Moving Integrated Product Development to Service Clouds in the Global Economy
J. Cha et al. (Eds.)

doi:10.3233/978-1-61499-440-4-317

Extending BPMN for Configurable Process Modeling

Hongyan ZHANG [a,1], Weilun HAN [a], and Chun OUYANG [b]
[a] *School of Software Engineering, Beijing Jiaotong University, Beijing 100044,China*
[b] *Faculty of Science and Technology, Queensland University of Technology, Australia*

Abstract. Configurable process modeling provides a key approach to capture possible process variations into one (reference) model on the one hand and to retrieve individual process variants through configuration of the model on the other hand. BPMN, a standard for business process modeling and a mainstream language being widely adopted in practice, lacks the configurable modeling capability. In this paper, we propose an extension to BPMN to support configurable process modeling with a focus on control-flow perspective. Using configurable workflow net as the theoretical foundation, we formally define the semantics of the proposed extension to BPMN, its correctness-preserving conditions, and its configuration semantics. We name the resulting language Configurable BPMN, i.e. C-BPMN, and provide a running example to illustrate how C-BPMN supports configurable process modeling as well as process configuration.

Keywords. Configurable process modeling, Configuration semantics, configurable reference model, BPMN

Introduction

With the rising of computation abstract level, business process has become both the organizational asset and computational object, which builds the bridge between business system and information system and helps them tightly integrated. In the contemporary of “big program”, the programming technology development is experiencing some innovations focusing its computation on algorithm logic complexity to business logic complexity, and wants to build a Turing machine which can directly understand business logic and process in a visible way. However, the existing languages of business process modeling are normally weak in the capability of multi-scenario description and seldom to support configurable process modeling. Even though some of them support it, the problem to retrieve individual process variants through configuration of a reference model is still not to be solved. These problems above heavily take influences on process reuse in design phase and additionally, stiff process logic brings heavy burdens of maintenance to its corresponding information systems during their execution. The root of the problems is due to lack of techniques to support context-awareness information system and adaptable programming or modeling languages [1, 2]. In these cases, configurable and adaptable process modeling

[1] Corresponding Author.

techniques are needed to enhance the capability of process reuse and the degree of information system flexibility.

Even though reference model technique [3, 4] invented can improve the capability of process reuse in some extent through supporting multi-scenario description, it doesn't change modeling languages themselves. Process modeling languages still can't support to capture possible process variations into one reference model and retrieve the individual process variant which best meets the specific requirements from context. In this case, a reference model is normally larger in size, more complex in logic and lower efficient in execution than its individualized model [5]. So, there is necessary to build a computation environment which supports configurable process modeling. The environment has three fundamental elements: 1) a configurable process modeling language; 2) a modeling tool supporting configurable process modeling; 3) a Turing machine which can understand the configurable process model and customize it into an configured it according to specific requirements. To build this environment, two theoretical problems as followed should be solved.

BPMN, as a standard for business process modeling language and a mainstream language widely adopted by industry, hasn't had its configurable modeling language until now. So, it is meaningful and valuable for the paperwork to solve the theoretical problems which are critical and useful to build a configurable BPMN modeling environment.

In this paper, the related works on the research are, firstly, introduced to help readers better understanding of the paper's work and its results, and then the research approach is put into discussion so that a clear problem-solving path of this technical field will be determined. Based on the formal definition of BPMN and process configuration semantics, configurable BPMN (C-BPMN) will be defined in the aspects of syntax and semantics; Model correctness verification and validation are very important for language creation or improvement; the paper provides the correctness-preserving conditions and constrains for C-BPMN. At last, a running example is made to demonstrate how C-BPMN supports configurable process modeling as well as process configuration.

1. Related Work

There are formal and non-formal two types of process modeling languages. Yawl [6] and Petri net [7] are typical formal languages, UML, EPC and BPMN belongs to non-formal type [8-10]. Compared with other process modeling languages, BPMN is richer of process modeling expression not only in semantics but also in syntax.
Configurable modeling and its individualization mainly used in design phase, which is before model execution [5]. By adding a configuration session into the whole life cycle of process engineering, a reference model can be automatically customized into an individual process according to specific requirements. The configured model is slimmer than its reference model not only in size but also in logic complexity. In the case, the corresponding information system of the configured model will be executed more efficiently than the original reference model.

Nowadays, configurable modeling techniques become a popular field in academic research. The paper [5, 11-14] introduced the configurable extension solutions to EPC and YAWL, which including C-EPC, C-iEPC, ADOM-EPC and C-YAWL, the authors of the paper also proposed a configurable extension to BPMN called C-BPMN [15] on the perspective of control flow.

According to the papers [16, 17], two fundamental approaches to extend process modeling languages can be summarized:

Extending modeling languages through building configurable nodes. C-EPC [5] is a configurable modeling language of EPC, which provides configurable statement through transforming EPC's Function and Connector nodes into configurable ones; Based on the definition of C-EPC, ADOM-EPC[13] builds a configurable extension by adding configurable attributes into Event entity and building configurable event node in a model. C-iEPC [11, 12] is the successive result of iEPC [18], which provides a resource-oriented and object-oriented configurable solution through building configurable Role and Object nodes in EPC models. A configurable extension to BPMN (called C-BPMN) was recently proposed in the paper [15], the solution provided a formal definition and appropriate syntax correctness-preserving conditions for configurable BPMN. However, it regrets that the formal definition is only a syntax-oriented static description without dynamic semantics.

Extending language based on hiding and blocking. C-YAWL [14] is a configurable extension to YAWL, which changes YAWL in this approach by adding the attributes of input and output port into Action entity and setting the port status with hiding and blocking. So, C-YAWL supports to retrieve an individual process variant from its reference model by configuring Action node with three statuses as follows: Normal Execution, Blocking then Execution, Skip then Execution. Even though many mainstream languages have had their configurable modeling solution based on language's extension, it is still blank in the research of process configuration semantics and their individualization computation.

In this paper, we will develop a renewed version of configurable BPMN based on the configuration semantics and provide C-BPMN model with the correctness-preserving conditions as well as individualization algorithm. Finally, a test case with the logic coverage of seven types of configuration patterns will be done to demonstrate the algorithm to support complex process logic.

2. Approach of Research

A programming or modeling language supporting one kind of metadata object is usually developed along the three dimensions: time, space and context. The results from the research along space dimension usually help creating or improving the language's entities and their syntax based upon the insight into its computational object ontology; the research along time dimension mainly focuses on the behavior semantics of metal object, and its results will help enhance language's capability in dynamic semantics description. As to process modeling languages, the change normally happens on their syntax definitions. The research along context dimension often focuses on the communication between language and its context; the results can help building adaptable languages to construct a context-awareness information system. The contents of the research along space and context dimensions are out of the scope of this paper's work, we only focus the paper's work on the extension to BPMN based on configuration semantics along time dimension, that means we need to transform BPMN into C-BPMN through implementation of hiding and blocking -- the fundamental operations of process configuration semantics ,define the correctness-preserving conditions of C-BPMN's syntax and the correctness-preserving constrains of C-BPMN model's execution semantics, and finally provide appropriate individualization algorithm based on the definition of configured C-BPMN model and configuration semantics. The details are explained below:

Step1: Extend BPMN into C-BPMN based on process configuration semantics. The precise definition of process configuration semantics can be achieved through the study of configurable workflow net called C-WF net, hiding and blocking are the fundamental operations of the semantics. Mapping BPMN model into appropriate C-WF net, C-BPMN, the configurable solution to BPMN, can be found through implementation of the two fundamental operations.

Step2: Validate C-BPMN model's syntax correctness in terms of the correctness-preserving conditions of BPMN syntax. Verify C-BPMN model's semantics correctness through verifying that its equivalent induced Petri net is a C-WF net, and the C-WF net's semantics is correct according to its correctness-preserving constrains of execution semantics [19].

Step3: Develop the individualization algorithm for C-BPMN based on process configuration semantics. The syntax correctness validation of its result -- the configured model can be done following the correctness-preserving conditions of C-BPMN. The semantics correctness verification of the configured process is still a problem needed to be solved in the next paper.

Step4: A process with the coverage of seven types of configuration patterns [20] will be chosen as a running example to demonstrate how C-BPMN supports configurable process modeling as well as configuration semantics.

3. Configurable BPMN(C-BPMN)

Even though there already existed a formal definition of C-BPMN in the previous research [5], it was still only a static definition without dynamic semantics. In this section, a renewed formal definition of C-BPMN will be discussed based on hiding and blocking semantics [16] as well as the original formal definition of BPMN [15].

3.1. Syntax of C-BPMN

Before introducing C-BPMN syntax, it is needed to introduce BPMN at first. As a mainstream process modeling language in industry, BPMN provides a set of graphic notations for business process modeling. Business Process Diagram (BPD), a kind of flowchart discussed in graphic theory, provides the formal description for BPMN.

The elements of BPD belong to the subset of BPMN's elements, which only consists of the core elements of BPMN including of Event set with two special instance Start and End, Activity set and Gateway set. Start even is the node to the beginning of process, End event is the node representing the end to a process. Other events are out of the scope that the paper wants to discuss. Activity set has two types of elements: Task and Sub process. A task is an atomic activity and represents a work to be performed within a process. Sub-process is out of the contents that the paper wants to discuss. Gateway is a kind of routing construct used to control the divergence and convergence of sequence flow. There are four main types of gateways, they are parallel fork gateway (AND-split), parallel join gateway (AND-join), data-based XOR decision gateway

(XOR-split), XOR merged gateway (XOR-join). The other types of gateway are out of the paper scope.

A core BPMN process using the core subset of BPMN elements can be completely formalized as a BPD. First we define the syntax of a core BPMN process.

Definition 1 (Core BPMN Process). A core BPMN is a BPMN = (O, T, E, G, C, F) where:

- O is a set of object which can be partitioned into disjoint sets of tasks T, events E, and gateways G, i.e., $O = T \cup E \cup G$,
- $t \in T$ is a finite (non-empty) set of tasks,
- E is a finite (non-empty) set of events, can be partitioned into disjoint sets of Start events E^S, End events E^E and Intermediate event E^I,
- G is a finite set of gateways, can be partitioned into disjoint sets of parallel fork gateways G^F, parallel join gateways G^J, data-based XOR decision gateways G^D, and XOR merge gateways G^M, i.e., $G^F \cap G^J \cap G^D \cap G^M = \Phi$ and $G^F \cup G^J \cup G^D \cup G^M = G$,
- $C \in G \rightarrow \{\wedge, XOR, \vee\}$ is a function which maps each gateway onto a control logic, $C_\wedge = \{g \in G | C = \wedge\}$, $C_{XOR} = \{g \in G | C = XOR\}$, $C_\vee = \{g \in G | C = \vee\}$,
- $C_J = \{g \in G | input(g) \geq 2\}$ is the set of join gateways,
- $C_S = \{g \in G | output(g) \geq 2\}$ is the set of split gateways,
- $G^F = C_S \cap C_\wedge$, $G^J = C_J \cap C_\wedge$, $G^D = C_S \cap C_{XOR}$, $G^M = C_J \cap C_{XOR}$.
- $F \subseteq O \times O$ is the control flow relation.

The paper [15] gave C-BPMN a formal definition with configurable task and configurable gateway through extending the core BPMN entities in configurable ones. Configurable task may be set as ON, OFF and OPT. A configurable gateway can be mapped onto a concrete Choice gateway, which represents the logic construct of split or join considered even can be configured to a sequence.

Definition 2 (Configurable BPMN Process). A configurable BPMN is a BPMN = $(O, T, E, G, E^S, E^E, G^F, G^D, G^J, G^M, F, T^C, G^C, R^C)$ where:

- $O, T, E, G, E^S, E^E, G^F, G^D, G^J, G^M$ and F are specified in Definition 1,
- $T^C \subseteq T \rightarrow \{ON, OFF, OPT\}$ is a set of configurable tasks,
- $G^C \subseteq G \rightarrow \{CT\}$ is a set of configurable gateways, $CT = \{\wedge, XOR, \vee\} \cup CTS$, $CTS = \{ SEQ_n | n \in T \cup E \cup G \}$,
- R^C is a configuration requirements.

3.2. Semantics of C-BPMN

The operations of hiding and blocking in C-WF net represent the semantics of process configuration, which can be used as a foundational framework to extend BPMN into C-BPMN.

Figure1 shows a relationship between C-BPMN and C-WF nets. The first column represents the type of configuration operations; the 2nd column contains a C-BPMN model with a configurable node represented by a double-line rectangle. The model for the 3rd column is the configured result corresponding to the model in the 2nd column; In the 4th column, a semantics-equivalent C-WF net with the C-BPMN model in the 2nd column is displayed, the model in the 5th column is the configured result of the C-

WF net. The Configured WF net is semantics-equivalent with the configured BPMN; it is the semantic net of the corresponding configurable BPMN.

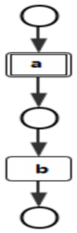
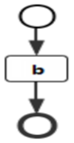
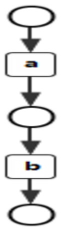
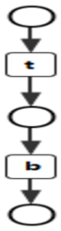
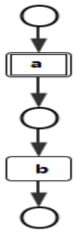
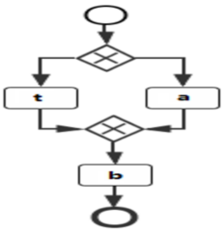
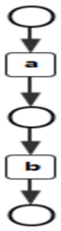
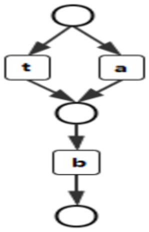
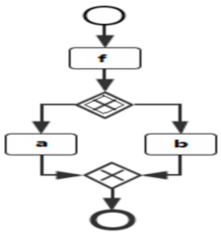
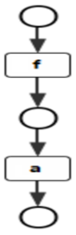
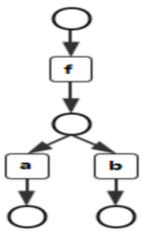
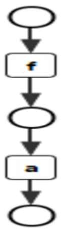
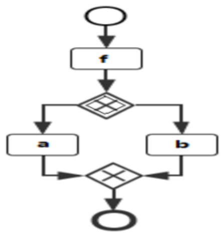
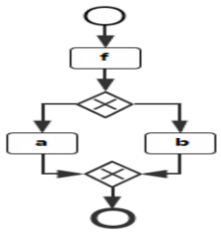
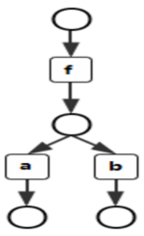
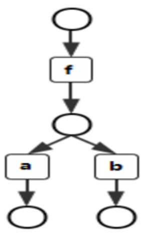
	C-BPMN	**Configured BPMN**	**Configurable WF-nets**	**Configured WF-nets**
Configurable Task is OFF				
Configurable Task is OPT				
Configurable gateway restricted to sequence				
Configurable gateway remained before				

Figure 1. Relationship between C-BPMN and C-WF nets

The first row in Figure1 depicted configurable task mapping to C-WF nets. The task within the C-BPMN process fragment is switched "OFF". This confirms to a hidden transition within the corresponding C-WF nets.

The second row also depicted configurable task mapping to C-WF nets. The task within the C-BPMN process fragment is switched "OPT". This means an option hidden transition corresponding C-WF nets.

The third row in Figure 1 depicted configurable gateway mapping to C-WF nets. Configurable XOR-split gateway restricted to a sequence, this means a blocked transition corresponding C-WF nets.

The fourth row also depicted configurable gateway mapping to C-WF nets. Configurable XOR-split can be configured to an XOR gateway. In fact, this confirms to an option blocked transition within the corresponding C-WF nets.

4. Correctness-Preserving of C-BPMN

C-BPMN, as a new language, is different from its original language BPMN, the correctness validation and verification in syntax and execution semantics are both needed to it. In this section, the correctness-preserving of C-BPMN will be discussed with respects to the conditions for its syntax validation, and the constrains for its semantics verification.

4.1. Syntactical Correctness

The paper [21, 22] described the correctness-preserving conditions of BPMN syntax as follow:

Definition 3 (Well-formed BPMN Process). A core BPMN Process given in Definition 1 is well formed if relation F satisfies the following requirements:

- $\forall$ s $\in E^S$, input(s) = Φ $\wedge$|output(s)| = 1, i.e. start event have an indegree of zero and an outdegree of one.
- $\forall$ e $\in E^E$, output(s) = Φ $\wedge$|input(s)| = 1, i.e. end event have an indegree of zero and an outdegree of one.
- $\forall$ g $\in G^F \cup G^D$: |input(g) | = 1 $\wedge$|output(g)|> 1, i.e. fork or decision gateways have an indegree of one and an outdegree of more than one,
- $\forall$ g $\in G^J \cup G^M$: |output(g)| = 1 $\wedge$|input(g)|> 1, i.e. join or merge gateways have an indegree of one and an outdegree of more than one,
- $\forall$ x $\in$ O, $\exists$ (s, e) $\in E^S \times E^E$, sF*x$\wedge$xF*e, i.e. every object is on a path from a start event to an end event.

Since C-BPMN only brings BPMN changes in entity attributes but not entities, So, C-BPMN doesn't change the rules of BPMN syntax, and its model's semantics correctness validation can be taken straight following the conditions of BPMN correctness-preserving. However, for the semantics of process configuration has been added into BPMN while C-BPMN model has no any behavior semantics, it's impossible for C-BPMN to verify its semantics correctness straight on C-BPMN model itself.

4.2. Behavioral Correctness

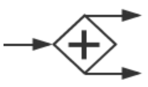
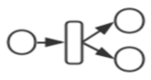
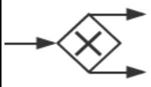
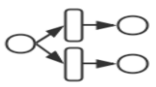
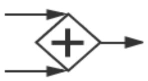
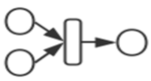
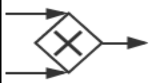
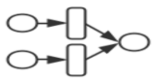
C-BPMN is ambiguous without behavior semantics; it is impossible to check the model for consistency and completeness semantics. WF-nets have formal semantics. It is sufficient to map BPMN onto WF-nets to specify the behavior unambiguously. The correctness-preserving results described in C-WF nets can be exploited to achieve the C-BPMN models, where each configuration step is soundness-preserving.

Induced Petri net is a semantics-equivalent net of C-BPMN model, which can be obtained through mapping C-BPMN onto Petri net. The verification of C-BPMN model's semantics correctness is equal to the correctness verification of its induced

Petri net. For there already existed the result about the correctness-preserving of C-WF net [19], So, the semantics correctness verification of C-BPMN model can be realized by means of verifying that its induced net is a configurable workflow net.

Table 1 shows the basic strategy that is used to map BPMN onto WF-nets: Start event or End event correspond to a similar module with a place and a silent transition. A task corresponds to a transition named by the task with one input place and one output place. The translation of gateways is more complex than event and task. Gateways are mapped onto small Petri nets modules with silent transitions capturing their routing behavior. For example, the behavior of Parallel Fork (AND-Split) mapped onto a silent transition with one input place and more than one output place.

Table 1. Mapping BPMN objects to WF-nets module

	BPMN Object	WF-nets Module		BPMN Object	WF-nets Module
Parallel Fork (AND-Split)			Data-based XOR (XOR-Split)		
Parallel Join (AND-Join)			XOR Merge (XOR-Join)		

The paper[19] provided correctness-preserving semantics of C-WF nets. To construct a WF-net from a C-BPMN model, we simply use the mapping from BPMN object to WF-nets module.

We can formally define the induced Petri net as follows:

Definition 4 (Induced Petri net). Let C-BPMN = (O, T, E, G, E^S, E^E, F, T^C, G^C, R^C) be a syntactically correct C-BPMN, PN(C-BPMN) = (P^{PN}, T^{PN}, F^{PN}) is the Petri net induced by C-BPMN such that:

- $P^{PN} = E \cup \cup_{c\in C} P_C^{PN}$.
- $T^{PN} = T \cup \cup_{c\in C} T_C^{PN}$.
- $F^{PN} = (F \cap (E\times T)) \cup \cup_{c\in C} F_C^{PN}$.

where P_C^{PN}, T_C^{PN}, F_C^{PN} are defined as Table 2.

Table 2. Configuration patterns in the C-BPMN

	P_C^{PN}	T_C^{PN}	F_C^{PN}
$g \in C_J \cap C_{AND}$	$\{p_x^c \mid x\in input(g)\}$	$\{t^c\}$	$\{(x, p_x^c) \mid x\in in(g) \cup\{(p_x^c, t_x^c) \mid x\in in(g)\} \cup\{(t_x^c,x) \mid x\in in(g)\}$
$g \in C_S \cap C_{AND}$	$\{p^c\}$	$\{t_x^c \mid x\in input(g)\}$	$\{(x, t_x^c) \mid x\in in(g) \cup\{(t_x^c, p_x^c) \mid x\in in(g)\} \cup\{(p_x^c,x) \mid x\in in(g)\}$
$g \in C_J \cap C_{XOR}$	$\{p_x^c \mid x\in output(g)\}$	$\{t^c\}$	$\{(x, t_x^c) \mid x\in out(g) \cup\{(t_x^c, p_x^c) \mid x\in out(g)\} \cup\{(p_x^c,x) \mid x\in out(g)\}$
$g \in C_S \cap C_{XOR}$	$\{p^c\}$	$\{t_x^c \mid x\in input(g)\}$	$\{(x, p_x^c) \mid x\in out(g) \cup\{(p_x^c, t_x^c) \mid x\in out(g)\} \cup\{(t_x^c,x) \mid x\in out (g)\}$

Lemma 1. Let C-BPMN = (O, T, E, G, E^S, E^E, F, T^C, G^C, R^C) be a syntactically correct C-BPMN and PN(C-BPMN) be its induced Petri net, then: PN(C-BPMN) is a WF-nets.

Proof.

Follows directly from the construction of PN (C-BPMN).

Figure 2 depicts the method how to transit a BPMN model into an Induced Petri net. Induced Petri net specifies the execution semantics of C-BPMN model. This allows us to identify those C-BPMNs which can be correctly executed.

Definition 5 (Sound C-BPMN). Let C-BPMN = (O, T, E, G, E^S, E^E, F, T^C, G^C, R^C) be a syntactically correct C-BPMN and PN(C-BPMN) be its induced Petri net. C-BPMN is sound iff PN(C-BPMN) is sound.

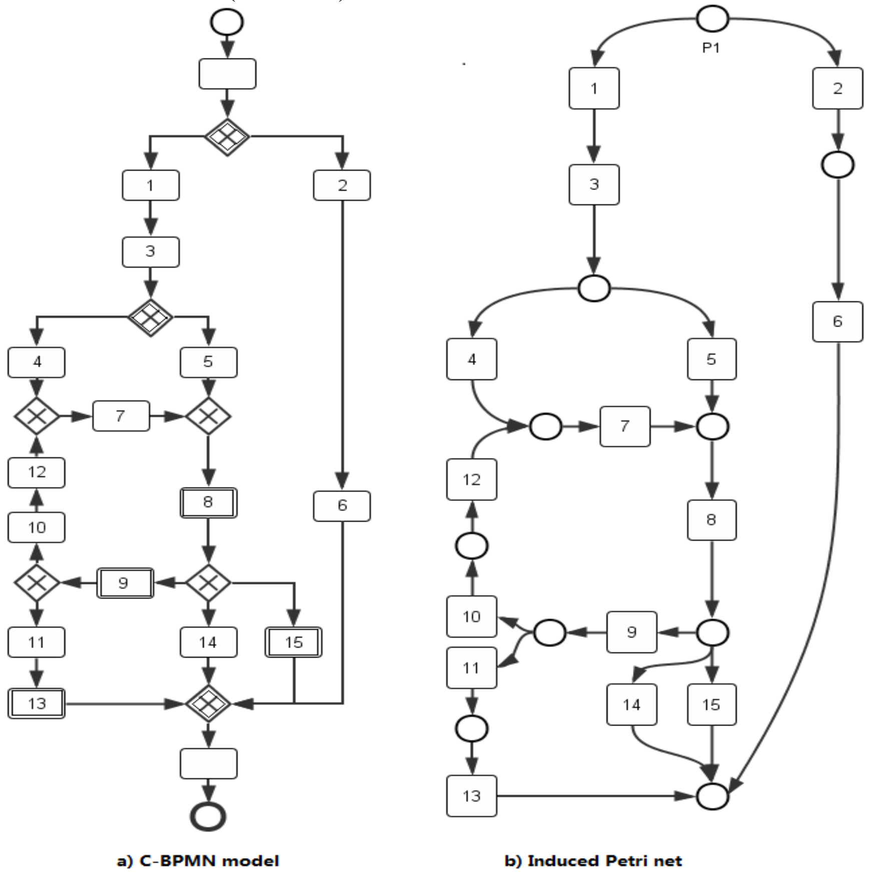


Figure 2. C-BPMN model and its Induced Petri net

5. Semantics of C-BPMN Process configuration

In this section we can now discuss how a C-BPMN model corresponds to a concrete model and how process correctors can be preserved during the configuration of a C-BPMN.

The paper [5] defined configuration BPMN as follow:

Definition 6 (Configuration of C-BPMN Process). Let C-BPMN = (O, T, E, G, E^S, E^E, G^F, G^D, G^J, G^M, F, T^C, G^C, R^C) be a C-BPMN. $C^C \in (T^C \rightarrow \{ON, OFF, OPT\}) \cup \{G^C \rightarrow CT\}$ is a configuration of C-BPMN if for each $g \in G^C$:

- $C^C(g) \leq^C C(g)$, $\leq^C = \{(n, n) | n \in CT\} \cup \{(XOR, \vee), (\wedge, \vee)\} \cup \{(n_1, n_2) | n_1 \in CTS \wedge n_2 \in \{XOR, \vee\}\} \cup \{(n,n) | n \in CTS\}$
- If $C^C(g) \in CTS$ and $g \in C_J$, then there exist an $n \in$ input(g) such that $C^C(g) = SEQ_n$
- If $C^C(g) \in CTS$ and $g \in C_S$, then there exist an $n \in$ output(g) such that $C^C(g) = SEQ_n$

The paper [5] also depicts some examples of configuration and valid configuration. The former section, in addition, gives the execution semantics of C-BPMN. Now we provide an algorithm to construct a concrete BPMN model based on a C-BPMN. Note that a C-BPMN defines a number of concrete BPMN models, and each valid configuration correspond a C-BPMN to a concrete BPMN. The function β maps a C-BPMN and its configuration onto a concrete BPMN β(C-BPMN, l^C_{BPMN}).

Definition 7 (Semantics of Configurations). Let C-BPMN= (O, T, E, G, E^S, E^E, F, T^C, G^C, R^C) be a C-BPMN, and l^C_{BPMN} a configuration of C-BPMN. The corresponding BPMN β(C-BPMN, l^C_{BPMN}) is constructed as follows:

- $BPMN_1 = (O, T, E, G, C_1, E^S, E^E, F_1)$ with $C_1 = \{(g, l(g)) | g \in G | G^C\} \cup \{(g, l^c(g)) | g \in G\}$ and $F_1 = F | (\{(g, n) \in C_S \times out(g) | \exists_{n' \in g\bullet} C^c(c) = SEQ_{n'} \wedge n \neq n'\} \cup \{(n, g) \in in(g) \times C_J | \exists_{n' \in \bullet g} C^c(c) = SEQ_{n'} \wedge n \neq n'\}$.
- $BPMN_2 = (O, T_2, E, G_2, C_2, E^S, E^E, F_2)$, with $T_2 = T | \{t = OFF\}$ and $F_2 = \{(n_1, n_2) \in F | \{n_1, n_2\} \cap (\bullet t \cup t \bullet) = \Phi\}$.
- $BPMN_3 = (O, T_3, E, G_3, C_3, E^S, E^E, F_3)$, for each t = OPT, with $T_3 = T_2 \cup \{skip_t\}$, $G_3 = G \cup \{split_t, join_t\}$, $C_3 = C_1 \cup \{(split_t, XOR), (join_t, XOR)\}$, $F_3 = \{(n_1, n_2) \in F_2 | f \notin \{n_1, n_2\}\} \cup \{(split_t, f), (split_t, split_t), (split_t, join_t), (f, join_t)\} \cup \{(n, split_t) | (n, f) \in F_2\} \cup \{(join_t, n) | (f, n) \in F_2\}$.
- Remove all gateways with just one input and output arcs, $BPMN_4 = (O, T_3, E, G_4, C_3, E^S, E^E, F_3)$, with $G_4 = G_3 | \{g \in in(g) = 1 \cap out(g) = 1\}$
- Re-apply Step 2 of the algorithm, i.e., try to remove the remaining functions labeled "$skip_t$", $BPMN_5 = (O, T_4, E, G_4, C_3, E^S, E^E, F_3)$, with $T_4 = T_3 | \{t = skip_t\}$
- Remove all nodes not on some path from a start event to a final event,
- Re-apply Step 4 of the algorithm, i.e., remove connector, $BPMN_4 = (O, T_4, E, G_5, C_3, E^S, E^E, F_3)$, with $G_5 = G_4 | \{g \in in(g) = 1 \cap out(g) = 1\}$

The following Theorem shows that the resulting β(C-BPMN, l^C_{BPMN}) is syntactically correct provided the initial C-BPMN is syntactically correct:

Theorem 1 (β(C-BPMN, l^C_{BPMN}) is an BPMN). Let C-BPMN = (O, T, E, G, E^S, E^E, F, T^C, G^C, R^C) be a C-BPMN, and l^C_{BPMN} a configuration of C-BPMN. β(C-BPMN, l^C_{BPMN}) is an BPMN satisfying all requirements stated in **Definition 7**

Proof. BPMN = (O, T, E, G, C, E^S, E^E, F) satisfies all requirements by definition.

- The sets E, T, G are disjoint Although not always stated explicitly we assume no name clashes,
- There is at least one event $e \in E^S$, such that input(e) = 0. Start event are not removed,
- There is at least one event $e \in E^E$, such that output(e) = 0. End event are not removed,

- For each $t \in T$: input(t) = 1 and output(t) = 1,
- For each $g \in G$: input(g) ≥ 1 and output(g) ≥ 1. Existing gateways and newly added connectors satisfy this requirement.

In the former section, we discussed that a C-BPMN configuration can be represented by using the hiding and blocking operations that defined in C-WF nets[19]. Therefore, we can construct a C-BPMN configuration onto the Induced WF-nets. For example, if a configuration task in a C-BPMN switched OPT, this indicated that the transition in the WF-nets is option hide. If a configurable XOR gateways is restricted, the transition in the WF-nets is blocked.

Definition 8 (Induced WF-net Configuration). Let C-BPMN = (O, T, E, G, E^S, E^E, F, T^C, G^C, R^C) be a syntactically correct C-BPMN, C_{PN} be one of its configuration and PN(C-BPMN) be its Induced Petri net. $C_{PN}^{C_{C-BPMN}} \in T^{PN} \rightarrow$ {allow, hide, block, option hide, option block} is the configuration of PN(C-BPMN) induced by C_{PN}.

If we start from a C-BPMN that has been checked for soundness, and we apply a configuration step, we can check the correctness of the resulting C-BPMN by reasoning on the Induced WF-net before and after the configuration.

Proposition 1(Soundness-preserving C-BPMN configuration). Let C-BPMN be a sound C-BPMN, C_{C-BPMN} be one of its configuration, PN(C-BPMN) be the Induced WF-net. Let also $C_{PN}^{C_{C-BPMN}}$ be the configuration of PN(C-BPMN) induced by C_{C-BPMN} and β_{PN}^{*}(PN(C-BPMN), $C_{PN}^{C_{C-BPMN}}$) be the configured net which all the nodes not on a directed path from the input to the output place. If PN(β(C-BPMN, l_{BPMN}^{C})) is equal to β_{PN}^{*}(PN(C-BPMN), $C_{PN}^{C_{C-BPMN}}$) then β(C-BPMN, l_{BPMN}^{C}) is sound.

Proof.

It is obviously that: 1) C-BPMN is sound. Hence its configured BPMN β(C-BPMN, l_{BPMN}^{C}) is syntactically correct(Theorem 1) and PN(C-BPMN) is sound(Definition 5). 2) Since β(C-BPMN, l_{BPMN}^{C}) is syntactically correct, its Induced Petri net PN(β(C-BPMN, l_{BPMN}^{C})) is a WF-net. Thus β_{PN}^{*}(PN(C-BPMN), $C_{PN}^{C_{C-BPMN}}$) is sound, since it is the configured WF-net of PN(C-BPMN) which is sound. If PN(β(C-BPMN, l_{BPMN}^{C})) is equal to β_{PN}^{*}(PN(C-BPMN), $C_{PN}^{C_{C-BPMN}}$) then PN(β(C-BPMN, l_{BPMN}^{C})) is sound. Hence β(C-BPMN, l_{BPMN}^{C}) is sound.

6. Case Study

To show the configuration steps of configuration semantics, a C-BPMN model is introduced and analyzed. Table 3 shows nine configuration patterns [20] and seven patterns in the extension of BPMN. a configurable task of C-BPMN corresponds to a Optionality pattern, and the task can be configured as ON, OFF and OPT. configurable gateway in C-BPMN corresponds six patterns, including split gateway and join gateway.

Figure 3 depicts a configurable model for configuration. The configuration aspects are denoted by double-line border, and we ignore the meaning of each element within the model. In examples, we have three components of configurable nodes: configurable task B, configurable gateway XOR and two configurable gateways XOR or AND connected with arcs, i.e. task A, E and F are configurable tasks and gateway XOR_1,

XOR_2 and AND_3 are configurable gateways. We also ignore the configuration requirements (R^C) in the model. The configurable nodes in the figure may have:

- The configurable task A has been switched OPT,
- The configurable task E remained ON,
- The configurable task F has been switched OFF,
- The configurable XOR_1 gateway has been configured to XOR,
- The configurable XOR_2 gateway has been configured to SEQ_E,
- The configurable AND_3 gateway has been configured to AND.

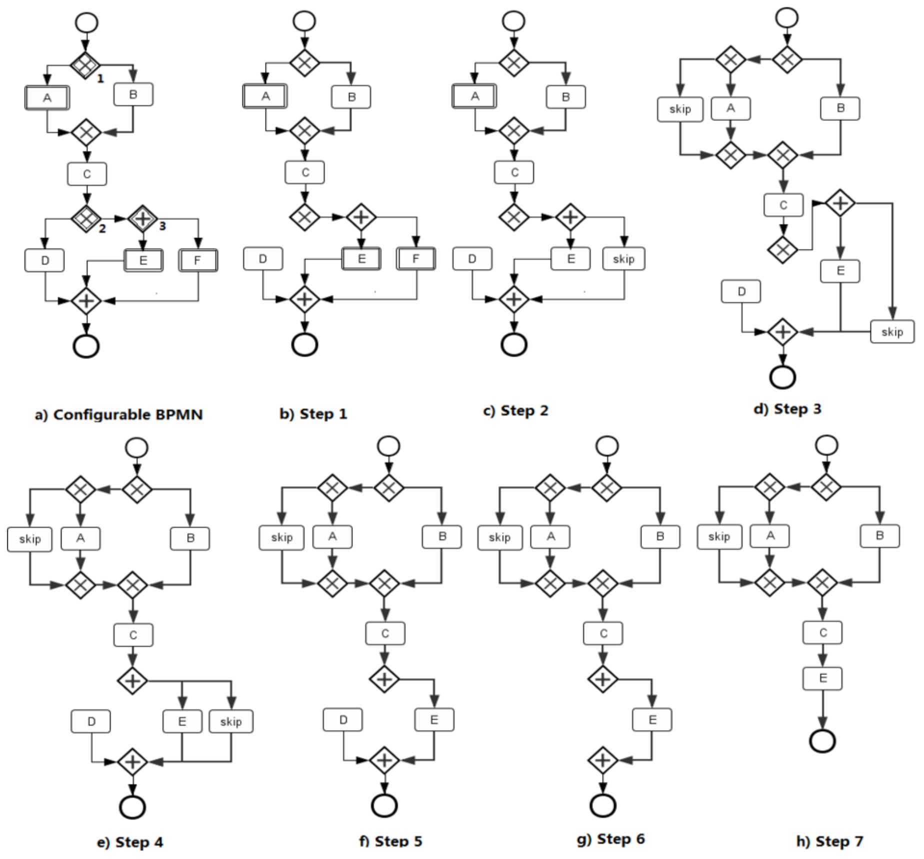


Figure 3. Configuration semantics of a C-BPMN model

Table 3. Configuration patterns in the C-BPMN

Configurable Task	**Optionality**	√
Configurable Gateway(Split)	Parallel Split	√
	Exclusive Choice	√
	Multi Choice	√
Configurable Gateway(Join)	Synchronization	√
	Simple Merge	√
	Synchronizing Merge	√
Others	Interleaved Parallel Routing	
	Sequence Inter-relationships	

Now, we should follow the configuration semantics. Step 1, we choice all configurable gateways correspond to a concrete gateway, i.e. XOR_1 and AND_3 remained before and XOR_2 have a decision on SEQ_E, so we delete the arcs form XOR_2 to SEQ_D. Then, we remain task E and replace task F by skip in Step 2. Next, we correspond task A. In Step 3, we add an XOR-split gateway, a skip task and a XOR-join gateway. Then we remove XOR_2 in Step 4 because of the gateway only have one input and one output. We delete task skip which come from task F in Step 5. Finally we remove all nodes not on some path from a start event to a final event in Step 6, and reapply Step 4 to prevent one input and one output gateways.

7. Conclusion and outlook

The paper provides a whole solution of the configurable extension to BPMN, which focuses on the extension of its core entities and provides them executable semantics. Moreover , the correctness-preserving of C-BPMN is discussed according to its formal definition , the correctness preserving conditions of C-BPMN for syntax validation and the correctness-preserving constrains of C-BPMN model for semantics verification are separately proposed. Additionally, the paper introduces an individualization algorithm of C-BPMN based on process configuration semantics and provides an efficient and effective method to automatically customize C-BPMN model with accordance to specific requirements. At last, a running example with logic coverage of seven configuration patterns is taken to demonstrate how well the algorithm supports complex C-BPMN models and process configuration semantics.
Based on the results above, there still some problems needed to be solved in the future:

1. Developing the other solutions of the configurable extension to BPMN not only focusing on the core entities of BPMN but more on the other entities of control flow, for example, middle event, complex gateways etc.
2. Up to now, there doesn't exists a scientific approach to identify which entities are suitable to be set configurable and the method to implement the configurable modeling technique.
3. Building C-BPMN modeling environment based on the existing results above.

References

[1] H. Klaus, M. Rosemann, G.G. Gable, What is ERP? , *Information systems frontiers*, 2(2): (2000) 141-162.

[2] M. Rosemann, ERP software: characteristics and consequences, *7th European Conference on Information Systems,* 1999.

[3] P. Fettke, P.Loos, Classification of reference models: a methodology and its application, *Information Systems and e-Business Management*, 1(1) (2003) 35-53.

[4] M. Rosemann, Application Reference Models and Building Blocks for Management and Control, In Bernus et al. (eds.), *Handbook on Enterprise Architecture*. Springer Berlin Heidelberg, 2003: 595-615.

[5] M. Rosemann, W. M. P. van der Aalst, A configurable reference modelling language, *Information Systems*, 32(1) (2007) 1-23.

[6] W. M. P. van der Aalst, A. H. M. Ter Hofstede YAWL, Yet another workflow language, *Information systems*, 30(4) (2005) 245-275.

[7] T. Murata, Petri nets: Properties, analysis and applications, *Proceedings of the IEEE*, 77(4) (1989) 541-580.

[8] G. Engels, A. Förster, R. Heckel et al., Process modeling using UML, *Process-Aware Information Systems,* (2005) 85-117.

[9] Object Management Group. *Business Process Modeling Notation (BPMN) Version2.0. OMG Final Adopted Specification*. Object Management Group, 2011.

[10] G. Keller, A.W. Scheer, M. Nüttgens, *Semantische Prozeßmodellierung auf der Grundlage Ereignisgesteuerter Prozeßketten (EPK)*, Inst. für Wirtschaftsinformatik, 1992.

[11] M. La Rosa, M. Dumas, A.H.M. ter Hofstede et al., *Beyond control-flow: Extending business process configuration to resources and objects*, Queensland University of Technology , 2007.

[12] M. La Rosa, M. Dumas, A.H.M. ter Hofstede et al., Configurable multi-perspective business process models, *Information Systems*, 36(2) (2011) 313-340.

[13] I. Reinhartz-Berger, P. Soffer, A. Sturm, Extending the adaptability of reference models, *Systems, Man and Cybernetics, Part A: Systems and Humans*, IEEE Transactions on, , 40(5) (2010) 1045-1056.

[14] F. Gottschalk, W.M.P. van der Aalst, M.H. Jansen-Vullers et al., Configurable workflow models, *International Journal of Cooperative Information Systems,* 17(02) (2008) 177-221.

[15] HAN Wei-lun, ZHANG Hong-yan., Configurable Process Modeling Techniques for BPMN, *Computer Integrated Manufacturing System,* 19(8) (2013) 1928-1934.

[16] F. Gottschalk, W.M.P. van der Aalst, M.H. Jansen-Vullers, *Configurable process models — a foundational approach* , Reference Modeling. Physica-Verlag HD, 2007: 59-77.

[17] M. La Rosa, M. Dumas, A.H.M. ter Hofstede, Modelling business process variability for Design-Time Configuration. In J. Cardoso, W.M.P. van der Aalst (editors), *Handbook of Research on Business Process Modeling*, IDEA Group – Information Science Reference, 2009.

[18] A.W. Scheer, *ARIS - Business Process Frameworks*, Springer, Berlin, 3rd edition, 1999.

[19] W.M.P van der Aalst, M. Dumas, F. Gottschalk et al., Preserving correctness during business process model configuration, *Formal Aspects of Computing,* 22(3-4) (2010) 459-482.

[20] A. Dreiling, M. Rosemann, W.M.P. van der Aalst et al., Model-driven process configuration of enterprise systems*, Wirtschaftsinformatik* 2005, Physica-Verlag HD, (2005) 687-706.

[21] C. Ouyang, W.M.P. van der Aalst, M. Dumas et al., *From business process models to process-oriented software systems: The BPMN to BPEL way,* http://bpmcenter.org/wp-content/uploads/reports/2006/BPM-06-27.pdf, 2006.

[22] R.M. Dijkman, M. Dumas, C. Ouyang, *Formal semantics and analysis of BPMN process models,* Technical Report Preprint 7115, Queensland University of Technology, 2007. https://eprints.qut.edu.au/archive/00007115.
.