Compositional Control Synthesis for Discrete Event Systems: A Semantic Framework Based on Open Petri Nets

Reiko Heckel
University of Paderborn, Paderborn, Germany
reiko@uni-paderborn.de
Mourad Chouikha
EXTESSY AG and Technical University of Braunschweig, Braunschweig, Germany
m.chouikha@extessay.com

ABSTRACT: Open nets are place-transition Petri nets with interfaces, which support a notion of composition and a corresponding compositional interpretation of the concurrent behaviour of nets. The control synthesis problem of generating a controller for a given plant from an abstract specification of the controller’s behaviour can be formulated in terms of open nets by modelling the plant as an open net whose interfaces correspond to the sensors and actuators of the controller and specifying the desired behaviour as a set of processes for this net. Then, the problem consists in synthesising a controller net which, when composed with the net modelling the plant, leads to the specified restriction of the plant’s processes.

Based on this observation, which provides an abstraction of the actual synthesis algorithm, we study the problem of generating controllers consisting of several components. In particular, we analyse requirements for the logic used for specifying the controller in order to allow for a compositional, component-wise synthesis.

I. INTRODUCTION

Petri nets are widely used in engineering and computer science. They provide an intuitive visual, yet formally based language with a direct representation of concurrency, powerful analysis methods and tool support. Besides visual specifications, also the behaviour of a net can be represented graphically by means of processes, that is, deterministic and acyclic nets that represent concurrent computations. This feature makes it possible to present the concurrent semantics of nets in a semi-formal way, e.g., in order to explain it to domain experts. A well-known drawback, however, of Petri nets is that, in contrast to more “syntactic” methods like process calculi, they do not provide an easy notion of composition which would allow for a compositional interpretation of their behaviour.

Practical methodologies in engineering and computer science take a structured approach, designing systems from smaller subsystems and components, which can be combined and reused. For example, in workflow specification, one important application area of Petri nets [Aal98], a common problem is the integration or coupling of workflows of different departments or enterprises [Aal99]. Also technical system are designed from smaller components. Using Petri nets for modelling such systems, it is important that they support a notion of composition and a notion of behaviour which allows to understand and reason about individual components without requiring complete knowledge of their environment.

Most approaches that allow the composition of nets based on some notion of interface consider semantics only globally, or they do not consider a semantics based on processes, but on abstract mathematical models like partial orders, event structures, or traces [BDH92], [NPS95], [Kin97], [Bas00]. The approach of open Petri nets is intended to solve the problem of compositionality in nets without loosing the benefits of a visual concurrent semantics. For this purpose, place-transition Petri nets are endowed with an interface of open places which represent the gluing points of the net with some unknown context. Then a corresponding notion of open processes is defined, which behave compositionally in the sense that the open processes of a composed net can be obtained by pairwise merging of open processes of the components nets [BCEH01].

Another problem with the application of Petri nets is the gap between Petri nets in theory and Petri net-based languages in practical use. In fact, the latter often combine features like abstract data types or object-oriented concepts, time, read and inhibitor arcs, etc., for which no relevant theory exists. Also there is a lack of theoretical support for domain-specific abstractions and methodologies in application-oriented languages.

The aim of this paper is to shorten this gap by analysing, from a theoretical point of view, a practical application of nets in control engineering and mapping the concepts found in this application to constructions and results in the theory of open nets. In particular, we shall employ open nets to formalise a component-based modelling approach and to provide an abstract formulation of the methodology of control synthesis which allows to generate a controller model for an automation system from a declarative specification. Then, based on the correspondence between theory and application concepts, we derive requirements for the further development of the theory to improve the support of practi-
cal problems.

The paper is organised as follows. First, we review the approach to control synthesis and discuss the various places where decomposition of nets is relevant in this context. Then, in Section III we present the basic concepts of open Petri nets, their composition operator and their compositional process semantics. Section IV is devoted to the formulation of the control synthesis problem in terms of open nets. Moreover, in this section, a further development of the approach towards a compositional control synthesis is proposed. Section V concludes the paper by a summary and discussion of extensions to the theory that would be needed to provide full support for the control synthesis approach.

II. MODEL-BASED CONTROL SYNTHESIS

The control theoretic approach to the synthesis of automation systems is based on a decomposition into a controller and plant which resembles a control loop in control engineering (Fig. 1). In [CSO98] a Petri net-based methodology is presented which supports this approach in four steps.

1. Modelling of the plant (the subsystem that shall be controlled) by means of a Petri net,
2. Specification of the controller (the component to be developed) using some form of temporal logic
3. Synthesis of a controller model (i.e., a Petri net) from the temporal specification, and
4. Generation of the controller code from the Petri net model

In the following, we shall review the first three steps of this methodology (see also [LOS95], [Cho99]) based on a small example, adapted from [CSO98].

A. Modelling of the plant

The model-based methodology for controller synthesis will be explained by means of a simplified stamping process (see Fig. 2). In this process a pusher moves coins from a store into a mould. Then the coins are stamped by a die, and afterwards ejected by a second pusher and blown out of the process by compressed air.

This process is modelled by the Petri net shown in Fig. 3. The net is structured into several components (or objects) representing resources like pushers or the die, or the workpiece to be processed. To support reuse, such objects are stored in libraries, from where they are imported, composed and configured using a design tool [CSO98].

The plant model consists of several layers, which represent different aspects of the plant. The first layer describes the state changes of the processed objects, the coins. The second layer represents the material flow. Here, the possible locations and the corresponding transportation processes of the workpiece are described. The third layer represents the dynamics of the process resources which influence the state and the location of the workpiece.

The actuators by means of which the resources shall be controlled are modelled as places, shown with bold borders. Sensors, which shall allow to detect the current state of resources or the location and state of a workpiece, are filled grey. The net contains read-arcs, shown as bidirectional arrows, inhibitor arcs, distinguished by a small black circle at the transition end, and two different dependencies between transitions: The two stamp transitions in the material flow and the workpiece layer are synchronised, that is, they may only fire simultaneously. The transition of the pusher1 component which represents the move from left to right is implied by the transition push in. That means, the implied transition has to fire whenever push in fires, but if push in is disabled the implied transition can also fire in isolation. This behaviour is achieved by the pattern shown in Fig. 4.

B. Specification of the controller

To specify requirements on the behaviour of the system, a temporal specification language is deployed. As atomic propositions, properties of states (markings) can be ex-
pressed by requiring e.g., the presence or absence of tokens in certain places. Using temporal operators like is forbidden or is forbidden after etc. safety properties can be expressed. On the other hand, state properties can be combined to requirements on sequences, which must hold for all executions of the model. Such combinations express liveness properties [Obe98].

To specify the stamping process we first specify the states which are relevant for all processes, e.g. start and end state as well as certain intermediate states. Furthermore we specify states which must not be reached, e.g., because they represent dangerous situations. For example, the state in which the die is in the low position and the pusher is simultaneously in the right position must be forbidden, because otherwise there is a collision between pusher and die. Moreover there are situations which are not absolutely dangerous, but cause a waste of resources, like switching on the compressed air before stamping the workpiece.

C. Control synthesis

The control synthesis problem now consists in generating a controller model, i.e., a second Petri net, which, when composed with the plant net, ensures that the temporal constraints of the specification are fulfilled by all firing sequences of the composed net. This is achieved in two steps. First, a search is performed for firing sequences of the plant net which conform to the specification. Then, a controller net is synthesised which forces the plant net to execute one of these sequences.

A sequence consists of a list of firing steps. Each step represents a number of transitions of the plant which can be concurrently executed, and defines the control outputs necessary to fire these transitions as well as the marking of the sensor places when executing them. The fundamental structure of a control net for a simple step is represented in Fig. 5. Here the transition t1 sets the actuator place enabling one or more transitions in the plant model. As soon as the sensor place indicates the end of the step, transition t2 resets the actuator place.

The controller net of the stamping process is shown in Fig. 6. The net consists of three parts. The central control flow cycle representing the control algorithm is connected to actuator places to specify when controls are set and reset. Connections between sensor places and the transitions of the control cycle model the inputs of the controller.

Summarising, we observe three different ways in which the nets in the control synthesis approach are structured. Plant nets are structured in different layers, like resources, material flow, and workpiece. The resource layer is further structured horizontally into subnets corresponding to different resources. The overall model is composed of the plant and the controller part. Orthogonally, as indicated in the top of Fig. 3 and 6, we may distinguish phases like the input, processing, and output phase.

Next, we shall introduce open nets to support the decomposition of nets at interface places that allow a compositional understanding of their behaviour. In Section III below, we will first turn to the decomposition of the (output
III. A COMPOSITIONAL APPROACH TO PETRI NETS

An open net is an ordinary P/T Petri net with a distinguished set of open places which represent the interface of the net towards the environment. For example, the open net in Fig. 3 modelling the plant provides an interface to the controller given by actuator places. From the point of view of the plant net, tokens can freely appear in and disappear from the open places. Concretely, an open place can be either an input or an output place (or both), meaning that the context can put or remove tokens from that place.

A. Open Petri Nets

We assume the standard definition [Rei85] of a place-transition Petri net \( N \) defined as a bipartite multigraph over disjoint sets of places \( S \) and transitions \( T \). By \( \sigma, \tau : T \rightarrow S^\bullet \) we denote the functions assigning to each transition its pre- and post-multiset, where \( X^\bullet \) is the set of finite multisets over a set \( X \). Thus, a net is at the same time a hypergraph with places as vertices and transitions as hyperedges [MM90]. Sometimes we will find it convinient to use this hypergraph presentation. The powerset over \( X \) shall be denoted by \( 2^X \).

Given a place \( s \in S \), the pre- and post-set of \( s \) are defined by \( \bullet s = \{ t \in T \mid s \in \sigma(t) \} \) and \( s^\bullet = \{ t \in T \mid s \in \tau(t) \} \).

A net \( N_0 \) is a subnet of \( N_1 \), denoted by \( N_0 \hookrightarrow N_1 \) if \( T_0 \subseteq T_1, S_0 \subseteq S_1 \) such that source and target functions \( \sigma_0 = \sigma_1|_{T_0}, \tau_0 = \tau_1|_{T_0} \) are defined by restriction of the corresponding functions of \( N_1 \) to the smaller set of transitions. That means, the connections of a given transition in \( N_0 \) are exactly the same as in \( N_1 \).

Definition III.1 (open net) An open net is a pair \( Z = (N_Z, O_Z) \), where \( N_Z = (S_Z, T_Z, \sigma_Z, \tau_Z) \) is an ordinary P/T Petri net and \( O_Z = (O^+_Z, O^-_Z) \subseteq 2^{S_Z} \times 2^{S_Z} \) are the input and output open places of the net.

Observe that the sets \( O^+_Z \) and \( O^-_Z \) are not necessarily disjoint, hence a place can be both an input and an output open place at the same time.

The open nets for the resources and material flow of the output phase of the stamping process are shown in Fig. 7. Ingoing and outgoing arcs without source or target designate the input and output places, respectively. Bidirectional arcs are used to abbreviate pairs of in and outgoing arcs, that is, the corresponding places are both in \( O^+_Z \) and \( O^-_Z \). The open places of the nets in Fig. 7 are the actuator and sensor places of the plant, which are both input and output open, as well as the entry and exit places of the material flow. The latter have to be connected to the corresponding places in other subprocesses in order to make the system working. Note that read arcs (bidirectional links) are interpreted as cycles while for inhibitor arcs we assume the usual implementation by means of complement places.

The notion of enabledness for a transition (or multiset of transitions) of an open net is the usual one, but, besides the changes produced by the firing of transitions, we represent interactions with the environment by spontaneous actions producing/consuming tokens in input/output places of the net. The actions of the environment which produce and consume tokens in an open place \( s \) are denoted by \( +s \) and \( -s \), respectively. Thus, if \( Z \) is an open net, a (sequential) firing step can be either

(i) the firing of a transition, i.e., \( m \oplus \bullet t \mid t \) \( m \oplus t^\bullet \), with \( m \in S^\oplus_Z, t \in T_Z \);
(ii) the creation of a token by the environment, i.e., \( m \oplus s \mid s \) \( m \oplus s \), with \( s \in O^+_Z, m \in S^\oplus_Z \);
(iii) the deletion of a token by the environment, i.e., \( m \ominus s \mid -s \) \( m \ominus s \), with \( m \in S^\ominus_Z, s \in O^-_Z \).

Embeddings of open nets are based on the notion of subnet. They formalise the idea of an “insertion” of a net into a large context where part of the unknown environment gets more specified. To capture this idea, we have to know for an inclusion of Petri nets \( N_0 \hookrightarrow N_1 \) those places where the net \( N_1 \) interferes with the context provided by \( N_1 \). Let \( \text{in}(N_0 \hookrightarrow N_1) \subseteq S_0 \) be the set of all places \( s_0 \) of \( N_0 \) such that there exists a transition \( t_1 \in T_1 \setminus T_0 \) with \( s_0 \in \tau(t_1) \). These are the places where context transitions my drop tokens, i.e., potential input places. Dually, \( \text{out}(N_1 \hookrightarrow N_2) \) is the set of all places \( s_0 \) of \( N_0 \) such that there exists \( t_1 \in T_1 \setminus T_0 \) with \( s_0 \in \sigma(t) \). These are potential output places where context
transitions may remove tokens.

**Definition III.2 (open net embedding)** An open net embedding $Z_1 \hookrightarrow Z_2$ is a Petri net inclusion on the underlying nets $N_0 \hookrightarrow N_1$ such that

(i) $O^+_1 \cup \text{in}(N_0 \hookrightarrow N_1) \subseteq O^+_0$ and

(ii) $O^-_1 \cup \text{out}(N_0 \hookrightarrow N_1) \subseteq O^-_0$.

Conditions (i) and (ii) require that open places are reflected and hence that places which are “internal” in $Z_0$ cannot be promoted to open places in $Z_1$. Furthermore, the context in which $Z_0$ is inserted can interact with $Z_0$ only through open places.

As an example of open net embedding, consider the embedding of net Output Flow into net Output of Fig. 7. Observe that the constraints characterising open net embeddings have an intuitive graphical interpretation:

- the connections of transitions to their pre-set and post-set have to be preserved. New connections cannot be added;
- in the larger net, a new arc may be attached to a place only if the corresponding place of the subnet has a dangling arc in the same direction. Dangling arcs may be removed, but cannot be added in the larger net. For instance, without the bidirectional dangling arc at the place high in net Output Flow (i.e., if place high were not input and output open) the embedding into Output in Fig. 7 would not have been a legal open net embedding.

Open net embeddings form a special case of open net morphisms [BCEH01]. In particular, they are closed under composition, and they induce a projection of the behaviour of the larger net onto the smaller one. This fact is made precise in the next subsection in terms of deterministic processes of open nets.

**B. Deterministic processes of open nets**

A deterministic process represents a unique concurrent computation of a Petri net [GR83]. A deterministic process is given by a Petri net, called occurrence net, satisfying suitable acyclicity and conflict freeness requirements together with a mapping to the original net. The idea is that each place of the occurrence net represents a token occurrence, and each transition represents a firing of a transition in the original Petri net. Therefore, a place may be in the post-set of at most one transition, i.e., the firing who generated the corresponding token. This is called absence of backward conflicts. Dually, absence of forward conflicts means that each place is in the pre-set of at most one transition. Otherwise this place would represent a branching point which would violate the idea of a deterministic computation.

In analogy to ordinary deterministic processes, a deterministic process of an open net is an open occurrence net together with a mapping to an open net. An open occurrence net is an open net $K$ such that $N_K$ is an ordinary deterministic occurrence net satisfying additional conditions for open places.

The open places in $K$ are intended to represent tokens which are produced/consumed by the environment in the considered computation. Consequently, every input open place is required to have an empty pre-set, i.e., to be minimal with respect to the causal order. In fact, an input open place in the post-set of some transition would correspond to a kind of generalised backward conflict: a token on this place could be generated in two different ways and this would prevent us to interpret the place as a token occurrence.

Analogously, each output open places must be maximal with respect to the causal order, i.e., an output open place...
cannot be in the pre-set of any transition. In fact, an output open place \( s \) which is in the pre-set of a transition \( t \) represents a token occurrence which can be consumed either by the environment or by transition \( t \).

**Definition III.3 (open occurrence net and process)** An open (deterministic) occurrence net is an open net \( K \) such that its underlying net \( K \) is an ordinary deterministic occurrence net with causal order \( <_K \), each input open place is minimal w.r.t. \( <_K \) (i.e., \( \forall s \in O^+_K, _K s = \emptyset \)) and each output open place is maximal (i.e., \( \forall s \in O^-_K, _K s = \emptyset \)).

A deterministic process of an open net \( Z \) is a mapping \( \pi : K \rightarrow Z \) where \( K \) is an open deterministic occurrence net and \( \pi : N_K \rightarrow N_Z \) is a Petri net morphism (that is, a mapping of places and transitions which preserves the pre- and post-sets of transitions) such that

\[
\pi(O^+_K) \subseteq O^+_Z \quad \text{and} \quad \pi(O^-_K) \subseteq O^-_Z.
\]

The class of all deterministic processes over an open net \( Z \) is denoted by \( \text{Proc}(Z) \).

In order to simplify the notation, in the following we will sometimes identify a process \( \langle K, \pi \rangle \) with the mapping \( \pi \).

The last condition states that the image of an open place in \( K \) must be an open place in \( Z \), since tokens can be produced (consumed) by the environment only in input (output) open places of \( Z \).

Processes for the open nets in Fig. 7 are shown in Fig. 8. The mappings back to the original nets in Fig. 7 are implicitly represented by the labelling of places and transitions of the occurrence nets. Observe that the requirement that each input place is minimal and each output place is maximal w.r.t. the causal order of the process has a natural graphical interpretation: the absence of backward and forward conflicts extends to dangling arcs, i.e., in total, each place may have at most one ingoing and one outgoing arc.

Let \( Z_0 \rightarrow Z_1 \) be an open net embedding. As mentioned before, each process of \( Z_1 \) can be projected to \( Z_0 \) by considering only that part of the computation which is visible in the smaller net.

**Construction III.1 (projection of a process)** Let \( Z_0 \rightarrow Z_1 \) be an open net embedding and let \( \pi_1 : K_1 \rightarrow Z_1 \) be a process of \( Z_1 \). The projection \( \pi_1|_{Z_0} = \pi_0 : K_0 \rightarrow Z_0 \) of \( \pi_1 \) to \( Z_0 \) is constructed as follows. \( N_{K_0} = \pi_1^{-1}(N_{Z_0}) \) is the inverse image of \( N_{Z_0} \rightarrow N_{K_1} \) under \( \pi_1 \).

\[
\pi_0 = \pi_1|_{K_0} : K_0 \rightarrow Z_0 \quad \text{is the restriction of} \quad \pi_1 \quad \text{to} \quad N_{K_0}.
\]

The open places of \( K_0 \) are determined by taking the smallest sets of open places which makes the inclusion \( K_0 \rightarrow K_1 \) an open net embedding, namely

- \( O^+_0 = O^+_1 \cup \text{in}(K_0 \rightarrow K_1) \)
- \( O^-_0 = O^-_1 \cup \text{out}(K_0 \rightarrow K_1) \)

The embedding of Output Flow into Output in Fig. 7 induces a projection of open net processes. For instance, the projection of the Output process in the bottom of Fig. 8 to the smaller net Output Flow yields the right net in Fig. 8. Notice how, e.g., the left-most occurrence of the transition from off to on, which produces a token in place on, is replaced in the projection by a dangling input arc: an internal action in the larger process becomes an interaction with the environment in the smaller one.

**C. Composing open nets**

In this section we introduce a basic mechanism for composing open nets. It is based on the gluing of two nets sharing a common subnet that represents the intended overlap.

Consider, for instance, the open nets for resources and material flow of the output phase of the stamping process Output Resources and Output Flow in the middle of Fig. 7. The two nets share the subnet Shared depicted in the top of the same figure, which represents the “glue” between the two components. The net Output resulting from the composition of Output Resources and Output Flow over the shared subnet Shared is shown in the bottom part of Fig. 7. This composition is only defined if the embeddings of the components into the resulting net satisfy the constraints of open net embedding. For example, if we remove the dangling arc of the place high in the net Output Flow, the embedding of Shared into Output Flow would still represent a legal open net embedding. However, in this case the embedding of Output Flow into Output would become illegal because of the new transitions in the pre- and post-sets of high (cf. condition (i) of Definition III.2).

Next we formalise the conditions under which the construction is defined and the way the open places for the composed net are determined.

**Definition III.4 (composable nets)** Two open nets \( Z_1 \) and \( Z_2 \) are called composable if their intersection \( Z_0 = Z_1 \cap Z_2 \) is defined, such that \( Z_1 \leftrightarrow Z_0 \leftrightarrow Z_2 \) are open net embeddings and

1. \( \text{in}(Z_0 \leftrightarrow Z_1) \subseteq O^+_{Z_2} \) and \( \text{out}(Z_0 \leftrightarrow Z_1) \subseteq O^-_{Z_2} \),
2. \( \text{in}(Z_0 \leftrightarrow Z_2) \subseteq O^+_{Z_2} \) and \( \text{out}(Z_0 \leftrightarrow Z_2) \subseteq O^-_{Z_1} \).

Conditions 1 and 2 mean that places which are used as interfaces by \( Z_1 \), namely the places in \( (Z_0 \leftrightarrow Z_1) \) and out \( (Z_0 \leftrightarrow Z_1) \), are input and output open places in \( Z_2 \), and symmetrically. If these condition are satisfied the composition of \( Z_1 \) and \( Z_2 \) is defined as follows.

**Construction III.2 (composition of open nets)** Let \( Z_1, Z_2 \) be composable open nets and \( Z_0 \leftrightarrow Z_0 \leftrightarrow Z_2 \) be the corresponding embeddings. Then, the composition of \( Z_1 \) and \( Z_2 \), denoted by \( Z_3 = Z_1 \oplus Z_2 \), is computed by forming the union of the underlying nets of \( Z_1 \) and \( Z_2 \), and then taking as open places, for \( x \in \{+, -, \} \),

\[
O^+_Z = \{ x \in S_3 \mid x \in S_1 \Rightarrow x \in O^+_Z \wedge x \in S_2 \Rightarrow x \in O^+_Z \}.
\]

Thus, the open places of \( Z_3 \) are determined in the most permissible way that is consistent with the constraints imposed by \( Z_1 \) and \( Z_2 \) (i.e., such that the conditions for open net embeddings are satisfied).

In [BCEH01] this construction is formalised as a pushout of open net morphisms. Such axiomatic characterisation of the idea of a union allows to apply this concept to
more sophisticated structures (like nets) while abstracting from “implementation issues” like name handling. In fact, category-theoretic constructions, like the pushout, are only defined up to isomorphism (consistent renaming) of structures. In our presentation, which is set-theoretic in flavour, we implicitly assume renaming of places and transitions whenever appropriate, relying on [BCEH01] for the consistency of this assumption.

D. Amalgamating deterministic processes

For a composition of nets like in Fig. 7, we would like to establish a relationship between the behaviours of the involved nets: The processes of the composed net should be constructed by merging matching pairs of local processes. Given two deterministic processes \( \pi_1 \) of \( Z_1 \) and \( \pi_2 \) of \( Z_2 \) which agree on \( Z_0 \) in the sense that they could be seen as two local views of the same global computation, we construct a deterministic process \( \pi_3 \) of \( Z_3 \) by amalgamating \( \pi_1 \) and \( \pi_2 \). Vice versa, each deterministic process \( \pi_3 \) of \( Z_3 \) shall be projected to deterministic processes \( \pi_1 \) and \( \pi_2 \) of \( Z_1 \) and \( Z_2 \), respectively, which can be amalgamated to produce \( \pi_3 \) again.

Since processes are essentially open nets (subject to some additional constraints and equipped with a mapping to another net), it is natural to expect that the amalgamation of processes is based on the composition of these occurrence nets. Thus, we have to identify a condition which ensures that the composition of the deterministic open processes (resp. their underlying occurrence nets) is defined and produces a net in the same class. This condition shall formalise the intuitive idea of processes of different nets which “agree” on a common part.

First, given occurrence nets \( K_1 \leftrightarrow K_0 \leftrightarrow K_2 \) we introduce the notion of causality relation induced by \( K_1 \) and \( K_2 \) over \( K_0 \). When the two nets are composed, the corresponding causality relations get “fused”. Hence, to avoid the creation of cyclic causal dependencies in the resulting net, the induced causality will be required to be a strict partial order.

**Definition III.5 (consistent occurrence nets)** Let \( K_1, K_2 \) be open occurrence nets with \( K_0 = K_1 \cap K_2 \) such that \( K_1 \leftrightarrow K_0 \leftrightarrow K_2 \) are open net embeddings. The causality relation \( <_{1,2} \) induced on \( K_0 \) by \( K_1 \) and \( K_2 \) is the least transitive relation such that for any \( x_0, y_0 \) in \( K_0 \), if \( x_0 <_{1,2} y_0 \) then \( x_0 <_{1,2} y_0 \).

We say that the occurrence nets are consistent if they are composable in the sense of Def. III.4 and the induced causality \( <_{1,2} \) is a finitary strict partial order.

The composition of open nets, when applied to consistent deterministic occurrence nets, produces a deterministic occurrence net [BCEH01].

As mentioned before, two deterministic processes \( \pi_1 \) of \( Z_1 \) and \( \pi_2 \) of \( Z_2 \) can be amalgamated only when they agree on the common subnet \( Z_0 \). This idea is formalised by resorting to the notion of consistent occurrence nets. In the rest of this section we will refer to fixed composition \( Z_3 = Z_1 \oplus Z_2 \) of open nets \( Z_1 \) and \( Z_2 \) over \( Z_0 \).
Definition III.6 (agreement of deterministic processes)
We say that two deterministic processes \( \pi_1 : K_1 \to Z_1 \) and \( \pi_2 : K_2 \to Z_2 \) agree on \( Z_0 \) if \( \pi_1|_{Z_0} = \pi_2|_{Z_0} = \langle K_0, \pi_0 \rangle \), that is, their projections to \( Z_0 \) coincide, and the corresponding occurrence nets \( K_1 \) and \( K_2 \) are consistent.

If two processes of \( Z_1 \) and \( Z_2 \) agree on \( Z_0 \), they can be amalgamated as follows.

Construction III.3 (amalgamation of processes) Let \( \pi_i : K_i \to Z_i \) (\( i \in \{1,2\} \)) be deterministic processes that agree on \( Z_0 \) and let \( K_1, K_2 \) be the corresponding consistent occurrence nets with \( K_0 = K_1 \cap K_2 \). Then, the amalgamation \( \pi_3 : K_3 \to Z_3 \) of \( \pi_1 \) and \( \pi_2 \), denoted \( \pi_3 = \pi_1 \oplus \pi_2 \), is given by the composed occurrence net \( K_3 = K_1 \oplus K_2 \) together with the mapping \( \pi_3 = \pi_1 \cup \pi_2 \) obtained as the union of the given process mappings.

Moreover, given two sets of processes \( P_1 \) over \( Z_1 \) and \( P_2 \) over \( Z_2 \), by \( P_1 \oplus P_2 \) we denote the set \( \{ \pi_1 \oplus \pi_2 \mid \pi_i \in P_i \, \text{and} \, \pi_1, \pi_2 \text{agree on } Z_0 \} \).

In [BCEH01] it is shown that each deterministic process \( \pi_3 \) of the composed net \( Z_3 \) can be decomposed into processes \( \pi_1 \) and \( \pi_2 \) over \( Z_1 \) and \( Z_2 \), respectively, such that \( \pi_1 \) and \( \pi_2 \) agree on \( Z_0 \). Moreover, amalgamation and decomposition of processes are inverse to each other, up to isomorphism of processes. That means, the set of processes of \( Z_3 \) can be computed from the processes of \( Z_1 \) and \( Z_2 \).

Theorem III.1 (amalgamation) Let \( Z_1 \) and \( Z_2 \) be composable open nets with intersection \( Z_0 \). Then, up to isomorphism of processes, \( \text{Proc}(Z_1 \oplus Z_2) = \text{Proc}(Z_1) \oplus \text{Proc}(Z_2) \).

This theorem is exemplified by Fig. 8: Two processes for the component nets Output Resources and Output Flow, which agree on the shared subnet Interface, can be amalgamated to produce a process for the composed net Output. Vice versa, each process of the net Output can be reconstructed as amalgamation of compatible processes of the component nets.

IV. CONTROL SYNTHESIS AS COMPOSITION OF OPEN NETS

After having introduced open nets as a compositional notion of Petri net with interfaces, next we shall formulate the semantics of the control synthesis approach [CSO98] in terms of this concept. Further, we shall extend the synthesis idea to plants and controllers consisting of several components themselves.

A. Specification and realization

In the terminology of Section III-A, a plant model is an open net \( P \) where actuators and sensors are open places both for input and output. A controller specification \( S \) expresses constraints on the set of open processes \( \text{Proc}(P) \) of the net \( P \). As the specification \( S \) refers to the names of places of \( P \), we can think of specifications as typed over open nets. The synthesis problem consists in constructing an open net \( C \) from a (correctly typed) specification \( S \), such that \( P \) and \( C \) are composable, and if we project the open processes over \( P \oplus C \) back to \( P \), we obtain the processes over \( P \) satisfying the constraints in \( S \).

To formalise this idea in an axiomatic way (that is, independently of the temporal specification language) we assume a logic for open nets with a notion of satisfaction of temporal constraints by open processes.

Definition IV.1 (specification logic for open nets) A specification logic for open nets \( \text{Spec} \) provides for every open net \( P \) a set \( \text{Spec}(P) \) of specifications over \( P \) together with a function \( \text{Proc}(P,S) \) which associates to each specification \( S \) over \( P \) the set \( \text{Proc}(P,S) \) of all processes over \( P \) that satisfy \( S \).

The specification \( S \), expressing properties of \( P \)'s processes, is realizable if a controller net exists which restricts the behaviour of \( P \) to just the open processes satisfying the specification.

Definition IV.2 (realization) Assume a specification \( S \) over an open net \( P \). A realization for \( S \) is an open net \( C \) such that \( C \) and \( P \) are composable and for the embedding of \( P \) into the composed net \( P \oplus C \) we have \( \text{Proc}(P,S) = \{ \pi | \pi \text{ is open process over } P \oplus C \} \).

Since we do not make further assumptions, not all specifications have a realization. For example, if we consider a plant net without open places, the only possible composition \( P \oplus C \) is a disjoint union. In this case, we cannot influence the behaviour of \( P \) by \( C \), that is, only trivial specifications would have a realization.

Notice a difference between the notions of synthesis and realization: The synthesis produces a controller net which enforces the plant net to execute (at least) one process out of the set of admissible processes. The realization asks for a controller net which realizes all the admissible processes. In fact, ideally, the synthesis algorithm should select among the many realizations one that leads to an efficient use of the properties of the plant. This optimisation problem is out of the scope of this paper.

B. Compositional realization

In our sample automation system, the plant is structured into the phases of input and processing, and of output. Thus, it may be desirable both for reducing the complexity of the synthesis problem and for supporting reuse of controller components, to apply the synthesis algorithm to each single phase and to compose afterwards the resulting controllers. In order to do so, we have to start from separate specifications for the two controllers which, when combined, lead to the desired restriction of the overall plant model. Figure 9 shows the composition of our sample controller net (cf. Fig. 6) from two components for the input & processing and the output phase. Notice the creation of the cycle by gluing the final place of either phase with the initial place of the other.
A specification logic which supports this kind of composition of specifications over open net components shall be called compositional. This requirement is captured by the following axioms.

**Definition IV.3 (compositional specification logic)** A specification logic for open nets \( \langle \text{Spec}, \text{Proc} \rangle \) is called compositional if for composable open nets \( P_1 \) and \( P_2 \)

1. \( S_1 \in \text{Spec}(P_1) \) and \( S_2 \in \text{Spec}(P_2) \) implies that \( S_1 \cup S_2 \in \text{Spec}(P_1 \uplus P_2) \)

2. \( \text{Proc}(P_1, S_1) \uplus \text{Proc}(P_2, S_2) = \text{Proc}(P_1 \uplus P_2, S_1 \cup S_2) \)

The first axiom means that, if \( S_1 \) is correctly typed over \( P_1 \) and the same holds for \( S_2 \) and \( P_2 \), then the union (conjunction) of \( S_1 \) and \( S_2 \) is defined and well-typed over \( P_1 \uplus P_2 \). The second axiom requires that the combination of specifications is reflected by the amalgamation of the admissible processes.

Under these assumptions we can prove the following theorem, which states the correctness of the compositional realization.

**Theorem IV.1 (compositional realization)** Assume a compositional specification logic, composable open nets \( P_1 \) and \( P_2 \), and specifications \( S_1 \) and \( S_2 \) over \( P_1 \) and \( P_2 \), respectively. Assume, moreover, realizations \( C_1 \) of \( S_1 \) and \( C_2 \) of \( S_2 \) which are also composable. Then, the open net \( C_1 \uplus C_2 \) realizes the specification \( S_1 \cup S_2 \) over \( P_1 \uplus P_2 \).

**Proof:** We have to show that \( \text{Proc}(P_1 \uplus P_2, S_1 \cup S_2) = \text{Proc}(P_1 \uplus P_2 \uplus C_1 \uplus C_2) \mid |_{P_1 \uplus P_2} \). By the axioms of Definition IV.3, the left-hand side is equal to \( \text{Proc}(P_1, S_1) \uplus \text{Proc}(P_2, S_2) \) which, in turn, is equal to \( \text{Proc}(P_1 \uplus C_1) \mid |_{P_1 \uplus P_2} \text{Proc}(P_2 \uplus C_2) \mid |_{P_1 \uplus P_2} \) because \( \text{Proc}(P_1, S_1) = \text{Proc}(P_1 \uplus C_1) \mid |_{P_1} \) and \( \text{Proc}(P_2, S_2) = \text{Proc}(P_2 \uplus C_2) \mid |_{P_2} \) by the assumption that \( C_1 \) and \( C_2 \) are realizations of \( S_1 \) and \( S_2 \), respectively. Due to the compatibility of unions and inverse images, where these constructions are based upon, projection and composition can be exchanged. Therefore, \( \uplus \) is equal to \( \text{Proc}(P_1 \uplus C_1 \uplus \text{Proc}(P_2 \uplus C_2)) \mid |_{P_1 \uplus P_2} \). Finally, this is the same like \( \text{Proc}(P_1 \uplus P_2 \uplus C_1 \uplus C_2) \mid |_{P_1 \uplus P_2} \) by the amalgamation theorem.

Notice that the proof makes use of the fact that the global net can be put together in different ways: first horizontally, then vertically as \( (P_1 \uplus P_2) \uplus (C_1 \uplus C_2) \) or first vertically and then horizontally as \( (P_1 \uplus C_1) \uplus (P_2 \uplus C_2) \). Due to the formalisation of composition by colimits [BCEH01], the result is essentially the same in both cases, that is, \( \uplus \) is commutative and associative up to isomorphism.

**V. Conclusion**

The long-term aim of this ongoing work is the transfer of ideas and technology between engineering and computer science. In this exchange, engineering provides a rich experience in designing and implementing reliable complex systems, which could help to improve the quality of software development processes. Computer science can supply computational concepts with sound semantic foundations which can back up existing methodologies in engineering and inspire new developments. In particular, in this paper we have studied properties of a logic for controller specification which would allow for the compositional synthesis.
of controllers.

From new applications, new questions arise, which drive the further development of the concepts and their theory. In studying the application of open nets to control synthesis, we noticed a number of shortcomings, in particular, with respect to the expressiveness of nets. First of all, the approach of [CSO98] is based on high-level rather than place-transition nets. In fact, a high-level version of open nets has already been discussed [PJE+01]. It remains, however, to lift the compositionality results to a notion of high-level (open) process. Based on high-level nets, a notion of nets with time can be realized by considering time as a distinguished data type [GMMP91].

An issue related to the net structure is the use of read arcs and inhibitor arcs, which have been simulated by cycles and complement places in this paper. Here, recent work on processes and unfoldings for both extensions is available [MR95], [Bal00] which one could lift to open nets as it has been done for processes of ordinary nets. Finally, the notion of composition employed here is based on merging nets at common places, while there is no way to synchronise two transitions unless they have exactly the same places in the pre- and post-domain. A corresponding extension would require a different notion of subnet which allows to add places to the pre- and post-domain of a transition.

Solutions to all these problems have been studied in isolation, but it is an open question if they can be integrated into one formalism.

REFERENCES


