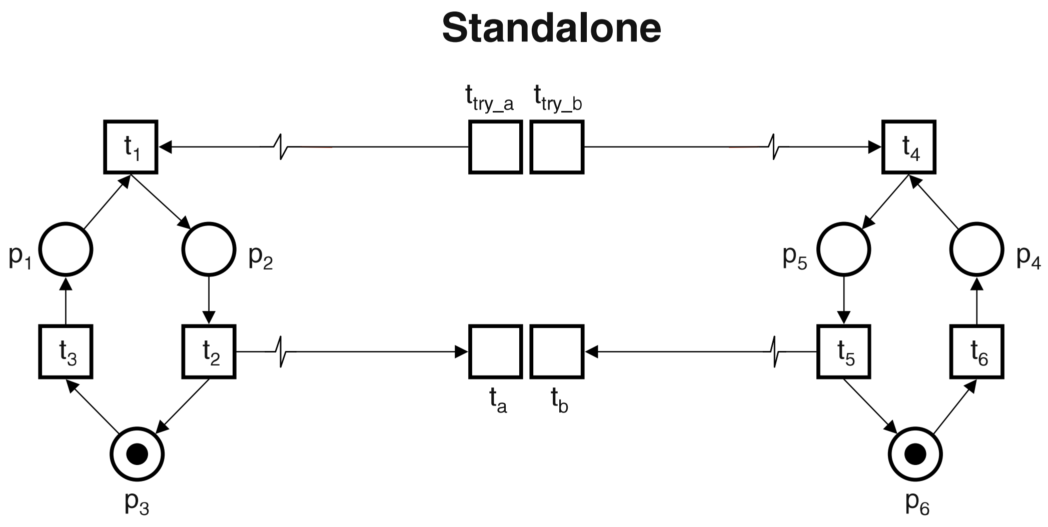

Fig. 10. Standalone of the plant.

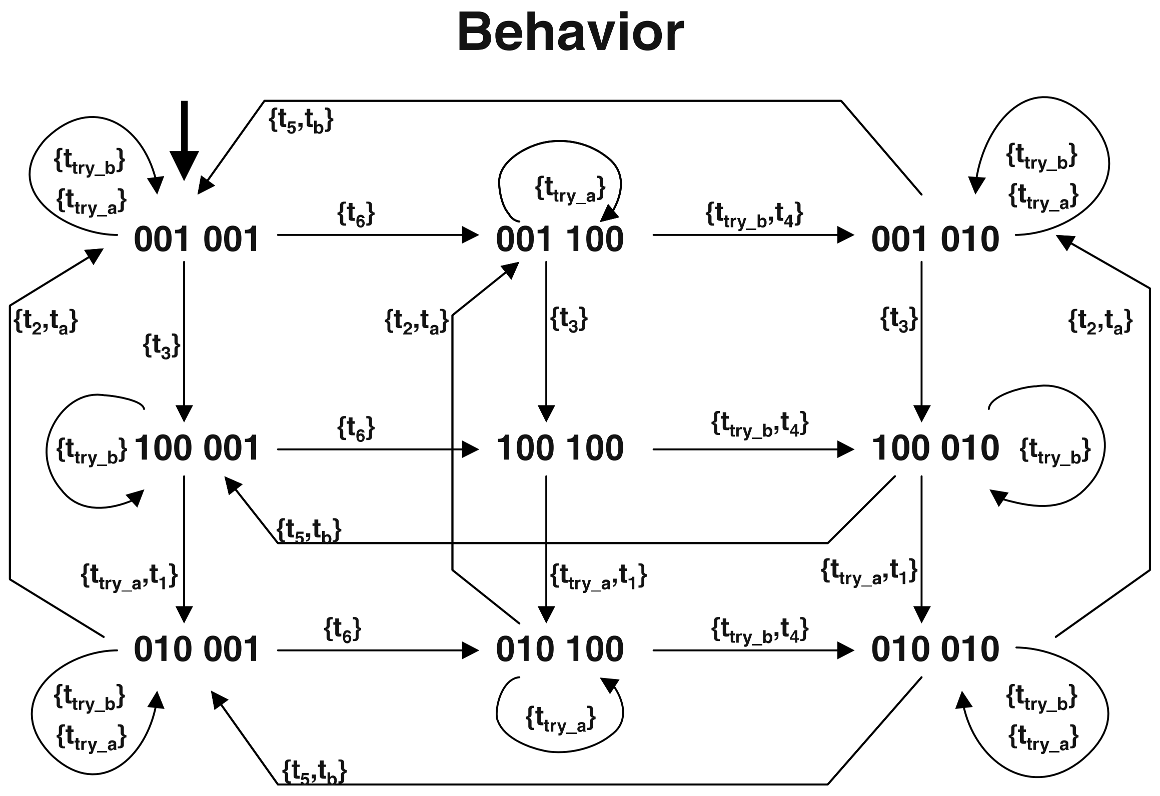

Fig. 11. Behavior of the standalone of the plant.

be completed. If $L_{\mathcal{P}}$ already satisfies the specification, $\mathcal{E}$ is the desired control module $\mathcal{C}$. Therefore, $\mathcal{E}$ can be seen as a first approximation of $\mathcal{C}$.

In our example, the projection of the occurrence sequence $\{t_6\}\{t_{try\_b}, t_4\}\{t_5, t_b\}$ onto O yields the string $\{t_b\}$, which is not a prefix of a word in $L_c = (t_a t_b)^*$. In such cases, the aim is to add new net elements to $\mathcal{E}$, thus defining causal dependencies between input and output transitions of $\mathcal{E}$ to prohibit such undesired occurrence sequences.

To summarize, the aim is to construct from $\mathcal{E}$ a control module $\mathcal{C}$ which composed with $\mathcal{P}$ by the same composition mapping Ω defining the composition of $\mathcal{P}$ and $\mathcal{E}$ satisfies: Each occurrence sequence of the underlying signal net of $\mathcal{C} *_{\Omega} \mathcal{P}$ respects the desired behavior in the sense that this occurrence sequence can be completed to another occurrence sequence of this underlying signal net whose projection onto (input and) output transitions of $\mathcal{C}$, which replace event signal (inputs and) outputs of $\mathcal{P}$, is a string of the desired behavior.

We synthesize $\mathcal{C}$ in two steps. First, we define conditions of *controllability* of a regular sublanguage of $L_{\mathcal{P}}$ (analogously to [23]) and compute the maximally permissive controllable nonblocking subbehavior L_{nbsafe} of $L_{\mathcal{P}}$ satisfying L_c as a finite automaton. This is done by manipulating regular languages in subsection 3.1. Second, we synthesize a signal net simulating the control structure given by this automaton and add this signal net to $\mathcal{E}$. By this signal net the input and output transitions of $\mathcal{E}$ are coordinated in such a way that L_{nbsafe} is realized.

3.1 The Behavior of the Controlled Plant

We formulate our approach similar as it is done in classical supervisory control. The main technical differences to the classical supervisory control approach are due to the mentioned step semantics. Nevertheless, some algorithms of classical supervisory control can at least be adapted to our framework. While omitting therefore most details of these algorithms, our paper remains self contained, i.e., it can be understood without previous knowledge of supervisory control.

As mentioned in the last section, our aim is to compute a regular sublanguage L_{nbsafe} of $L_{\mathcal{P}}$ representing the maximally permissive controllable nonblocking subbehavior of $L_{\mathcal{P}}$ satisfying the specification. Roughly speaking, controllable means that L_{nbsafe} can be realized by a control.

The computation of L_{nbsafe} is done in several steps by manipulating regular languages. For this we need a special projection operator λ_Y. Applying λ_Y to a language over an alphabet of the form 2^X means to project each word in this language onto $2^{X \setminus Y}$. In [18,19] we showed that projection operations of the form λ_Y and pumping operations of the form λ_Y^{-1} preserve the regularity of languages.

“Good" occurrence sequences. In a first step we delete all occurrence sequences w from $L_{\mathcal{P}}$ satisfying $\lambda_{I \cup T}(w) \notin \overline{L_c}$, i.e. whose projections onto output transitions are not prefixes of L_c ($\overline{L_c}$ denotes the prefix closure of L_c). We call such occurrence sequences *"bad" occurrence sequences* and the remaining ones *"good" occurrence sequences*.

For example, $w = \{t_3\}\{t_{try\_a}, t_1\}\{t_2, t_a\}$ is a good occurrence sequence since $\lambda_{I \cup T}(w) = \{t_a\} \in \overline{L_c}$, and $v = \{t_3\}\{t_{try\_a}, t_1\}\{t_2, t_a\}\{t_3\}\{t_{try\_a}, t_1\}\{t_2, t_a\}$ is a bad occurrence sequence since $\lambda_{I \cup T}(v) = \{t_a\}\{t_a\} \notin \overline{L_c}$.

The set of all good occurrence sequences is denoted by L_{psafe}. It can formally be computed by

$$L_{psafe} = \lambda_{I \cup T}^{-1}(\overline{L_c}) \cap L_{\mathcal{P}},$$

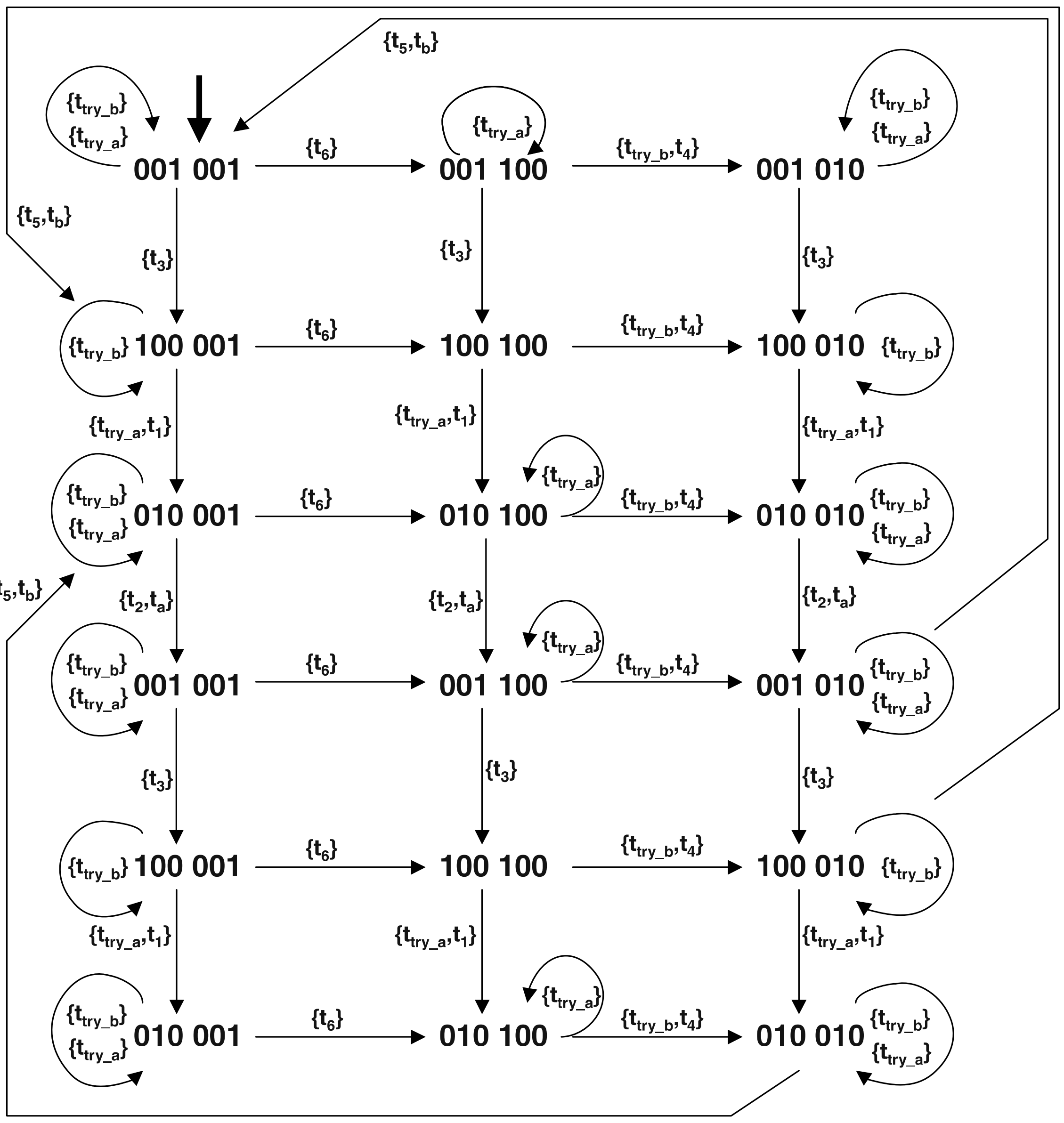

Fig. 12. Automaton recognizing the language L_{psafe}.

and is therefore regular. For our running example, Figure 12 shows a representation of L_{psafe} as a finite automaton.

Controllable occurrence sequences. In L_{psafe} there may exist good occurrence sequences which can be extended within $L_{\mathcal{P}}$ to bad occurrence sequences. For example, the good occurrence sequence $w = \{t_6\}\{t_{try\_b}, t_4\}$ can be extended by the step $\{t_5, t_b\}$ to the bad occurrence sequence $v = w\{t_5, t_b\} = \{t_6\}\{t_{try\_b}, t_4\}\{t_5, t_b\}$. If this extension is not controllable, i.e. contains no controllable transition, it can not be avoided by the control. In our example, once w is allowed, v can not be avoided: The step $\{t_5, t_b\}$ is not controllable, but can only be observed by the control. Therefore, we require L_{nbsafe} to satisfy the property

(1) *What cannot be prevented, should be legal*

which we call the *first condition of controllability*. It corresponds to the classical notion of *controllability* in supervisory control.

Good occurrence sequences which can be extended by uncontrollable steps to bad occurrence sequences are called *dangerous occurrence sequences*. They must be cut off at the last possibility of control, i.e. the last possible signal input (if there is one). In our example, after the step $\{t_6\}$, sending a signal input via occurrence of the transition t_{try_b} should be forbidden, so w is cut after $\{t_6\}$. By this, all dangerous occurrence sequences ending with a step containing an input and their futures are deleted from L_{psafe}.

Due to unobservable transitions, there may remain good occurrence sequences which cannot be distinguished by the control from dangerous occurrence sequences deleted in the last computation step. For example, the control can not decide whether the occurrence sequence $\{t_6\}$ has occurred or not since t_6 is an unobservable transition. Therefore, forbidding the occurrence sequence $\{t_6\}\{t_{try\_b}, t_4\}$ means to forbid also the occurrence sequence $\{t_{try\_b}\}$. This follows the rule

(2) *What cannot be distinguished, cannot call for different control actions*,

called the *second condition of controllability*. It corresponds to the notion of *observability* in supervisory control.

In our example, since the signal input $\{t_{try\_b}\}$ is forbidden after the occurrence of $\{t_6\}$ according to the last computation step, it must also be forbidden from the initial marking (after the empty occurrence sequence). The cutting off of appropriate inputs can be represented by deleting corresponding edges in the automaton representing L_{psafe}, see Figure 13.

Deleting all dangerous occurrence sequences and all occurrence sequences which are undistinguishable to a dangerous occurrence sequence L_{psafe} gives the language L_{safe}.

L_{safe} again can be expressed by a closed formula over regular languages, and is therefore regular. Moreover, it was proven to be the maximally permissive controllable sublanguage of $L_{\mathcal{P}}$ in [19]. For our running example, Figure 14 shows a representation of L_{safe} as a finite automaton.

In general, there can be an occurrence sequence w from $L_{\mathcal{P}}$ which contains no input transition and whose projection onto O is not a prefix of a string in L_c. In this case we cannot control the plant in such a way that no undesired behavior will happen. Thus, only if there is no such sequence, the maximally permissive controllable sublanguage of $L_{\mathcal{P}}$ exists.

Blocking occurrence sequences. By construction, every projection of an occurrence sequence in L_{safe} onto O is a prefix of a string in L_c. However, it might happen that there are occurrence sequences that cannot be completed within L_{safe} to an occurrence sequence whose projection onto O is in L_c, i.e. the desired behavior is blocked. We call such occurrence sequences *blocking*.

In our example, L_{safe} does not contain blocking occurrence sequences and therefore is already the searched language L_{nbsafe}. Therefore, we illustrate the existence of

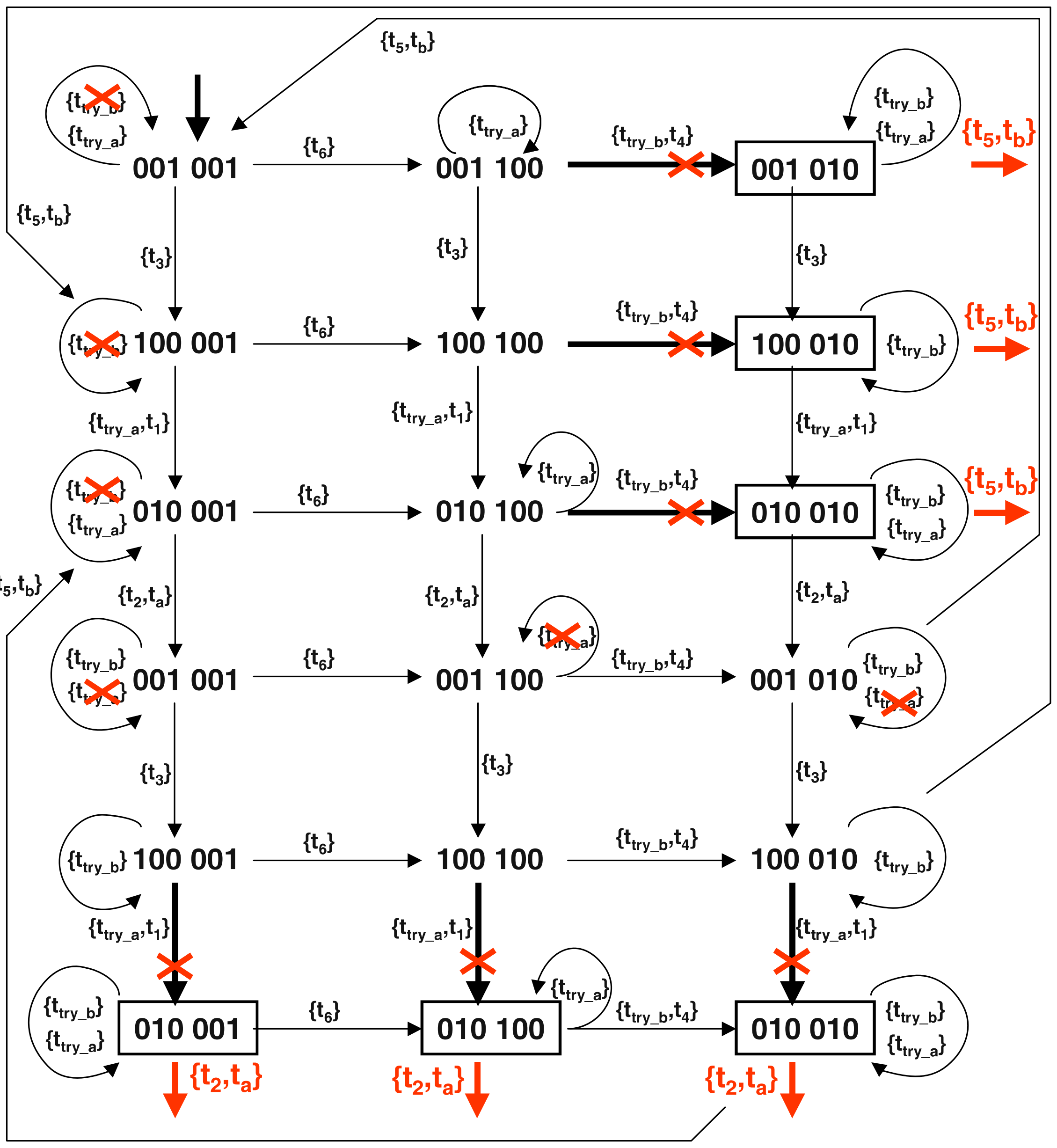

Fig. 13. Removing dangerous occurrence sequences and their future from the language L_{psafe}.

blocking occurrence sequences by another example of the standalone of a module of a plant $\mathcal{P}'$ given in Figure 15. The behavior $L_{\mathcal{P}'}$ of $\mathcal{P}'$ is given by the automaton in Figure 16. The automaton in Figure 17, also representing $L_{\mathcal{P}'}$, is more appropriate to illustrate the procedure of deleting blocking occurrence sequences.

The control specification is given again by the regular expression $(t_a t_b)^*$. All projections of occurrence sequences of the standalone onto $O = \{t_a, t_b\}$ are prefixes of strings in L_c. Thus, $L_{safe} = L_{\mathcal{P}'}$. However, the occurrence sequence

$$\{t_4\}\{t_{i\_1}, t_1\}\{t_5, t_7, t_a\}\{t_{i\_3}, t_3\}\{t_6, t_8, t_b\}\{t_5, t_9, t_a\}$$

is blocking.

Since every future of a blocking occurrence sequence is also blocking, we can delete blocking occurrence sequences by cutting them off at the last possible input (if there

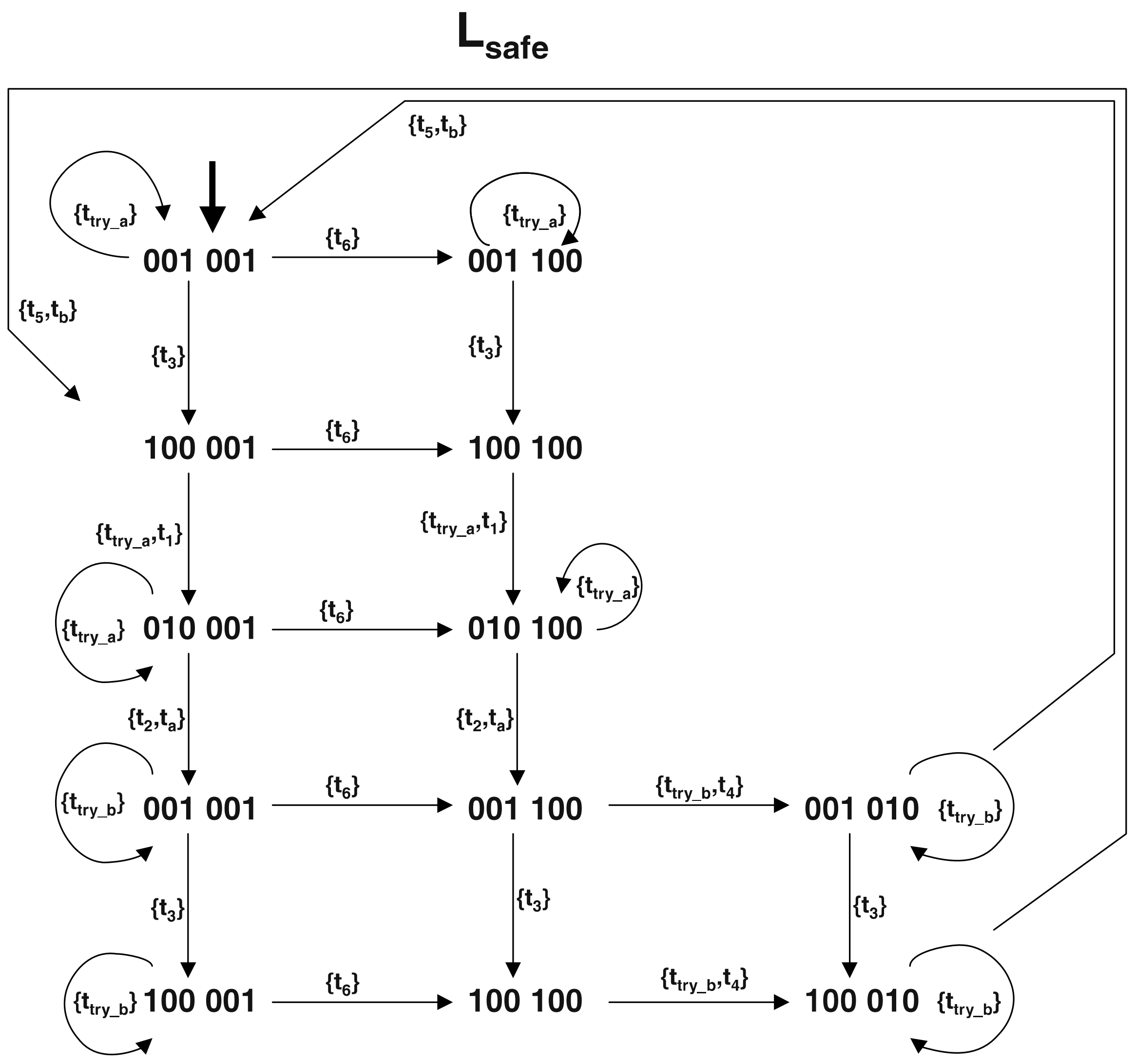

Fig. 14. Automaton recognizing the language L_{safe}.

is one). The prefixes ending with these inputs are called *real bad choices*. In the above example, the last possible input is $\{t_{i\_3}, t_3\}$ and

$$\{t_4\}\{t_{i\_1}, t_1\}\{t_5, t_7, t_a\}\{t_{i\_3}, t_3\}$$

is the corresponding real bad choice (see Figure 17).

Due to the second condition of controllability, such an input must be forbidden for all occurrence sequences which are undistinguishable to a real bad choice. Such occurrence sequences are called *bad choices*. In our example,

$$\{t_{i\_1}, t_1\}\{t_5, t_a\}\{t_{i\_3}, t_3\}$$

is a bad choice undistinguishable to the previous real bad choice (see Figure 17).

Deleting all bad choices and their futures from L_{safe} possibly produces new blocking occurrence sequences. For example, after deleting the real bad choice

$$\{t_4\}\{t_{i\_1}, t_1\}\{t_5, t_7, t_a\}\{t_{i\_3}, t_3\}$$

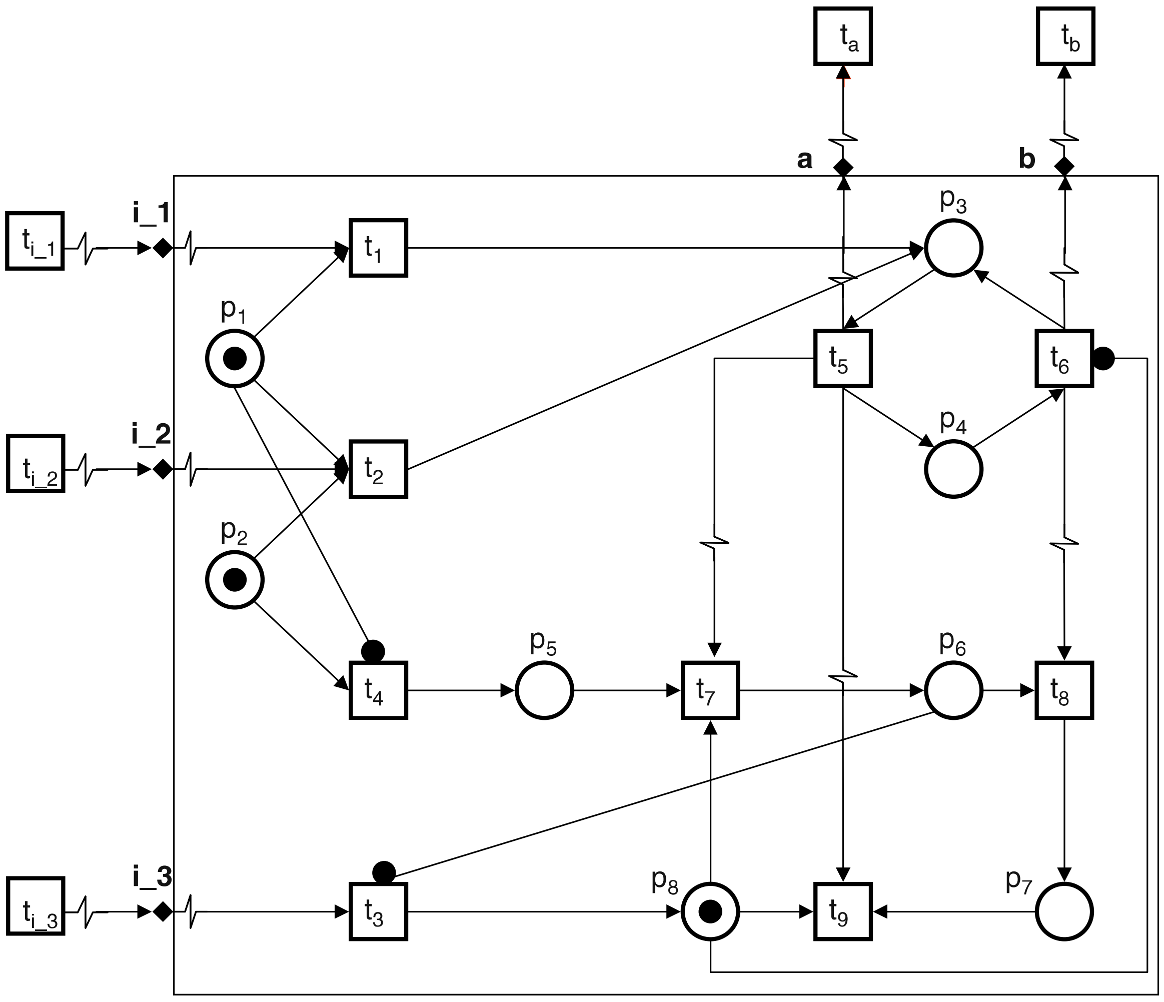

Fig. 15. Standalone of a module.

and its future produces the new blocking occurrence sequence

$$\{t_4\}\{t_{i\_1}, t_1\}\{t_5, t_7, t_a\}.$$

Therefore we have to iterate this procedure (see Figure 17).

To state the algorithm, we denote for any sublanguage K of L_{safe} by $K_{blocking}$ the set of all blocking words of K and by $K_{badchoice}$ the corresponding bad choices in K. We showed in [19] that if K is regular then also $K_{blocking}$ and $K_{badchoice}$ are regular, since they can then be expressed by a closed formula over regular languages. The following algorithm deletes subsequently all blocking words from L_{safe}:
Input: Language $K^0 = L_{safe}$, Integer $i = 0$.

Step 1:
Compute $K^i_{blocking}$.

Step 2:
If $K^i_{blocking}$ contains at least one word without any input **return** "L_{nbsafe} *does not*

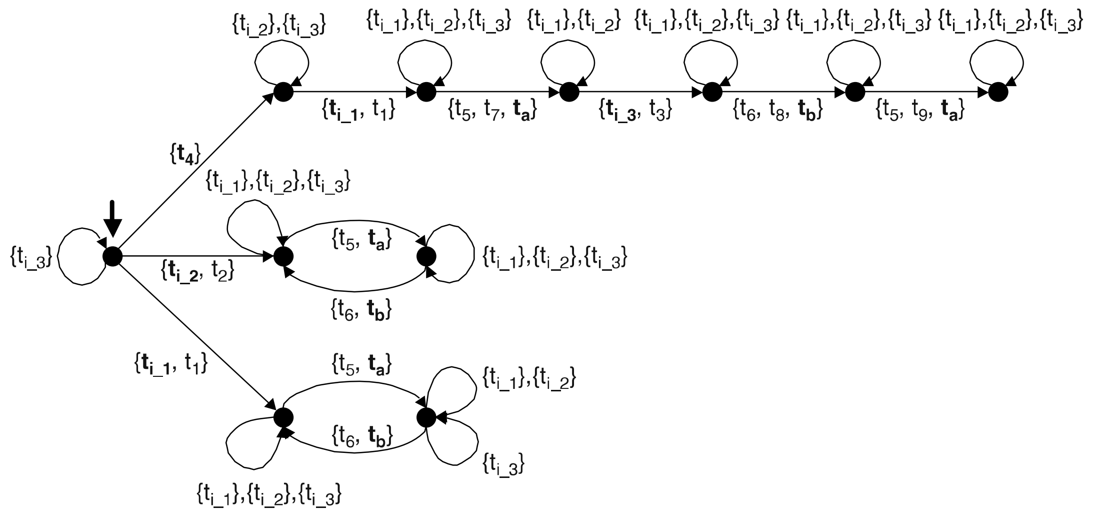

Fig. 16. Automaton representing the behavior of the standalone from Figure 15.

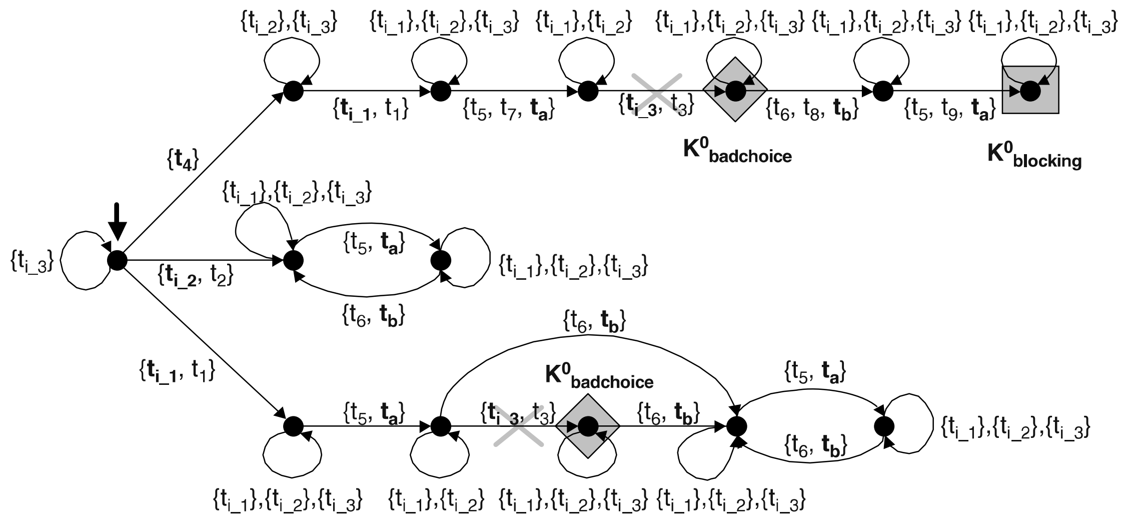

Fig. 17. Automaton representing the behavior of the standalone from Figure 15 which splits the bad choices from other words.

exist" (because there is no possibility to avoid this word by control).
If $K^i_{blocking}$ is empty **return** K^i.

Step 3:
Compute $K^i_{badchoice}$.
Set K^{i+1} by removing all words of $K^i_{badchoice}$ and their future from K^i.
Set $i = i + 1$.
Goto Step 1.

Starting with $K^0 = L_{safe}$ the algorithm iteratively deletes blocking occurrence sequences by cutting them off at the last possible inputs and by additionally deleting all undistinguishable occurrence sequences. This is done until either no new blocking

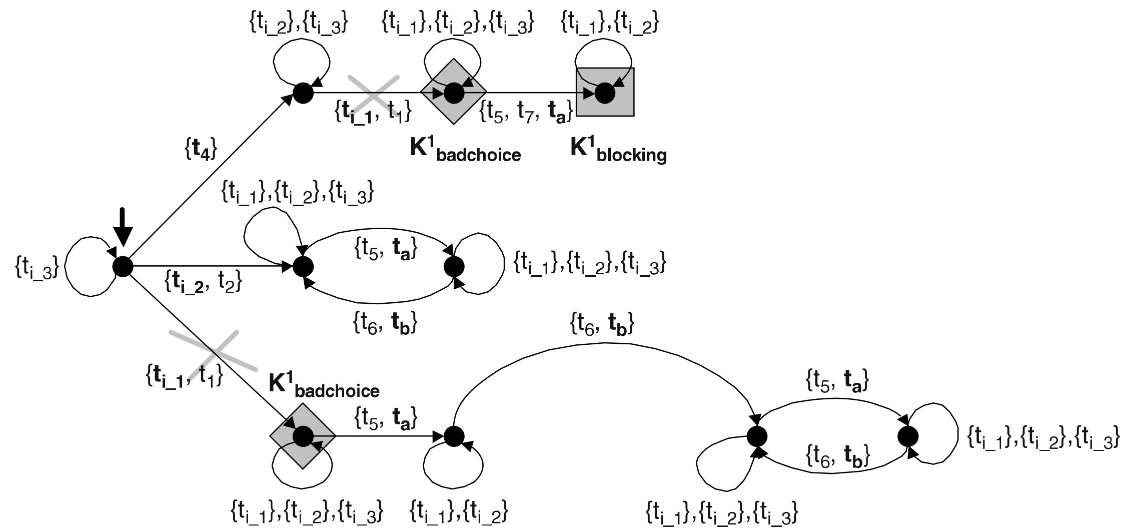

Fig. 18. Automaton obtained by deleting edges representing bad choices from automaton in Figure 17.

occurrence sequences are produced or there is a blocking occurrence sequence without any input in the actually computed language (in which case no controllable language without blocking occurrence sequences exists).

The algorithm returns a language if and only if the maximally permissive nonblocking controllable sublanguage exists, which we call L_{nbsafe}. A proof can be found in [19].

We briefly show that the algorithm always terminates. The main idea is to find a deterministic finite automaton G recognizing L_{safe} such that *deleting words* of $K^i_{badchoice}$ and their future corresponds to *deleting edges* in G. A necessary *and* sufficient condition for this is that the states of G distinguish words in $K^i_{badchoice}$ from words not in $K^i_{badchoice}$.

In general, not each finite automaton A recognizing L_{safe} satisfies this property. For example, the previously mentioned occurrence sequence

$$\{t_{i\_1}, t_1\}\{t_5, t_a\}\{t_{i\_3}, t_3\}$$

is a bad choice. However, in the automaton representing L_{safe} from Figure 16 the occurrence sequence

$$\{t_{i\_1}, t_1\}\{t_5, t_a\}\{t_6, t_b\}\{t_5, t_a\},$$

which is no bad choice in K^0, leads to the same state. In this case, deleting the corresponding edge would cut also nonblocking behavior off, i.e. it would cut too much.

In [19] we showed that such an automaton G always exists by an effective construction.

Figure 17 shows an automaton recognizing the same behavior as the automaton in Figure 16 which distinguishes between bad choices. In the automaton of Figure 17 the bad choice (of the language K^0)

$$\{t_{i\_1}, t_1\}\{t_5, t_a\}\{t_{i\_3}, t_3\}$$

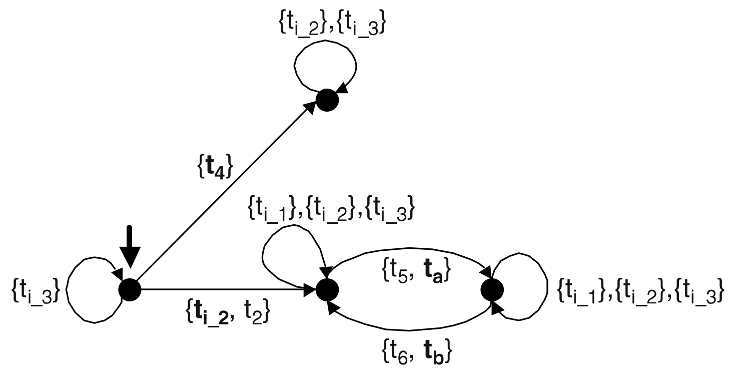

Fig. 19. Automaton obtained by deleting edges representing bad choices from automaton in Figure 18.

is now recognized by a different state as the occurrence sequence

$$\{t_{i\_1}, t_1\}\{t_5, t_a\}\{t_6, t_b\}\{t_5, t_a\}.$$

The application of the algorithm to the automaton in Figure 17 is illustrated in Figures 18 and 19. In the resulting automaton in Figure 19 the transition t_4, which can occur spontaneously, leads to a deadlock in the module. However, this is an "allowed" deadlock because the control specification "if a happens then b must happen too" is still satisfied.

3.2 Synthesis of Control Modules

In this subsection we show how to synthesize a control module $\mathcal{C}$ from the controllable subbehavior L_{nbsafe} (see Figure 14) of $L_{\mathcal{P}}$ (see Figure 11) by adding new net elements to the environment module $\mathcal{E}$ (see Figure 9). The composition of the resulting control module $\mathcal{C}$ with the plant module $\mathcal{P}$ has exactly the behavior L_{nbsafe}, in the following sense: The projection of the behavior of this composed module onto the set $I \cup O \cup T$ equals L_{nbsafe}.

To synthesize $\mathcal{C}$, we consider the observable part of L_{nbsafe}, i.e. its projection $\lambda_T(L_{nbsafe})$ onto input and output transitions. Figure 21 shows a possible automaton A recognizing $\lambda_T(L_{nbsafe})$. This automaton can be computed in two steps: First take the automaton recognizing L_{nbsafe} from Figure 14 and hide all transitions of T. The result is the nondeterministic ϵ-automaton shown in Figure 20 (ϵ is the empty word). This automaton can be transformed by a standard procedure into an equivalent deterministic automaton. The main idea for the construction of $\mathcal{C}$ is to simulate the control flow given by A by a signal net and to add this signal net to $\mathcal{E}$.

For an automaton with states drawn as filled circles and with labelled directed edges between states drawn as arrows with annotation, an edge leading from state s to state s' with label x is denoted by $s \xrightarrow{x} s'$.

In our example, it is quite straightforward to compute a signal net which simulates A: Every state s becomes a place p_s, and every edge $s \xrightarrow{x} s'$ becomes a transition $t_{s \xrightarrow{x} s'}$. In case $s \neq s'$, $t_{s \xrightarrow{x} s'}$ has the preset $\{p_s\}$ and the postset $\{p_{s'}\}$. In case $s = s'$, $t_{s \xrightarrow{x} s'}$ tests p_s to be marked by a condition arc. Initially the place corresponding to the initial

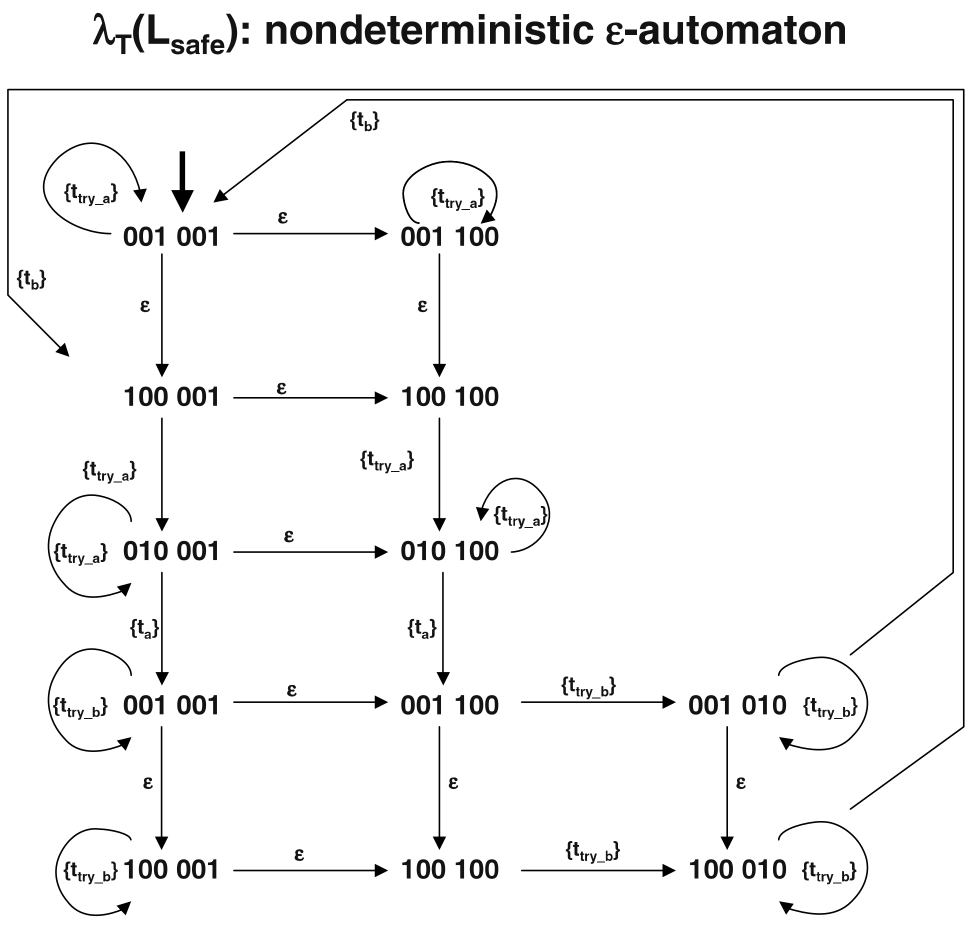

Fig. 20. The nondeterministic ϵ-automaton recognizing $\lambda_T(L_{safe})$ computed from the automaton recognizing L_{safe} from Figure 14.

state is marked. The procedure is illustrated in Figure 22. For convenience, we use a slightly simpler notation for the transitions. In case $x = \{t_{try\_a}\}$ we write $t_{to\_try\_a}$ for $t_{s \xrightarrow{x} s'}$. This illustrates the intended meaning of $t_{to\_try\_a}$ to send the input signal try_a to the plant. In case $x = \{t_a\}$ we write t_{from_a} for $t_{s \xrightarrow{x} s'}$. This shows the intended meaning of t_{from_a} to observe the output signal a sent from the plant. The labels $x = \{t_{try\_b}\}$ and $x = \{t_b\}$ are treated analogously. Each state s of the automaton corresponds to the marking in the signal net where exactly p_s is marked.

When the resulting signal net is connected with the input and output transitions of $\mathcal{C}$ inherited from $\mathcal{E}$, the control module shown in Figure 23 is obtained. Using the connections between the plant and its environment $\mathcal{E}$, we get the controlled composed module shown in Figure 24. The control module formalizes the intuition of how to send inputs in order to observe prefixes of the desired output behavior $(t_a t_b)^*$: First the input try_a is sent at least once. It is sent as long as one observes the output a. After that it is no longer possible to send try_a. Then try_b is sent at least once. This input is sent as long as one observes the output b. Finally the behavior starts from the beginning.

In general, the automaton A is more complicated because the arc labels are steps consisting of more than one transition. Therefore, the construction of $\mathcal{C}$ is more sophisticated. In [19] the construction of $\mathcal{C}$ for general automata A is presented. This paper

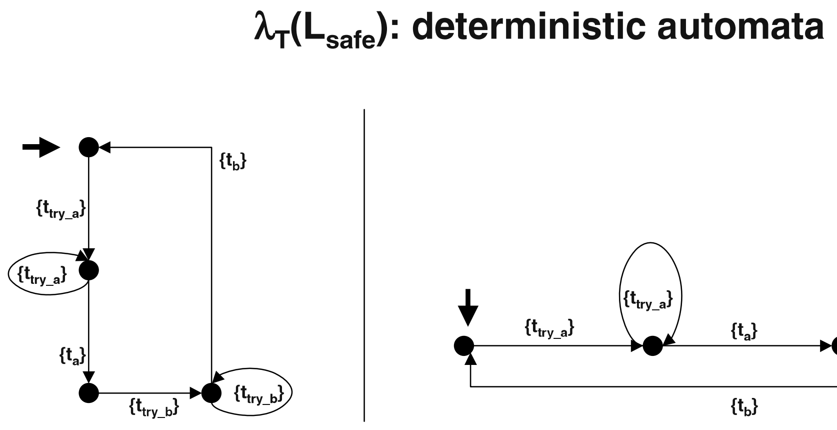

Fig. 21. The deterministic automaton recognizing $\lambda_T(L_{safe})$ computed from the nondeterministic ϵ-automaton from Figure 20. It is drawn in two different forms: The left Figure corresponds to the shape of the automaton recognizing L_{psafe} from Figure 12. The right Figure is convenient to deduce the signal net representing the control flow.

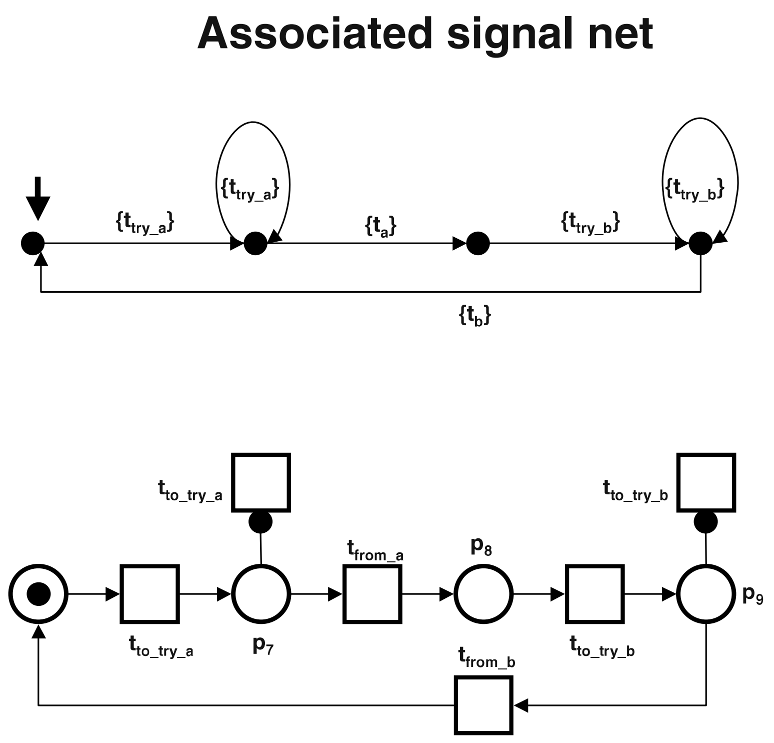

Fig. 22. The signal net simulating the control flow given by the automaton shown above.

contains a proof that the projection of the behavior of the composition of $\mathcal{P}$ and $\mathcal{C}$ onto the set $I \cup O \cup T$ equals L_{nbsafe}. Since the general construction is very involved, we only give some examples for possible problems and their solutions.

Basically, each state s of A can be modelled as a place p_s, and each labelled edge $s \xrightarrow{x} s'$ of A as an AND-synchronized transition $t_{s \xrightarrow{x} s'}$. In the case $x \subseteq O$, the transition $t_{s \xrightarrow{x} s'}$ is intended to observe the step of outputs x. It is therefore synchronized by all

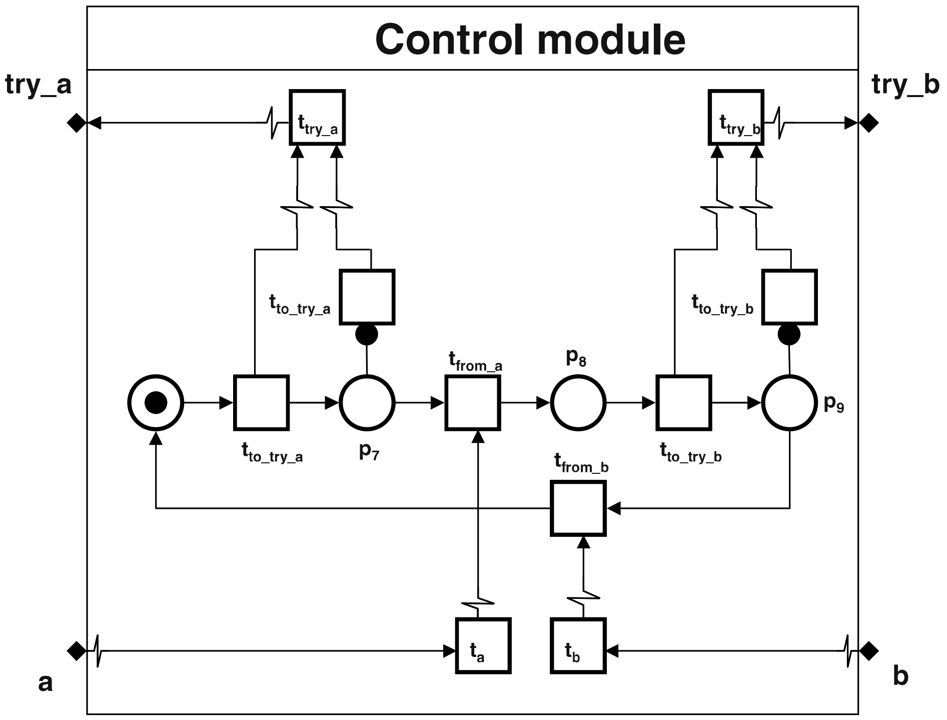

Fig. 23. The resulting control module given by the signal net from Figure 22 connected with the input and output transition from the environment.

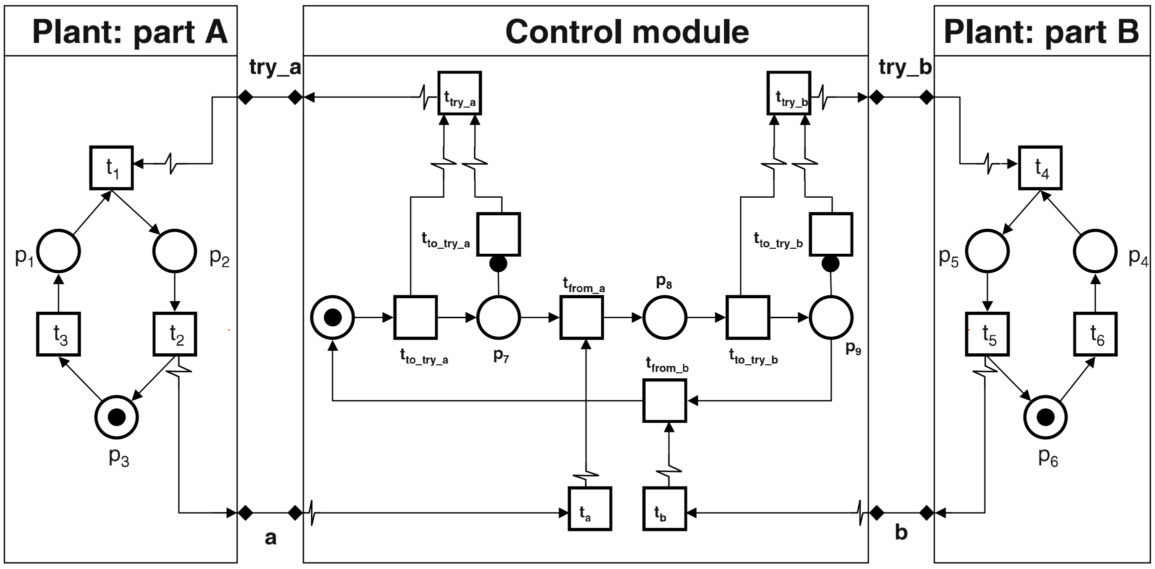

Fig. 24. The control module connected with the two plant modules. The composed module satisfies the specification in a minimal restrictive way.

output transitions in the set $\{t_a \in O \mid a \in x\}$. If x contains signal inputs, the situation is more complicated. Each state is represented by a marking which consists in general of more than one place.

Avoiding undesired conflicts caused by shared pre-places. Consider the automaton A shown in the left part and the signal net simulating A in the middle part of Figure 25

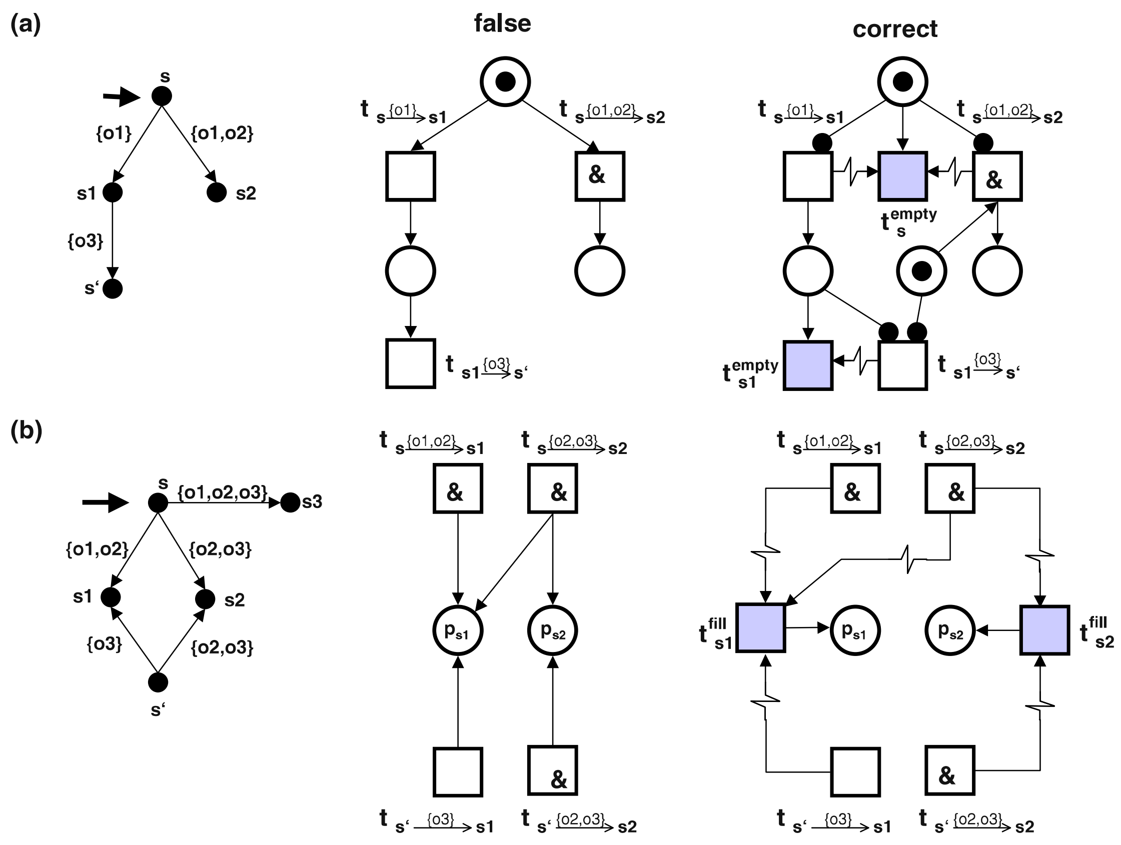

Fig. 25. Part (a): Avoiding undesired conflicts caused by shared preplaces and checking exact markings. Part (b): Avoiding undesired conflicts caused by shared postplaces.

(a). Since the transitions $t_{s\xrightarrow{\{o1\}}s1}$ and $t_{s\xrightarrow{\{o1,o2\}}s2}$ share a pre-place, the occurrence of the step of output signals $\{o1, o2\}$ either synchronizes $t_{s\xrightarrow{\{o1\}}s1}$ or $t_{s\xrightarrow{\{o1,o2\}}s2}$. That means, in the marking corresponding to state s two steps are in conflict. Moreover, one of the steps (when $t_{s\xrightarrow{\{o1\}}s1}$ is synchronized) does not correspond to the control flow.

The right part of Figure 25 (a) shows a signal net simulating A, where the mentioned conflict is avoided. To this end, a transition t_s^{empty} removing the token from the place p_s is introduced. It is synchronized by both transitions $t_{s\xrightarrow{\{o1\}}s1}$ and $t_{s\xrightarrow{\{o1,o2\}}s2}$. Both transitions have empty preset and test the place p_s to be marked via condition arcs. The occurrence of the step of output signals $\{o1, o2\}$ now synchronizes both transitions $t_{s\xrightarrow{\{o1\}}s1}$ and $t_{s\xrightarrow{\{o1,o2\}}s2}$.

Checking exact markings. In the last example, the occurrence of the step of output signals $\{o1, o2\}$ produces the follower marking $\{p_{s1}, p_{s2}\}$. Therefore, the marking corresponding to the state $s2$ is $\{p_{s1}, p_{s2}\}$. The occurrence of the step of output signals $\{o1\}$ produces the follower marking $\{p_{s1}\}$, which therefore is the marking corresponding to the state $s1$.

Since the transition $t_{s1\xrightarrow{\{o3\}}s'}$ should only be enabled under the marking $\{p_{s1}\}$, but not under the marking $\{p_{s1}, p_{s2}\}$, $t_{s1\xrightarrow{\{o3\}}s'}$ has to test the place p_{s2} not to be marked.

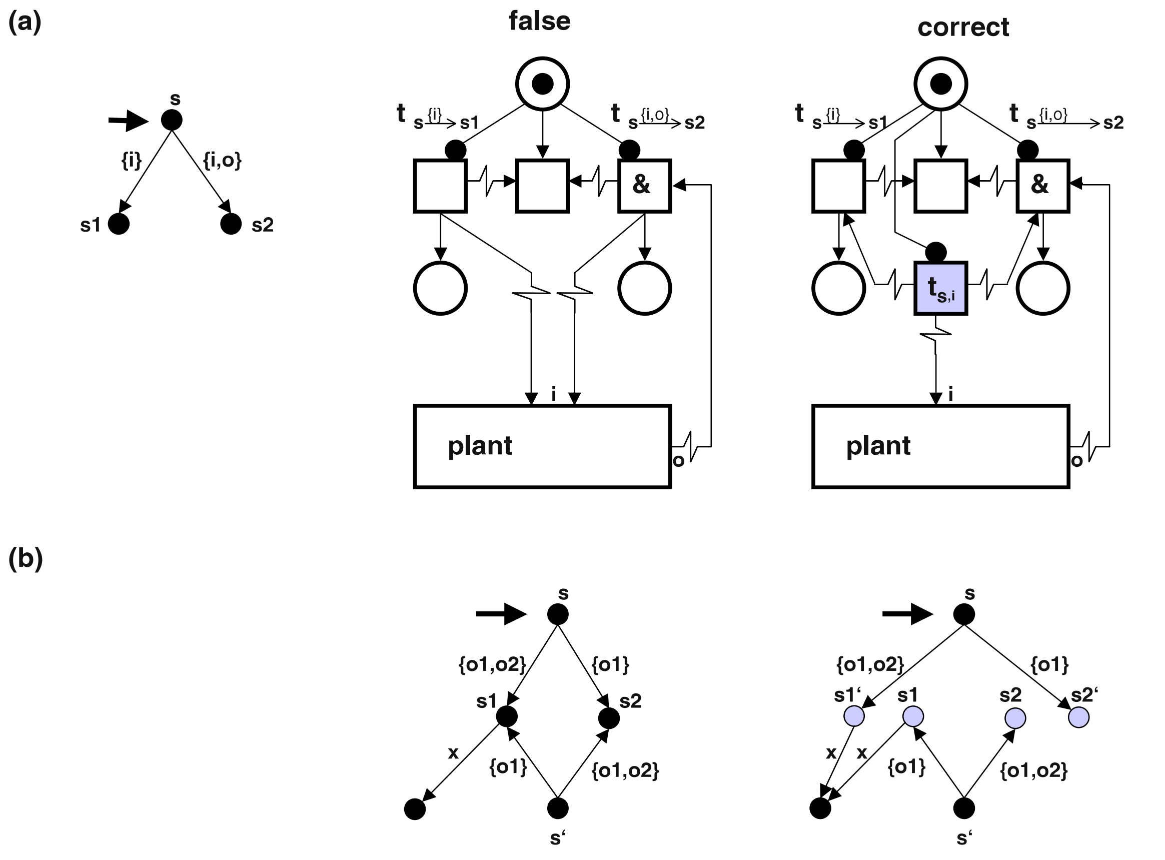

Fig. 26. Part (a): Avoiding cycles of event arcs caused by signal inputs. Part (b): Distinguishing states by markings.

This can be achieved by introducing a complementary place to p_{s2} and to test this place to be marked.

Avoiding undesired conflicts caused by shared post-places. Consider the automaton A shown in the left part and the signal net simulating A in the middle part of Figure 25 (b). The occurrence of the step of output signals $\{o2, o3\}$ in the marking corresponding to state s' synchronizes both transitions $t_{s' \xrightarrow{\{o3\}} s1}$ and $t_{s' \xrightarrow{\{o2,o3\}} s2}$. Therefore, the marking $\{p_{s1}, p_{s2}\}$ corresponds to the state $s2$. Since the occurrence of the transition $t_{s \xrightarrow{\{o2,o3\}} s2}$ must lead to the same marking $\{p_{s1}, p_{s2}\}$, the transitions $t_{s \xrightarrow{\{o2,o3\}} s2}$ and $t_{s \xrightarrow{\{o1,o2\}} s1}$ share a post-place. Therefore, the occurrence of the step of output signals $\{o1, o2, o3\}$ in the marking corresponding to state s either synchronizes $t_{s \xrightarrow{\{o2,o3\}} s2}$ or $t_{s \xrightarrow{\{o1,o2\}} s1}$. That means, in the marking corresponding to state s two steps are in conflict.

The right part of Figure 25 (b) shows a signal net simulating A where the mentioned conflict is avoided. For this, transitions t_{s1}^{fill} resp. t_{s1}^{fill}, which mark the places p_{s1} resp. p_{s2}, are introduced in a similar way as the transitions t_s^{empty} which unmarks places.

Avoiding cycles of event arcs caused by signal inputs. Consider the automaton A shown in the left part and the signal net simulating A in the middle part of Figure 26 (a). By the occurrence of transition $t_{s \overset{\{i,o\}}{\to} s2}$ input i is sent to the plant. This produces a cycle of event arcs, since i possibly synchronizes the step of outputs $\{o\}$, which again synchronizes $t_{s \overset{\{i,o\}}{\to} s2}$.

The right part of Figure 26 (a) shows a signal net simulating A, where such cycles are avoided. For this inputs are modelled by additional transitions as shown.

Distinguishing states by markings. Consider the automaton A given in the left part of Figure 26 (b). On the one hand, the occurrence of the step of output signals $\{o1, o2\}$ in the marking $\{p_s\}$ synchronizes the transitions $t_{s \overset{\{o1,o2\}}{\to} s1}$ and $t_{s \overset{\{o1\}}{\to} s2}$. Therefore, the follower marking is $\{p_{s1}, p_{s2}\}$. On the other hand, the occurrence of the step of output signals $\{o1, o2\}$ in the marking $\{p_{s'}\}$ synchronizes the transitions $t_{s' \overset{\{o1,o2\}}{\to} s1}$ and $t_{s' \overset{\{o1\}}{\to} s2}$ and the follower marking is also $\{p_{s1}, p_{s2}\}$. Therefore it corresponds to both states $s1$ and $s2$. That means that the control does not distinguish between observable different states of the plant.

This situation can be avoided if we modify the automaton A such that the states of A distinguish words according to their last character. More formally, we require $x = y$ for all edges of the form $s' \overset{x}{\to} s$ and $s'' \overset{y}{\to} s$. As long as this is not the case for a state s of A, i.e. $x \neq y$, one can split s into two copies, one for words ending with x and one for words ending with y.

3.3 Alternative Techniques for Synthesis

An approach that uses complete enumeration of the whole controlled behavior will obviously reach its limitation if the systems are large. It therefore makes sense to consider methods that use structural properties of the model instead. At the current state of research, a general approach for models without structural restrictions is unlikely. If one restricts, however, to models with simple structures, formal techniques for synthesis are possible. This holds in particular for pure Petri net models of the plant.

Methods for controller or supervisor synthesis for Petri nets are well developed. This holds especially for specification in terms of forbidden states. A comprehensive review can he found in [13]. Synthesis methods for sequential specifications, however, are yet rather sparse. One significant contribution in this area is the work of Giua [7], who studied an alternative design to Ramadge and Wonhams monolithic supervisor. His method entails the Petri net modelling of subsystems and specifications, which are then combined through a "concurrent composition" operation. Then, a "refinement" step of the combined model ensures the non-blocking and controllable properties. In this case, the existence of the Petri net supervisor is guaranteed whenever the given Petri net subsystems are conservative, although additional structure may be needed in the refinement step. The main distinction to [7] in the method described in [21] is the connection of the plant and specification models, which is done through the condition and event signals of Signal nets. This is also the case when we consider systems where plant and controller interact with each other in a closed loop. So, the model of the admissible behavior will not be a classical Petri net but a Signal Petri net. The framework [21] can also be applied as long as the Petri net model of the plant is safe.

The scope of [21] is limited to solving the problem of preventing an uncontrollable event at a specific “controller state.” The method starts with a safe Petri net model of the system and a sequential specification that is modelled with a special state machine. A more general modelling framework for sequential specifications is currently under investigation, where a temporal logic formula can be mapped to a signal net model. An initial discussion is given in [20]. The main contribution in [21] can be described as follows: The condition and event signals of Signal nets are used to combine the plant and specification models. Then, the structure of this combined model allows to determine which controllable events need to be restricted by the controller, and at what states they need to be restricted in order to obtain the legal or admissible behavior of the system. Thus, a complete description of the state space is avoided, and control of the system is performed in a minimally restrictive way.

Another issue that is currently under investigation is how to extend the method to reach some "target events" in the plant model. First results are presented in [22]. The major drawback, however, is that in general the set of all reachable markings are required to determine all feasible processes that lead to this target events. This means that even if a process exists that drives the plant model towards the target event, the process might be not feasible since it requires a marking that is not reachable from the initial marking.

4 Conclusion

In this paper we have illustrated methodologies for synthesis of control for DES with input and output structure. For input/output communication, we employ active signals which try to force events and passive signals which prohibit resp. enable event occurrences. As a modelling formalism, we use modules of signal nets. Signal nets offer a direct way to model typical actuators behavior. Another advantage of such modules consists in supporting input/output structuring, modularity, and compositionality in an intuitive graphical way.

Given a control specification in form of a regular language over output signals of the system, we showed how to automatically synthesize the control module. This forces the system to maximally permissive behavior preserving the control specification, in the sense that only sequences of outputs which respect the specification will be observed.

In the paper we did not focus on complexity issues. It is known that the complexity of the supervisory control problem is PSPACE-hard in general, and sometimes even undecidable ([28], pp. 15 - 36). For efficient algorithms, the setting must be restricted in some way, for example by considering only very special classes of control specifications.

We restricted our approach in several aspects: As a control specification, only sequences of signal outputs were considered. Moreover, the synthesized controller changes at most one of its inputs to the plant at each moment. An extension of our methodology to control specifications including signal inputs and to controllers sending steps consisting of more than one signal input to the plant is a straightforward exercise.

The presented approach considers only Petri nets on an elementary level. For complex industrial-size systems, these nets tend to be either very large or too abstract. In particular, data and time aspects cannot be modelled in a natural way. Therefore, we are working on an extension of the control methodology for modules of signal nets with special high-level Petri net features.

References

1. B. Caillaud, P. Darondeau, L. Lavagno and X. Xie (Eds.). Synthesis and Control of Discrete Event Systems. Kluwer Academic Press, 2002.
2. C. G. Cassandras and S. Lafortune. Introduction to Discrete Event Systems. Kluwer, 1999.
3. H. Cho and S.I. Marcus. On supremal languages of classes of sublanguages that arise in supervisor synthesis problems with partial observation. em Mathematics of Control, Signals, and Systems, Vol. 2, No. 2, pp. 47–69, 1989.
4. J. Desel, G. Juhás and R. Lorenz. Input/Output Equivalence of Petri Modules. In *Proc. of IDPT 2002*, Pasadena, USA, 2002.
5. J. Desel, V. Milijic and C. Neumair. Model Validation in Controller Design. In J. Desel, W. Reisig and G. Rozenberg (Eds.) *Lectures on Concurrency and Petri Nets*. Springer, LNCS 3098, 467 – 495, 2004.
6. P. Dietrich, R. Malik, W.M. Wonham and B.A. Brandin. Omlementation Consideration in Supervisory Control. In [1].
7. A. Giua. Petri Nets and Discrete Event Models for Supervisory Control. Ph.D. Thesis. Dept. of Computer and Systems Engineering, Rensselaer Polytechnic Institute. Troy, NY, 1992.
8. H.-M. Hanisch, A. Lüder. Modular Modeling of Closed-Loop Systems. Colloquium on Petri Net Technologies for Modeling Communication Based Systems, Berlin 1999, pp. 103–126.
9. H-M. Hanisch and A. Lüder. A Signal Extension for Petri nets and its Use in Controller Design. *Fundamenta Informaticae*, 41(4) 2000, 415–431.
10. H.-M. Hanisch, A. Lüder, M. Rausch. Controller Synthesis for Net Condition/Event Systems with Incomplete State Observation. European Journal of Control, Nr. 3, 1997, 292–303.
11. H.M. Hanisch, M. Rausch: Netz-Condition/Event-Systeme, 4. Fachtagung Entwurf Komplexer Automatisierungssysteme, Braunschweig, Germany, June 1995, Proceedings, pp. 55-71 (in German).
12. H.-M. Hanisch, J. Thieme and A. Lüder. Towards a Synthesis Method for Distributed Safety controllers Based on Net Condition/Event Systems. *Journal of Intelligent Manufacturing*, 5, 1997, 8, 357–368.
13. L.E. Holloway, B.H. Krogh and A. Giua. A Survey of Petri Net Methods for Controlled Discrete Event Systems. *Discrete Event Dynamic Systems: Theory and Applications*, 7, 1997, 151–190.
14. J.E. Hopcroft, R. Motwani and J.D.Ullman. Introduction to Automata Theory, Languages, and Computation. Addison Wesley, 2001.
15. R. Janicki and M. Koutny. Semantics of Inhibitor Nets. *Information and Computations*, 123, 1–16, 1995.
16. G. Juhás. On semantics of Petri nets over partial algebra. In J. Pavelka, G. Tel and M. Bartosek (Eds.) *Proc. of 26th Seminar on Current Trends in Theory and Practice of Informatics SOFSEM'99*, Springer, LNCS 1725, 408–415, 1999.
17. G. Juhás and R. Lorenz. Modeling with Petri Modules. In [1].
18. G. Juhás, R. Lorenz and C. Neumair. Synthesis of Controlled Behavior with Modules of Signal Nets. In J. Cortadella and W. Reisig (Eds.) *Proc. of 25th International Conference on Application and Theory of Petri Nets*, Springer, LNCS 3099, 238 – 257, 2004.
19. G. Juhás, R. Lorenz and C. Neumair. Modeling and Control with Modules of Signal Nets. In J. Desel, W. Reisig and G. Rozenberg (Eds.) *Lectures on Concurrency and Petri Nets*. Springer, LNCS 3098, 585 – 625, 2004.

20. L.E. Pinzon, H.-M. Hanisch and M.A. Jafari. Sequential Control Specifications with TL and NCES. Rutcor Research Report 4–98, RUTCOR, Rutgers University, Piscataway, NJ, 1998.
21. L.E. Pinzon, M.A. Jafari, H.-M. Hanisch and P. Zhao. Modeling admissible behavior using event signals. To be published in: IEEE Transactions on Sytems, Man and Cybernetics, Part B, 2004.
22. L.E. Pinzon, M.A. Jafari and H.-M. Hanisch. Controller Synthesis for Target Events. Submitted to Wodes 2004.
23. P.J. Ramadge, W.M. Wonham. The Control of Discrete Event Systems. Proceedings of the IEEE, 77 (1989) 1, 81–98.
24. G. Rozenberg, and J. Engelfriet. Elementary Net Systems. In W. Reisig and G. Rozenberg (Eds.) *Lectures on Petri Nets I: Basic Models*, Springer, LNCS 1491, 12–121, 1998.
25. R.S. Sreenivas und B.H. Krogh. On Condition/Event Systems with Discrete State Realizations. *Discrete Event Dynamic Systems: Theory and Applications*, 2, 1991, 1, 209–236.
26. R. S. Sreenivas and B. H. Krogh. Petri Net Based Models for Condition/Event Systems. In *Proceedings of 1991 American Control Conference*, vol. 3, 2899–2904, Boston, MA, 1991.
27. P. H. Starke. Das Komponieren von Signal-Netz Systemen. In Proc 7. Workshop Algorithmen und Werkzeuge für Petrinetze AWPN 2000, Universität Koblenz - Landau, 1–6, 2000.
28. P. Darondeau and S. Kumagai (Eds.). Proceedings of the *Workshop on Discrete Event System Control*. Satellite Workshop of ATPN 2003.
29. M.C. Zhou und F. DiCesare. *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems.* Kluwer Adacemic Publishers, Boston, MA, 1993.
30. Zhonghua Zhang and W.M. Wonham. STCT: An Efficient Algorithm for Supervisory Control Design. In [1].