Correctness Ensuring Process Configuration: An Approach Based on Partner Synthesis

Wil van der Aalst[1,3], Niels Lohmann[1,2], Marcello La Rosa[3], and Jingxin Xu[3]

[1] Eindhoven University of Technology, The Netherlands
w.m.p.v.d.aalst@tue.nl
[2] Universität Rostock, Germany
niels.lohmann@uni-rostock.de
[3] Queensland University of Technology, Australia
m.larosa@qut.edu.au,jingxin.xu@connect.qut.edu.au

Abstract. A configurable process model describes a family of similar process models in a given domain. Such a model can be configured to obtain a *specific* process model that is subsequently used to handle individual cases, for instance, to process customer orders. *Process configuration is notoriously difficult as there may be all kinds of interdependencies between configuration decisions.* In fact, an incorrect configuration may lead to behavioral issues such as deadlocks and livelocks. To address this problem, we present a novel verification approach inspired by the "operating guidelines" used for partner synthesis. We view the configuration process as an external service, and compute a characterization of all such services which meet particular requirements using the notion of *configuration guideline*. As a result, we can characterize all feasible configurations (i. e., configurations without behavioral problems) at design time, instead of repeatedly checking each individual configuration while configuring a process model.

Key words: Configurable process model, operating guideline, Petri nets

1 Introduction and Background

Although large organizations support their processes using a wide variety of process-aware information systems, the majority of business processes are still not directly driven by process models. Despite the success of Business Process Management (BPM) thinking in organizations, Workflow Management (WfM) systems — today often referred to as *BPM systems* — are not widely used. One of the main problems of BPM technology is the "lack of content", that is, providing just a generic infrastructure to build process-aware information systems is insufficient as organizations need to support specific processes. Organizations want to have "out-of-the-box" support for standard processes and are only willing to design and develop system support for organization-specific processes. Yet most BPM systems expect users to model basic processes from scratch. Enterprise Resource Planning (ERP) systems such as SAP and Oracle, on the other hand, focus on the support of these common processes. Although all ERP systems have workflow engines comparable to the engines of BPM systems, the majority of processes are not supported by software which is driven by models. For example, most of SAP's functionality is not grounded in their workflow component, but hard-coded in application software. ERP vendors try to capture "best practices" in dedicated applications designed for a particular purpose. Such systems can be configured by setting parameters. System configuration can be a time consuming and complex process. Moreover, configuration parameters are exposed as "switches in the application software", thus making it difficult to see the intricate dependencies among certain settings.

A model-driven process-oriented approach toward supporting business processes has all kinds of benefits ranging from improved analysis possibilities (verification, simulation, etc.) and better insights, to maintainability and ability to rapidly develop organization-specific solutions. Although obvious, this approach has not been adopted thus far, because BPM vendors have failed to provide content and ERP vendors suffer from the "Law of the handicap of a head start". ERP vendors manage to effectively build data-centric solutions to support particular tasks. However, the complexity and large installed base of their products makes it impossible to refactor their software and make it process-centric.

Based on the limitations of existing BPM and ERP systems, we propose to use *configurable process models*. A configurable process model represents a *family of process models*, that is, a model that through configuration can be customized for a particular setting. Configuration is achieved by *hiding* (i. e., bypassing) or *blocking* (i. e., inhibiting) certain fragments of the configurable process model [12]. In this way, the desired behavior is selected. From the viewpoint of generic BPM software, configurable process models can be seen as a mechanism to add content to these systems. By developing comprehensive collections of configurable models, particular domains can be supported. From the viewpoint of ERP software, configurable process models can be seen as a means to make these systems more process-centric, although in the latter case quite some refactoring is needed as processes are hidden in table structures and application code.

Various configurable languages have been proposed as extensions of existing languages (e. g., C-EPCs [22], C-iEPCs [17], C-WF-nets [3], C-SAP, C-BPEL) but few are actually supported by enactment software (e. g., C-YAWL [13]). In this paper, we are interested in models in the latter class of languages, which, unlike traditional reference models [9,8,11], are executable after they have been configured. Specifically, we focus on the *verification of configurable executable process models*. In fact, because of hiding and/or blocking selected fragments, the instances of a configured model may suffer from behavioral anomalies such as deadlocks and livelocks. This problem is exacerbated by the total number of possible configurations a model may have, and by the complex domain dependencies which may exist between various configuration options. For example, the configurable process model we constructed from the VICS documentation — an industry standard for logistics and supply chain management — comprises 50 activities. Each of these activities may be "blocked", "hidden", or "allowed", depending on the configuration requirements. This results in $3^{50} \approx 7.18e{+}23$ possible configurations. Clearly, checking the *feasibility* of each single configuration can be time consuming as this would typically require to perform state-space analysis. Moreover, characterizing the "family of correct models" for a particular configurable process model is even more difficult and time-consuming as a naive approach would require to solve an exponential number of state-space problems.

As far as we know, our earlier approach [3] is the only one focusing on the verification of configurable process models which takes into account behavioral correctness and avoids the state-space explosion problem. Other approaches either only discuss syntactical correctness related to configuration [22,10,8], or deal with behavioral correctness but run into the state-space problem [14]. In this paper, we propose a completely novel verification approach where we consider the configuration process as an "external service" and then synthesize a "most permissive partner" using the approach described by Wolf [24] and implemented in the tool Wendy [21]. This most permissive partner is closely linked to the notion of *operating guidelines* for service behavior [20]. In this paper, we define for any configurable model a so-called *configuration guideline* to characterize all correct process configurations. This approach provides the following advantages over our previous approach [3]:

- We provide a *complete characterization of all possible configurations at design time*, that is, the *configuration guideline*.
- Computation time is moved *from configuration time to design time* and results can be reused more easily.
- *No restrictions are put on the class of models* which can be analyzed. The previous approach [3] was limited to sound free-choice WF-nets. Our new approach can be applied to models which do not need to be sound, which can have complex (non-free choice) dependencies, and which can have multiple end states.

To prove the practical feasibility of this new approach, we have implemented it as a component of the toolset supporting C-YAWL.

The remainder of this paper is organized as follows. Section 2 introduces basic concepts such as open nets and weak termination. These concepts are used in Section 3 to formalize the notion of process configuration. Section 4 presents the solution approach for correctness ensuring configuration. Section 5 discusses tool support and Section 6 concludes the paper.

2 Business Process Models

For the formalization of the problem we use Petri nets which offer a formal model of concurrent systems. However, the same ideas can be applied to other languages (e. g., C-YAWL, C-BPEL).

Definition 1 (Petri net). *A* marked Petri net *is a tuple* $N = (P, T, F, m_0)$ *such that:* P *and* T *(*$P \cap T = \emptyset$*) are finite sets of places and transitions, respectively,* $F \subseteq (P \times T) \cup (T \times P)$ *is a flow relation, and* $m_0 : P \rightarrow \mathbb{N}$ *is an initial marking.*

A Petri net is a directed graph with two types of nodes: places and transitions, which are be connected by arcs as specified in the flow relation. If $p \in P$, $t \in T$, and $(p, t) \in F$, then place p is an input place of t. Similarly, $(t, p) \in F$ means that p is an output place of t.

The *marking* of a Petri net describes the distribution of tokens over places and is represented by a *multiset of places*. For example, the marking $m = [a^2, b, c^4]$ indicates

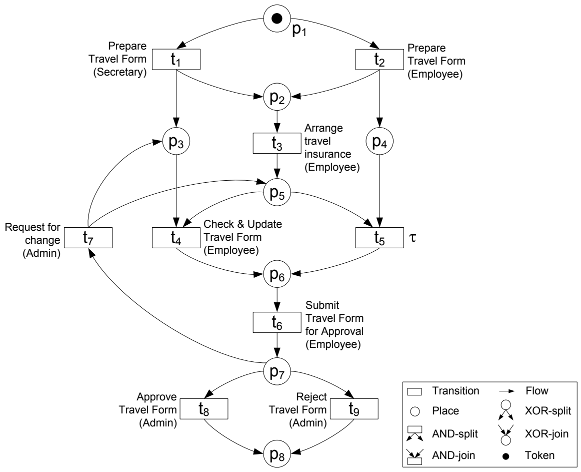

p1
Prepare Travel Form (Secretary)
t1
t2
Prepare Travel Form (Employee)
p2
p3
t3
Arrange travel insurance (Employee)
p4
p5
Request for change (Admin)
t7
t4
Check & Update Travel Form (Employee)
t5
τ
p6
t6
Submit Travel Form for Approval (Employee)
p7
Approve Travel Form (Admin)
t8
Reject Travel Form (Admin)
t9
p8
Transition
Flow
Place
XOR-split
AND-split
XOR-join
AND-join
Token

Fig. 1. The open net for travel request approval ($\Omega = \{[p_8]\}$).

that there are two tokens in place a, one token in b, and four tokens in c. Formally m is a function such that $m(a) = 2$, $m(b) = 1$, and $m(c) = 4$. We use $\oplus$ to compose multisets; for instance, $[a^2, b, c^4] \oplus [a^2, b, d^2, e] = [a^4, b^2, c^4, d^2, e]$.

A transition is *enabled* and can *fire* if all its input places contain at least one token. Firing is atomic and consumes one token from each of the input places and produces one token on each of the output places. $m_0 \xrightarrow{t} m$ means that t is enabled in marking m_0 and the firing of t in m_0 results in marking m. We use $m_0 \xrightarrow{*} m$ to denote that m is reachable from m_0, that is, there exists a (possibly empty) sequence of enabled transitions leading from m_0 to m.

For our configuration approach, we use *open nets*. Open nets extend classical Petri nets with the identification of final markings Ω and a labeling function ℓ.

Definition 2 (Open net). *A tuple* $N = (P, T, F, m_0, \Omega, L, \ell)$ *is an* open net *if*

- (P, T, F, m_0) *is a marked Petri net (called the* inner net *of* N*),*
- $\Omega \subset P \to \mathbb{N}$ *is a finite set of* final markings*,*
- L *is a finite set of* labels*,*
- $\tau \notin L$ *is a label representing* invisible *(also called silent)* steps*, and*
- $\ell : T \to L \cup \{\tau\}$ *is a* labeling function.

We use *transition labels* to represent the activity corresponding to the execution of a particular transition. Moreover, if an activity appears multiple times in a model, we use the same label to identify all the occurrences of that activity. The special label τ refers to an invisible step, sometimes referred to as "silent". Invisible transitions are typically used to represent internal actions which do not mean anything at the business level (cf. the "inheritance of dynamic behavior" framework [2,7]). We use visible labels to denote activities that may be configured while in Section 4 we use these labels to synchronize two open nets.

Figure 1 shows an example open net which models a typical travel request approval. The process starts with the preparation of the travel form. This can either be done by an employee or be delegated to a secretary. In both cases, the employee personally needs to arrange the travel insurance. If the travel form has been prepared by the secretary, the employee needs to check it before submitting it for approval. An administrator can then approve or reject the request, or make a request for change. Now, the employee can update the form according to the administrator's suggestions and resubmit it. In Fig. 1 all transitions bear a visible label, except for t_5 which bears a τ-label as it has only been added for routing purposes.

Unlike our previous approach [3] based on WF-nets [1] and hence limited to a single final place, we allow for *multiple final markings* here. Good runs of an open net end in a marking in set Ω. Therefore, an open net is considered to be erroneous if it can reach a marking from which no final marking can be reached any more. An open net *weakly terminates* if a final marking is reachable from every reachable marking.

Definition 3 (Weak termination). *An open net* $N = (P, T, F, m_0, \Omega, L, \ell)$ weakly terminates *if and only if (iff) for any marking* m *with* $m_0 \overset{*}{\rightarrow} m$ *there exists a final marking* $m_f \in \Omega$ *such that* $m \overset{*}{\rightarrow} m_f$.

The net in Fig. 1 is weakly terminating. Weak termination is a weaker notion than soundness, as it does not require transitions to be quasi-live [1]. This correctness notion is more suitable as parts of a correctly configured net may be left dead intentionally.

3 Process Model Configuration

We use open nets to model configurable process models. An open net can be configured by blocking or hiding transitions which bear a visible label. Blocking a transition means that the corresponding activity is no longer available and none of the paths with that transition cannot be taken any more. Hiding a transition means that the corresponding activity is bypassed, but paths with that transition can still be taken. If a transition is neither blocked nor hidden, we say it is allowed, meaning it remains in the model. Configuration is achieved by setting visible labels to *allow*, *hide* or *block*.

Definition 4 (Open net configuration). *Let* N *be an open net with label set* L*. A mapping* $C_N : L \rightarrow \{allow, hide, block\}$ *is a* configuration for N*. We define:* $A_N^C = \{t \in T \mid \ell(t) \neq \tau \ \wedge \ C_N(\ell(t)) = allow\}$, $H_N^C = \{t \in T \mid \ell(t) = \tau \ \vee \ C_N(\ell(t)) = hide\}$*, and* $B_N^C = \{t \in T \mid \ell(t) \neq \tau \ \wedge \ C_N(\ell(t)) = block\}$.

An open net configuration implicitly defines an open net, called *configured net*, where the blocked transitions are removed and the hidden transitions are given a τ-label.

Definition 5 (Configured net). *Let* $N = (P, T, F, m_0, \Omega, L, \ell)$ *be an open net and* C_N *a configuration of* N*. The resulting* configured net $\beta_N^C = (P, T^C, F^C, m_0, \Omega, L, \ell^C)$ *is defined as follows:* $T^C = T \setminus (B_N^C)$, $F^C = F \cap ((P \cup T_C) \times (P \cup T_C))$*, and* $\ell^C(t) = \ell(t)$ *for* $t \in A_N^C$ *and* $\ell^C(t) = \tau$ *for* $t \in H_N^C$.

As an example, Fig. 2(a) shows the configured net derived from the open net in Fig. 1 and the configuration $C_N(\textit{Prepare Travel Form (Secretary)}) = \textit{block}$ (to allow only

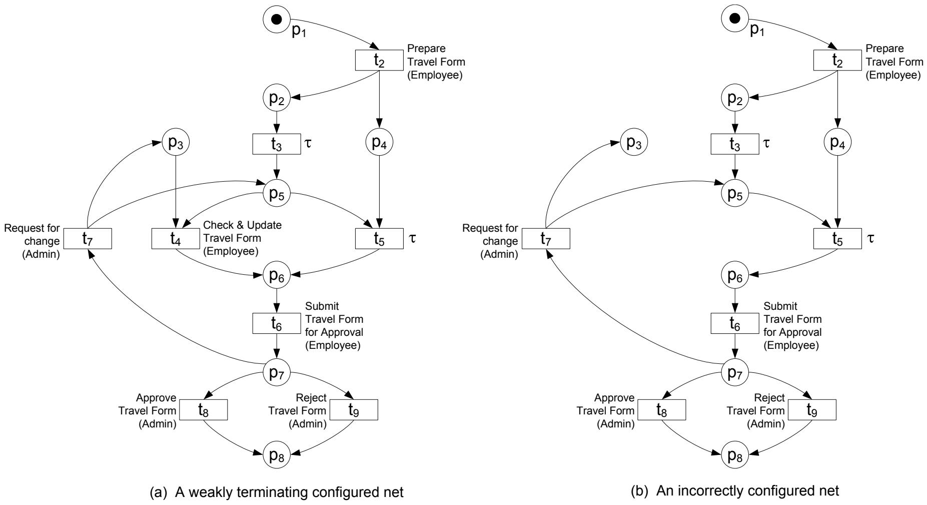

p_1
t_2
Prepare Travel Form (Employee)
p_2
t_3
τ
p_3
p_4
p_5
Request for change (Admin)
t_7
t_4
Check & Update Travel Form (Employee)
t_5
τ
p_6
t_6
Submit Travel Form for Approval (Employee)
p_7
Approve Travel Form (Admin)
t_8
Reject Travel Form (Admin)
t_9
p_8
(a) A weakly terminating configured net
(b) An incorrectly configured net

Fig. 2. Two possible configured nets based on the model in Fig. 1.

employees to prepare travel forms), $C_N(\mathit{Arrange\ Travel\ Insurance}\ (\mathit{Employee})) = \mathit{hide}$ (to skip arranging the travel insurance), and $C_N(x) = \mathit{allow}$ for all other labels x.

Typically, configurable process models cannot be freely configured, because the use of hiding and blocking has to comply with the application domain in which the model has been constructed. For instance, in the travel request example we cannot hide the labels of both t_1 and t_2, because all the other activities depend on the preparation of the travel form, nor block the label of t_8, because there must be an option to approve the travel request. The link between configurable process models and domain decisions was explored in [18] and can be incorporated easily (see Sect. 6)

A configured net may have disconnected nodes and some parts may be dead (i. e., can never become active). Such parts can easily be removed. However, as we impose no requirements on the structure of configurable models, these disconnected or dead parts are irrelevant with respect to weak termination. For example, if we block the label of t_2 in Fig. 1, transition t_5 becomes dead as it cannot be enabled any more, and hence can also be removed without causing any behavioral issues. Nonetheless, not every configuration of an open net results in a weakly terminating configured net. For example, by blocking the label of t_4 in the configured net of Fig. 2(a), we obtain the configured net in Fig. 2(b). This net is not weakly terminating because after firing t_7 tokens will get stuck in p_3 (as this place does not have any successor) and in p_5 (as t_5 can no longer fire).

Blocking can cause behavioral anomalies such as the deadlock in Fig. 2(b). However, hiding cannot cause such issues, because it merely changes the labels of an open net. Hence, we shall focus on blocking rather than hiding. In this paper we are interested in all configurations which yield weakly terminating configured nets. We use the term *feasibility* to refer to such configured nets.

Definition 6 (Feasible configuration). *Let N be an open net and C_N a configuration of N. C_N is* feasible *iff the configured net β_N^C weakly terminates.*

Given a configurable process model N, we are interested in the following two questions: i) Is a *particular* configuration C_N feasible? ii) How to characterize the set of *all* feasible configurations?

The remainder of this paper is devoted to a new verification approach answering these questions. This approach extends the work in [3] in two directions: (i) it imposes *no unnecessary requirements* on the configurable process model (allowing for non-free-choice nets and nets with multiple end places/markings), and (ii) it checks a *weaker correctness* notion (i. e., weak termination instead of soundness). For instance, the net in Fig. 1 is not free-choice because t_4 and t_5 share an input place, but their sets of input places are not identical. The non-free-choice construct is needed to model that after firing t_1 or t_7, t_5 cannot be fired, and similarly, after firing t_2, t_4 cannot be fired.

4 Correctness Ensuring Configuration

To address the two main questions posed in the previous section, we could use a direct approach by enumerating all possible configurations and simply checking whether each of the configured nets β_N^C weakly terminates. As indicated before, the number of possible configurations is exponential in the number of configurable activities. Moreover, most techniques for checking weak termination typically require the construction of the state space. Hence, traditional approaches are computationally expensive and do not yield a useful characterization of the set of all feasible configuration. Consequently, we propose a completely different approach using the synthesis technique described in [24]. *The core idea is to see the configuration as an "external service" and then synthesize a "most permissive partner".* This most permissive partner represents all possible "external configuration services" which yield a feasible configuration. The idea is closely linked to the notion of *operating guidelines* for service behavior [20]. An operating guideline is a finite representation of all possible partners. Similarly, our *configuration guideline characterizes all feasible process configurations.* This configuration guideline can also be used to *efficiently check the feasibility of a particular configuration without exploring the state space of the configured net.* Our approach consists of three steps:

1. Transform the configurable process model N into a *configuration interface* N^{CI}.
2. Synthesize the "most permissive partner" (our *configuration guideline*) Q^{C_N} for the configuration interface N^{CI}.
3. Study the composition of N^{CI} with Q^{C_N}.

We first introduce the notion of *composition*. Open nets can be composed by synchronizing transitions according to their visible labels. In the resulting net, all transitions bear a τ-label and labeled transitions without counterpart in the other net disappear.

Definition 7 (Composition). *For $i \in \{1,2\}$, let $N_i = (P_i, T_i, F_i, m_{0_i}, \Omega_i, L_i, \ell_i)$ be open nets. N_1 and N_2 are* composable *iff the inner nets of N_1 and N_2 are pairwise disjoint. The* composition of two composable open nets *is the open net $N_1 \oplus N_2 = (P, T, F, m_0, \Omega, L, \ell)$ with:*

- $P = P_1 \cup P_2$,
- $T = \{t \in T_1 \cup T_2 \mid \ell(t) = \tau\} \cup \{(t_1, t_2) \in T_1 \times T_2 \mid \ell(t_1) = \ell(t_2) \neq \tau\}$,

- $F = (F_1 \cup F_2) \cap ((P \times T) \cup (T \times P)) \cup \{(p, (t_1, t_2)) \in P \times T \mid (p, t_1) \in F_1 \vee (p, t_2) \in F_2\} \cup \{((t_1, t_2), p) \in T \times P \mid (t_1, p) \in F_1 \vee (t_2, p) \in F_2\}$,
- $m_0 = m_{0_1} \oplus m_{0_2}, \quad \Omega = \{m_1 \oplus m_2 \mid m_1 \in \Omega_1 \wedge m_2 \in \Omega_2\}$,
- $L = \emptyset$, *and* $\ell(t) = \tau$ *for* $t \in T$.

Composition can limit the behavior of each original net; for instance, transitions may no longer be available or may be blocked by one of the two original nets. Hence, it is possible that N_1 and N_2 are weakly terminating, but $N_1 \oplus N_2$ is not. Similarly, $N_1 \oplus N_2$ may be weakly terminating, but N_1 and N_2 are not. The labels of the two open nets in Def. 7 serve now a different purpose: they are not used for configuration, but to synchronize the two nets as described in [24].

With the notions of composition and weak termination, we define the concept of *controllability*, which we need to reason about the existence of feasible configurations.

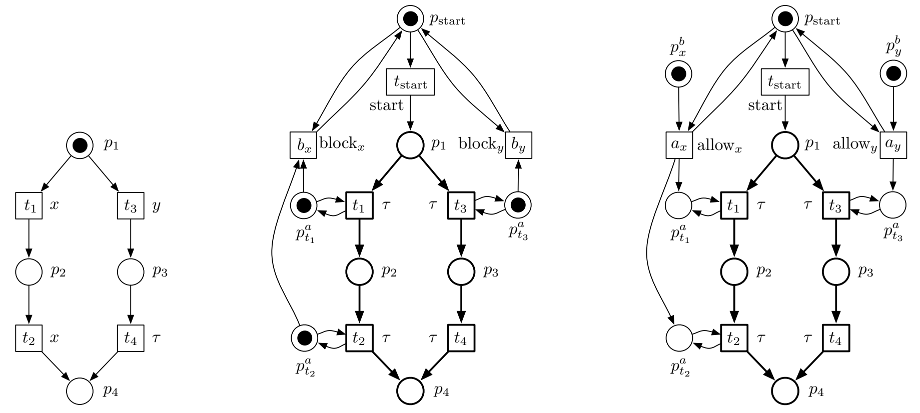
Definition 8 (Controllability). *An open net N is* controllable *iff there exists an open net N' (called* partner*) such that $N \oplus N'$ is weakly terminating.*

In [24], Wolf presents an algorithm to check controllability: if an open net is controllable, this algorithm can synthesize a partner.

After these preliminaries, we define the notion of a *configuration interface*. One of the objectives of this paper was to characterize the set of all feasible configurations by synthesizing a "most permissive partner". To do this, we transform a configurable process model (i. e., an open net N) into an open net N^{CI}, called the configuration interface, which can communicate with services which configure the original model. In fact, we shall provide two configuration interfaces: one where everything is *allowed by default* and the external configuration service can block labels, and the other where everything is *blocked by default* and the external configuration service can allow labels. These two interfaces allow us to configure both nets where all transitions are initially allowed (and configuration is done by blocking transitions) and nets where all transitions are initially blocked (and configuration is done by allowing transitions). In either case, the resulting open net N^{CI} is controllable iff there exists a feasible configuration C_N of N. Without loss of generality, we assume a 1-safe initial marking, that is, $m_0(p) > 0$ implies $m_0(p) = 1$. This assumption helps to simplify the configuration interface.

Definition 9 (Configuration interface – allow by default). *Let $N = (P, T, F, m_0, \Omega, L, \ell)$ be an open net. We define the open net with configuration interface $N_a^{CI} = (P^C, T^C, F^C, m_0^C, \Omega^C, L^C, \ell^C)$ with:*

- $T^V = \{t \in T \mid \ell(t) \neq \tau\}$,
- $P^C = P \cup \{p_{start}\} \cup \{p_t^a \mid t \in T^V\}, \quad T^C = T \cup \{t_{start}\} \cup \{b_x \mid x \in L\}$,
- $F^C = F \cup \{(p_{start}, t_{start})\} \cup \{(t_{start}, p) \mid p \in P \wedge m_0(p) = 1\} \cup \{(t, p_t^a) \mid t \in T^V\} \cup \{(p_t^a, t) \mid t \in T^V\} \cup \{(b_x, p_{start}) \mid x \in L\} \cup \{(p_{start}, b_x) \mid x \in L\} \cup \{(p_t^a, b_{\ell(t)}) \mid t \in T^V\}$,
- $m_0^C = [p^1 \mid p \in \{p_{start}\} \cup \{p_t^a \mid t \in T^V\}]$,
- $\Omega^C = \{m \oplus \bigoplus_{t \in T} m_t^* \mid m \in \Omega \ \wedge \ \forall_{t \in T} \ m_t^* \in \{[\,], [p_t^a]\}\ \}$,
- $L^C = \{start\} \cup \{block_x \mid x \in L\}$
- $\ell^C(t_{start}) = start$, $\ell^C(b_x) = block_x$ *for* $x \in L$, *and* $\ell^C(t) = \tau$ *for* $t \in T$.



(a) Open net N ($\Omega = \{[p_4]\}$)

(b) Configurable interface N_a^{CI} (allow by default)

(c) Configurable interface N_b^{CI} (block by default)

Fig. 3. An example open net (a) and its two configuration interfaces (b,c).

Figure 3 illustrates the two configuration interfaces for a simple open net N. In both interfaces, the original net N consisting of places $\{p_1, p_2, p_3, p_4\}$ and transitions $\{t_1, t_2, t_3, t_4\}$ is retained, but all transition labels are set to τ. Let us focus on the configuration interface where all activities are allowed by default (Fig. 3(b)). Here transitions b_x and b_y are added to model the blocking of labels x and y, respectively. Places $p^a_{t_1}$, $p^a_{t_2}$, and $p^a_{t_3}$ are also added to connect the new transitions to the existing ones, and are initially marked as all configurable transitions are allowed by default. Firing b_x will block t_1 and t_2 by removing the tokens from $p^a_{t_1}$ and $p^a_{t_2}$. These two transitions are blocked at the same time as both bear the same label x in N. Firing b_y will block t_3. Transitions b_x and b_y are labeled respectively block$_x$ and block$_y$. This means that in the composition with a partner they can only fire if a corresponding transition in the partner can fire. Transition *start* has been added to ensure configuration actions take place *before* the original net is activated. In this way, we avoid "configuration on the fly". Figure 3(c) shows the construction of the configuration interface where all activities are blocked and is discussed later.

Consider now a configuration service represented as an open net Q. $N_a^{CI} \oplus Q$ is the composition of the original open net (N) extended with a configuration interface (N_a^{CI}), and the configuration service Q. First, blocking transitions such as b_x and b_y can fire (apart from unlabeled transitions in Q). Next, transition *start* fires after which blocking transitions such as b_x and b_y can no longer fire. Hence, only the original transitions in N_a^{CI} can fire in the composition after firing *start*. The configuration service Q may still execute transitions, but these cannot influence N_a^{CI} any more. Hence, Q represents a feasible configuration iff N_a^{CI} can reach one of its final markings from any reachable marking in the composition. So Q corresponds to a feasible configuration iff $N_a^{CI} \oplus Q$ is weakly terminating, that is, Q is a partner of N_a^{CI}.

To illustrate the basic idea, we introduce the notion of a *canonical configuration partner*, that is, the representation of a configuration $C_N : L \rightarrow \{allow, hide, block\}$ in terms of an open net which synchronizes with the original model extended with a configuration interface.

Definition 10 (Canonical configuration partner – allow by default). *Let N be an open net and let $C_N : L \to \{allow, hide, block\}$ be a configuration for N. $Q_a^{C_N} = (P, T, F, m_0, \Omega, L^Q, \ell)$ is the* canonical configuration partner *with:*

- $B = \{x \in L \mid C_N(x) = block\}$ *is the set of blocked labels,*
- $P = \{p_x^0 \mid x \in B\} \cup \{p_x^\omega \mid x \in B\}$, $T = \{t_x \mid x \in B\} \cup \{t_{start}\}$,
- $F = \{(p_x^0, t_x) \mid x \in B\} \cup \{(t_x, p_x^\omega) \mid x \in B\} \cup \{(p_x^\omega, t_{start}) \mid x \in B\}$,
- $m_0 = [(p_x^0)^1 \mid x \in B]$, $\Omega = \{\, [\,]\, \}$,[4]
- $L^Q = \{block_x \mid x \in B\} \cup \{start\}$, *and* $\ell(t_x) = block_x$ *for* $x \in B$, $\ell(t_{start}) = start$.

The set of labels which need to be blocked to mimic configuration C_N is denoted by B. The canonical configuration partner $Q_a^{C_N}$ has a transition for each of these labels. These transitions may fire in any order after which the transition with label *start* fires. We observe that in the composition $N_a^{CI} \oplus Q_a^{C_N}$ first all transitions with a label in $\{\text{block}_x \mid x \in B\}$ fire in a synchronous manner, followed by the transition with label *start* (in both nets). After this, the net is configured and $Q_a^{C_N}$ plays no role in the composition $N_a^{CI} \oplus Q_a^{C_N}$ any more.

The following lemma formalizes the relation between the composition $N_a^{CI} \oplus Q_a^{C_N}$ and feasibility.

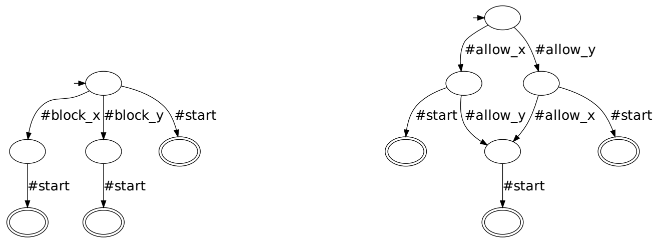
Lemma 1. *Let N be an open net and let C_N be a configuration for N. C_N is a feasible configuration iff $N_a^{CI} \oplus Q_a^{C_N}$ is weakly terminating.*

Proof. ($\Rightarrow$) Let C_N be a feasible configuration for N and let N_a^{CI} be as defined in Def. 9. Consider the composition $N_a^{CI} \oplus Q_a^{C_N}$ after the synchronization via label *start* has occurred. By construction, (1) $N_a^{CI} \oplus Q_a^{C_N}$ reached the marking $m = m_0 \oplus m_1 \oplus m_2$ such that m_0 is the initial marking of N, m_1 marks all places p_t^a of transitions $t \in A_N^C \cup H_N^C$, and m_2 is the empty marking of Q^{C_N}. Furthermore, (2) all transitions which bear a synchronization label (i. e., t_{start} and all b_x transitions) and all $t \in B_N^C$ are dead in m and cannot become enabled any more. From N_a^{CI}, construct the net N^* by removing these transitions and their adjacent arcs, as well as the places p_{start} and p_t^a for all $t \in T^V$. The resulting net N^* coincides with β_N^C (modulo renaming). Hence, $N_a^{CI} \oplus Q_a^{C_N}$ weakly terminates.
($\Leftarrow$) Assume $N_a^{CI} \oplus Q_a^{C_N}$ weakly terminates. From $Q_a^{C_N}$, we can straightforwardly derive a configuration C for N in which all labels are blocked which occur in $N_a^{CI} \oplus Q_a^{C_N}$. With the same observation as before, we can conclude that β_N^C coincides with the net N^* constructed from N_a^{CI} after the removal the described nodes. Hence, β_N^C weakly terminates and C is a feasible configuration for N. □

Lemma 1 states that checking the feasibility of a particular configuration can be reduced to checking for weak termination of the composition. However, the reason for modeling configurations as partners is that we can synthesize partners and test for the existence of feasible configurations.

Theorem 1 (Feasibility coincides with controllability). *Let N be an open net. N_a^{CI} is controllable iff there exists a feasible configuration C_N of N.*

[4] $[x^k \mid x \in X]$ denotes the multiset where each element of X appears k times. $[\,]$ denotes the empty multiset.



(a) CG^a Allow by default

(b) CG^b Block by default

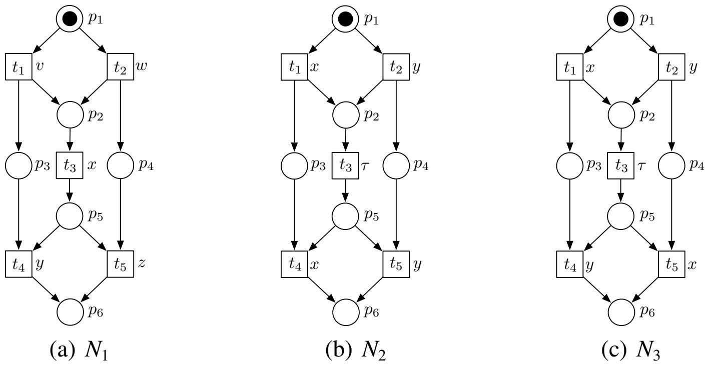
Fig. 4. Two configuration guidelines characterizing all possible configurations.

Proof. (⇒) If N_a^{CI} is controllable, then there exists a partner N' of N_a^{CI} such that $N_a^{CI} \oplus N'$ is weakly terminating. Consider a marking m of the composition reached by a run σ from the initial marking of $N_a^{CI} \oplus N'$ to the synchronization via label *start*. Using the construction from the proof of Lemma 1, we can derive a net N^* from N_a^{CI} which coincides with a configured net β_N^C for a configuration C_N. As $N_a^{CI} \oplus N'$ is weakly terminating, C_N is feasible.
(⇐) If C_N is a feasible configuration of N, then by Lemma 1, $N_a^{CI} \oplus Q_a^{C_N}$ weakly terminates and by Def. 8, N_a^{CI} is controllable. □

As shown in [24], it is possible to synthesize a partner which is *most-permissive*. This partner simulates any other partner and thus characterizes all possible feasible configurations. In previous papers on partner synthesis in the context of service oriented computing, the notion of an *operating guideline* was used to create a finite representation capturing all possible partners [20]. Consequently, we use the term *Configuration Guideline* (CG) to denote the most-permissive partner of a configuration interface. Fig. 4(a) shows the configuration guideline CG^a for the configurable model in Fig. 3(a), computed from the configuration interface N_a^{CI} in Fig. 3(b).

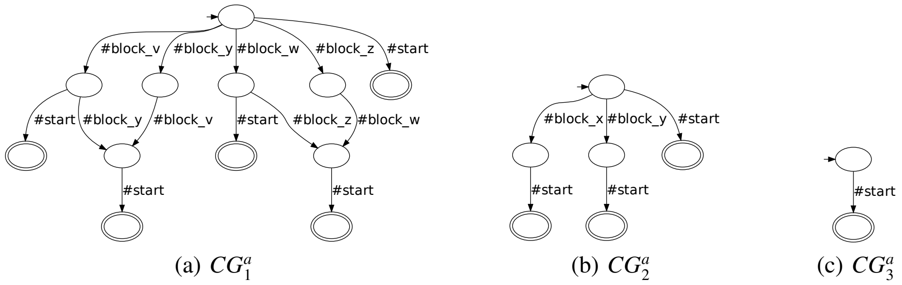
A configuration guideline is an automaton with one start state and one or more final states. *Any path in the configuration guideline starting in the initial state and ending in a final state corresponds to a feasible configuration.* The initial state in Fig. 4(a) is denoted by a small arrow and the final states are denoted by double circles. The leftmost path in Fig. 4(a) (i. e., $\langle \text{block}_x, \text{start} \rangle$), corresponds to the configuration which blocks label x. Path $\langle \text{block}_y, \text{start} \rangle$ corresponds to the configuration which blocks label y. The rightmost path (i. e., $\langle \text{start} \rangle$) does not block any label. The three paths capture all three feasible configurations. For example, blocking both labels is not feasible. Figure 4(a) is trivial because there are only two labels and three feasible configurations.

Thus far, we used a configuration interface that allows all configurable activities by default, that is, blocking is an explicit action of the partner. It is also possible to use a completely different starting point and initially block all activities. As this “block by default” strategy is analogous to the “allow by default” approach we discussed before, we refer to [5] for formal definitions and proofs. Figure 3(c) depicts the “block by default” configuration interface for the net N_1 (Fig. 3(a)) and Fig. 4(b) shows the respective configuration guideline.



(a) N_1 (b) N_2 (c) N_3

Fig. 5. Three open nets ($\Omega = \{[p_6]\}$).



(a) CG_1^a (b) CG_2^a (c) CG_3^a

Fig. 6. The configuration guidelines (allow by default) for N_1 (a), N_2 (b) and N_3 (c).

Let us now consider a more elaborated example to see how configuration guidelines can be used to rule out unfeasible configurations. Figure 5 shows three open nets. The structures are identical, only the labels are different. For example, blocking x in N_2 corresponds to removing both t_1 and t_4 as both transitions bear the same label, while blocking x in N_3 corresponds to removing t_1 and t_5. For these three nets, we can construct the configuration interfaces using Def. 9 and then synthesize the configuration guidelines, as shown in Fig. 6.

Figure 6(a) reveals all feasible configurations for N_1 in Fig. 5(a). From the initial state in the configuration guideline CG_1^a, we can immediately reach a final state by following the rightmost path $\langle \text{start} \rangle$. This indicates that all configurations which block nothing (i. e., only allow or hide activities) are feasible. It is possible to just block v (cf. path $\langle \text{block}_v, \text{start} \rangle$) or block both v and y (cf. paths $\langle \text{block}_v, \text{block}_y, \text{start} \rangle$ and $\langle \text{block}_y, \text{block}_v, \text{start} \rangle$). However, it is not allowed to block y only, otherwise a token would deadlock in p_3. For the same reasons, one can block w only or w and z, but not z only. Moreover, it is not possible to combine the blocking of w and/or z on the one hand and v and/or y on the other hand, otherwise no final marking can be reached. Also x can never be blocked, otherwise both v and w would also need to be blocked (to avoid a token to deadlock in p_2) which is not possible. There are $3^5 = 243$ configurations for N_1. If we abstract from hiding as this does not influence feasibility, there remain $2^5 = 32$ possible configurations. Of these only 5 are feasible configurations which correspond to the final states in Fig. 6(a). This illustrates that the configuration guideline can indeed represent all feasible configurations in an intuitive manner.

Figure 6(b) shows the three feasible configurations for N_2 in Fig. 5(b). Again all final states correspond to feasible configurations. Here one can block the two leftmost transitions (labeled x) or the two rightmost transitions (labeled y), but not both.

The configuration guideline in Fig. 6(c) shows that nothing can be blocked for N_3 (Fig. 5(c)). Blocking x or y will yield an unfeasible configuration as a token will get stuck in p_4 (when blocking x) or p_3 (when blocking y). If both labels are blocked, none of the transitions can fire and thus no final marking can be reached.

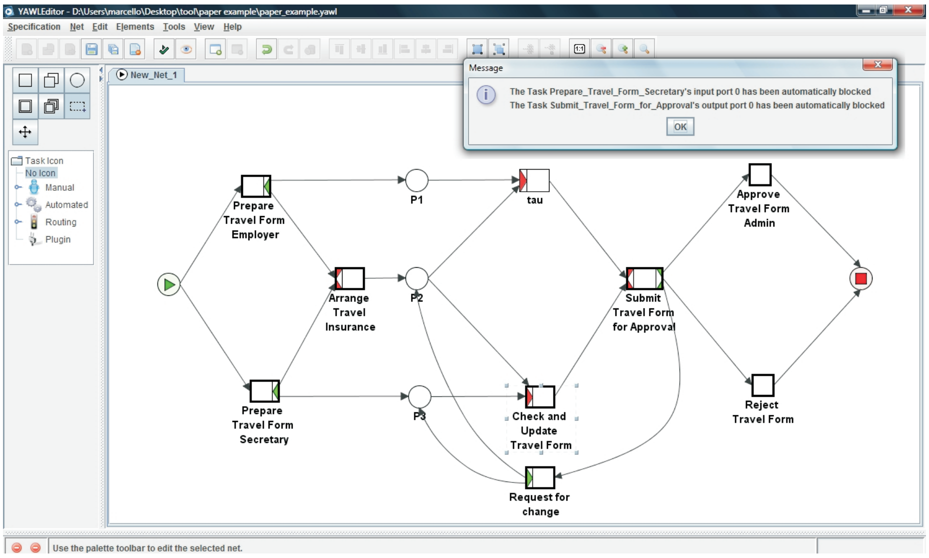
5 Tool Support

To prove the feasibility of our approach, we applied it to the configuration of C-YAWL models [13] and extended the YAWL system accordingly. YAWL is based on the well-know workflow patterns [4] and is one of the most widely used open source workflow systems [15]. For configuration we restrict ourselves to the basic control-flow patterns and do not use YAWL's cancelation sets, multiple instance tasks and OR-joins.

A C-YAWL model is a YAWL model where some tasks are annotated as *configurable*. Configuration is achieved by restricting the routing behavior of configurable tasks via the notion of *ports*. A configurable task's joining behavior is identified by one or more *inflow* ports, whereas its splitting behavior is identified by one or more *outflow* ports. The number of ports for a configurable task depends on the task's routing behavior. For example, an AND-split/join and an OR-join are each identified by a single port, whereas an XOR-split/join is identified by one port for each outgoing/incoming flow. An OR-split is identified by a port for each combination of outgoing flows. To restrict a configurable task's routing behavior, inflow ports can be hidden (thus the corresponding task will be skipped) or blocked (no control will be passed to the corresponding task via that port), whereas outflow ports can only be blocked (the outgoing paths from that task via that port are disabled). For instance, Fig. 7 shows the C-YAWL model for the travel request approval in the YAWL Editor, where configurable tasks are marked with a ticker border.

The new *YAWL Editor* can be downloaded from `www.yawlfoundation.org`. It provides a graphical interface to conveniently configure and check C-YAWL models and subsequently generate configured models. The *C-YAWL Correctness Checker* [5] which is embedded in the editor converts C-YAWL models into open nets and passes these on to the tool *Wendy* [21] to produce configuration guidelines. Wendy implements the algorithms to synthesize partners [24] and calculates operating guidelines [20]. The complexity of the partner synthesis is exponential in the size of the Petri net with the configuration interface (the reachability graph needs to be generated) and the size of the interface. However, practical experiences show that Wendy is able to analyze industrial models with up to 5 million states and to synthesize partners of about the same size [21].

At each configuration step, the Correctness Checker scans the set of outgoing edges of the current state in the configuration guideline, and prevents users from blocking those ports not included in this set. This is done by disabling the block button for those ports. As users block a valid port, the Correctness Checker traverses the configuration guideline through the corresponding edge and updates the current state. If this is not a



YAWLEditor - D:\Users\marcello\Desktop\tool\paper example\paper_example.yawl

Message

The Task Prepare_Travel_Form_Secretary's input port 0 has been automatically blocked
The Task Submit_Travel_Form_for_Approval's output port 0 has been automatically blocked

Use the palette toolbar to edit the selected net.

Fig. 7. The C-YAWL model for travel request approval.

consistent state, that is, a state with an outgoing edge labeled "start", further ports need to be blocked, because the current configuration is unfeasible. In this case the component provides an *"auto complete" option*. This is achieved by traversing the shortest path from the current state to a consistent state and automatically blocking all ports in that path. After this, the component updates the current state and notifies the user with the list of ports that have been automatically blocked. For example, Fig. 7 shows that after blocking the input port of task *Check and Update Travel Form*, the component notifies the user that the input port of task *Prepare Travel Form for Approval (Secretary)* and the output port of task *Submit Travel Form for Approval* to task *Request for Change* have also been blocked. Similarly, the component maintains a consistent state in case users decide to allow a previously blocked port. In this case it traverses the shortest backward path to a consistent state and allows all ports in that path. By traversing the shortest path we ensure that the number of ports being automatically blocked or allowed is minimal.

The C-YAWL example of Fig. 7 comprises ten inflow ports and nine outflow ports. In total more than 30 million configurations are potentially possible. If we abstract from hiding we obtain 524,288 possible configurations, of which only 1,593 are feasible according to the configuration guideline. Wendy took an average of 336 seconds (on a 2.4 GHz processor with 2GB of RAM) to generate this configuration guideline which consumes 3.37 MB of disk space. Nonetheless, the shortest path computation is a simple depth-first search which is linear on the number of nodes in the configuration guideline. Thus, once the configuration guideline has been generated, the component's response time at each user interaction is instantaneous.

6 Conclusion

Configurable process models are a means to compactly represent families of process models. However, the verification of such models is difficult as the number of possible configurations grows exponentially in the number of configurable elements. Due to concurrency and branching structures, configuration decisions may interfere with each other and thus introduce deadlocks, livelocks and other anomalies. The verification of configurable process models is challenging and only few researchers have worked on this. Moreover, existing results impose restrictions on the structure of the configurable process model and fail to provide insights into the complex dependencies among different process model configuration decisions.

The main contribution of this paper is an innovative approach for ensuring correctness during process configuration. Using partner synthesis we compute the configuration guideline — a compact characterization of all feasible configurations, which allows us to rule out configurations that lead to behavioral issues. The approach is highly generic and imposes no constraints on the configurable process models that can be analyzed. Moreover, all computations are done at design time and not at configuration time. Thus, once the configuration guideline has been generated, the response time is instantaneous thus stimulating the practical (re-)use of configurable process models. The approach is implemented in a checker integrated in the YAWL Editor. This checker uses the Wendy tool to ensure correctness while users configure C-YAWL models.

Several interesting extensions are possible. First, the partner synthesis could be further refined using *behavioral constraints* [19] in order to rule out specific partners. This could be used to encode knowledge about a process' application domain [16] in the configuration interface. For example, domain knowledge may state that two activities cannot be blocked or allowed at the same time. Similarly, one could study techniques to identify semantic inconsistencies between control-flow and data-flow that can arise from configuration, and use behavioral constraints to encode these inconsistencies (e.g., extend the approach in [23]). Second, one could consider configuration at run-time, that is, while instances are running, configurations can be set or modified. This can be easily embedded in the current approach. Finally, one could devise more compact representations of configuration guidelines (e.g. exploiting concurrency [6]).

References

1. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
2. 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.
3. W.M.P. van der Aalst, M. Dumas, F. Gottschalk, A.H.M. ter Hofstede, M. La Rosa, and J. Mendling. Preserving Correctness During Business Process Model Configuration. *Formal Aspects of Computing*, 22(3):459–482, 2010.
4. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
5. W.M.P. van der Aalst, N. Lohmann, M. La Rosa, and J. Xu. Correctness Ensuring Process Configuration: An Approach Based on Partner Synthesis (extended version). BPM Center Report BPM-10-02, BPMcenter.org, 2010.

6. E. Badouel and P. Darondeau. Theory of regions. In *Advanced Course on Petri Nets*, LNCS 1491, pages 529–586. Springer, 1996.
7. T. Basten and W.M.P. van der Aalst. Inheritance of Behavior. *Journal of Logic and Algebraic Programming*, 47(2):47–145, 2001.
8. J. Becker, P. Delfmann, and R. Knackstedt. Adaptive Reference Modeling: Integrating Configurative and Generic Adaptation Techniques for Information Models. In *Reference Modeling: Efficient Information Systems Design Through Reuse of Information Models*, pages 27–58. Physica-Verlag, Springer, 2007.
9. T. Curran and G. Keller. *SAP R/3 Business Blueprint: Understanding the Business Process Reference Model*. Upper Saddle River, 1997.
10. K. Czarnecki and M. Antkiewicz. Mapping Features to Models: A Template Approach Based on Superimposed Variants. In *GPCE 2005*, pages 422–437. Springer, 2005.
11. P. Fettke and P. Loos. Classification of Reference Models - A Methodology and its Application. *Information Systems and e-Business Management*, 1(1):35–53, 2003.
12. F. Gottschalk, W.M.P. van der Aalst, and H.M. Jansen-Vullers. Configurable Process Models: A Foundational Approach. In *Reference Modeling: Efficient Information Systems Design Through Reuse of Information Models*, pages 59–78. Physica-Verlag, Springer, 2007.
13. F. Gottschalk, W.M.P. van der Aalst, M.H Jansen-Vullers, and M. La Rosa. Configurable Workflow Models. *Int. J. Cooperative Inf. Syst.*, 17(2):177–221, 2008.
14. A. Hallerbach, T. Bauer, and M. Reichert. Guaranteeing Soundness of Configurable Process Variants in Provop. In *CEC*, pages 98–105. IEEE, 2009.
15. A.H.M. ter Hofstede, W.M.P. van der Aalst, M. Adams, and N. Russell. *Modern Business Process Automation: YAWL and its Support Environment*. Springer, 2010.
16. M. La Rosa, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Questionnaire-based Variability Modeling for System Configuration. *Software and Systems Modeling*, 8(2):251–274, 2009.
17. 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 *ER 2008*, volume 5231 of *LNCS*, pages 199–215. Springer, 2008.
18. M. La Rosa, J. Lux, S. Seidel, M. Dumas, and A.H.M. ter Hofstede. Questionnaire-driven Configuration of Reference Process Models. In *CAiSE'07*, volume 4495 of *LNCS*, pages 424–438. Springer, 2007.
19. N. Lohmann, P. Massuthe, and K. Wolf. Behavioral Constraints for Services. In *BPM 2007*, volume 4546 of *LNCS*, pages 271–287. Springer, 2007.
20. N. Lohmann, P. Massuthe, and K. Wolf. Operating Guidelines for Finite-State Services. In *ICATPN 2007*, volume 4546 of *LNCS*, pages 321–341. Springer, 2007.
21. N. Lohmann and D. Weinberg. Wendy: A tool to synthesize partners for services. In *PETRI NETS 2010*, LNCS. Springer, 2010.
22. M. Rosemann and W.M.P. van der Aalst. A Configurable Reference Modelling Language. *Information Systems*, 32(1):1–23, 2007.
23. N. Trcka, W.M.P. van der Aalst, and N. Sidorova. Data-Flow Anti-Patterns: Discovering Data-Flow Errors in Workflows. In *CAiSE'09*, volume 5565 of *LNCS*, pages 425–439. Springer, 2009.
24. K. Wolf. Does my service have partners? *LNCS T. Petri Nets and Other Models of Concurrency*, 5460(2):152–171, 2009.