Preserving Correctness During Business Process Model Configuration

Wil M.P. van der Aalst[1,3], Marlon Dumas[2,3], Florian Gottschalk[1],
Arthur H.M. ter Hofstede[3], Marcello La Rosa[3] and Jan Mendling[4]

[1]Eindhoven University of Technology, The Netherlands.
[2]University of Tartu, Estonia.
[3]Queensland University of Technology, Australia.
[4]Humboldt-University, Berlin.

Abstract. A configurable process model captures a family of related process models in a single artifact. Such models are intended to be configured to fit the requirements of specific organizations or projects, leading to individualized process models that are subsequently used for domain analysis or solution design. This article proposes a formal foundation for individualizing configurable process models incrementally, while preserving correctness, both with respect to syntax and behavioral semantics. Specifically, assuming the configurable process model is behaviorally sound, the individualized process models are guaranteed to be sound. The theory is first developed in the context of Petri nets and then extended to a process modeling notation widely used in practice, namely Event-driven Process Chains.

Keywords: Configurable process model, configuration, Petri net

1. Introduction

The design of business process models is labor-intensive, especially when such models are required to be detailed enough to support the development of software systems. To avoid the effort of creating process models from scratch, several consortia and vendors have defined so-called *reference process models*. These models capture proven practices and recurrent business operations in a given domain. They are designed in a generic manner and are intended to be individualized to fit the requirements of specific organizations or IT projects. Commercial process modeling tools come with standardized libraries of reference process models such as the IT Infrastructure Library (ITIL)[1] or the Supply Chain Operations Reference (SCOR) model [Ste01]. Also, the SAP Reference Model [CK97] incorporates a collection of process models corresponding to common business operations supported by SAP's Enterprise Resource Planning (ERP) system.

Reference process models in commercial use lack an explicit representation of configuration alternatives and decisions. As a result, their individualization is entirely manual [RA07]. Analysts take the reference models merely as a source of inspiration, but ultimately, they design their own model on the basis of the reference model, with little guidance as to which model elements need to be removed, added or modified

---

[1] `www.itil-officialsite.com`.

Correspondence and offprint requests to: Wil M.P. van der Aalst, Eindhoven University of Technology P.O. Box 513, 5600MB Eindhoven, The Netherlands. e-mail: w.m.p.v.d.aalst@tue.nl

to address a given requirement. To address this shortcoming, we introduced the concept of *configurable process models* [RA07] in previous work. A configurable process model represents multiple variants of a business process model in an integrated manner. In line with methods from the field of software product lines [PBL05], these alternatives are captured as *variation points*. That means, instead of having to add or remove model elements manually, the fact that a task in a reference process model may or may not appear in an individualized model is captured by attaching a variation point to that task allowing users to select or deselect it. Individualized models are obtained from configurable models by interpreting the values for each variation point. In separate work we also reported on experiences from using configurable process models in practice [LHRS08, GWJV$^+$09]. These studies showed that configurable process modeling provides an intuitive mechanism to combine different process variants in a single model and is preferred to a traditional process modeling paradigm.

While configurable process models provide guidance to analysts during individualization, they do not guarantee that the individualized models are correct, whether syntactically or semantically. For example, if a model element or an entire path in a reference process model is removed during configuration, the remaining model elements need to be re-connected to maintain syntactic correctness. Also, the configuration of variation points attached to parallel splits, decision points and synchronization points in a configurable process model may lead to the introduction of deadlocks. And if the individualized process model contains such semantic errors, it needs to be manually fixed.

The contribution of this article is a formal framework for configuring reference process models in a correctness-preserving manner. The framework includes a technique to derive propositional logic constraints that, if satisfied by a configuration step, guarantee the syntactic correctness of the resulting model. We prove that for a large class of process models, these constraints also ensure that semantic correctness is preserved. The framework supports *staged configuration* [CHE04]. In other words, it allows correctness to be checked at each intermediate step of the configuration procedure. Whenever a value is assigned to a variation point, the current set of constraints is evaluated. If the constraints are satisfied, the configuration step is applied. If on the other hand the constraints are violated, we compute a reduced propositional logic formula, from which we can identify additional variation points that need to be configured simultaneously in order to preserve correctness (e.g. if an edge in the process model is removed, all nodes in a path starting with that edge need to be removed). The set of constraints is incrementally updated after each step of the configuration procedure.

The proposal is intended as a foundation for analyzing properties of configurable process models, particularly with respect to correctness. Accordingly, we initially adopt a Petri net-based representation of process models, thus abstracting from the specificities of process modeling notations used in practice such as UML Activity Diagrams (UML ADs), Event-driven Process Chains (EPCs) or the Business Process Modeling Notation (BPMN). We use a class of Petri nets, namely workflow nets, which are specifically designed to represent business processes [Aal97]. Workflow nets come with a notion of behavioral correctness known as soundness, which ensures the absence of deadlocks and improper completion. In this article, we enhance workflow nets with the notion of variation point, leading to the concept of a configurable Workflow net. We then define a notion of configuration step over such nets and we show how to derive correctness-preserving constraints for such steps. A core result of the article is that, for workflow nets that satisfy the "free-choice" property [DE95], if the outcome of a configuration step starting from a sound Workflow net is a Workflow net, then this latter Workflow net is sound. This means that for this class of nets, configuration steps that preserve syntactic correctness also preserve behavioral correctness.

Having established a formal foundation for process model configuration, we apply it to Configurable Event-driven Process Chains (C-EPCs) [RA07] – a configurable version of the EPC notation. We reuse previous results to link C-EPCs to Petri nets, and we show how the notion of configuration step defined on Petri nets can be adapted to fit the specificities of C-EPCs.

The article is structured as follows. Section 2 introduces workflow nets and the notion of soundness while Section 3 introduces the notion of configurable Workflow net and configuration step. Section 4 discusses the derivation of constraints that guarantee the preservation of syntactic correctness, and proves that these constraints also guarantee soundness for free-choice nets. Chapter 5 shows how the framework can be applied to the individualization of configurable EPCs. The article concludes with a section on related work, a summary, and an outlook on open issues.

2. Background

Petri nets are a formal model of concurrent systems [Mur89]. Petri nets benefit from a rich body of theoretical results, analysis techniques and tools. They have been extensively applied to the formal verification of business process models [VBA01]. These features make Petri nets suitable for establishing a formal foundation for business process model configuration. In addition, mappings exist between process modeling languages used in practice (e.g. UML ADs, EPC, BPMN, BPEL) and Petri nets. These mappings provide a basis for extending the results outlined in this article to concrete process modeling notations.

We use a class of Petri nets, namely workflow nets, specifically designed for business process modeling. Workflow nets have a single starting point and ending point, which captures the intuition that business processes are instantiated, and each process instance progresses independently through a series of activities until completion. A desirable property is that an instance of a Workflow net always completes properly. This is captured by the notion of soundness. To make the article self-contained, we provide an introduction to workflow nets and soundness.

2.1. Workflow nets: Syntax

Petri nets are composed of two types of elements, namely transitions and places, connected by directed arcs. Transitions represent tasks while places represent the status of the system before or after the execution of a transition. Formally:

Definition 1 (Petri net, Preset, Postset). A Petri net is a triple $PN = (P, T, F)$, such that:

- P is a finite set of places,
- T is a finite set of transitions ($P \cap T = \varnothing$),
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (flow relation).

For each node $n \in P \cup T$, we use $\bullet n$ and $n \bullet$ to denote the set of inputs to n (preset) and the set of outputs of n (postset). $\square$

Figure 1 shows a process model for travel requisition approval as a Petri net. It consists of two variants: the left one for international travel and the right one for domestic travel. After requesting a quote for international travel, either the employee or an assistant prepares the travel requisition form. In case of the latter, the employee needs to check the form before submitting it for approval. The administrator can then approve or reject the requisition, or make a request for change. At this point, the employee can update the form according to the administrator's suggestions and re-submit it, or drop the case. In contrast, the application for domestic travel only requires the employee to ask for a quote and to report the travel requisition to the administration.

A business process model may be executed a number of times to deal with different cases (e.g. different travel requests in the example). Each of these cases (called *process instances*) has a distinct start (input) and an end (output). Accordingly, we are only interested in Petri nets with a unique source place (representing the input) and a unique sink place (output), and such that all other nodes are on a directed path between the input and the output places. A Petri net satisfying these conditions represents a *structurally correct* process model and is known as a *Workflow net* [Aal97]. Formally:

Definition 2 (Workflow net). Let $PN = (P, T, F)$ be a Petri net and F^* be the reflexive transitive closure of F. PN is a Workflow net (WF-net) iff:

- there exists exactly one input place, i.e. $\exists!_{p_I \in P} \ \bullet p_I = \varnothing$, and
- there exists exactly one output place, i.e. $\exists!_{p_O \in P} \ p_O \bullet = \varnothing$, and
- each node is on a directed path from the input place to the output place, i.e. $\forall_{n \in P \cup T} \ [(p_I, n) \in F^* \ \wedge (n, p_O) \in F^*]$. $\square$
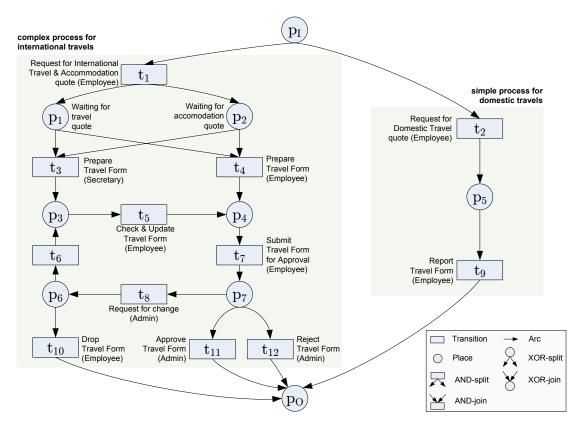
The Petri net in Figure 1 is a *WF*-net.

Fig. 1. Reference model for travel form approval.

2.2. Workflow nets: Semantics

Behavioral correctness of a *WF*-net is defined with respect to the states that a process instance can be in during its execution. A state of a *WF*-net is represented by the marking of its places with tokens. In other words, in a given state, each place is either empty, or it contains one or more tokens (i.e. it is marked). A transition is enabled in a given marking, if all the places in the transition's preset are marked. Once enabled, the transition can fire (i.e. can be executed) by removing a token from each place in the preset and putting a token into each subsequent place of the transition's postset. This leads to a new state. Formally:

Definition 3 (Marking, Enabling Rule, Firing Rule). Let $N = (P, T, F)$ be a *WF*-net with source place p_I and sink place p_O:

- $M : P \to \mathbb{N}$ is a marking of N and $\mathbb{M}(N)$ is the set of markings of N,
- M_I is the initial marking of N with one token in place p_I, i.e. $M_I = [p_I]$,
- M_O is the final marking of N with one token in place p_O, i.e. $M_O = [p_O]$,
- $M(p)$ returns the number of tokens in place p if $p \in dom(M)$,
- For any two markings $M, M' \in \mathbb{M}(N)$, $M \geq M'$ iff $\forall_{p \in P} M(p) \geq M'(p)$,
- For any transition $t \in T$ and any marking $M \in \mathbb{M}(N)$, t is *enabled* at M, denoted as $M[t\rangle$, iff $\forall_{p \in \bullet t} M(p) \geq 1$. Marking M' is reached from M by firing t and $M' = M - \bullet t + t \bullet$,
- For any two markings $M, M' \in \mathbb{M}(N)$, M' is reachable from M in N, denoted as $M' \in N[M\rangle$, iff there exists a firing sequence $\sigma = \langle t_1, t_2, ..., t_n \rangle$ leading from M to M', and we write $M \xrightarrow{\sigma}_N M'$. If $\sigma = \langle t \rangle$, we use the notation $M \xrightarrow{t}_N M'$. N can be omitted if clear from the context. □

The execution of a process instance starts with the state in which the input place has one token and no other place is marked. The execution of this process instance should then progress through transition firings until

a proper completion state. This intuition is captured by three requirements [Aal97]. Firstly, every process instance should always have the option to complete. If a *WF*-net satisfies this requirement, it will never run into a deadlock or livelock. Secondly, every process instance should eventually reach the state in which there is one token in the output place p_O, and no tokens are left behind in any other place, since this would signal that there is still work to be done. Thirdly, for every transition, there should be at least one execution sequence from the initial marking (where only p_I is marked) to the final marking (where only p_O is marked) that includes at least one firing of this transition. In other words, no transition in the *WF*-net should be spurious. A *WF*-net fulfilling these requirements is *sound*. Formally:

Definition 4 (Sound WF-net). Let $N = (P, T, F)$ be a *WF*-net and M_I, M_O be the initial and end markings. N is sound iff:

- option to complete: for every marking M reachable from M_I, there exists a firing sequence leading from M to M_O, i.e. $\forall_{M \in N[M_I\rangle} \ M_O \in N[M\rangle$, and
- proper completion: the marking M_O is the only marking reachable from M_I with at least one token in place p_o, i.e. $\forall_{M \in N[M_I\rangle}[\ M \geq M_O \Rightarrow M = M_O]$, and
- no dead transitions: every transition can be reached by the initial marking, i.e. $\forall_{t \in T} \ \exists_{M \in N[M_I\rangle} \ M[t\rangle$. $\square$

3. Process Model Configuration

There are several ways to capture variation points for the purpose of representing a configurable process model [CA05, GAJV07, RA07]. In this article we choose the approach presented in [GAJV07], which is based on the concept of inheritance of process behavior [AB02], since it abstracts from vendor-specific process modeling notations and can easily be applied to Petri nets. Accordingly, we define the notion of *configurable WF-net*, where each transition captures a variation point whose possible values (or *variants*) are: *allowed*, *hidden* and *blocked*.

Hiding a transition refers to skipping its execution while it is fired, without affecting the rest of the process flow. Consider for example the *WF*-net in Figure 1. Some organizations may not require a quote for domestic travels. Thus, the task to request a quote can be skipped from the process model by hiding transition t_2. The process continues without forcing the employee to request a quote.

Blocking a transition implies to inhibit it in the process model. Blocked transitions cannot forward cases and all the subsequent transitions will never be executed if they cannot be enabled via other paths. For example, if t_2 in Figure 1 is blocked, the process for domestic travels cannot be triggered and all travel approvals must be done via the complex variant.

If a transition is neither blocked nor hidden, we say it is allowed, meaning nothing changes in the model. To configure a *WF*-net each transition has to be assigned one value among hidden, blocked or allowed. Formally:

Definition 5 (WF-net Configuration). Let $N = (P, T, F)$ be a *WF*-net, then $\mathcal{C}_N \in T \rightarrow \{allow, hide, block\}$ is a *configuration* for N. We define:

- $A_N^{\mathcal{C}} = \{t \in T \mid \mathcal{C}_N(t) = allow\}$ as the set of all allowed transitions,
- $H_N^{\mathcal{C}} = \{t \in T \mid \mathcal{C}_N(t) = hide\}$ as the set of all hidden transitions,
- $B_N^{\mathcal{C}} = \{t \in T \mid \mathcal{C}_N(t) = block\}$ as the set of all blocked transitions.[2] $\square$

Based on these configuration values, a configured net is obtained representing the new behavior of the process model. This new Petri net is a restriction of the behavior of the starting model (the reference model), where all the hidden transitions are replaced by silent *skip* transitions and all the blocked transitions are removed. Also, all the places connected only to blocked transitions and all the flow relations from/to blocked transitions have to be removed too. Formally:

Definition 6 (Configured Petri net). Let $N = (P, T, F)$ be a WF-net and let $\mathcal{C}_N$ be a configuration of N. The resulting configured net $\beta_N(N, \mathcal{C}_N) = (P^{\mathcal{C}}, T^{\mathcal{C}}, F^{\mathcal{C}})$ is defined as follows:

- $T^{\mathcal{C}} = (T \setminus (B_N^{\mathcal{C}} \cup H_N^{\mathcal{C}})) \ \cup \ \{skip_t \mid t \in H_N^{\mathcal{C}}\}$,

---

[2] $A_N^{\mathcal{C}} \cap H_N^{\mathcal{C}} \cap B_N^{\mathcal{C}} = \varnothing$ follows from the definition of N.
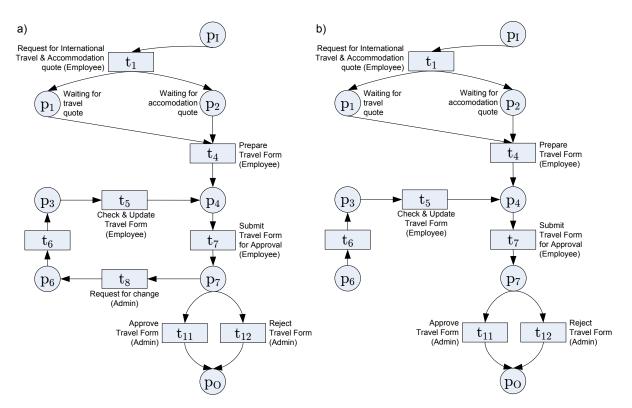
Fig. 2. a) Correct configured net b) Incorrect configured net.

- $F^{\mathcal{C}} = (F \cap ((P \cup T^{\mathcal{C}}) \times (P \cup T^{\mathcal{C}}))) \cup \{(p, skip_t) \mid (p, t) \in F \wedge t \in H_N^{\mathcal{C}}\} \cup \{(skip_t, p) \mid (t, p) \in F \wedge t \in H_N^{\mathcal{C}}\},$
- $P^{\mathcal{C}} = (P \cap \bigcup_{(x,y) \in F^c} \{x, y\}) \cup \{p_I, p_O\}.$ □

As an example, Figure 2a shows a configured net derived from the *WF*-net in Figure 1, where the transitions t_2 and t_9 have been blocked to allow the complex approval process only. In this net employees have to prepare the approval form on their own, as t_3 has been blocked, and cannot drop a form application if a change is requested after approval (t_{10} also blocked). Place p_5 has been removed as it became disconnected after removing t_2 and t_9.

A process configuration has to comply with the requirements of the domain. This may prevent users from configuring the values of transitions freely. For example, in the travel management domain, if an employee submits a travel form for approval there must be at least an option to accept the request and an option to reject it. This is clearly a requirement of the domain, which forbids users to block both t_{11} and t_{12} in the process model. In [LLS+07] we showed how propositional logic expressions can be used to encode domain constraints. By evaluating each transition's value against these constraints with a SAT solver, it is possible to prevent all the configurations which would violate the constraints.

Nonetheless, the set of constraints derived from the domain are in most cases not sufficient to guarantee the syntactic and semantic correctness of the configured model. Indeed, as per Definition 6, a configured net can be any Petri net, which means that it can contain elements that are not on a path from p_I to p_O, or which are completely disconnected. For example, forbidding the request for a change by blocking t_8 in the *WF*-net of Figure 2a would make p_6, t_6, p_3 and t_5 unreachable, yielding the net of Figure 2b. This configured net is not syntactically correct and hence not semantically correct either, according to Definition 4. So, as soon as t_3 and t_8 are blocked, it would be desirable to suggest the user to block t_6 and t_5 too, so as to get rid of the unreachable branch. In the following section we present an approach to automatically derive a set of constraints from a *WF*-net that preserve the model correctness during its configuration.

4. Correctness-Preserving Configuration

Existing tools like Woflan [VBA01] support the verification of Petri net-based process models. These tools could be used to check every single configured net that can be derived from a reference process model. If the net is incorrect, the configuration that has generated this net should be excluded from the set of possible configurations. However, this approach is costly, considering that reference process models can potentially yield thousands of individualized process models.

Our aim is therefore to define a framework which allows incorrect configuration steps to be discarded incrementally and without computing all possible configurations of the reference model. In addition, the framework needs to seamlessly integrate the domain constraints, so that a user can derive a correct process model which also satisfies any domain constraints.

To this end, we complement the domain constraints with a set of process constraints to guarantee the preservation of syntactic and semantic correctness in the configured net. Both sets of constraints are captured in propositional logic over the nodes of a *WF*-net and are reduced by a BDD solver. In this way we can provide interactive support to the user, by pinpointing the impact of each configuration step on the resulting net and by eliminating unfeasible options.

4.1. Preserving syntactic correctness

In a staged configuration, users make configuration decisions one after another in steps, and the set of configuration options is recalculated after each step. To remain syntactically correct, a *WF*-net must thus be checked on which configuration options are still viable among the transitions that have not been configured yet. For this, we have to consider the configuration decisions already taken.

To distinguish nodes which remain in the net from nodes which do not, we use a boolean variable for each node. If the variable is set to *true*, the node remains part of the net; if it is set to *false*, the node is dropped in the configured net. Accordingly, we assign a blocked transition the value *false*, while a transition that is allowed or hidden is assigned the value *true*. Since silent transitions have the same routing behavior as the original transitions, we do not need to distinguish hidden from allowed transitions. All transitions that are not explicitly configured remain as variables (i.e. unset).

According to Definition 6, any internal place remains in the net if there is a non-blocked transition in its preset or postset. Translating this definition in boolean logic, if one such transition is *true*, the place has also to be set to *true*; if all the connected transitions are *false*, the place has to be set to *false*; if some transitions have no value assigned yet, the place remains unset. Since a configuration is defined over the transitions of a net, we have to derive the values of the places. We do that by imposing that each transition set to *true* implies *true* for all the places in its preset and in its postset. Formally: $\bigwedge_{t \in T^c} [\ t \Rightarrow \bigwedge_{p \in \bullet t} p \ \wedge \bigwedge_{p \in t\bullet} p].$[3]

Assuming the original net is a *WF*-net, to guarantee the configured net is still a *WF*-net, we have to ensure that each node that remains in the configured net be on a directed path from p_I to p_O. This is the only requirement of *WF*-net to be verified, as p_I and p_O are part of the configured net by definition. This means all the nodes composing the directed path should not be *false*. For each node, we can decompose this path into two sub-paths: one from p_I to the node in question and the other from the node to p_O, and verify the property over the nodes of each sub-path. However, as per Definition 6, we can restrict the verification to the places of each sub-path, by deriving the places' values from the ones of the transitions. Indeed, if a non-blocked transition has at least one place in its preset on a directed path from p_I and at least one place in its postset on a directed path to p_O, then the transition is on a directed path from p_I to p_O. When searching for such paths we can restrict our analysis to acyclic paths. In fact a cycle always leads back to the same node, but does not provide any valuable progress from p_I to p_O. Formally, we define an acyclic path as follows:

Definition 7 (Acyclic Path). Let $PN = (P, T, F)$ be a Petri net:

- $\phi = \langle n_1, n_2, ..., n_k \rangle$ is an *acyclic path* of PN such that $(n_i, n_{i+1}) \in F$ for $1 \leq i \leq k-1$ and $i \neq j \Rightarrow n_i \neq n_j$,
- $\alpha(\phi) = \{n_1, n_2, ..., n_k\}$ is the alphabet of ϕ,
- Φ_{PN} is the set of all acyclic paths of PN;

---

[3] Where with t, p we indicate a transition, resp. a place, which is set to *true*.

- for all $n \in P \cup T$, $AC_I(n) = \{\phi \in \Phi_{PN} \mid \phi = \langle p_I, ..., n \rangle\}$ is the set of all acyclic paths from p_I to n,
- for all $n \in P \cup T$, $AC_O(n) = \{\phi \in \Phi_{PN} \mid \phi = \langle n, ..., p_O \rangle\}$ is the set of all acyclic paths from n to p_O. $\quad\square$

The set of process constraints is called PC and is defined as follows:

Definition 8 (Process Constraint). Let $N = (P, T, F)$ be a WF-net. Treating each place and each transition of N as a propositional variable, the process constraint $PC(N)$ is a propositional logic formula over these variables, given by the conjunction of the following expressions:

- p_I and p_O are always true, i.e. $p_I \wedge p_O$;
- each place p implies the disjunction of all acyclic paths from p_I to p and the disjunction of all acyclic paths from p to p_O: $\bigwedge_{p \in P} [p \Rightarrow \bigvee_{\phi \in AC_I(p)} (\bigwedge_{n \in \alpha(\phi)} n) \wedge \bigvee_{\phi \in AC_O(p)} (\bigwedge_{n \in \alpha(\phi)} n)]$. $\quad\square$

The following theorem shows that any configured net derived from a configuration that satisfies PC is a WF-net.

Theorem 1. Let $N = (P, T, F)$ be a WF-net and $PC(N)$ be its process constraint. Let $\mathcal{C}_N$ be a configuration of N and let $\beta_N(N, \mathcal{C}_N) = (P^{\mathcal{C}}, T^{\mathcal{C}}, F^{\mathcal{C}})$ be the resulting configured net. Let $v \in T \cup P \to \{true, false\}$ be such that $v(q) = true$ iff $q \in T^{\mathcal{C}} \cup P^{\mathcal{C}}$. Then $\beta_N(N, \mathcal{C}_N)$ is a WF-net $\Leftrightarrow v \models PC(N)$.

Proof. ($\Rightarrow$) Let $\beta_N(N, \mathcal{C}_N)$ be a WF-net and let $v \in T \cup P \to \{true, false\}$ such that $v(n) = true$ iff $n \in T^{\mathcal{C}} \cup P^{\mathcal{C}}$. As $p_I \in P^{\mathcal{C}}$ and $p_O \in P^{\mathcal{C}}$ (Definition 6), $v(p_I) = true$ and $v(p_O) = true$, hence $v \models p_I \wedge p_O$. Since $\beta_N(N, \mathcal{C}_N)$ is a WF-net, for all $p \in P^{\mathcal{C}}$ there exists at least one directed path from p_I to p. Let $\phi \in AC_I(p)$ be such a path, thus for all $n \in \alpha(\phi)$ we have $n \in P^{\mathcal{C}} \cup T^{\mathcal{C}}$, hence $v(n) = true$. Therefore, $v \models \bigwedge_{n \in \alpha(\phi)} n$. Hence, $v \models \bigvee_{\phi \in AC_I(p)} (\bigwedge_{n \in \alpha(\phi)} n)$. Similarly, as there is at least one path from p to p_O, $v \models \bigvee_{\phi \in AC_O(p)} (\bigwedge_{n \in \alpha(\phi)} n)$, hence $v \models \bigvee_{\phi \in AC_I(p)} (\bigwedge_{n \in \alpha(\phi)} n) \wedge \bigvee_{\phi \in AC_O(p)} (\bigwedge_{n \in \alpha(\phi)} n)$. Thus, for all $p \in P^{\mathcal{C}}$ $v \models \bigvee_{\phi \in AC_I(p)} (\bigwedge_{n \in \alpha(\phi)} n) \wedge \bigvee_{\phi \in AC_O(p)} (\bigwedge_{n \in \alpha(\phi)} n)$ and therefore for all $p \in P^{\mathcal{C}}$ $v \models p \Rightarrow \bigvee_{\phi \in AC_I(p)} (\bigwedge_{n \in \alpha(\phi)} n) \wedge \bigvee_{\phi \in AC_O(p)} (\bigwedge_{n \in \alpha(\phi)} n)$. If $p \in P \setminus P^{\mathcal{C}}$ then $v(p) = false$ and thus $v \models p \Rightarrow \bigvee_{\phi \in AC_I(p)} (\bigwedge_{n \in \alpha(\phi)} n) \wedge \bigvee_{\phi \in AC_O(p)} (\bigwedge_{n \in \alpha(\phi)} n)$. Hence $v \models \bigwedge_{p \in P} [p \Rightarrow \bigvee_{\phi \in AC_I(p)} (\bigwedge_{n \in \alpha(\phi)} n) \wedge \bigvee_{\phi \in AC_O(p)} (\bigwedge_{n \in \alpha(\phi)} n)]$.

($\Leftarrow$) Let $v \models PC(N)$. Assume $\beta_N(N, \mathcal{C}_N)$ is not a WF-net. Since p_I and p_O belong to $\beta_N(N, \mathcal{C}_N)$ by definition, choose $p \in P^{\mathcal{C}}$ such that there is either (1) no path from p_I to p or (2) no path from p to p_O. If (1) then for all $\phi \in AC_I(p)$ there is a node $n \in \alpha(\phi)$ such that $n \notin P^{\mathcal{C}} \cup T^{\mathcal{C}}$ and thus $v(n) = false$, $v \not\models n$ and hence for all $\phi \in AC_I(p)$ $v \not\models \bigwedge_{n' \in \alpha(\phi)} n'$ and thus $v \not\models \bigvee_{\phi \in AC_I(p)} (\bigwedge_{n' \in \alpha(\phi)} n')$. If (2) then for all $\phi \in AC_O(p)$ there is a node $n \in \alpha(\phi)$ such that $n \notin P^{\mathcal{C}} \cup T^{\mathcal{C}}$ and thus $v(n) = false$, $v \not\models n$ and hence for all $\phi \in AC_O(p)$ $v \not\models \bigwedge_{n' \in \alpha(\phi)} n'$ and thus $v \not\models \bigvee_{\phi \in AC_O(p)} (\bigwedge_{n' \in \alpha(\phi)} n')$. From both cases we can conclude $v \not\models \bigvee_{\phi \in AC_I(p)} (\bigwedge_{n' \in \alpha(\phi)} n') \wedge \bigvee_{\phi \in AC_O(p)} (\bigwedge_{n' \in \alpha(\phi)} n')$. Given that $v \models p$, $v \not\models p \Rightarrow \bigvee_{\phi \in AC_I(p)} (\bigwedge_{n' \in \alpha(\phi)} n') \wedge \bigvee_{\phi \in AC_O(p)} (\bigwedge_{n' \in \alpha(\phi)} n')$. This implies that $v \not\models \bigwedge_{p \in P} [p \Rightarrow \bigvee_{\phi \in AC_I(p)} (\bigwedge_{n' \in \alpha(\phi)} n') \wedge \bigvee_{\phi \in AC_O(p)} (\bigwedge_{n' \in \alpha(\phi)} n')]$, hence $v \not\models PC(N)$ (Contradiction). $\quad\square$

PC has to be satisfied over a system of variables represented by the nodes of the net, where the values of the transitions are configured by the user and the values of the places are derived automatically. Checking the satisfiability of PC is an NP-complete problem. To overcome this issue, we propose to use a SAT solver[4] based on Shared Binary Decision Diagrams (SBDDs). Existing SBDD solvers can efficiently deal with systems made up of around one million possibilities [MIY90]. Hence they are reasonably adequate to capture all the configurations produced by a reference process model.

We propose to use the solver to obtain a reduced representation of PC in conjunctive normal form, where each variable is initially unset. Then we conjunct this formula with each new transition valuation as provided by the user during the configuration process, and further reduce the formula. In this way we do not recalculate PC for each configuration step. The solver can only reduce the formula if this is satisfiable, i.e. if the configuration can yield a syntactically correct process model. This may imply to automatically force to *true* or *false* the conjunction or disjunction of other transitions which are still unset, in order to keep the

---

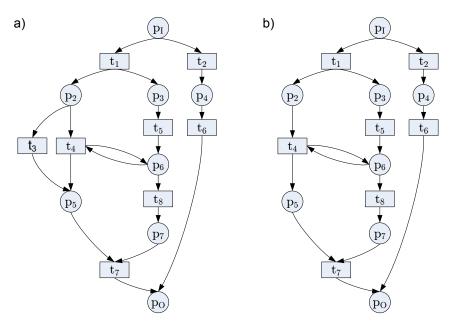[4] Available at `www-verimag.imag.fr/~raymond/tools/bddc-manual`.

Fig. 3. Blocking t_3 in (a) leads to an unsound WF-net (b)

formula satisfiable. For example, after blocking t_8 in the model of Figure 2a, the solver would force to *false* t_5 and t_6 as well.

This solver can be embedded in a tool to support staged configuration of process models, where invalid configurations are identified when a configuration step is applied and alternatives are suggested to keep the model correct.

4.2. Preserving Semantic Correctness

In addition to structural correctness, a configuration should be semantically correct. The example in Figure 3 shows that a configuration conforming to the WF-net properties is not automatically sound, even if it is derived from a sound WF-net. The WF-net in (a) is a sound WF-net: if t_8 fires before t_4, the token in p_2 can reach p_5 via t_3. However, if t_3 is blocked (b), t_4 needs to fire before t_8 as t_4 depends on the token in p_6 which is removed when t_8 fires. Since this behavior is not enforced in the net, the process might deadlock, and is therefore not sound, although (b) is still a valid WF-net.
Soundness is only defined for WF-nets (Definition 2), but it can be generalized to any Petri net with a designated source and sink place. However, it is easy to show that any non WF-net would still violate this generalized soundness notation. Therefore, the process constraint defined in Definition 8 is a necessary requirement for soundness, but as Figure 3 shows, it is not sufficient.

Below, we prove that PC is a sufficient requirement to guarantee soundness of a configured net, if the original model is a sound extended free-choice WF-net. The restriction to this class of Petri nets provides a good compromise between expressiveness and verification complexity. Not only do extended free-choice WF-nets have several desirable properties [DE95], but the large majority of constructs of process modeling languages such as EPCs, BPMN or BPEL can be mapped to Petri nets in this class [Aal99, AL08, ODA$^+$09]. An extended free-choice is defined as follows [Mur89]:

Definition 9 (Extended Free-choice WF-net). Let $N = (P, T, F)$ be a Petri net. N is *extended free-choice* (*eFC*) if for every couple of places sharing transitions in their postset, these postsets coincide, i.e. $\forall_{p_1, p_2 \in P} [p_1 \bullet \cap p_2 \bullet \neq \varnothing \Rightarrow p_1 \bullet = p_2 \bullet]$. □

Assuming the reference process model is a sound, *eFC* WF-net, we are able to identify several configuration properties relevant for the preservation of soundness during the configuration process:

Proposition 1 (Properties of WF-net Configuration). Let $N = (P, T, F)$ be a sound, *eFC* WF-net

with source place p_I and sink place p_O, let $\mathcal{C}$ be a configuration of N, and let $\beta_N(N,\mathcal{C}_N) = (P^{\mathcal{C}}, T^{\mathcal{C}}, F^{\mathcal{C}})$ be the configured net resulting from $\mathcal{C}$. If $\beta_N(N,\mathcal{C}_N)$ is a *WF*-net (i.e. $PC(N)$ evaluates to *true*), then:

a) $\forall_{t \in T^{\mathcal{C}}} \ [(\bullet_N t = \bullet_{\beta_N(N,\mathcal{C}_N)} t) \ \wedge \ (t \bullet_N = t \bullet_{\beta_N(N,\mathcal{C}_N)})]$.

b) $p_I \in P^{\mathcal{C}}$ and $p_O \in P^{\mathcal{C}}$.

c) $\forall_{t \in B_N^{\mathcal{C}}} \ [(\bullet_N t \cap P^{\mathcal{C}} = \varnothing) \ \vee \ \exists_{t' \in T^{\mathcal{C}}} \ (\bullet_N t = \bullet_N t')]$ (a blocked transition is either not consuming any tokens from $P^{\mathcal{C}}$ or there is a transition in $T^{\mathcal{C}}$ with the same input set).

d) $\forall_{\sigma \in T^{\mathcal{C}*}} \ [(M_I \xrightarrow{\sigma}_N) \Leftrightarrow (M_I \xrightarrow{\sigma}_{\beta_N(N,\mathcal{C}_N)})]$ (the input and output sets of transitions in $T^{\mathcal{C}}$ are the same in both nets, therefore, the respective behaviors are identical when considering only firing sequences $\sigma \in T^{\mathcal{C}*}$).

e) $\forall_{\sigma \in T^{\mathcal{C}*}} \ \forall_M \ [(M_I \xrightarrow{\sigma}_N M) \Leftrightarrow (M_I \xrightarrow{\sigma}_{\beta_N(N,\mathcal{C}_N)} M)]$.

f) $\beta_N(N,\mathcal{C}_N)[M_I\rangle \subseteq N[M_I\rangle$ (all firing sequences of $\beta_N(N,\mathcal{C}_N)$ are also possible in N).

g) $\beta_N(N,\mathcal{C}_N)$ is *eFC*.

h) $\forall_{M \in \beta_N(N,\mathcal{C}_N)[M_I\rangle \setminus \{M_O\}} \ \exists_{t' \in T^{\mathcal{C}}} \ [M[t'\rangle]$ ($\beta_N(N,\mathcal{C}_N)$ has no deadlock markings).

Proof.

a) Follows directly from the construction of $\beta_N(N,\mathcal{C}_N)$.

b) Idem.

c) Suppose that some $t \in B_N^{\mathcal{C}}$ consumes a token from a place $p \in P^{\mathcal{C}}$ in N. Because $\beta_N(N,\mathcal{C}_N)$ is a *WF*-net with source place p_I and sink place p_O, there has to be a path from p to p_O. Hence there is a transition $t' \in T^{\mathcal{C}}$ consuming a token from p. Hence $\bullet_N t \cap \bullet_N t' \neq \varnothing$, thus $\bullet_N t = \bullet_N t'$ (N is *eFC*).

d) Follows directly from (a).

e) Follows directly from (d).

f) Follows directly from (e).

g) Let $t, t' \in T^{\mathcal{C}}$ such that $\bullet_{\beta_N(N,\mathcal{C}_N)} t \cap \bullet_{\beta_N(N,\mathcal{C}_N)} t' \neq \varnothing$. Given that $\bullet_N t' = \bullet_{\beta_N(N,\mathcal{C}_N)} t'$ and $\bullet_N t = \bullet_{\beta_N(N,\mathcal{C}_N)} t$, we have $\bullet_{\beta_N(N,\mathcal{C}_N)} t \cap \bullet_{\beta_N(N,\mathcal{C}_N)} t' = \bullet_N t \cap \bullet_N t' \neq \varnothing$. Hence $\bullet_N t = \bullet_N t'$ and thus $\bullet_{\beta_N(N,\mathcal{C}_N)} t = \bullet_{\beta_N(N,\mathcal{C}_N)} t'$. Therefore $\beta_N(N,\mathcal{C}_N)$ is *eFC*.

h) Let $M \in \beta_N(N,\mathcal{C}_N)[M_I\rangle \setminus \{M_O\}$. Then using (e) we can deduce $M_I \twoheadrightarrow_N M$, thus there exists a $t \in T$ such that $M[t\rangle$ (as N is sound). If $t \in T^{\mathcal{C}}$ then we are done. If $t \in B_N^{\mathcal{C}}$ then there exists a $t' \in T^{\mathcal{C}}$ such that $\bullet_N t = \bullet_{\beta_N(N,\mathcal{C}_N)} t'$ (c). Therefore $M[t'\rangle$. $\qquad\square$

While propositions a, b, d, e and f follow directly from the construction of configured nets and hold for non *eFC* *WF*-nets, propositions c, g, and h are particularly interesting for soundness. The problem in the example of Figure 3 is that the configuration may yield an unsound model when a transition is blocked which shares part of its preset with another transition. By definition, in an *eFC* *WF*-net such a situation cannot exist and therefore a deadlock marking cannot occur (propositions c and h). Further on, the deadlock in the example prevents all tokens from reaching the final place. As the configured net derived from an *eFC* *WF*-net remains *eFC* (proposition g), the *eFC* property prevents also this problem as it permits any token to move towards the final place.

These properties allow us to prove that if a configured net, derived from a sound *eFC* *WF*-net, is a *WF*-net, it fulfills the soundness criteria. Formally:

Theorem 2. Let $N = (P, T, F)$ be a sound, *eFC* WF-net with source place p_I and sink place p_O, let $\mathcal{C}$ be a configuration of N and let $\beta_N(N,\mathcal{C}_N) = (P^{\mathcal{C}}, T^{\mathcal{C}}, F^{\mathcal{C}})$ be the resulting configured net. If $\beta_N(N,\mathcal{C}_N)$ is a *WF*-net, then $\beta_N(N,\mathcal{C}_N)$ is sound.

Proof. Note that changing a transition into a silent transition (hiding) has no implications for soundness analysis.

- *proper completion*: since $\beta_N(N,\mathcal{C}_N)[M_I\rangle \subseteq N[M_I\rangle$ (Proposition 1f), M_O is the only state marking p_O.
- *option to complete*: because $\beta_N(N,\mathcal{C}_N)$ is an *eFC WF*-net (Proposition 1g), any token can decide to move towards p_O. If p_O is marked, all other places are empty ($\beta_N(N,\mathcal{C}_N)$ has proper completion). Hence, marking M_O can be reached (and the property holds) or the net is in a deadlock M. However, this is not possible as $\beta_N(N,\mathcal{C}_N)$ has no deadlock markings (Proposition 1h).

- *no dead transitions*: we define a length function as follows: $L : T^{\mathcal{C}} \to \mathbb{N}$. If $p_I \in \bullet t$ then $L(t) = 0$. Otherwise $L(t) = 1 + min_{p \in \bullet t, t' \in \bullet p} L(t')$. Given that every transition in $\beta_N(N, \mathcal{C}_N)$ is on a path from p_I, the function is well-defined. Using induction we prove $\forall_{n \in \mathbb{N}} \forall_{t \in T^{\mathcal{C}}} [L(t) = n \Rightarrow t$ is not dead in $\beta_N(N, \mathcal{C}_N)]$.
  (Base case) If $n = 0$ then $\bullet t = \{p_I\}$ and as $p_I \in P^{\mathcal{C}}$ (Proposition 1b), $M_I[t\rangle$, hence t is not dead.
  (Induction Hypothesis (IH)) If $t \in T'^{\mathcal{C}}$ is such that $L(t) = n + 1$, there exists a transition t' such that $L(t') = n$ and $t' \bullet \cap \bullet t \neq \varnothing$. t' is not dead (IH), hence there exists an $M \in \beta_N(N, \mathcal{C}_N)[M_I\rangle$ such that $M[t'\rangle$. Let M' be such that $M \xrightarrow{t'} M'$, then M' marks at least one input place (i.e., p) of t. As $\beta_N(N, \mathcal{C}_N)$ has the option to complete, $M' \twoheadrightarrow M_O$. This implies that some transition t'' exists which removes the token from p in some marking M', hence $p \in \bullet t''$. Therefore $\bullet t \cap \bullet t'' \neq \varnothing$, and thus, given that $\beta_N(N, \mathcal{C}_N)$ is eFC (Proposition 1g) $\bullet t = \bullet t''$. Therefore $M'[t\rangle$ and t is not dead. $\square$

Theorems 1 and 2 can be combined to show that a configured net is sound if and only if the process constraint PC is satisfied for the corresponding configuration. If the configured net is not an eFC WF-net, the implication only holds in one direction and in the other direction soundness cannot be guaranteed. In these cases PC can be used to rule out all the syntactically incorrect process models and conventional analysis tools such as Woflan [VBA01] have to be used in addition.

5. Application to Configurable EPCs

To demonstrate that inducing constraints is not only formally feasible, but also applicable to languages used by practitioners, in the following we show how the described approach can be applied when configuring Event-driven Process Chains (EPCs). EPCs are an easy-to-understand language for modeling business processes [SL05]. Supported by modeling tools like ARIS from IDS Scheer or Microsoft Visio, nowadays they are one of the most popular notations used by practitioners to depict and discuss the flow of business processes. EPCs focus on the control-flow of business processes using functions to represent the activities that need to be performed, events to depict both pre-conditions for the execution of functions and the result of these executions, and logical connectors to determine the control-flow behavior whenever the process splits into or joins from various process branches [KNS92]. Connectors can be of type *XOR*, *AND* and *OR*. An *XOR*-split models an exclusive decision: only one of its outgoing branches can be taken at a time, while an *XOR*-join acts as a pass-through to merge two or more branches. An *AND*-split models parallelism: all the outgoing branches are taken, while an *AND*-join is used to synchronize control from all incoming branches. An *OR*-split models an inclusive decision: one or more outgoing branches can be taken at a time, while the *OR*-join allows the partial synchronization of the incoming branches. As an example, Figure 4 depicts the travel requisition approval process from Figure 1 in EPC.

In [RA07] configuration options have been intuitively added to EPCs in a language called Configurable EPCs (C-EPCs). C-EPCs allow users to identify functions and connectors that can vary as *configurable nodes* by marking them with a bold border in the model. Variation is achieved by restriction. Configurable functions can be left *ON* or restricted to *OFF* or to *OPT* (optional). If a configurable function, such as function *Check & Update Travel Form (Employee)* in Figure 4, is left *ON*, it must be executed as a normal function of the process. If it is switched *OFF*, the function's execution is skipped at runtime. Switching *OFF* the function *Check & Update Travel Form (Employee)* therefore implies that travel forms arriving at the employee's desk are immediately ready for submission, i.e. without performing any further check or update by the employee. Therefore, a parallelism can be drawn between switching a function *OFF* in EPCs and hiding a transition in a Workflow net, since both imply some work to be skipped. Finally, the configuration value *OPT* is just a combination of the previous two options, as it allows a user to defer the decision of whether to execute or skip an optional function until runtime, on an instance-by-instance basis.

Configurable connectors can be configured by restricting their routing behavior. A configurable *XOR* can be left as a regular *XOR* or restricted to an outgoing (in case of a split) or incoming (in case of a join) sequence SEQ_n of events and functions, where n is the node starting the sequence and belongs to the postset of a split, or the preset of a join. An *OR* connector can be left as a regular *OR* or restricted to a regular *XOR*, *AND*, or to a sequence of nodes. A connector of type *AND* cannot be restricted since it does not capture any choice between different execution paths. As the configuration of connectors prevents certain process paths from being taken, a parallelism can be drawn with the blocking operator in Workflow nets.
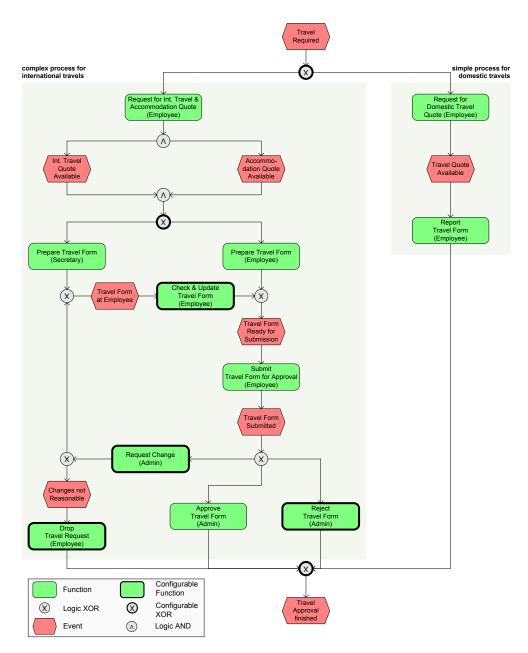
Fig. 4. The reference model for travel form approval in the C-EPC notation.

For example, configuring an *XOR*-split connector to one of its outgoing branches in EPC would correspond to blocking all but one transitions in the postset of a place in Workflow nets.

Therefore, the application of hiding and blocking operators to Workflow nets can be used as a foundational framework to formally describe the configuration of EPCs. Furthermore, the correctness results exposed in Section 4, can be exploited to achieve the staged configuration of C-EPC process models, where each configuration step is soundness-preserving. To show this, we first formalize the syntax of a C-EPC process model and then we define its semantics in terms of the induced Petri net.

5.1. C-EPCs: Syntax

Before turning to the formal definition of EPCs we briefly motivate some simplifications that we took into account. These simplifications are conditioned by the mapping to Petri nets which we will use later on to describe the semantics of EPCs.

The formal mapping of EPCs to Petri nets has been discussed for more than a decade. While the mapping of XOR and AND connectors is rather trivial, the OR-join poses considerable challenges. In essence, the problem stems from the fact that its informal description as a "partial synchronization" of incoming branches (i.e. wait until tokens can arrive) implies a recursive definition if there are multiple OR-joins in a loop. As shown in [Kin06], a unique fixed point is not guaranteed for evaluating such a definition. Nevertheless, it is important to note that the OR-join does *not add* behavior, but it only represents a process in a more compact way. Furthermore, it has been shown that a behavior-equivalent Petri net can always be constructed [MDA08] using the theory of regions [ER89, CKLY98], although the resulting model may be rather complex. This significant increase in the complexity also holds for OR-splits although their mapping is generally less challenging than the one of the OR-join. Since the OR connector does not add expressiveness but complicates the mapping dramatically, we abstract from this construct in the remainder of this article.

In light of the above, we formally define EPCs as the combination of a set of events, a set of functions and a set of connectors as the nodes of a graph. These nodes are connected through a set of arcs, while each connector must either be of type XOR or of type AND:

Definition 10 (Event-driven Process Chain). An Event-driven Process Chain (EPC) is a five-tuple $\Upsilon = (E, F, C, l, A)$ where:

- E is a finite non-empty set of events,
- F is a finite non-empty set of functions,
- C is a finite set of logical connectors ($E \cap F = \varnothing$, $E \cap C = \varnothing$, $F \cap C = \varnothing$),
- $l \in C \to \{AND, XOR\}$ is a mapping defining the type of each connector (AND or XOR), and
- $A \subseteq (E \times F) \cup (F \times E) \cup (E \times C) \cup (C \times E) \cup (F \times C) \cup (C \times F) \cup (C \times C)$ is a set of arcs. □

Further on, we define auxiliary sets, such as preset and postset of a node, and predicates, such as a path of nodes, which allow us to describe EPCs in a more compact way.

Definition 11 (EPC auxiliary sets and predicates). Let $\Upsilon = (E, F, C, l, A)$ be an EPC. Then:

- $\forall_{n \in E \cup F \cup C}$ $\bullet n = \{x \in E \cup F \cup C \mid (x, n) \in A\}$ is the preset of n,
- $\forall_{n \in E \cup F \cup C}$ $n \bullet = \{x \in E \cup F \cup C \mid (n, x) \in A\}$ is the postset of n,
- $p = \langle n_1, n_2, \ldots, n_k \rangle$ is a path such that $(n_i, n_{i+1}) \in A$ for $1 \leq j < k$,
- $C_{AND} = \{c \in C \mid l(c) = AND\}$ is the set of AND connectors,
- $C_{XOR} = \{c \in C \mid l(c) = XOR\}$ is the set of XOR connectors,
- $C_S = \{c \in C \mid |\bullet c| = 1 \wedge |c \bullet| > 1\}$ is the set of split connectors,
- $C_J = \{c \in C \mid |\bullet c| > 1 \wedge |c \bullet| = 1\}$ is the set of join connectors,
- $C_{EF} \subseteq C$ such that $c \in C_{EF}$ iff there is a path $p = \langle n_1, n_2, \ldots, n_{k-1}, n_k \rangle$ such that $n_1 \in E$, $n_2, \ldots, n_{k-1} \in C$, $n_k \in F$ and $c \in \{n_2, \ldots, n_{k-1}\}$ is the set of connectors between events and functions, and
- $C_{FE} \subseteq C$ such that $c \in C_{FE}$ iff there is a path $p = \langle n_1, n_2, \ldots, n_{k-1}, n_k \rangle$ such that $n_1 \in F$, $n_2, \ldots, n_{k-1} \in C$, $n_k \in E$ and $c \in \{n_2, \ldots, n_{k-1}\}$ is the set of connectors between function and events. □

C-EPCs extend EPCs with two kinds of variation points: configurable functions and configurable connectors. Both configurable functions and configurable connectors are integrated into a C-EPC as regular functions and connectors with the only difference that they are marked as configurable. Thus to define C-EPCs, we just need to mark a subset of the functions and a subset of the connectors as configurable. Since we abstract from OR connectors, a configurable connectors must only be of type XOR.

Definition 12 (Configurable EPC). A configurable EPC (C-EPC) is a seven-tuple $\Gamma = (E, F, C, l, A, F^C, C^C)$ where:

- (E, F, C, l, A) is an EPC,
- $F^C \subseteq F$ is the set of *configurable functions*,

- $C^C \subseteq C_{XOR}$ is the set of *configurable connectors*.   □

If sets F^C and C^C are empty a C-EPC corresponds to a regular EPC. Hence, many of the following definitions hold for both EPCs and C-EPCs which we will make clear by referring to (C-)EPCs whenever this is applicable.

Finally, we introduce a notion of syntactically correct (C-)EPC. First of all, in a syntactically correct (C-)EPC, connectors can only be of type split or join, events must have at most one incoming and one outgoing arc, while functions must have exactly one incoming and one outgoing arc. As functions can be triggered by events or be triggers to events, this order must be retained even if connectors are located between them, i.e. it is not possible to have a connector between two functions or two events. Furthermore, all (C-)EPC nodes need to be on a path between the unique start and the unique end event. This latter condition ensures that: (i) there is no function/event in the EPC that can not be reached from the initial node, and that would therefore never be executed; and (ii) there is no "trap" in the EPC, such that once the flow of control enters that "trap", it can no longer leave it (leading to an infinite execution). Finally, we impose that a syntactically correct C-EPC must have a unique start event and a unique end event. This condition is not strictly necessary but we introduce it here to simplify the presentation of our proposal. Importantly, this condition does not restrict the expressive power: (C-)EPCs with multiple start and end events can be transformed to (C-)EPCs with a single start and a single end event by merging all the start events into a new start event, and all the end events into a new end event [MDA08]. The following definition captures these conditions. It is based on previous work on formalization of EPCs [Aal99].

Definition 13 (Syntactically correct (C-)EPC). Let $\Gamma = (E, F, C, l, A, F^C, C^C)$ be a C-EPC and A^* be the reflexive transitive closure of A. Γ is syntactically correct iff:

- events have at most one incoming and one outgoing arc, i.e. $\forall_{e \in E} [| \bullet e| \leq 1 \ \wedge \ |e \bullet | \leq 1]$,
- functions have exactly one incoming and one outgoing arcs, i.e. $\forall_{f \in F} [| \bullet f| = 1 \ \wedge \ |f \bullet | = 1]$,
- C_S and C_J partition C, i.e. $C_S \cap C_J = \varnothing$ and $C_S \cup C_J = C$,
- C_{EF} and C_{FE} partition C, i.e. $C_{EF} \cap C_{FE} = \varnothing$ and $C_{EF} \cup C_{FE} = C$,
- there exists exactly one start event, i.e. $\exists!_{e_S \in E} \ \bullet e_S = \varnothing$,
- there exists exactly one end event, i.e. $\exists!_{e_E \in E} \ e_E \bullet = \varnothing$, and
- every node is on a directed path from start to end event, i.e. $\forall_{n \in E \cup F \cup C} [(e_S, n) \in A^* \ \wedge \ (n, e_E) \in A^*]$.□

The C-EPC of Figure 4 is syntactically correct.

5.2. C-EPCs: Semantics

Since there is no standard definition of the execution semantics of (C-)EPCs, we describe such a semantics in terms of a Petri net that we induce from a (C-)EPC. To obtain such a Petri net, we first create an *expanded* (C-)EPC in which we get rid of all chains of connectors, by replacing any arc between two connectors by a "silent function", a "silent event" and arcs to connect these new elements with the two connectors in question. In this way, connectors can only be linked with functions and events and not with other connectors. These additional nodes do not correspond to observable behavior and thus do not change the overall behavior depicted by the model, but are merely added to simplify the mapping to Petri nets.

Definition 14 (Expanded (C-)EPC). Let $\Gamma = (E, F, C, l, A, F^C, C^C)$ be a syntactically correct C-EPC. $\Gamma' = (E', F', C, l, A', F^C, C^C)$ is the expanded net of Γ such that:

- $E' = E \ \cup \ \{e^a \mid a \in A \cap (C \times C)\}$,
- $F' = F \ \cup \ \{f^a \mid a \in A \cap (C \times C)\}$,
- $A' = A \setminus (C \times C) \ \cup \ \{(c, f^{(c,d)}) \mid f^{(c,d)} \in F' \wedge \ c \in C_{EF}\} \ \cup \ \{(c, e^{(c,d)}) \mid e^{(c,d)} \in E' \wedge \ c \in C_{FE}\} \ \cup \ \{(f^{(c,d)}, e^{(c,d)}) \mid f^{(c,d)} \in F' \wedge \ e^{(c,d)} \in E' \wedge c \in C_{EF}\} \ \cup \ \{(e^{(c,d)}, f^{(c,d)}) \mid f^{(c,d)} \in F' \wedge \ e^{(c,d)} \in E' \wedge c \in C_{FE}\} \ \cup \ \{(e^{(c,d)}, d) \mid e^{(c,d)} \in E' \wedge \ c \in C_{EF}\} \ \cup \ \{(f^{(c,d)}, d) \mid f^{(c,d)} \in F' \wedge \ c \in C_{FE}\}$.   □

In the remainder, when referring to an EPC Υ or to a C-EPC Γ, we always mean their expanded net Υ' and Γ'.
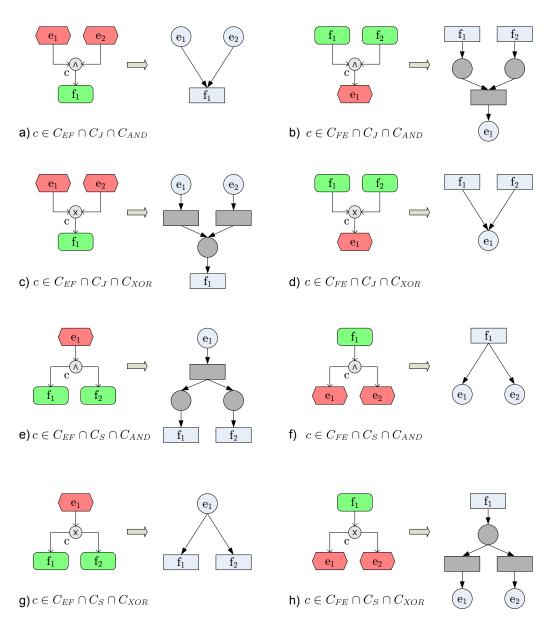
Fig. 5. Mapping (C-)EPC connectors to Petri net places, transitions and arcs.

To construct a Petri net from a (C-)EPC we simply map each event of the (C-)EPC to a place in the Petri net and each function of the (C-)EPC onto a transition in the Petri net. If events are directly connected to functions or vice versa in the (C-)EPC, we can also add the corresponding arcs as flows in the Petri net.

If, however, a connector is located between an event and a function, we have to re-build the splitting or joining behavior of the connector in the Petri net. For this, we use the transformations depicted in Figure 5 which might require inserting additional "silent" transitions and places. As a transition synchronizes the flow of its incoming branches in a Petri net, the places conforming to events preceding an *AND*-join can be connected directly to the transition corresponding to the function succeeding the *AND*-join connector (Figure 5a). However, if an *AND*-join connector is followed by an event in the (C-)EPC, it is necessary to introduce an additional transition to the Petri net to synchronize the incoming flows before the place corresponding to this event, since Petri net places do not synchronize any incoming flow. Given that the

	P_c^{PN}	T_c^{PN}	F_c^{PN}
$c \in C_{EF} \cap C_J \cap C_{AND}$	$\varnothing$	$\varnothing$	$\{(x,y) \mid x \in \bullet c \ \wedge y \in c\bullet\}$
$c \in C_{FE} \cap C_J \cap C_{AND}$	$\{p_x^c \mid x \in \bullet c\}$	$\{t^c\}$	$\{(x,p_x^c) \mid x \in \bullet c\} \ \cup \{(p_x^c,t^c) \mid x \in \bullet c\} \ \cup \{(t^c,x) \mid x \in c\bullet\}$
$c \in C_{EF} \cap C_J \cap C_{XOR}$	$\{p^c\}$	$\{t_x^c \mid x \in \bullet c\}$	$\{(x,t_x^c) \mid x \in \bullet c\} \ \cup \{(t_x^c,p^c) \mid x \in \bullet c\} \ \cup \{(p^c,x) \mid x \in c\bullet\}$
$c \in C_{FE} \cap C_J \cap C_{XOR}$	$\varnothing$	$\varnothing$	$\{(x,y) \mid x \in \bullet c \ \wedge y \in c\bullet\}$
$c \in C_{EF} \cap C_S \cap C_{AND}$	$\{p_x^c \mid x \in c\bullet\}$	$\{t^c\}$	$\{(x,t^c) \mid x \in \bullet c\} \ \cup \{(t^c,p_x^c) \mid x \in c\bullet\} \ \cup \{(p_x^c,x) \mid x \in c\bullet\}$
$c \in C_{FE} \cap C_S \cap C_{AND}$	$\varnothing$	$\varnothing$	$\{(x,y) \mid x \in \bullet c \ \wedge y \in c\bullet\}$
$c \in C_{EF} \cap C_S \cap C_{XOR}$	$\varnothing$	$\varnothing$	$\{(x,y) \mid x \in \bullet c \ \wedge y \in c\bullet\}$
$c \in C_{FE} \cap C_S \cap C_{XOR}$	$\{p^c\}$	$\{t_x^c \mid x \in c\bullet\}$	$\{(x,p^c) \mid x \in \bullet c\} \ \cup \{(p^c,t_x^c) \mid x \in c\bullet\} \ \cup \{(t_x^c,x) \mid x \in c\bullet\}$

Table 1. Mapping a (C-)EPC connector $c \in C$ to places, transitions and arcs (see Figure 5).

inserted transition cannot be directly connected to the transitions corresponding to the functions preceding the *AND*-join connector, an additional place must also be introduced for each of the functions preceding this connector (Figure 5b).

For *XOR*-join connectors, synchronization has to be avoided. Thus, transitions corresponding to functions preceding an *XOR*-join connector can be directly connected to the place corresponding to the event succeeding the *XOR*-join connector (Figure 5d) while an additional place with preceding transitions must be introduced when the *XOR*-join connector is succeeded by a function (Figure 5c). The Petri net constructs for split connectors are in line with this as illustrated in Figure 5e-h.

Formally we can define the induced Petri net as follows:

Definition 15 (Induced Petri net). Let $\Gamma = (E, F, C, l, A, F^C, C^C)$ be a syntactically correct (C-)EPC. $\mathcal{N}(\Gamma) = (P^{PN}, T^{PN}, F^{PN})$ is the Petri net induced by Γ' such that:

- $P^{PN} = E \ \cup \ \bigcup_{c \in C} P_c^{PN}$,
- $T^{PN} = F \ \cup \ \bigcup_{c \in C} T_c^{PN}$,
- $F^{PN} = (A \cap ((E \times F) \cup (F \times E))) \ \cup \ \bigcup_{c \in C} F_c^{PN}$,

where P_c^{PN}, T_c^{PN}, and F_c^{PN} are defined as per Table 1. $\qquad\square$

It is easy to see that for any syntactically correct (C-)EPC Γ, $\mathcal{N}(\Gamma) = (P^{PN}, T^{PN}, F^{PN})$ is a Petri net, since by definition $P^{PN} \cap T^{PN} = \varnothing$ and $F \subseteq (P^{PN} \times T^{PN}) \cup (T^{PN} \times P^{PN})$. Moreover, the Petri net is an *extended free-choice WF*-net, as illustrated by the following lemma which extends a lemma in [Aal99].

Lemma 1. Let $\Gamma = (E, F, C, l, A, F^C, C^C)$ be a syntactically correct (C-)EPC and $\mathcal{N}(\Gamma)$ be its induced Petri net, then:

a) $\mathcal{N}(\Gamma)$ is a *WF*-net.
b) $\mathcal{N}(\Gamma)$ is *eFC*.

Proof.

a) Follows directly from the construction of $\mathcal{N}(\Gamma)$.
b) We have to prove that for any two transitions t, t' sharing an input place, $\bullet t = \bullet t'$. Therefore, we have to check every place with two or more output arcs. An event cannot have more than one output arc (Definition 13). The only way to obtain a place with multiple output arcs is the mapping of *XOR*-split connectors onto Petri net constructs (see Figure 5). However, the rules given in Table 1 guarantee that the output transitions have identical sets of input places. Therefore $\mathcal{N}(\Gamma)$ is *eFC*. $\qquad\square$

Figure 6 depicts the expanded net for the C-EPC of Figure 4 and its induced Petri net. Also here it is easy to see that the induced Petri net is indeed an extended free-choice WF-net as all transitions which are preceded by places with multiple outgoing arcs are not synchronizing any path.
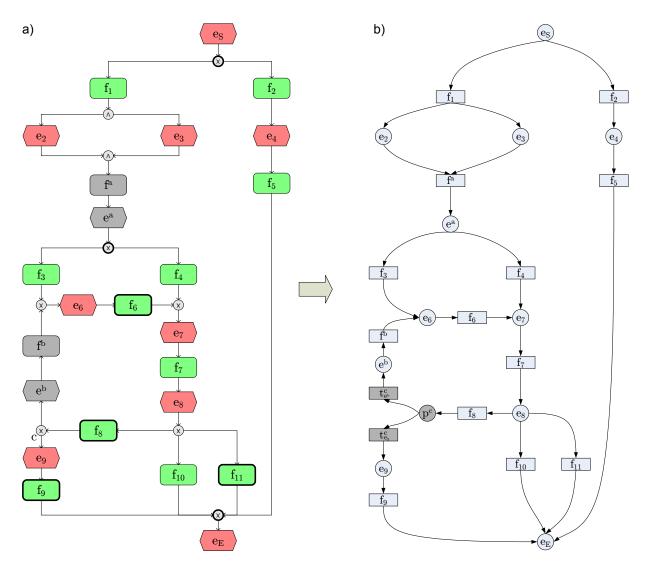
Fig. 6. a) The expanded net for the C-EPC of Figure 4 and b) its induced Petri net.

We can use the induced Petri net to describe the semantics of a (C-)EPC. This allows us to identify those (C-)EPCs which can be correctly executed, i.e. which are sound, by exploiting the definition of soundness for WF-nets (Definition 4):

Definition 16 (Sound (C-)EPC). Let $\Gamma = (E, F, C, l, A, F^C, C^C)$ be a syntactically correct (C-)EPC and $\mathcal{N}(\Gamma)$ be its induced WF-net. Γ is sound iff $\mathcal{N}(\Gamma)$ is sound. □

In this way, we can identify if a (C-)EPC Γ contains semantic problems, i.e. if it is unsound, by checking whether the induced net $\mathcal{N}(\Gamma)$ presents semantic issues. As an example, let us consider the syntactically correct EPC in Figure 7. Here the control-flow is split-up through an *XOR*-split connector and later on joined again by a synchronizing *AND*-join connector. In this case it is easy to see that the final event will never be reached because the *XOR* connector triggers only one of the two subsequent paths while the *AND* waits for the completion of both. Hence, this EPC is not sound. In more complex (C-)EPCs such a mismatch between split and join connectors might be far trickier to spot. When mapping this EPC onto the corresponding Workflow net, this mismatch translates into a so-called PT-handle [ES90] where two paths exist between a place and a transition in the net which do not share any further nodes. Algorithms that detect such PT-handles as well as other issues that cause the unsoundness of a Petri net are, as mentioned
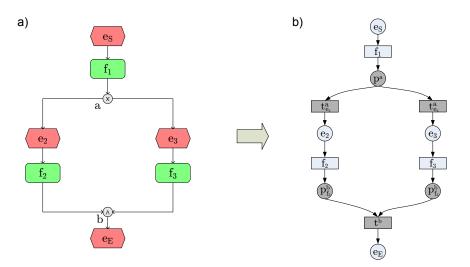
Fig. 7. a) An unsound C-EPC and b) its induced Petri net.

before, implemented in tools like Woflan [VBA01]. Thus by defining the semantics of a (C-)EPC via its induced Workflow net, we can use these algorithms to automatically determine whether a (C-)EPC is sound or not.

5.3. C-EPCs: Configuration and Correctness

We can now discuss how process correctness can be preserved during the configuration of a C-EPC. As illustrated in the beginning of this section, in a C-EPC configurable functions may be included or skipped while configurable connectors may be restricted.

To capture the restriction of connectors, we introduce the partial order $\leq^C$ to order connectors from specific to more generic ones, before we define the notion of C-EPC configuration. $\leq^C$ forces a configurable XOR to be configured to a regular XOR or a sequence operator SEQ_n starting with node n.

Definition 17 (Partial Order for Connectors). The partial order $\leq^C$ is defined on $CT \cup CTS$ where $CT = \{AND, XOR\}$ is the set of connector types and $CTS = \{SEQ_n \mid n \in E \cup F \cup C\}$ is the set of sequence operators. $\leq^C = \{(n,n) \mid n \in CT \cup CTS\} \cup (CTS \times \{XOR\})$. $\qquad\qquad\square$

For example $SEQ_n \leq^C XOR$ implies that the configurable connector XOR can be mapped onto the connector type SEQ_n, i.e., a choice may be replaced by a sequence but not the other way around.

A *configuration* is a function that maps a configurable node onto an allowed value according to the node type. It also ensures that SEQ_n can be chosen as value, only if it n is the final node of an incoming branch for a configurable join, or the initial node of an outgoing branch for a configurable split.

Definition 18 (C-EPC Configuration). Let $\Gamma = (E, F, C, l, A, F^C, C^C)$ be a syntactically correct C-EPC. The mapping $\mathcal{C}_\Gamma \in (F^C \nrightarrow \{ON, OFF\}) \cup (C^C \nrightarrow CT \cup CTS)$ is a *configuration* of Γ iff for each $c \in C^C \cap dom(\mathcal{C}_\Gamma)$:[5]

- $\mathcal{C}_\Gamma(c) \leq^C l(c)$,
- if $c \in C_S$ and $\mathcal{C}_\Gamma(c) = SEQ_n$ for some $n \in E \cup F \cup C$, then $n \in c\bullet$,
- if $c \in C_J$ and $\mathcal{C}_\Gamma(c) = SEQ_n$ for some $n \in E \cup F \cup C$, then $n \in \bullet c$. $\qquad\square$

By applying a configuration $\mathcal{C}_\Gamma$ to a C-EPC, we can derive a new EPC from the original net if $\mathcal{C}_\Gamma$ assigns a value to all configurable nodes, or a partly configured C-EPC if only some configurable nodes are set. This is done in four steps. Firstly, we change the type of all configurable connectors that have been restricted by

---

[5] $f \in A \nrightarrow B$ denotes a partial function, i.e. the domain of f is a subset of A.

configuration to their new types, and remove all arcs to or from the connectors that are not permitted to be taken any longer. Secondly, all functions f that have been configured as *OFF* are replaced with functions $skip_f$ which correspond to no actual behavior. In this way, the structure of the net does not really change in this second step. Thirdly, all elements that after the first two steps are no longer on a path between the start and the end event are removed. Finally, all connectors with a single incoming and a single outgoing arc are replaced with arcs, as they are no longer required. The following definition formalizes this algorithm adapted from [RA07].

Definition 19 (Configured EPC). Let $\Gamma = (E, F, C, l, A, F^C, C^C)$ be a syntactically correct C-EPC and let $\mathcal{C}_\Gamma$ be one of its configurations. $\beta_\Gamma(\Gamma, \mathcal{C}_\Gamma)$ defines a (C-)EPC Ψ constructed as follows:

1. Map the configurable connectors $c \in C^C \cap dom(\mathcal{C}_\Gamma)$ onto their concrete type and remove arcs not involving the selected sequence, i.e. $\Psi_1 = (E, F, C, l_1, A_1)$ with $l_1 = l \oplus \{(c, \mathcal{C}_\Gamma(c)) \mid c \in C^C\}$ and $A_1 = A \setminus (\{(c, n) \in C_S \times c \bullet \mid \exists_{n' \in c \bullet, n' \neq n} [\mathcal{C}_\Gamma(c) = SEQ_{n'}]\} \cup \{(n, c) \in \bullet c \times C_J \mid \exists_{n' \in \bullet c, n' \neq n} [\mathcal{C}_\Gamma = SEQ_{n'}]\})$.[6]

2. For each $f \in F^C \cap dom(\mathcal{C}_\Gamma)$ such that $\mathcal{C}_\Gamma(f) = OFF$, replace the function with a new function $skip_f$ to reflect that the original function is not executed, i.e. $\Psi_2 = (E, F_2, C, l_1, A_1)$ with $F_2 = F \oplus \{skip_f \mid f \in F^C \cap dom(\mathcal{C}_\Gamma) \wedge \mathcal{C}_\Gamma(f) = OFF\}$.

3. Remove all nodes not on some path from the start event $e_S \in E$ to the end event $e_E \in E$. Let $\Psi_3 = \{E_3, F_3, C_3, l_3, A_3\}$ be the resulting EPC.[7]

4. Remove all connectors with just one input and one output node, i.e. $\beta_\Gamma(\Gamma, \mathcal{C}_\Gamma) = \Psi = (E_3, F_3, C_4, l_4, A_4)$ with $C_4 = \{c \in C_3 \mid |c \bullet| > 1 \vee |\bullet c| > 1\}$, $l_4 = \{(c, x) \in l_3 \mid c \in C_4\}$ and $A_4 = \{(n_1, n_2) \in A_3 \mid \{n_1, n_2\} \cap (C_3 \setminus C_4) = \varnothing\} \cup \{(n_1, n_2) \mid \exists_{c \in C_3 \setminus C_4} [\{(n_1, c), (c, n_2)\} \in A_3]\}$. □

Figure 8 shows the EPC resulting from a configuration of the C-EPC in Figure 6 where

- the configurable *XOR* connector at the top has been configured to SEQ_{f_1},
- the configurable *XOR* connector after e^a has been configured to SEQ_{f_4},
- the configurable function f_9 has been switched *OFF*,
- all other configurable connectors have not been restricted, and
- all other configurable functions remained *ON*.

The configuration of the top *XOR* connector leads to the removal of the arc from this connector to f_2. Thus, all nodes subsequent to f_2 until the joining connector right before e_E are not reachable from e_S anymore and thus removed in the third step of the configuration algorithm. In the same way, the configuration of the *XOR* connector after e^a makes f_3 not reachable anymore. After this, the *XOR* connector subsequent to f_3 has only a single incoming arc from f^b and a single outgoing arc to e_6 and is therefore removed by the fourth algorithm step. Switching function f_9 off leads to its replacement with the $skip_{f_9}$ function.

The following Theorem shows that the resulting (C-)EPC $\beta_\Gamma(\Gamma, \mathcal{C}_\Gamma)$ is syntactically correct provided the initial C-EPC is syntactically correct:

Theorem 3. Let $\Gamma = (E, F, C, l, A, F^C, C^C)$ be a syntactically correct C-EPC and let $\mathcal{C}_\Gamma$ be one of its configurations. $\beta_\Gamma(\Gamma, \mathcal{C}_\Gamma)$ is a syntactically correct (C-)EPC.

Proof. See *[RA07]*. □

In the beginning of this section we indicated that a C-EPC configuration can be represented by using the blocking and hiding operators that we defined for workflow nets. Thus, from a C-EPC configuration we can project a configuration onto the induced Workflow net. If a configurable function in a C-EPC is switched *OFF*, this implies that the corresponding transition in the Workflow net is hidden. If a configurable *XOR* connector is restricted, the transition corresponding to the particular incoming (in case of a join connector) or outgoing (in case of a split connector) arc(s) must be blocked.

Definition 20 (Induced WF-net Configuration). Let $\Gamma = (E, F, C, l, A, F^C, C^C)$ be a syntactically correct C-EPC, $\mathcal{C}_\Gamma$ be one of its configurations and $\mathcal{N}(\Gamma) = (P^{PN}, T^{PN}, F^{PN})$ be its induced WF-net. $\mathcal{C}_\mathcal{N}^{\mathcal{C}_\Gamma} \in T^{PN} \to \{allow, hide, block\}$ is the configuration of $\mathcal{N}(\Gamma)$ induced by $\mathcal{C}_\Gamma$. For $t \in T^{PN}$:

---

[6] $\oplus$ is the override operator.
[7] This step may lead to an empty net if e_S and/or e_E have been disconnected by step 1.
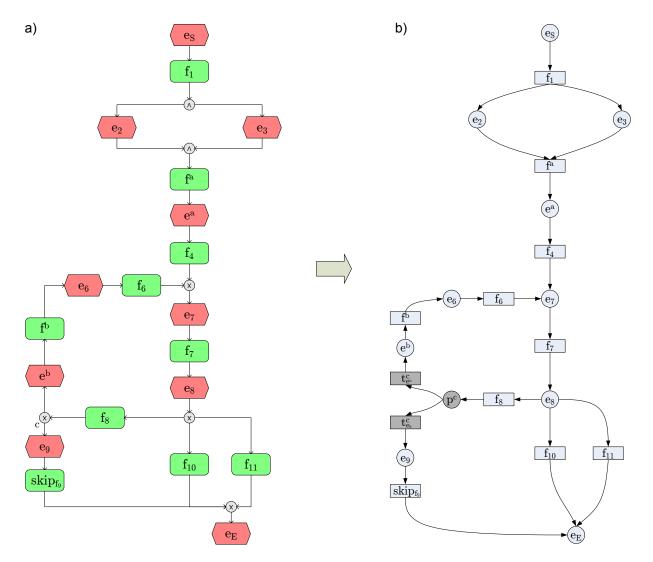
Fig. 8. a) The configured EPC from the C-EPC of Figure 6 and b) its induced WF-net.

$$
\mathcal{C}_{\mathcal{N}}^{\mathcal{C}_\Gamma}(t) = \begin{cases}
hide, & t \in F^C \text{ and } \mathcal{C}_\Gamma(t) = OFF, \\
block, & t \in c \bullet \setminus \{n\} \text{ for } c \in C^C \cap C_{XOR} \cap C_S \cap C_{EF} \\
& \text{so that } \mathcal{C}_\Gamma(c) = SEQ_n, \\
block, & t \in \{t_x^c \mid x \in c \bullet \setminus \{n\}\} \text{ for } c \in C^C \cap C_{XOR} \cap C_S \cap C_{FE} \\
& \text{so that } \mathcal{C}_\Gamma(c) = SEQ_n, \\
block, & t \in \{t_x^c \mid x \in (\bullet c) \setminus \{n\}\} \text{ for } c \in C^C \cap C_{XOR} \cap C_J \cap C_{EF} \\
& \text{so that } \mathcal{C}_\Gamma(c) = SEQ_n, \\
block, & t \in \bullet c \setminus \{n\} \text{ for } c \in C^C \cap C_{XOR} \cap C_J \cap C_{FE} \\
& \text{so that } \mathcal{C}_\Gamma(c) = SEQ_n, \\
allow, & \text{otherwise.}
\end{cases}
$$

Figure 9a shows the induced WF-net configuration applied to our example model (Figure 6b), where f_2 and f_3 have been blocked and f_9 has been hidden. To determine which further nodes need to be removed in order to preserve the syntactic correctness of this model, we use the process constraint $PC(\mathcal{N}(\Gamma))$ and a SAT solver as illustrated at the end of Section 4.1. In our example, $PC(\mathcal{N}(\Gamma))$ imposes to remove e_4 and f_5 because these nodes would no longer be on a directed path from the input to the output place after
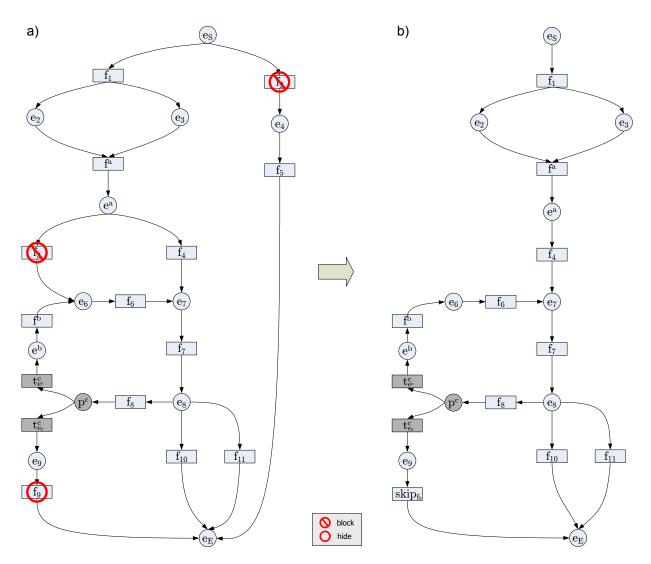
Fig. 9. a) The WF-net configuration induced by the C-EPC configuration is applied to the induced WF-net of Figure 6b. b) The configured net after enforcing $PC(\mathcal{N}(\Gamma))$.

removing f_2. Figure 9b depicts the resulting WF-net, which we call $\beta_N^*(\mathcal{N}(\Gamma), \mathcal{C}_\mathcal{N}^{\mathcal{C}_\Gamma})$. This is the configured net obtained after applying $\mathcal{C}_\mathcal{N}^{\mathcal{C}_\Gamma}$ and removing all nodes that are not on a directed path from the input to the output place in order to fulfil $PC(\mathcal{N}(\Gamma))$.

As we have shown that the induced WF-net $\mathcal{N}(\Gamma)$ is eFC (Lemma 1), the application of $PC(\mathcal{N}(\Gamma))$ guarantees that $\beta_N^*(\mathcal{N}(\Gamma), \mathcal{C}_\mathcal{N}^{\mathcal{C}_\Gamma})$ is both syntactically and semantically correct (Theorem 2). This net is identical to the Workflow net $\mathcal{N}(\beta_\Gamma(\Gamma, \mathcal{C}_\Gamma))$ which we induced from the configured C-EPC (see Figure 8b). Thus we can conclude for our example that the configured EPC $\beta_\Gamma(\Gamma, \mathcal{C}_\Gamma)$ is not only syntactically correct (Theorem 3), but also semantically correct, i.e. sound. More generally, we can formulate the following proposition.

Proposition 2 (Soundness-preserving C-EPC configuration). Let Γ be a sound C-EPC, $\mathcal{C}_\Gamma$ be one of its configurations and $\mathcal{N}(\Gamma)$ be its induced WF-net. Let also $\mathcal{C}_\mathcal{N}^{\mathcal{C}_\Gamma}$ be the configuration of $\mathcal{N}(\Gamma)$ induced by $\mathcal{C}_\Gamma$ and $\beta_N^*(\mathcal{N}(\Gamma), \mathcal{C}_\mathcal{N}^{\mathcal{C}_\Gamma})$ be the configured net in which all the nodes not on a directed path from the input to the output place have been removed to fulfil $PC(\mathcal{N}(\Gamma))$. If $\mathcal{N}(\beta_\Gamma(\Gamma, \mathcal{C}_\Gamma))$ is equal to $\beta_N^*(\mathcal{N}(\Gamma), \mathcal{C}_\mathcal{N}^{\mathcal{C}_\Gamma})$, then $\beta_\Gamma(\Gamma, \mathcal{C}_\Gamma)$ is sound.

Proof. We observe that: (i) Γ is sound, hence its configured EPC $\beta_\Gamma(\Gamma, \mathcal{C}_\Gamma)$ is syntactically correct (Theorem 3) and $\mathcal{N}(\Gamma)$ is an *eFC* WF-net (Lemma 1) and is sound (Definition 16); (ii) since $\beta_\Gamma(\Gamma, \mathcal{C}_\Gamma)$ is syntactically correct, its induced Petri net $\mathcal{N}(\beta_\Gamma(\Gamma, \mathcal{C}_\Gamma))$ is an *eFC* WF-net (Lemma 1). Thus $\beta_N^*(\mathcal{N}(\Gamma), \mathcal{C}_N^{\mathcal{C}_\Gamma})$ is sound, since it is the configured *eFC* WF-net of $\mathcal{N}(\Gamma)$ which is sound (Theorem 2). If $\mathcal{N}(\beta_\Gamma(\Gamma, \mathcal{C}_\Gamma))$ is equal to $\beta_N^*(\mathcal{N}(\Gamma), \mathcal{C}_N^{\mathcal{C}_\Gamma})$, then $\mathcal{N}(\beta_\Gamma(\Gamma, \mathcal{C}_\Gamma))$ is sound. Hence $\beta_\Gamma(\Gamma, \mathcal{C}_\Gamma)$ is sound (Definition 16).   □

This proposition provides a basis for staged correctness-preserving configuration of C-EPCs. If we start from a C-EPC that has been checked for soundness, and we apply a configuration step, we can then check the correctness of the resulting C-EPC by reasoning on the induced Workflow nets before and after the configuration.

6. Related Work

Variability modeling has been widely studied in the field of Software Product Line Engineering (SPLE) [PBL05]. Techniques developed in this field enable the configuration of software artifacts based on models that relate these artifacts to domain concepts (e.g. parameters, options or features). The techniques differ in the way domain models are captured and related to software artifacts, and also in the way they capture constraints. The Adele Configuration Manager [EC94] and the Cosmic Configurable Middleware [TGN04] use first-order logic to capture constraints. In contrast, we use propositional logic, for which we can apply efficient techniques to discard incorrect configuration steps or to suggest ways of repairing them. Batory [Bat05] presents a Feature-Oriented Domain Analysis (FODA) technique in which constraints are captured in propositional logic. The respective tool uses a SAT solver to determine if a configuration is valid. A similar approach is adopted in [AC04]. Our work is inspired by these approaches but it is targeted at business process model configuration. Thus, we deal with graph-oriented models (hence, structural correctness needs special attention) and we are concerned with ensuring absence of deadlocks or livelocks and other behavioral properties.

The literature exhibits a heterogeneous set of methods for capturing variability in process models. The approach in [BDK07] is based upon the principle of model projection. Since a *reference* process model typically contains information for multiple application scenarios, it is possible to create a projection for a specific scenario (e.g. a class of users) by fading out those process branches that are not relevant to the scenario in question. Plain EPCs are used to represent process models. Another proposal for capturing variability in process models is defined in [SP06], which relies on stereotype annotations to accommodate variability in a so-called *variant-rich process model*. Accordingly, process activities can be annotated to become variation points and variants can be attached to these variation points in the form of other stereotyped activities. Although stereotypes are an extensibility mechanism of UML, in this proposal they are applied to both UML ADs and BPMN models. A subset of these stereotypes also appears in [RK08]. Specifically, in this approach the focus is on two types of variation points: *optional* and *alternative* ones. An optional variation point allows the selection of at most one variant among the available ones. An alternative variation point allows the selection of exactly one variant. This idea of annotating model elements to represent variability has also been investigated in [CA05]. In this approach, any control-flow element of an UML AD can be annotated using *presence conditions* (PCs) and *meta-expressions* (MEs). PCs indicate if the model element they refer to should be present or be removed. MEs are used to compute attributes of model elements relevant to the UML notation (e.g. the name of an activity).

Although there exists a variety of approaches, none of these deal with the matter of preserving model correctness during configuration. The only ones that get closer to providing some sort of support in this respect are [BDK07] and [CA05]. These approaches prompt users with a list of syntactic issues detected during process individualization. However these issues need to be manually fixed.

Our previous work includes the definition of variability mechanisms for existing process modeling languages such as EPCs [RA07], YAWL, SAP WebFlow and BPEL [GAJVL08]. In [LLS+07] we proposed a framework which ensures domain conformance (but not syntactic or behavioral correctness) by linking configurable process models to domain models expressed as questionnaires. Finally, the use of the hiding and blocking operators for variation points is sketched in [GAJV07].

In this article we outlined a technique to derive propositional logic constraints from process models. Similar techniques have been used for analyzing Petri nets [AIN04] and process graphs [SOS05]. However,

the constraints we derive are specifically aimed at checking that a configuration step preserves the structural properties of workflow nets.

Links between EPCs and Petri nets have been widely discussed in literature. The mapping from EPCs to Boolean nets, a variant of colored Petri nets, in [LSW98], is the first significant formal contribution in this area. The semantics is however restricted to EPCs in which OR connectors appear only at the boundaries of single-entry, single-exit regions. Another attempt to attach a formal to C-EPCs in a more general way can be found in [NR02], but it was later proven to be flawed in the sense that a fixed point is not guaranteed [Kin06]. Again, the problem lies with the formalization of OR connectors, and specifically the OR-join connector. This finding has triggered work on defining OR-joins based on timers, history or context [HOS05, MA07, HOS$^+$08]. While these formalizations implement the OR-join in different ways, they essentially agree on the semantics of XOR and AND connectors. The difficulties in reaching a consensus semantics of the OR-join in EPCs underpins our decision to leave this operator out of the scope of this article, and to adopt the semantics of EPCs without OR-joins proposed in [Aal99].

7. Conclusion

We have exposed a formal framework for staged correctness-preserving configuration of reference process models. Assuming the initial configurable process model is correct, the framework guarantees that the individualized process models are also correct at each stage of the configuration procedure. This is achieved by capturing the syntactic correctness constraints as a propositional logic formula. This formula, in conjunction with another formula capturing the domain constraints, is used to check the correctness-preservation of each configuration step. If a configuration step violates the constraints, a formula is derived to suggest ways of making the configuration step correctness-preserving. A cornerstone of the framework is a proof that, for free-choice process models, the enforcement of these syntactic constraints also ensures the preservation of semantic correctness.

Having established this formal framework using Petri nets, we have illustrated its application to a practical process modeling notation, namely EPCs. This application is achieved by reusing a previous formalization of EPCs and defining a mapping from configuration operators defined for C-EPCs into corresponding operators defined for Petri nets. This mapping then allows us to check the correctness-preservation of configuration steps applied to C-EPCs by reasoning on the induced Petri nets. In order to achieve a simple mapping, we restricted our study to C-EPCs without OR connectors. While this limitation does not affect the expressive power of the (C-)EPC notation, it is still worth lifting given that OR connectors can add convenience by allowing modelers to represent certain constructions in a more compact way. Extending the proposal presented here in order to directly support the OR connector is a direction for future work.

In previous work [LLS$^+$07] we developed a toolset, namely Synergia, for questionnaire-driven configuration of C-EPC process models. Based on the framework exposed in this article, we have extended this toolset to incrementally check the syntactic and semantic correctness of a C-EPC at each step of its configuration. The toolset can be downloaded from the Process Configuration web-site.[8]

The results presented in this article focus on the control-flow perspective of process modeling. We envisage extending the formal framework presented here to cover other dimensions than control-flow, such as dataflow between tasks in a process model and allocation of tasks to resources. Initial results in this direction are exposed in [LDH$^+$08].

Another direction for future work consists in investigating techniques for automating the construction of configurable process models. A possible starting point is to collect a number of related process models from different (preferably successful) process design projects, and to merge them together. But how this merger can be facilitated is still an open question. For example, since process models are usually represented as graphs, algorithms from the field of graph matching could prove useful [Bun00].

References

[Aal97]   W.M.P. van der Aalst. Verification of Workflow Nets. In P. Azéma and G. Balbo, editors, *Proceedings of the 18th*

---

[8] `www.processconfiguration.com`

International Conference on Application and Theory of Petri Net, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426. Springer, 1997.

[Aal99]   W.M.P. van der Aalst. Formalization and Verification of Event-driven Process Chains. *Information and Software Technology*, 41(10):639–650, 1999.

[AB02]   W.M.P. van der Aalst and T. Basten. Inheritance of Workflows: An Approach to Tackling Problems Related to Change. *Theoretical Computer Science*, 270(1-2):125–203, 2002.

[AC04]   M. Antkiewicz and K. Czarnecki. FeaturePlugIn: Feature Modeling Plug-In for Eclipse. In M.G. Burke, editor, *Proceedings of the 2004 OOPSLA workshop on Eclipse Technology eXchange, (ETX 2004)*, pages 67–72. ACM Press, 2004.

[AIN04]   P. A. Abdulla, S. P. Iyer, and A. Nylén. SAT-Solving the Coverability Problem for Petri Nets. *Formal Methods in System Design*, 24(1):25–43, 2004.

[AL08]   W.M.P. van der Aalst and K.B. Lassen. Translating Unstructured Workflow Processes to Readable BPEL: Theory and Implementation. *Information & Software Technology*, 50(3):131–159, 2008.

[Bat05]   D.S. Batory. Feature Models, Grammars, and Propositional Formulas. In J. H. Obbink and K. Pohl, editors, *Proceedings of the 9th International Conference on Software Product Lines (SPLC'05)*, volume 3714 of *Lecture Notes in Computer Science*, pages 7–20. Springer, 2005.

[BDK07]   J. Becker, P. Delfmann, and R. Knackstedt. Adaptive Reference Modeling: Integrating Configurative and Generic Adaptation Techniques for Information Models. In J. Becker and P. Delfmann, editors, *Proceedings of the Reference Modeling Conference (RM'06)*, pages 27–58. Springer, 2007.

[Bun00]   H. Bunke. Recent Developments in Graph Matching. In A. Sanfeliu, J.J. Villanueva, M. Vanrell, R. Alquezar, A.K. Jain, and J. Kittler, editors, *Proceedings of the 15th International Conference on Pattern Recognition (ICPR'00)*, volume 2, pages 117–124. IEEE Computer Society, 2000.

[CA05]   K. Czarnecki and M. Antkiewicz. Mapping Features to Models: A Template Approach Based on Superimposed Variants. In Robert Glück and Michael R. Lowry, editors, *Proceedings of the 4th International Conference on Generative Programming and Component Engineering*, pages 422–437. Springer, September 2005.

[CHE04]   K. Czarnecki, S. Helsen, and U. Eisenecker. Staged Configuration Using Feature Models. In R.L. Nord, editor, *Proceedings of the 3rd International Conference on Software Product Lines (SPLC'04)*, pages 266–283. Springer, 2004.

[CK97]   T. Curran and G. Keller. *SAP R/3 Business Blueprint: Understanding the Business Process Reference Model*. Upper Saddle River, 1997.

[CKLY98]   J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Deriving Petri Nets from Finite Transition Systems. *IEEE Transactions on Computers*, 47(8):859–882, 1998.

[DE95]   J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1995.

[EC94]   J. Estublier and R. Casallas. The Adele Software Configuration Manager. In *Configuration Management*, pages 99–139. John Wiley & Sons, 1994.

[ER89]   A. Ehrenfeucht and G. Rozenberg. Partial (Set) 2-Structures - Part 1 and Part 2. *Acta Informatica*, 27(4):315–368, 1989.

[ES90]   J. Esparza and M. Silva. Circuits, Handles, Bridges and Nets. In G. Rozenberg, editor, *Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 210–242. Springer, 1990.

[GAJV07]   F. Gottschalk, W.M.P. van der Aalst, and M.H. Jansen-Vullers. Configurable Process Models – A Foundational Approach. In J. Becker and P. Delfmann, editors, *Reference Modeling*, pages 59–78. Springer, 2007.

[GAJVL08]   F. Gottschalk, W.M.P. van der Aalst, M.H. Jansen-Vullers, and M. La Rosa. Configurable Workflow Models. *International Journal of Cooperative Information Systems*, 17(2):177–221, 2008.

[GWJV+09]   F. Gottschalk, T.A.C. Wagemakers, M.H. Jansen-Vullers, W.M.P. van der Aalst, and M. La Rosa. Configurable Process Models – Experiences from a Municipality Case Study. In J. Gordijn, editor, *Proceedings of the 27st International Conference on Advanced Information Systems Engineering (CAiSE'09)*. Springer, 2009.

[HOS05]   K.M. van Hee, O. Oanea, and N. Sidorova. Colored Petri Nets to Verify Extended Event-Driven Process Chains. In R. Meersman and Z. Tari, editors, *Proceedings of CoopIS/DOA/ODBASE 2005*, volume 3760 of *Lecture Notes in Computer Science*, pages 183–201. Springer, 2005.

[HOS+08]   K.M. van Hee, O. Oanea, A. Serebrenik, N. Sidorova, and M. Voorhoeve. History-based joins: Semantics, soundness and implementation. *Data & Knowledge Engineering*, 64(1):24–37, 2008.

[Kin06]   E. Kindler. On the semantics of EPCs: Resolving the vicious circle. *Data & Knowledge Engineering*, 56(1):23–40, 2006.

[KNS92]   G. Keller, M. Nüttgens, and A.-W. Scheer. Semantische Processmodellierung auf der Grundlage Ereignisgesteuerter Processketten (EPK). Veröffentlichungen des Instituts für Wirtschaftsinformatik, University of Saarland, Saarbrücken, 1992. (in German).

[LDH+08]   M. La Rosa, M. Dumas, A.H.M. ter Hofstede, J. Mendling, and F. Gottschalk. Beyond Control-Flow: Extending Business Process Configuration to Roles and Objects. In Q. Li, S. Spaccapietra, E. Yu, and A. Olivé, editors, *Proceedings of the 27th International Conference on Conceptual Modeling (ER'08)*, volume 5231 of *Lecture Notes in Computer Science*, pages 199–215. Springer, 2008.

[LHRS08]   M. La Rosa, A.H.M. ter Hofstede, M. Rosemann, and K. Shortland. Bringing Process to Post Production. In *Proceedings of the International Conference "Creating Value: Between Commerce and Commons"*. Queensland University of Technology, 2008.

[LLS+07]   M. La Rosa, J. Lux, S. Seidel, M. Dumas, and A.H.M. ter Hofstede. Questionnaire-driven Configuration of Reference Process Models. In J. Krogstie, A.L. Opdahl, and G. Sindre, editors, *Proceedings of the 19th International*

Conference on Advanced Information Systems Engineering (CAiSE'07), volume 4495 of *Lecture Notes in Computer Science*, pages 424–438. Springer, 2007.

[LSW98]   P. Langner, C. Schneider, and J. Wehler. Petri Net Based Certification of Event driven Process Chains. In J. Desel and M. Silva, editors, *Application and Theory of Petri Nets 1998*, volume 1420 of *Lecture Notes in Computer Science*, pages 286–305. Springer, 1998.

[MA07]   J. Mendling and W.M.P. van der Aalst. Formalization and Verification of EPCs with OR-Joins Based on State and Context. In J. Krogstie, A.L. Opdahl, and G. Sindre, editors, *Proceedings of the 19th International Conference on Advanced Information Systems Engineering (CAiSE'07)*, volume 4495 of *Lecture Notes in Computer Science*, pages 439–453. Springer, 2007.

[MDA08]   J. Mendling, B.F. van Dongen, and W.M.P. van der Aalst. Getting Rid of OR-Joins and Multiple Start Events in Business Process Models. *Enterprise Information Systems. Special Issue on EDOC 2007 Best Papers*, 2(4):403–419, 2008.

[MIY90]   S. Minato, N. Ishiura, and S. Yajima. Shared Binary Decision Diagram with Attributed Edges for Efficient Boolean Function Manipulation. In R.C. Smith, editor, *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pages 52–57. ACM Press, 1990.

[Mur89]   T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[NR02]   M. Nüttgens and F.J. Rump. Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In J. Desel and M. Weske, editor, *Proceedings of Promise 2002*, volume 21 of *Lecture Notes in Informatics*, pages 64–77. GI, 2002.

[ODA+09]   C. Ouyang, M. Dumas, W.M.P. van der Aalst, A.H.M. ter Hofstede, and J. Mendling. From Business Process Models to Process-oriented Software Systems: The BPMN to BPEL Way. *ACM Transactions on Software Engineering and Methodology*, 2009. (forthcoming).

[PBL05]   K. Pohl, G. Böckle, and F. van der Linden. *Software Product-line Engineering – Foundations, Principles and Techniques*. Springer, 2005.

[RA07]   M. Rosemann and W. M. P van der Aalst. A Configurable Reference Modelling Language. *Information Systems*, 32(1):1–23, 2007.

[RK08]   M. Razavian and R. Khosravi. Modeling Variability in Business Process Models Using UML. In S. Latifi, editor, *Proceedings of the 5th International Conference on Information Technology: New Generations (ITGN'08)*, pages 82–87, 2008.

[SL05]   K. Sarshar and P. Loos. Comparing the Control-Flow of EPC and Petri Net from the End-User Perspective. In W.M.P. van der Aalst, B. Benatallah, and F. Casati, editors, *Proceedings of the 3rd International Conference on Business Process Management (BPM'05)*, volume 3649 of *Lecture Notes in Computer Science*, pages 434–439. Springer, 2005.

[SOS05]   S.W. Sadiq, M.E. Orlowska, and W. Sadiq. Specification and Validation of Process Constraints for Flexible Workflows. *Information Systems*, 30(5):349–378, 2005.

[SP06]   A. Schnieders and F. Puhlmann. Variability Mechanisms in E-Business Process Families. In W. Abramowicz and H.C. Mayr, editors, *Proceedings of the 9th International Conference on Business Information Systems (BIS'06)*, volume 85 of *LNI*, pages 583–601. GI, 2006.

[Ste01]   S. Stephens. The Supply Chain Council and the Supply Chain Operations Reference Model. *Supply Chain Management – An International Journal*, 1(1):9–13, 2001.

[TGN04]   E. Turkay, A.S. Gokhale, and B. Natarajan. Addressing the Middleware Configuration Challenges using Model-based Techniques. In S.-M. Yoo and L.H. Etzkorn, editors, *Proceedings of the 42nd ACM Southeast Regional Conference*, pages 166–170. ACM Press, 2004.

[VBA01]   H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst. Diagnosing Workflow Processes using Woflan. *The Computer Journal*, 44(4):246–279, 2001.