Configurable Process Models – A Foundational Approach

F. Gottschalk, W.M.P. van der Aalst, and M.H. Jansen-Vullers

Department of Technology Management, Eindhoven University of Technology,
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.
{f.gottschalk|w.m.p.v.d.aalst|m.h.jansen-vullers}@tm.tue.nl

Abstract. Off-the-shelf packages such as SAP need to be configured to suit the requirements of an organization. Reference models support the configuration of these systems. Existing reference models use rather traditional languages. For example, the SAP reference model uses Event-driven Process Chains (EPCs). Unfortunately, the choice construct within traditional process modelling languages like EPCs do not capture different scopes or impacts of decisions. That means they offer no opportunities to distinguish between decisions made for a single case (i.e. process instance) when executing the process and decisions made in advance for numerous cases impacting bigger parts of the company.
This paper discusses the need for *configurable process models*. An analysis of configuration from a theoretical perspective provides a solid fundament for such models. Within the analysis a link is made to inheritance of dynamic behavior and previously defined inheritance concepts. By applying these concepts to process models the essence of configuration is determined, which enables the development of more mature configurable process modelling languages.

1 Introduction

Reference models streamline the design of particular models by providing a generic solution [25]. Motivated by the "Design by Reuse" paradigm they provide a repository of potentially relevant models which can be used to accelerate the modelling process. Ideally these models are "plug and play" but usually need some adjustment to individual requirements [9,13,7,3]. Hereby it is required to distinguish between *generating* and *non-generating adaptations*. Non-generating adaptations as aggregation, instantiation, specialization, and analogy are providing basic models with certain gaps which have to be filled in by the reference model user. That means, the individual part of the model is generated by the user and not by guidelines of the reference model. The reference model only provides interfaces. A generating approach on the other hand provides clear rules how the reference model can be configured and therefore adapted to the user's requirements [7,29,30,10]. Unfortunately, the languages used for reference modelling [8,11,24] provide little or no support to include such different configuration options. The goal of this paper is to discuss the theoretical requirements

for *configurable process modelling languages*, i.e., we restrict ourselves to the control-flow perspective [15].

Probably the most comprehensive reference model is the SAP reference model [11]. Its data model includes more than 4000 entity types and the reference process models cover more than 1000 business processes and inter-organizational business scenarios [25]. Most of the other dominant ERP vendors have similar or alternative approaches towards reference models. Foundational conceptual work for the SAP reference model has been conducted by SAP AG and the Institute for Information Systems (IWi) of the Saarland University in a collaborative research project in the years 1990-1992 [16]. The outcome of this project was the process modeling language Event-Driven Process Chains (EPCs) [16,17], which has been used for the design of the reference process models in SAP. EPCs also became the core modeling language in the Architecture of Integrated Information Systems (ARIS) [27,28]. It is now one of the most popular reference modeling languages and has also been used for the design of many SAP-independent reference models (e.g., the ARIS-based reference model for Siebel CRM or industry models for banking, retail, insurance, telecommunication, etc.). *Despite its success, the basic EPC model offers little support for process configuration.* It contains (X)OR connectors but it is unclear whether the corresponding decisions need to be taken at run-time (e.g., based on the stock-level), at build-time (e.g., based on the size of the organization using SAP), or somewhere in-between (e.g., based on the period of the year or resource availability). For that reason so-called *Configurable EPCs* (C-EPCs) were developed [25,12], extending EPCs (and previously developed extensions like build-time operators [29,26,23]), aiming at a generic-monolithic approach for constructing re-usable models [13]. Indeed C-EPCs allow for a clear distinction between run-time and build-time decisions. However, they only provide a partial solution as they are based on a specific language (i.e. EPCs). Within this paper we will look at configuration from an language-independent perspective. Afterwards we will use the results to analyse C-EPCs [25].

The remainder of the paper is organized as follows. First, we elaborate on the concept of "choice" which is essential for configurable process models. Second, we approach the problem from a theoretical viewpoint, i.e., we depict what the essence of configuration is. Finally, we briefly discuss Configurable EPCs as a first step towards such configurable process models.

2 It is all about making choices

There are many languages to model processes ranging from formal (e.g., Petri nets and process algebras such as Pi calculus) to informal (flow charts, UML activity diagrams, EPCs, etc.). Each of these languages provides some *notion of choice* (e.g., two transitions sharing a single input place in a Petri net, the "+"-operator in process algebra, the $\diamondsuit$-symbol in UML activity diagrams, or an (X)OR-split connector in an EPC). Typically, it is not possible to describe the nature of such a choice. At best one can either specify a Boolean condition based

on some data element (data-based decision) or one can specify events that have to occur for triggering paths (event-based decision) [21]. The usual interpretation is that a choice is made at run-time, based on such a Boolean condition or based on occurring events. *In the context of reference models, this interpretation is too narrow.*

The *scope* of a decision can vary. For example, if a hospital uses a rule like "If a patient has high blood pressure a day before the planned operation, the operation will be cancelled", then the scope of each choice (operate or not) is limited to a single patient. There may also be choices which affect more cases, e.g., consider the rule "If there is a major disaster in the region, all planned operations will be cancelled." or also an entire process, e.g., "The admittance process requires patients to pre-register.". There may even be choices that affect all processes within an organization. We call such choices that are made in advance and that are affecting more than a single instance of a process *configuration choices*. However note that the borderline between run-time choices and configuration choices may be a bit fuzzy as the following examples show.

- The organization's management chooses not to allow for pre-shipments.
- The Dutch branches require a deposit, while this is not needed for branches in other countries (nation-wide management decision).
- If stock is below 100 items, only preferred customers are serviced (local management decision).
- Based on the volume of the order, the goods are shipped by truck or mail (local management decision).
- On Saturday, goods are shipped by truck (local, temporal decision).

Each of these choices is at another level, i.e. they are made at other points in time with different validity limits and periods. However, classical process modeling languages, e.g., the languages used in workflow management systems [4,15] or in reference modeling [11], allow only for one level of choice. The examples demonstrate that reference models have to allow for a broader spectrum of choices. All decisions have in common that they restrict the actual available options for decisions at a later point in time. For that reason one can view configuration as *limiting choices by making choices*. However, at a certain point in time it is not longer possible to postpone a decision without delaying the actual process flow. These decisions are called run-time decision and must be distinguished from build-time or configuration decisions which can be postponed to a later point in time without delaying the process flow. Seen from this viewpoint, process modeling languages need to distinguish at least between run-time choices and configuration choices.

3 Configuration: A theoretical perspective

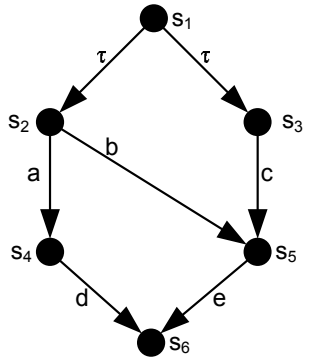
The aim of configurable process models is to provide generic models integrating possible process variations into one model. Afterwards such a model can be configured to a specific solution. This means a configurable model should guide

the user to a solution that fits to the user's requirements [7]. In [13] this is also classified as a generic-monolithic approach for model re-use. In order to provide such configuration opportunities *a configurable model must be able to provide a complete, integrated set of all possible process configurations.* Only in this case each individual model can be derived from the model. In other words the configurable process model can be described as the "least common multiple" of all process variations. The task of configuration is to create a new model by selecting that parts of the configurable model that are relevant to the user or – the other way around – by deselecting the irrelevant parts. In practice such a configured process model can probably not satisfy all individual requirements as the reference model will not include the complete set of all possible configurations. The gap has to be filled in manually by the user by applying non-generating adaptation mechanisms [7]. However this subsequent step is out of the scope of this paper.

To depict and analyse process models we will use the notion of *Labeled Transition Systems (LTS).*

Definition 1. *A labeled transition system is a five-tuple* $LTS = (S, L, T, S_I, S_F)$*, where*

- S *is the set of* states*,*
- L *is the set of* transition labels*,*
- $\tau \in L$ *is the label reserved for* silent transitions*,*
- $T \subseteq S \times L \times S$ *is the set of* transitions*,*
- $S_I \subseteq S$ *is the set of initial states, and*
- $S_F \subseteq S$ *is the set of final states.*



$$S = \{S_1, S_2, S_3, S_4, S_5, S_6\}$$
$$L = \{\tau, a, b, c, d, e\}$$
$$T = \{(s_1, \tau, s_2), (s_1, \tau, s_3), ..., (s_5, e, s_6)\}$$
$$S_I = \{S_1\}$$
$$S_F = \{S_6\}$$

Fig. 1. A labeled transition system

A state represents a complete set of properties, describing the actual situation within the process. A labeled transition describes the switching from one state to another. Therefore transitions are also representing any kind of activity or functionality that is executed and thereby changing the properties of the system. LTS can be depicted graphically as, e.g., in Figure 1. The actual process flow is

from top to bottom. E.g., the execution of the transition labeled "a" transforms S_2 into S_4. If more than one outgoing arc leaves a state, there is a choice between the arcs for the continuation of the process. A silent transition, labeled τ, is a special transition that transforms a state into another without changing any of the externally visible properties of the state. Note that in S_1 all three transitions a, b, and c can be executed, in S_2 only a and b can be executed, and in S_3 only c can be executed, i.e., although the τ transitions are not visible they may limit the possible ways to continue.

Although numerous process modeling languages are defined and used, all process models having formal semantics can be mapped onto labeled transition systems [6,14,20]. By using labeled transtition systems for our analysis, we are able to transfer the results into any of these languages.

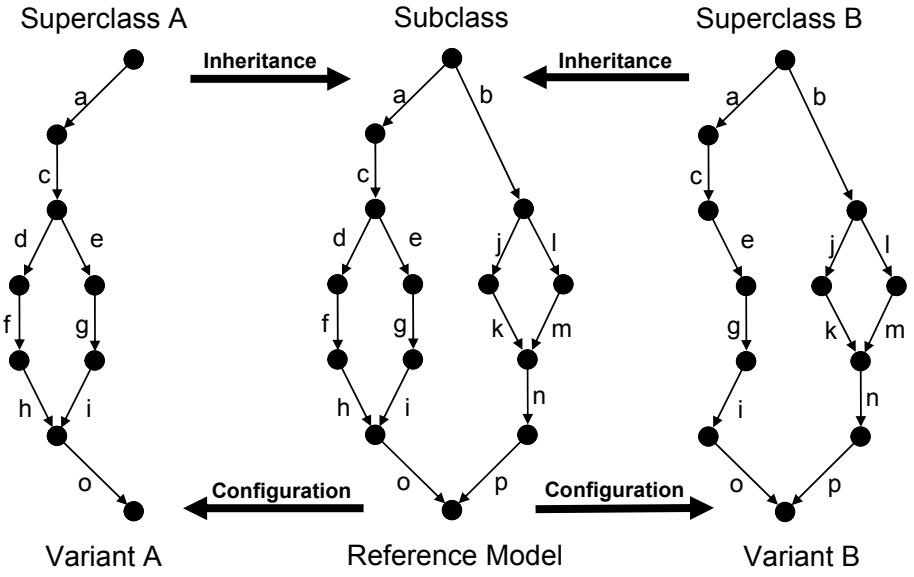


Fig. 2. Configuration - the inverse of inheritance

To depict the essence of configuration we make use of the concepts of *inheritance of dynamic behavior* [1,6]. The basic idea of inheritance – as also applied in object-oriented software development – is to provide a mechanism that allows constructing subclasses which are inheriting all behaviour and features of superclasses. The subclass extends the superclass with additional behavior or features, i.e., the superclass supports less functionality than the subclass. By using multiple inheritance it is also possible that a subclass is the subclass of multiple superclasses. Such a subclass includes the behavior of all superclasses, i.e. from the perspective of each single superclass the subclass is extended with the behaviour of the other superclasses. If such a subclass is minimal (i.e., each extension is motivated by some superclass), we refer to it as the least common multiple of all superclasses. In this paper, we will show that this least common

multiple corresponds to the unconfigured reference model. Each superclass of the subclass (i.e., the reference model) can be regarded as one of its configured variants. That means configuration is the process of transforming the subclass into the superclass, which is exactly the inverse of inheritance (see also Figure 2).

In [1,6] two different mechanisms for detecting inheritance in workflow models are defined. Both mechanisms are defined in the inverse direction. That means the behaviour of the superclass is regarded from the viewpoint of the subclass. The first mechanism inhibits the execution of additionally functionality. If it is not possible to distinguish the behaviors of model x and model y when only transitions of x that are also present in y are executed, then x will be a subclass of y. That means all transitions of the subclass x that are not present in the superclass y are *blocked* (encapsulation). The second mechanism compares the effects of the superclass y and subclass x by considering only that effects of the subclass x that also occur within the superclass y. If it is not possible to distinguish the behaviors of x and y when arbitrary tasks of x are executed, but when only the effects of tasks that are also present in y are considered, then x is a subclass of y. All effects of the subclass x not occurring in y are *hidden* in the superclass y (abstraction).

When configuring a process model, the complete, configurable model is restricted to a desired variant. As the two mechanisms of blocking and hiding are defined in the inverse direction and as we showed that configuration can be regarded as the inverse of inheritance we can use these mechanisms to depict configuration in the following. However, as shown above, configuration implies decision making. Each configuration decision of blocking/not blocking or hiding/not hiding determines if a transition will be executable at run-time or not. A decision, however, requires information which might not be available at build-time. Such decisions must be postponed to run-time and performed for each case individually. Therefore they must be integrated into a run-time (i.e., configured) process model. Thus, a transition can not only be configured as hidden or blocked but also as optional hidden or optional blocked. That means for an LTS:

Definition 2. *A configuration is a (partial) function* $c \in T \nrightarrow \{\tau, \delta, \tau_0, \delta_0\}$ *where* $dom\,(c)$ *is the set of configured transitions, and for* $t \in dom\,(c)$[1]*:*

- $c\,(t) = \tau$*, is a hidden transition,*
- $c\,(t) = \delta$*, is a blocked transition,*
- $c\,(t) = \tau_0$*, is a optionally hidden transition, and*
- $c\,(t) = \delta_0$*, is a optionally blocked transition.*

Of course not all configurations are possible and therefore valid. E.g., for sure certain functionality and therefore certain transitions are mandatory and cannot be blocked or hidden. Also interdependencies between various transitions will probably exist. Therefore we define:

[1] $f \in A \nrightarrow B$ denotes a partial function, $dom\,(f) \subseteq A$ is the domain of f.

Definition 3. *A configurable process model is a tuple* $CPM = (LTS, CS)$ *where:*

- $LTS = (S, L, T, S_I, S_F)$ *is a labeled transition system, and*
- $CS \subseteq T \nrightarrow \{\tau, \delta, \tau_0, \delta_0\}$ *is a set of configurations.*

Configuring the configurable process model means to select a configuration $c \in CS$. In order to get a configured model the configuration must be applied to the labeled transition system. Figure 3 depicts some configuration-examples within an LTS. The first column depicts the configurable models. The subsequent columns depict the configured models of certain configuration scenarios.

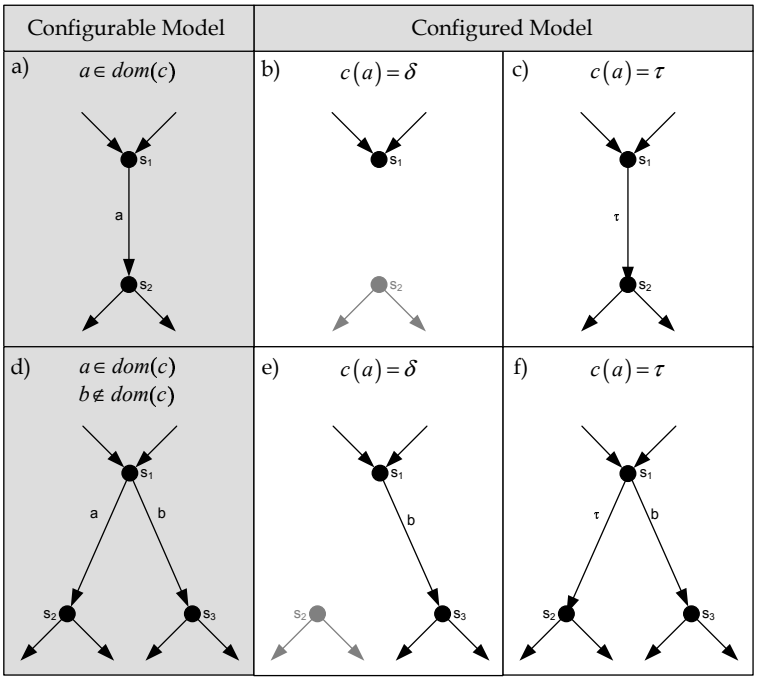


Fig. 3. Configuration in a labeled transition system

The configuration decision to *block* a transition implies that the transition will never be executed. That means the transition should not appear within the configured model. It must be removed from the model when transforming the configurable model into a configured model as depicted in Figure 3a/b. The configurable transition a in Figure 3a is removed in the configured model in Figure 3b. As no alternative transition can be executed from state S_1 the state becomes a deadlock. State S_2 and subsequent transitions and states become unreachable. They could be removed from the configured model, however as they are not reachable anyway this has no influence on the execution of the process. This situation differs from the situation if transition a is configured blocked in the second configurable model (Figure 3d). In this case transition b must be executed when reaching S_1 (Figure 3e).

If the configuration decision is to *hide* a transition, the transition's external, i.e. observable, effects will be ignored. However the effects within the model, that means on the execution of subsequent transitions, are kept. Therefore, when transforming the configurable model into a configured model, the transition must be transformed into a silent step without output by renaming the label into τ (see Figure 3a/c). The definition of hiding given above explicitly says that the task is executed, but the external effect is ignored. However, the desired result when configuring a process model differs slightly. In fact instead of ignoring the transition's external effects it should not even be executed. Only the non-observable, internal effect of reaching a subsequent state and triggering subsequent transitions has to occur. For that reason we will call this kind of configuration also *skipping* in the following. As the perceived results are identical, skipping can be handled in the same manner as hiding by introducing a silent step τ.

If it is not possible to decide on hiding/blocking at configuration time, a configured model can – as depicted above – include the choice between blocking and not blocking or between hiding and not hiding. To include such a postponed choice into the configured model, the choice must of course be included before the actual transition. Each postponed configuration decision needs to be resolved at run-time; either the transition will be hidden/blocked or it will not be hidden/not be blocked. In order to model such as run-time decision, we introduce new intermediate states into the model. Each state corresponds to the result of all postponed decisions in the particular state. We denote these as states $s_{H,NH,B,NB}$ where $H \subseteq T$ is the set of hidden transitions in the particular state, $NH \subseteq T$ are the non-hidden transitions, $B \subseteq T$ are the blocked transitions, and $NB \subseteq T$ are the non-blocked transitions. If it is obvious which transition is referred to, we just use the label to describe a labeled transition, i.e. we write l instead of (s, l, s'). E.g., in Figure 4b the transition labeled a is configured as optional blocked. For that reason two additional states are introduced within the configured model: $s_{1_{\{\},\{\},\{\},\{a\}}}$ for the case that a will not be blocked at run-time and $s_{1_{\{\},\{\},\{a\},\{\}}}$ for the case that a will be blocked at run-time. For the subsequent model each state matches exactly s_1 of the case that it would have been configured blocked or not blocked at build-time, i.e. for example $s_{1_{\{\},\{\},\{a\},\{\}}}$ matches s_1 in Figure 3b. It represents the deadlock. Both states $s_{1_{\{\},\{\},\{a\},\{\}}}$ and $s_{1_{\{\},\{\},\{\},\{a\}}}$ are reachable from s_1 by silent transitions. As these silent transitions have no output, the execution of the model will result in the same process as if the configuration decision would have been made at build-time.

Figure 4c depicts the same situation for the case that transition a is configured as optionally hidden. Here state $s_{1_{\{\},\{a\},\{\},\{\}}}$ represents the situation that transition a is not hidden and will be executed, whereas $s_{1_{\{a\},\{\},\{\},\{\}}}$ represents the result of the configuration decision to hide transition a and therefore corresponds to s_1 in Figure 3c. Figures 4e/f provide further examples by depicting the optional configurations of a in Figures 3e/f.

Figure 4h depicts a situation where more than one transition that is outgoing from the same state is configured optional. In this case it is required to generate 2^n intermediate states, where n is the number of transitions that are configured

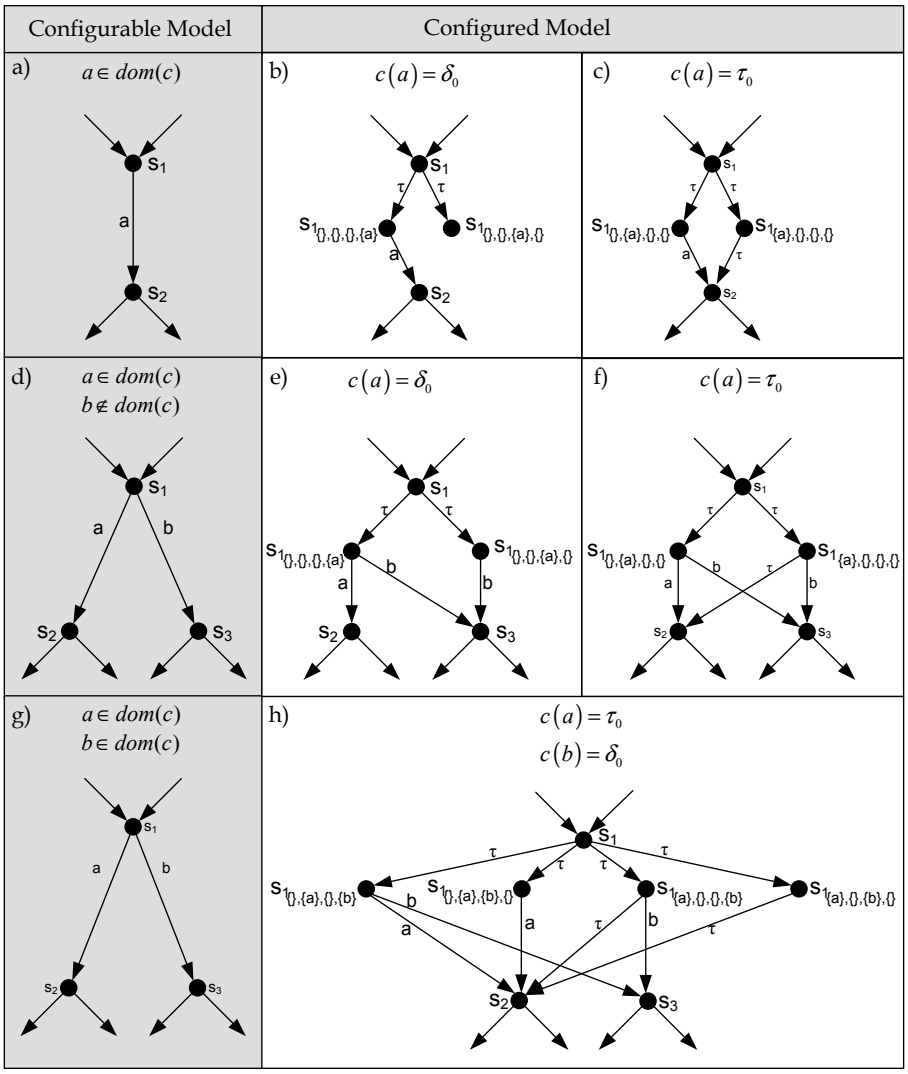


Fig. 4. Transitions configured optional in a labeled transition system

optional. The sets H, NH, B, and NB depict the configurations of the transition. They define which configuration is represented by the particular state.

To transform the configurable process model into a configured process model we provide the following algorithm:

Algorithm 1 *Let $LTS = (S, L, T, S_I, S_F)$ be a labeled transition system and $c \in T \nrightarrow \{\tau, \delta, \tau_0, \delta_0\}$ a configuration. The labeled transition system resulting from this configuration, notation $LTS^c = (S^c, L^c, T^c, S_I^c, S_F^c)$, is defined as follows*

- $T_0 = \{t \in dom\,(c)\,|c\,(t) \in \{\tau_0, \delta_0\}\}$,
- $S_0 = \{s \in S | \exists_{(s',l,s'') \in T_0} s' = s\}$,

- $T' = \{t \in T | t \in dom(c) \Rightarrow c(t) \in \{\tau_0, \delta_0\}\} \bigcup$
  $\{(s, \tau, s') | \exists_{l \in L} (s, l, s') \in dom(c) \wedge c((s, l, s')) = \tau\}$
- $S_{opt} = \{s_{H,NH,B,NB} | s \in S_0$
  $\wedge (H \cap NH = \emptyset)$
  $\wedge (H \cup NH = \{(s', l, s'') \in T_0 | s = s' \wedge c((s', l, s'')) = \tau_0\})$
  $\wedge (B \cap NB = \emptyset)$
  $\wedge (B \cup NB = \{(s', l, s'') \in T_0 | s = s' \wedge c((s', l, s'')) = \delta_0\})$
- $T_{opt} = \{(s, \tau, s_{H,NH,B,NB}) | s_{H,NH,B,NB} \in S_{opt}\} \bigcup$
  $\{(s_{H,NH,B,NB}, l, s') | s_{H,NH,B,NB} \in S_{opt} \wedge l \in (NH \cup NB) \wedge (s, l, s') \in T'\} \bigcup$
  $\{(s_{H,NH,B,NB}, \tau, s') | s_{H,NH,B,NB} \in S_{opt} \wedge \exists l \in H : (s, l, s') \in T'\} \bigcup$
  $\{(s_{H,NH,B,NB}, l, s') | s_{H,NH,B,NB} \in S_{opt} \wedge (s, l, s') \in (T' \setminus T_0) \wedge s \in S_0\}$
- $S^c = S \cup S_{opt}$
- $L^c = L$
- $T^c = \{(s', l, s'') \in T' | s' \notin S_0\} \cup T_{opt}$
- $S^c_I = S_I$
- $S^c_F = S_F$

T_0 is the set of transitions configured as either optionally hidden or optionally blocked. S_0 are all states which are sources of transitions configured as optional. T' are all transitions of the configurable model that are not configured as blocked or hidden, merged with the transitions that are configured as hidden with changed labels to τ. S_{opt} are all the additional intermediate states required for including postponed configuration choices. T_{opt} are all transitions required to include S_{opt} into the model. There are four types of transitions. First, T_{opt} includes all the silent transitions from the original states to the intermediate states. Second, it includes the original transitions repositioned between all new intermediate states in which they are listed in "NB" or in "NH" and their original targets. Third, T_{opt} includes all the renamed, silent transitions from the intermediate states where they are listed in "H" to their original target. Fourth, it also includes all non-configured transitions originally leaving a state in S_0, reallocated between the particular intermediate state and its original destination. Of course, blocked transitions must not be included here.

These sets enable us to specify the configured model. Labels, initial states, and final states remain the same as in the configurable model. The states of the configured model S^c are the states of the unconfigured model S plus the states required for the postponed choices S_{opt}. To define the transitions of the configured model, T^c consists of two types of transitions. First, T^c includes all the non-configured or hidden transitions defined in T', but without the transitions leaving a state that is split into intermediate states. Second, the transitions to include the intermediate states into the model are defined in T_{opt} and also included in T^c.

Some states and transitions become unreachable within such a configured labeled transition system. An additional cleanup algorithm could of course remove these elements. Since this is a trivial technicality it is not shown here.

After clearly defining what configuration of process models is and after defining what configurable process models formally are, we are now able to systematically analyse existing configurable process modelling languages and propose improvements.

4 Configurable EPCs: An example of a language

To conclude this paper we will introduce and analyse *Configurable EPCs* (C-EPCs) [25,12]. C-EPCs serve as an example of a configurable process modeling language and we compare its expressive power with the requirements for process configuration. C-EPCs are an extension of EPCs [16]. An EPC consists of functions, events, arcs and connectors. Events represent states and functions represent activities or functionality. Arcs and connectors define the process flow. Functions follow events and events follow functions. Moreover, to model splits and joins in a process, connectors may be used. There are three types of connectors: AND, OR and XOR. AND-splits and AND-joins may be used to model

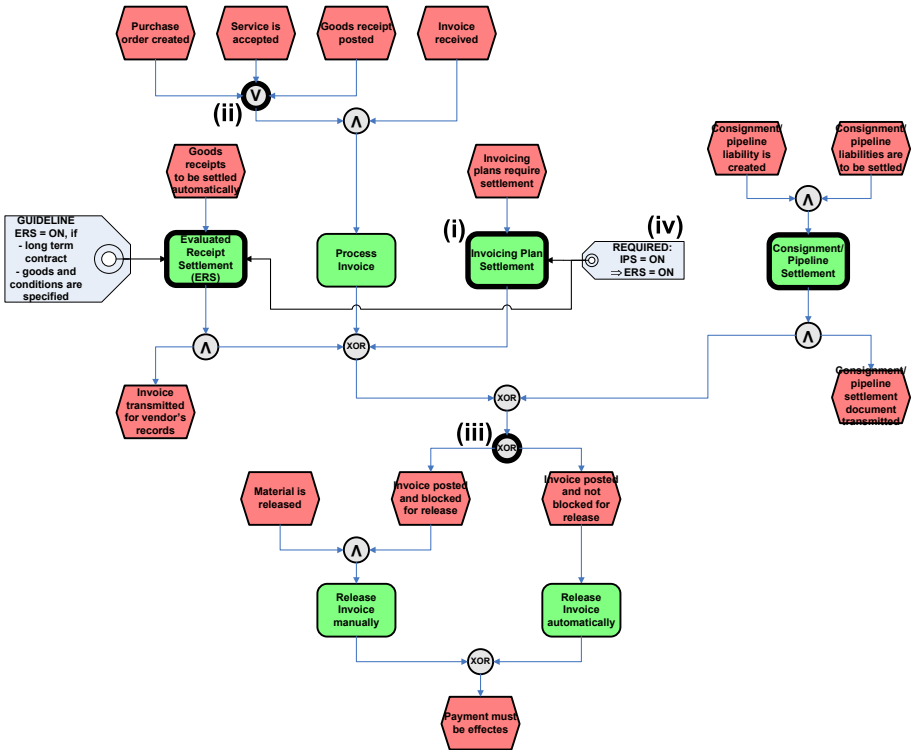


Fig. 5. A Configurable EPC.

parallel routing. XOR-splits and XOR-joins may be used to model the selection of specific routes (e.g., a "switch case" construct). OR-splits and OR-joins may be used to model a mixture of conditional and parallel routing. (The depicted semantics are informal. There is an on-going discussion about mathematical sound semantics of EPCs, especially on the non-locality of the OR-join, e.g. see [18,2].)

In a C-EPC, as defined in [25,12], *both functions and connectors may be configurable*. Configurable functions may be included (ON), skipped (OFF) or conditionally skipped (OPT). Configurable connectors may be restricted at build-time, e.g., a configurable connector of type OR may be mapped onto an AND connector, an XOR-connector or a sequence[2]. Local configuration choices like skipping a function may be limited by configuration requirements. For example, if one configurable function f_1 is configured as "ON", then another configurable function f_2 needs to be excluded. This configuration requirement may be denoted by the logical expression; $f_1 = ON \Rightarrow f_2 = OFF$. In addition to these requirements it is possible to add guidelines, supporting the configuration process.

Figure 5 shows a C-EPC describing an invoice verification process. The classical EPC is extended with configurable functions and connectors (indicated using thick lines) as well as with requirements and guidelines attached to functions. For example function *Invoicing Plan Settlement* (i) is configurable, i.e., it may be included (ON), skipped (OFF) or conditionally skipped (OPT) within the configured model. Note that skipping corresponds to the notion of hiding, i.e., if a function is skipped, the process flow continues after the function without actually executing the function. This is also depicted in the first row of Figure 6. The function a within the C-EPC process fragment is switched "OFF". This conforms to a hidden transition a within the corresponding LTS. Within the configured LTS, the transition is renamed into τ, whereas in the configured EPC function a is removed. In order to generate a lawful EPC also one of the events surrounding a must be removed, which is indicated by the brackets around event A in Figure 6. Comparing the configured EPC and the configured LTS, the sequences of executed activities correspond to each other. Also optional hiding is supported by C-EPCs [19,12]: If a function is configured as "OPT" this means that the decision about its execution is postponed to run-time.

The example C-EPC in Figure 5 also shows two configurable connectors. By configuring the OR-join (ii), it is possible to specify which of the events has to occur in order to start the process. E.g., it is possible to restrict the connector to an AND-join, which would mean that all events have to occur. It is also possible to restrict it to an XOR-join which would mean that only one of the events has to occur. The configurable XOR-split (iii) can be configured to an XOR-connector or it can be restricted to a sequence. E.g., in order to disable automatic invoice release, it can be configured to a sequence only executing the left path (i.e. always performing manual invoice release). In fact this conforms to blocking of the other path leaving the XOR connector. The second row of Figure 6 depicts a process fragment of a corresponding labeled transition system. In the third row of this figure it becomes obvious that direct blocking of functionality is not available

[2] For details see [25], Section 5.2.

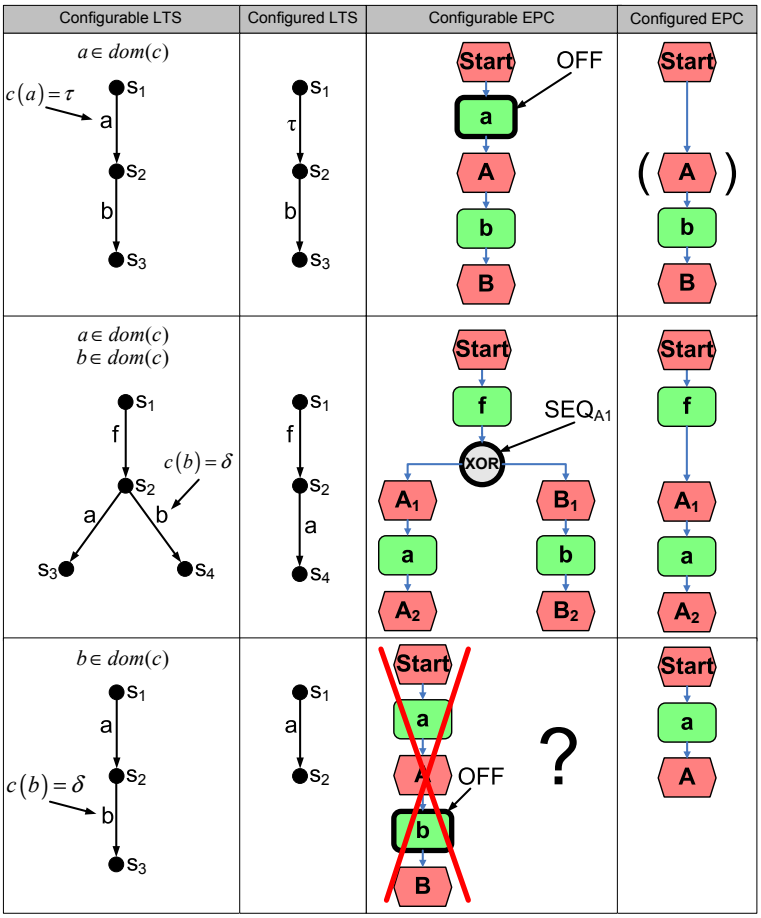


Fig. 6. Configuration within LTSs and C-EPCs.

within C-EPCs. There is no construct available that would enable the blocking of function b as it is in the labeled transition system. A configurable function can only be hidden, but not blocked. That means within C-EPCs blocking is only supported indirectly by configurable connectors. This also becomes obvious if it is required to postpone the choice of blocking to run-time. A configurable connector cannot be configured as optional. However if it is not restricted, it keeps all configuration opportunities. That means, the configuration choice will occur implicitly within the configured model, however it will not be modelled explicitly.

The third element type for configuration within C-EPCs are requirements. E.g., in Figure 5 the requirement attached to "Invoicing Plan Settlement" states that if it is switched "ON" also the function *Evaluated Receipt Settlement* has to be switched "ON" (iv) . Requirements therefore ensure that only configurations are generated that are valid within the configurable process model (see Definition 3).

Altogether the implementation of configuration within C-EPCs by blocking functionality within the process flow and directly skipping functionality without changing the process flow can be seen as a rather restrictive, but very intuitive, approach. Although it does not provide complete support for all possible configuration scenarios, it already provides some support for both configuration techniques blocking and hiding as well as for optional blocking and optional hiding. It also provides opportunities to inhibit generation of invalid process models.

5 Summary and Outlook

Within this paper we have argued that it is required to distinguish between at least two types of choices as the scope and level of decisions varies: (1) configuration choices made at build-time and (2) "normal" choices made at run-time. To allow for a language-independent discussion on configurable process-models we tried to capture the essence of configuration by describing configuration as the inverse of inheritance. Instead of adding functionality, configurations restrict the model. The two techniques used for restriction are called blocking and hiding. Blocking stops the process flow whereas hiding disables functionality by continuing the process flow. As decisions regarding blocking and hiding require information which might not be available at build-time, configurable process modelling languages must support postponement of decisions to run-time.

The important thing to note is that it is possible to extend a language like EPCs with configurable elements, supporting these requirements. Although the current definition of C-EPCs lacks of some configuration opportunities, the extension is intuitive making it easy to apply. The target of this research was to formally define configuration of process models. Further research has to analyze which of these configuration opportunities are sensible in a practical context. A reference model is always a trade-off between costs and benefits. I.e. configuration is the first step from the reference model towards the individual model. Afterwards further specialization has to be done individually by the user in order to include requirements not covered by the reference model.

Using the theory developed within this paper on the one hand and practical experiences using C-EPCs on the other hand, we hope to develop more mature configuration languages. To improve the configuration process of Enterprise systems it will also be required to transfer the presented ideas from process modelling to truly executable models which can be used for enactment. As a starting point, we plan to work on adding configurability features to workflow modelling languages like SAP workflow [22], Staffware [31], or YAWL [5].

References

1. 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, January 2002.

2. W.M.P. van der Aalst, J. Desel, and E. Kindler. On the Semantics of EPCs: A Vicious Circle. In M. Nüttgens and F.J. Rump, editors, *Proceedings of the EPK 2002: Business Process Management using EPCs*, pages 71–80, Trier, Germany, November 2002. Gesellschaft für Informatik, Bonn.
3. W.M.P. van der Aalst, A. Dreiling, F. Gottschalk, M. Rosemann, and M.H. Jansen-Vullers. Configurable Process Models as a Basis for Reference Modeling. In C.Bussler et al., editors, *BPM 2005 Workshops*, volume 3812 of *Lecture Notes in Computer Science*, pages 512–518, Berlin Heidelberg, 2006. Springer Verlag.
4. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA, 2002.
5. W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.
6. T. Basten and W.M.P. van der Aalst. Inheritance of behavior. *Journal of Logic and Algebraic Programming*, 47(2):47–145, 2001.
7. J. Becker, P. Delfmann, and R. Knackstedt. Konstruktion von Referenzmodellierungssprachen: Ein Ordnungsrahmen zur Spezifikation von Adaptionsmechanismen für Informationsmodelle (in German). *Wirtschaftsinformatik*, 46(4):251–264, 2004.
8. J. Becker, M. Kugeler, and M. Rosemann, editors. *Process Management: A Guide for the Design of Business Processes*. Springer-Verlag, Berlin, 2003.
9. P. Bernus. Generalised Enterprise Reference Architecture and Methodology, Version 1.6.3. IFIPIFAC Task Force on Architectures for Enterprise Integration, March 1999.
10. J. vom Brocke and C. Buddendick. Konstruktionstechniken für die Referenzmodellierung (in German). In J. Becker and P. Delfmann, editors, *Referenzmodellierung. Grundlagen, Techniken und domänenbezogene Anwendung, also Proceedings of the 8th Fachtagung Referenzmodellierung*, pages 19–48, Heidelberg, 2004.
11. T. Curran and G. Keller. *SAP R/3 Business Blueprint: Understanding the Business Process Reference Model*. Upper Saddle River, 1997.
12. A. Dreiling, M. Rosemann, W.M.P. van der Aalst, W. Sadiq, and S. Khan. Model-Driven Process Configuration of Enterprise Systems. In O.K. Ferstl, E.J. Sinz, S. Eckert, and T. Isselhorst, editors, *Wirtschaftsinformatik 2005. eEconomy, eGovernment, eSociety*, pages 687–706, Heidelberg, Germany, 2005. Physica-Verlag.
13. P. Fettke and P. Loos. Methoden zur Wiederverwendung von Referenzmodellen – Übersicht und Taxonomie (in German). Arbeitsbericht 52, Instituts für Wirtschaftsinformatik, Münster, 2002.
14. R.J. van Glabbeek and W.P. Weijland. Branching Time and Abstraction in Bisimulation Semantics. *Journal of the ACM*, 43(3):555–600, 1996.
15. S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture, and Implementation*. International Thomson Computer Press, London, UK, 1996.
16. G. Keller, M. Nüttgens, and A.W. Scheer. Semantische Prozeßmodellierung auf der Grundlage Ereignisgesteuerter Prozeßketten (EPK). Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89 (in German), University of Saarland, Saarbrücken, 1992.
17. E. Kindler. On the Semantics of EPCs: A Framework for Resolving the Vicious Circle. In J. Desel, B. Pernici, and M. Weske, editors, *International Conference on Business Process Management (BPM 2004)*, volume 3080 of *Lecture Notes in Computer Science*, pages 82–97. Springer-Verlag, Berlin, 2004.
18. E. Kindler. On the semantics of EPCs: Resolving the vicious circle. *Data & Knowledge Engineering*, 56(1):23–40, January 2006.

19. J. Mendling, J. Recker, M. Rosemann, and W.M.P. van der Aalst. Towards the Interchange of Configurable EPCs: An XML-based Approach for Reference Model Configuration. In J. Desel and U. Frank, editors, *Workshop on Enterprise Modelling and Information Systems Architectures (EMISA 2005)*, volume 75 of *Lecture Notes in Informatics*, pages 8–21, Klagenfurt, Austria, October 2005. Gesellschaft fuer Informatik, Bonn.
20. R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1980.
21. M. Owen and J. Raj. BPMN and Business Process Management – Introduction to the New Business Process Modeling Standard. Technical report, Popkin Software, 2003.
22. A. Rickayzen, J. Dart, C. Brennecke, and M. Schneider. *Practical Workflow for SAP: Effective Business Processes using SAP's WebFlow Engine*. Galileo, 2002.
23. M. Rosemann. *Komplexitätsmanagement in Prozessmodellen: methodenspezifische Gestaltungsempfehlungen für die Informationsmodellierung (in German)*. Wiesbaden, 1996.
24. M. Rosemann. Application Reference Models and Building Blocks for Management and Control (ERP Systems). In P. Bernus, L. Nemes, and G. Schmidt, editors, *Handbook on Enterprise Architecture*, pages 596–616. Springer-Verlag, Berlin, 2003.
25. M. Rosemann and W.M.P. van der Aalst. A Configurable Reference Modelling Language. *Information Systems*, 2005. (to appear, also available via BPMCenter.org).
26. M. Rosemann and R. Schütte. Grundsätze ordnungsmäßiger Referenzmodellierung (in German). Arbeitsbericht 52, Instituts für Wirtschaftsinformatik, Münster, 1997.
27. A.-W. Scheer. *Business Process Engineering, Reference Models for Industrial Enterprises*. Springer-Verlag, Berlin, 1994.
28. A.W. Scheer. *ARIS: Business Process Modelling*. Springer-Verlag, Berlin, 2000.
29. R. Schütte. *Grundsätze ordnungsmäßiger Referenzmodellierung – Konstruktion konfigurations- und anpassungsorientierter Modelle (in German)*. Gabler, Wiesbaden, 1998.
30. A. Schwegmann. *Objektorientierte Referenzmodellierung: theoretische Grundlagen und praktische Anwendung (in German)*. Gabler, Wiesbaden, 1999.
31. Staffware. *Staffware 2000 / GWD User Manual*. Staffware plc, Berkshire, United Kingdom, 2000.